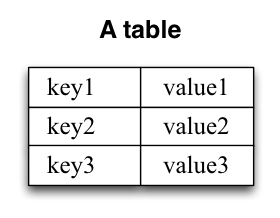
使用Streams DSL构建一个处理器拓扑，开发者可以使用KStreamBuilder类，它是TopologyBuilder的扩展。在Kafka源码的streams/examples包中有一个简单的例子。另外本节剩余的部分将通过一些代码来展示使用Streams DSL创建拓扑的关键的步骤。但是我们推荐开发者阅读更详细完整的源码。

**Duality of Streams and Tables（流和表的对偶性）**

我们讨论Kafka Streams聚合等概念之前，我们必须首先介绍表，和最重要的表和流之间的关系：所谓的流表对偶性。本质上，这种二元性意味着一个流可以被视为一个表，反之亦然。例如，Kafka的日志压缩功能也利用了对偶性。

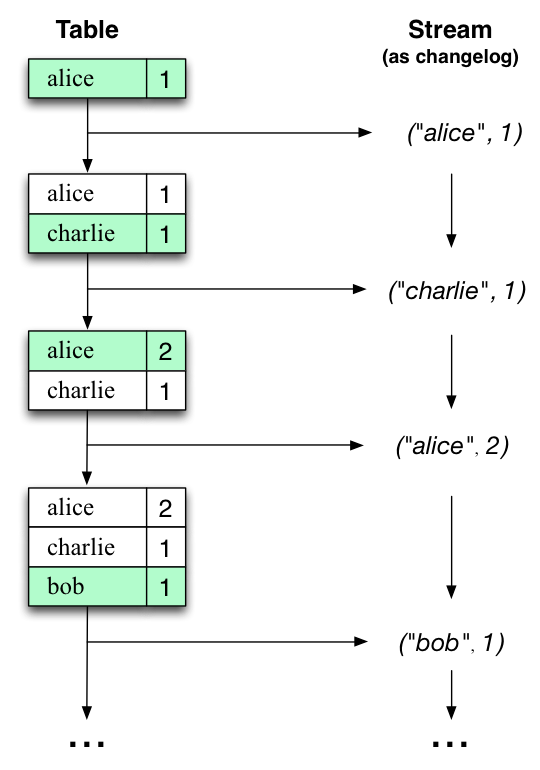
表的格式是一个简单的key-value对的集合，也称为map或关系数组。看起来像这样：

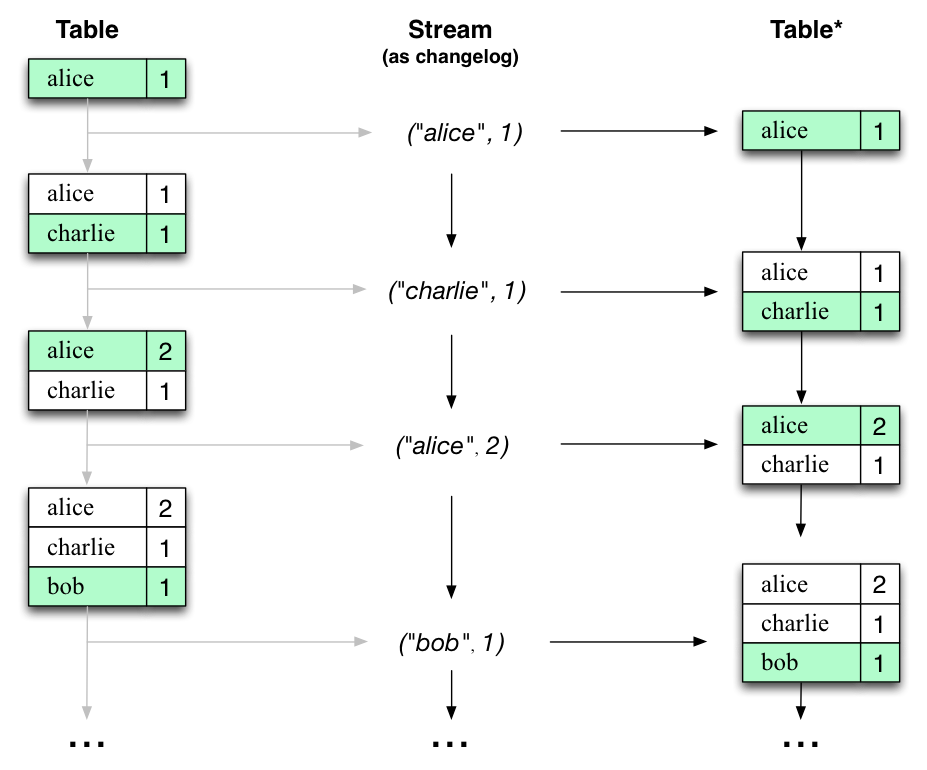


流表二元性描述了流和表之间的紧密关系。

* 流作为表：一个流可以认为是一个表的变更日志，其中在流中的每个的数据记录捕获表的状态变化。因此，流其实是一个伪装的表，并且可以通过从开始到结束重放变更日志来很容地重构“真实”表。同样，在更多类比中，在流中聚合数据记录 - 例如根据用户的访问事件统计总量。- 将返回一个表。（这里的key和value分别是用户和其对应的网页游览量。）
* 表作为流：表可以认为是在流中的每个key的最新value的一个时间点的快照（流的数据记录是key-value对）。因此，表也可以认为是伪装的流，它可以通过对表中每个key-value进行迭代而容易的转换成“真实”流。

让我们用一个例子来说明这一点，假设有一张表，用于跟踪用户的总游览量（下图第一列）。随着时间的推移，每当处理新的网页游览时，相应的更新表的状态。这里，不同时间点之间状态的改变 - 以及表的不同的更新- 表示为变更日志流（第二列）。



有趣的是，由于流表的对偶性，同一个流可以用来重建原始表（第三列）：  


例如，使用相同的机制，通过变更日志捕获（CDC）复制数据库，并在Kafka Streams中，在机器之间复制其所谓的状态存储，以实现容错。  
流表的对偶性是一个重要的概念，Kafka Streams通过KStream，KTable，和GlobalKTable接口模型。我们将在下面的章节中描述。

**KStream, KTable, GlobalKTable**

DSL有3个主要的抽象概念。KStream是一个消息流抽象，其中每个数据记录代表在无界数据集里的自包含数据。KTable是一个变更日志流的抽象，其中每个数据记录代表一个更新。更确切的说，数据记录中的value是相同记录key的最后一条的更新（如果key存在，如果key还不存在，则更新将被认为是创建）。类似于Ktable，GlobalKTable也是一个变更日志流的抽象。其中每个数据记录代表一个更新。但是，不同于KTable，它是完全的复制每个KafkaStreams实例。同样，GlobalKTable也提供了通过key查找当前数据值的能力（通过join操作）。为了说明KStreams和KTables/ GlobalKTables之间的区别，让我们想想一下两个数据记录发送到流中：

("alice", 1) --> ("alice", 3)

假设流处理应用程序是求总和，如果这个是KStream，它将返回4。如果是KTable或GlobalKTable，将返回的是3，因为最后的记录被认为是一个更新动作。

**创建源流**

记录流（KStreams）或变更日志流（KTable或GlobalkTable）可以从一个或多个Kafka主题创建源流，（而KTable和GlobalKTable，只能从单个主题创建源流）。

KStreamBuilder builder = new KStreamBuilder();

KStream<String, GenericRecord> source1 = builder.stream("topic1", "topic2");

KTable<String, GenericRecord> source2 = builder.table("topic3", "stateStoreName");

GlobalKTable<String, GenericRecord> source2 = builder.globalTable("topic4", "globalStoreName");

**Windowing a stream（窗口流）**

流处理器可能需要将数据记录划分为时间段。即，通过时间窗口。通常用于连接和聚合操作等。Kafka Streams当前定义了一下的类型窗口：

* 跳跃时间窗口是基于时间间隔的窗口。此模式固定大小，（可能）重叠的窗口。通过2个属性来定义跳跃窗口：窗口的大小和其前进间隔（又叫“跳跃”）。前进间隔是根据前一个窗口来指定向前移动多少。例如，你可以配置一个跳跃窗口，大小为5分钟，前进间隔是1分钟。由于跳跃窗口可以重叠。因此数据记录可以属于多于一个这样的窗口。
* 滚动时间窗口是跳跃时间窗口的特殊情况，并且像后者一样，也是基于时间间隔。其模型固定大小，非重叠，无间隔窗口。滚动窗口是通过单个属性来定义的：窗口的大小。滚动窗口等于其前进间隔的跳跃窗口大小。由于滚动窗口不会重叠，数据记录仅属于一个且仅有一个窗口。
* 滑动窗口模式是基于时间轴的连续滑动的固定大小的窗口。如果它们的时间戳的差在窗口大小内，则两个数据记录包含在同一个窗口中。因此，滑动窗口不和epoch对准，而是与数据时间戳对准。在Kafka Streams中，滑动窗口仅用于join操作，并且可通过JoinWindows类指定。
* 会话窗口(Session windows)是基于key事件聚合成会话。会话表示一个活动期间，由不活动间隔分割定义的。在任何现有会话的不活动间隔内处理的任何事件都将合并到现有的会话中。如果事件在会话间隔之外，那么将创建新的会话。会话窗口独立的跟踪的key（即，不同key的窗口通常开始和结束时间不同）和它们大小的变化（即使相同的key的窗口大小通常都不同）。因为这样session窗口不能被预先计算，而是从数据记录的时间戳分析获取的。

在Kafka Streams DLS中，开发者可以指定保留窗口的周期。允许保留旧的窗口段一段时间。为了等待晚到的记录（时间戳落在窗口间隔内的）。如果记录过了保留周期之后到达，则不能处理，并将该其删除。

在实时数据流中，晚到的记录始终是可能的。这取决于如何有效的处理延迟记录。利用处理时间，语义是何时处理数据，这意味着延迟记录的概念不适用这个，因为根据定义，没有记录会晚到。因此，晚到的记录实际上可以被认为是事件时间或咽下时间（ingestion-time）。在这两种情况下，Kafka Streams能正常处理晚到的消息。

**Join multiple streams（连接多个流）**

join(连接，加入)操作基于其数据记录的key来合并两个流，并产生一个新的流。在记录流上通常需要在窗口的基础上执行连接，否则为了执行连接必须保持记录的数量可以无限增长。在Kafka Streams中，可以执行以下连接操作：

* KStream对Kstreams连接始终基于窗口，否则内存和状态需要计算加入的无限增长大小。这里，从流中新接收的记录与指定窗口间隔内的其他流的记录相连接，为每个匹配生成一个结果（基于用户提供的ValueJoiner）。新KStream实例表示从此操作者返回join流的结果。
* KTable对KTable连接连接操作设计和关系型数据库中连接操作一致。这里，两个变更日志流首先是本地状态存储。当从流中接收新的记录时，它与其他流的状态仓库相结合，为每个匹配对生成一个结果（基于用户提供的ValueJoiner）。新KTable实例表示连接流的结果，它也代表表的变更日志流，从此操作人返回。
* KStream对KTable连接允许当你从另一个记录流（KStream）接受到新记录时，针对变更日志刘（KTabloe）执行表查询。例如，用最新的用户个人信息（KTable）来填充丰富用户的活动流（KStream)。只有从记录流接受的记录触发连接并通过ValueJoiner生成结果，反之（即，从变更日志流接收的记录将只更新状态仓库）。新的KStream表示该操作者返回的接入结果流。
* KStream对GlobalKTable连接允许你基于从其他记录流（KStream）接受到新记录时，针对一个完整复制的变更日志流（GlobalKTable）执行表查询。连接GlobalKTable不需要重新分配输入KStream，因为GlobalKTable的所有分区在每个KafkaStreams实例中都可用。与连接操作一起提供的KeyValueMapper应用到每个KStream记录，提取用于查找GlobalKTable的连接key，从而可以进行非记录key连接。例如，用最新的用户个人信息（GlobalKTable）来丰富用户活跃流（KStream）。只有从记录流接收的记录触发连接并产生结果（通过ValueJoiner），反之亦然（即，从变更日志流接收的记录仅被用于更新状态仓库）。新的KStream实例代表从该操作者返回的连接结果流。

根据操作数，支持以下连接操作：**内部连接**，**外部连接**和**左连接**。类似于关系型数据库。

**聚合流**

聚合操作采用一个输入流，并通过将多个输入记录合并成单个输出记录来产生一个新的流。计算数量或总数的例子，记录流上通常需要在窗口基础上执行聚合，否则为了执行聚合操作必须保持记录数可以无限地增长。

在Kafka Streams DSL中，聚合操作的输入流可以是KStream或KTable，但是输出流将始终是KTable，允许Kafka Streams在生成或发出之后，最后抵达的记录更新聚合的值。当这种晚到到达的记录发生，聚合KStream或KTtable只是发出一个新的聚合值。由于输出是KTable，所以在后续的处理步骤中，具有key的旧值将被新值覆盖。

**转换流**

除了join（连接）和聚合操作之外，KStream和KTable各自提供其他的转换操作。这些操作每一个都可以生成一个或多个KStream和Ktable对象，并可以转换成一个或多个连接的处理器到底层处理器拓扑中。所有这些转换方法可以链接在一起构成一个复杂的处理器拓扑。由于KSteram和KTable是强类型的，所有转换操作都被定义为泛型，用户可以在其中指定输出和输出数据的类型。

这些转换中，filter,map,myValues等是无状态操作，可应用于KStream和KTable，用户通常可以自定义函数作为参数传递给这些函数，如Predicate的filter，MapValueMapper的map等：

// written in Java 8+, using lambda expressions

KStream<String, GenericRecord> mapped = source1.mapValue(record -> record.get("category"));

无状态转换，不需要处理任何状态。因此在实现上它们不需要流处理器的状态仓库。另一方面，有状态的转换，则需要状态仓库。例如，在连接和聚合操作中，使用窗口状态来存储所有目前为止在定义窗口边界内的所有接收的记录。然后，操作员可以访问这些存储的记录，并基于它们进行计算。

// written in Java 8+, using lambda expressions

KTable<Windowed<String>, Long> counts = source1.groupByKey().aggregate(

() -> 0L, // initial value

(aggKey, value, aggregate) -> aggregate + 1L, // aggregating value

TimeWindows.of("counts", 5000L).advanceBy(1000L), // intervals in milliseconds

Serdes.Long() // serde for aggregated value

);

KStream<String, String> joined = source1.leftJoin(source2,

(record1, record2) -> record1.get("user") + "-" + record2.get("region");

);

**将流写回kafka**

在处理结束后，开发者可以通过KStream.to和KTable.to将最终的结果流（连续不断的）写回Kafka主题。

joined.to("topic4");

如果已经通过上面的to方法写入到一个主题中，但是如果你还需要继续读取和处理这些消息，可以从输出主题构建一个新流，Kafka Streams提供了一个便利的方法，through:

// equivalent to

//

// joined.to("topic4");

// materialized = builder.stream("topic4");

KStream materialized = joined.through("topic4");