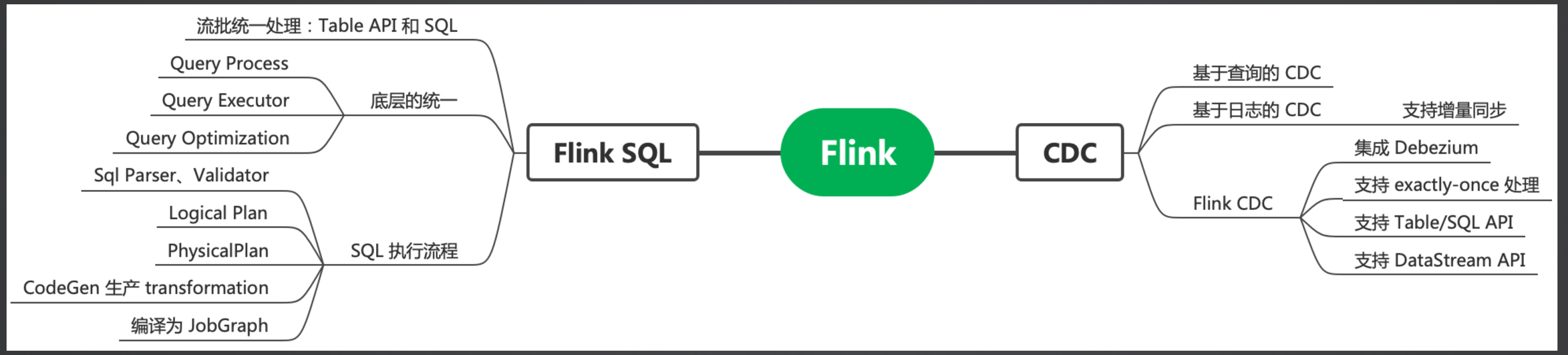
第十二周领教直播 · Flink（下篇）

张语

目录

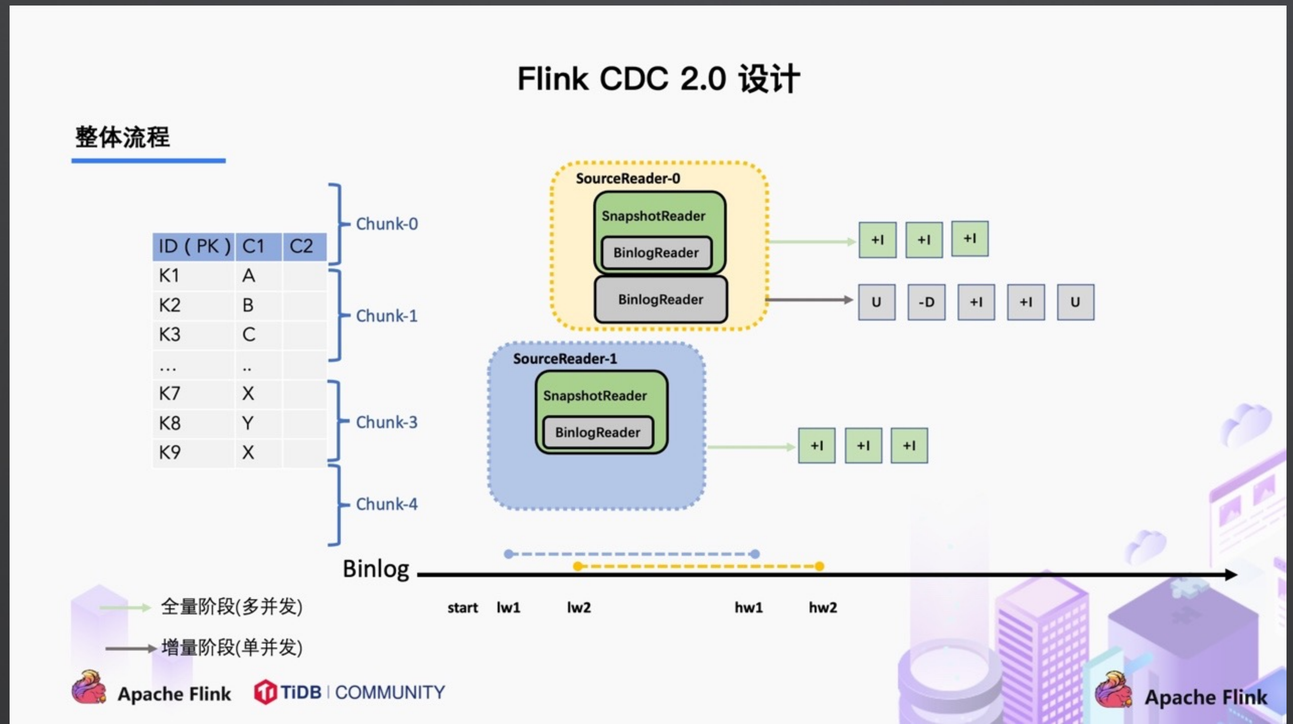
重点内容回顾


流批统一处理：Table API 和 SQL
Query Process
Query Executor
Query Optimization
底层的统一
Sql Parser、Validator
Logical Plan
PhysicalPlan
CodeGen 生产 transformation
编译为 JobGraph
SQL 执行流程
Flink SQL
Flink
CDC
基于查询的 CDC
基于日志的 CDC
支持增量同步
集成 Debezium
支持 exactly-once 处理
Flink CDC
支持 Table/SQL API
支持 DataStream API

Flink CDC

Flink CDC 2.0 的整体流程

1. 通过主键对表进行 Snapshot Chunk 划分，再将 Snapshot Chunk 分发给多个 SourceReader
2. 每个 Snapshot Chunk 读取时通过算法实现无锁条件下的一致性读，SourceReader 读取时支持 chunk 粒度的 checkpoint
3. 在所有 Snapshot Chunk 读取完成后，下发一个 binlog chunk 进行增量部分的 binlog 读取

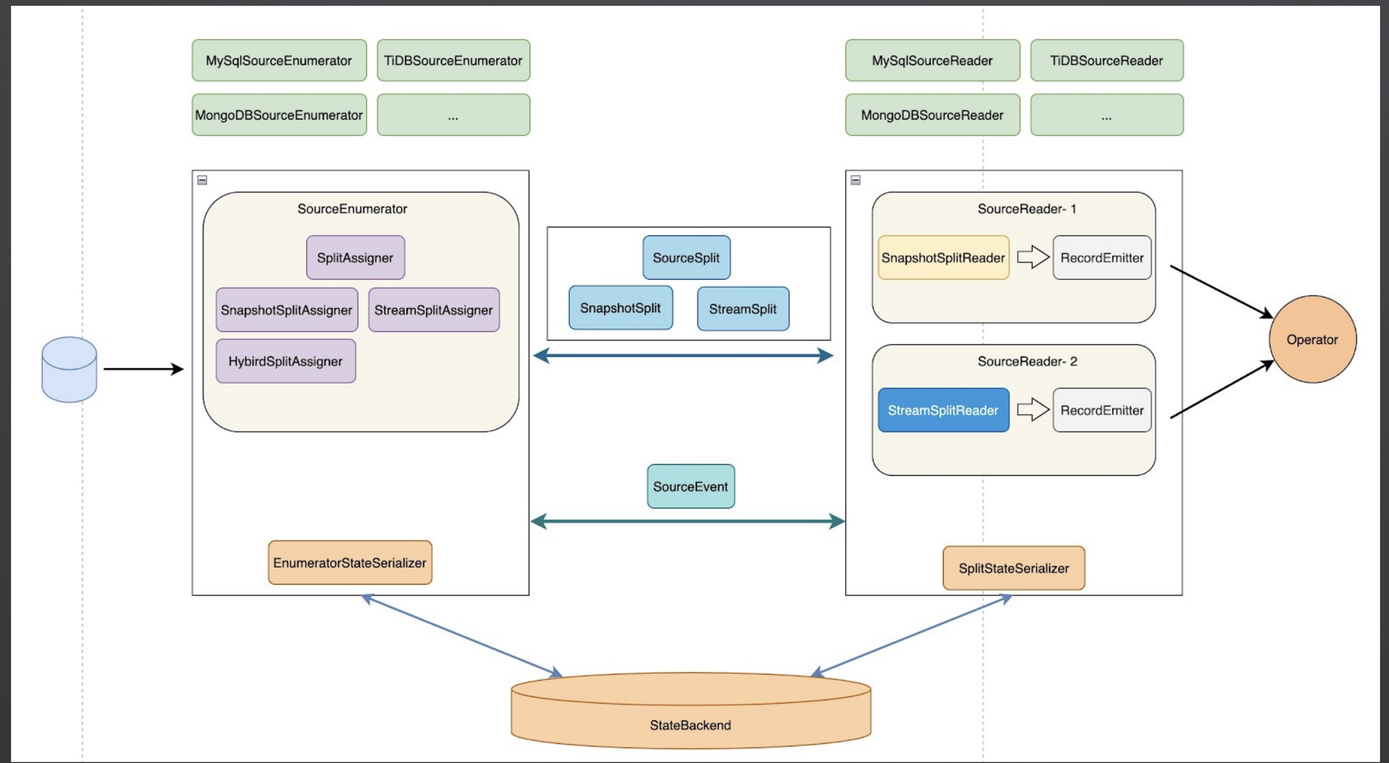


Flink CDC

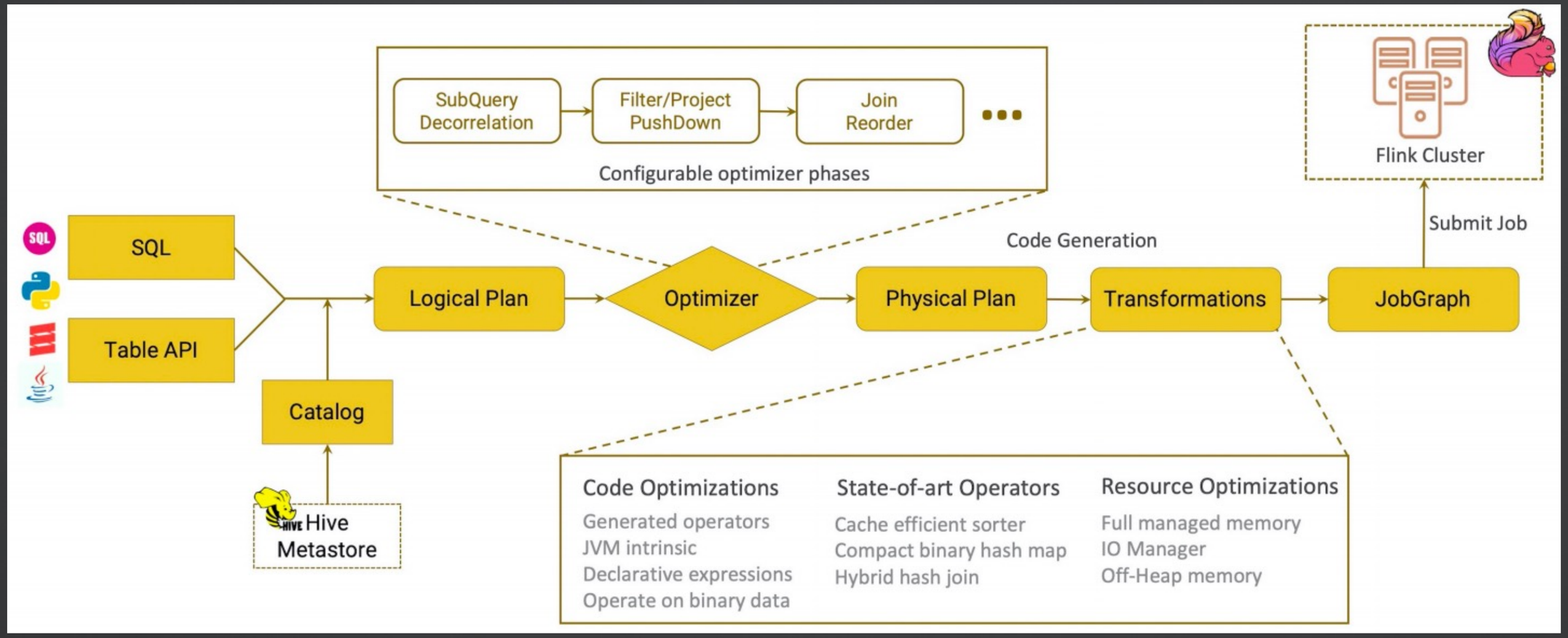
增量快照读取框架

在 2.0 版本只有 MySQL CDC 支持无锁，并发读取，断点续传等高级功能，这些功能基于定制的增量快照读取算法实现

2.2 版本将增量快照读取算法抽象成了公共框架，方便其他 connector 接入。



Flink SQL 执行流程



目录

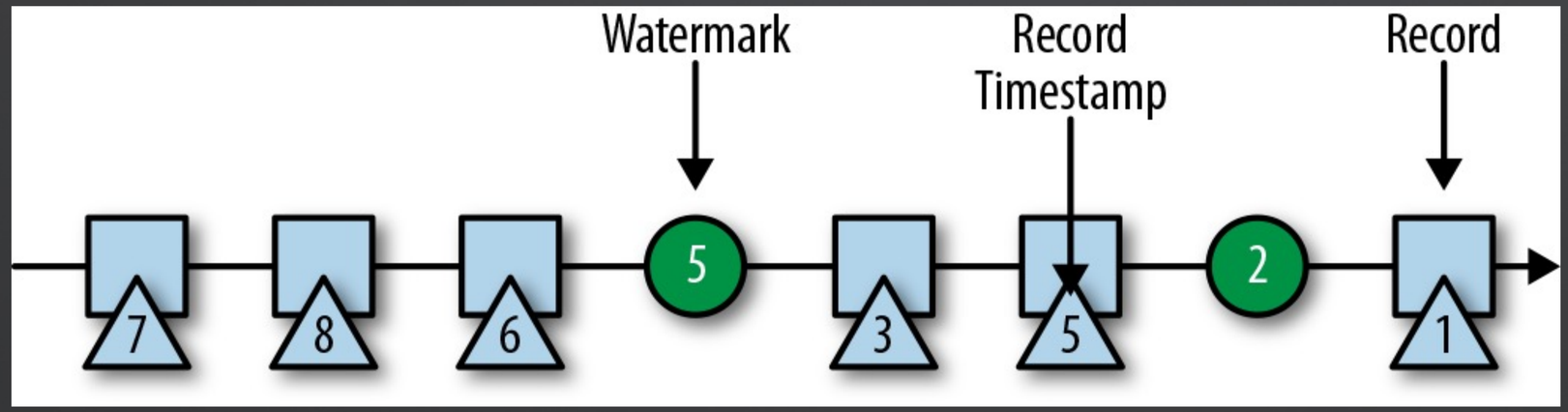
水位线

水位线是全局进度的度量标准。系统可以确信在一个时间点之后，不会有早于这个时间点发生的事件到来了

水位线是利用一些包含 Long 值时间戳的特殊记录来实现的

水位线拥有两个基本属性：

1. 必须单调递增。这是为了确保任务中的事件时间时钟正确前进，不会倒退
2. 和记录的时间戳存在联系。 一个时间戳为 T 的水位线表示，接下来所有记录的时间戳都大于 T



水位线

```java
public final class Watermark extends StreamElement {

    /** The watermark that signifies end-of-event-time. */
    public static final Watermark MAX_WATERMARK = new Watermark(Long.MAX_VALUE);
    /** The watermark that signifies is used before any actual watermark has been generat
    public static final Watermark UNINITIALIZED = new Watermark(Long.MIN_VALUE);

    // ------------------------------------------------------------------------

    /** The timestamp of the watermark in milliseconds. */
    private final long timestamp;

    /** Creates a new watermark with the given timestamp in milliseconds. */
    public Watermark(long timestamp) { this.timestamp = timestamp; }

    /** Returns the timestamp associated with this {@link Watermark} in milliseconds. */
    public long getTimestamp() { return timestamp; }
```

水位线

当任务接收到一个水位线时会执行以下操作：

1. 基于水位线记录的时间戳更新内部事件时间时钟
2. 任务的时间服务会找出所有触发时间小于更新后事件时间的计时器。对于每个到期的计时器，调用回调函数，利用它执行计算或发出记录
3. 任务根据更新后的事件时间将水位线发出

水位线

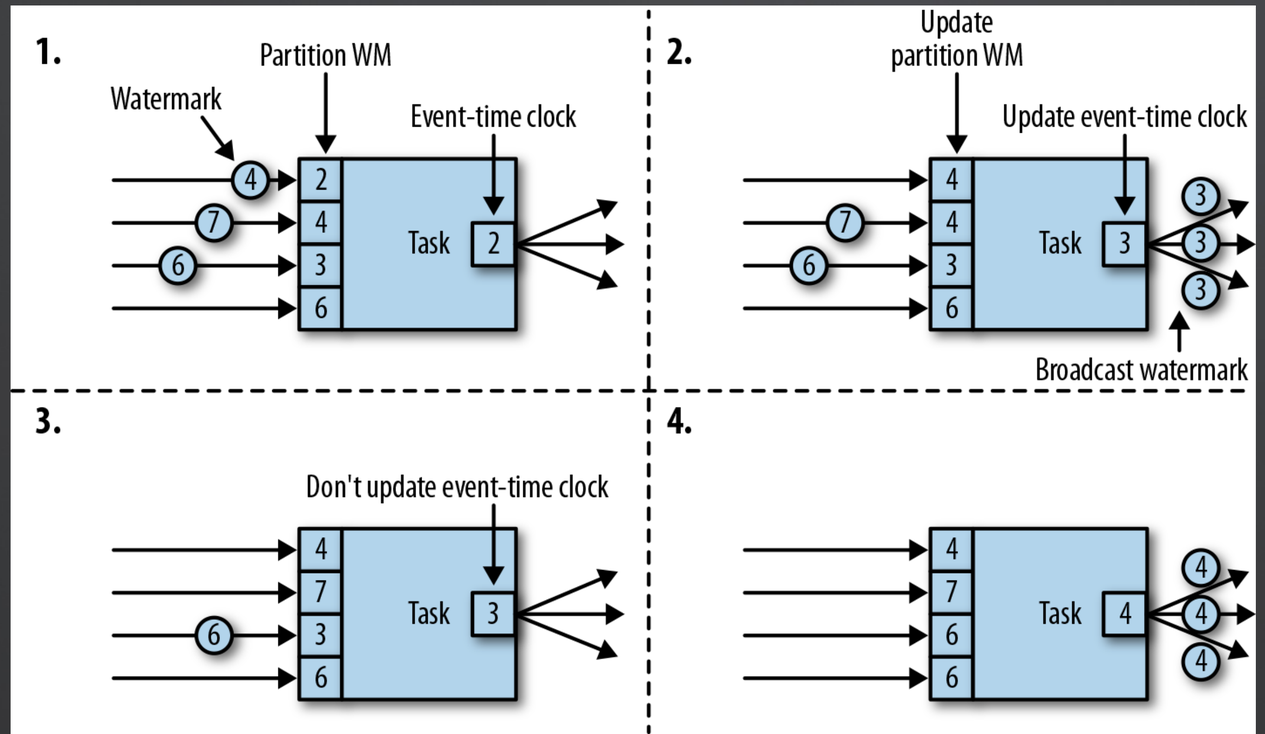
为每个输入分区都维护一个分区水位线。

当收到某个分区传来的水位线后，任务会以接收值和当前值中较大的那个去更新对应分区水位线的值。

任务会把事件时间时钟调为所有分区水位线中最小的那个值。

如果事件时间时钟向前推动，任务会先处理因此而触发的所有计时器

把对应的水位线发往所有连接的输出分区，以实现事件时间到全部下游任务的广播。



水位线生成

在 flink 1.11 中对水位线生成接口进行了重构，统一使用使用 assignTimestampsAndWatermarks 方法来构造水位线，新的接口需要传入一个 WatermarkStrategy 对象

WatermarkStrategy 内置的水位线生成策略

固定延迟：forBoundedOutOfOrderness(Duration maxOutOfOrderness)

单调递增： forMonotonousTimestamps()，以事件中的时间戳充当了水位线

提取时间戳

TimestampAssigner 接口，用于从读入流式应用的元素中提取时间戳。应该在数据源函数后面立即调用时间戳分配器，因为大多数分配器在生成水位线的时候都会做出一些有关元素顺序相对时间戳的假设。

```
public interface TimestampAssigner<T> {

    /** The value that is passed to {@link #extractTimestamp
    long NO_TIMESTAMP = Long.MIN_VALUE;

    /** Assigns a timestamp to an element, in milliseconds s
    long extractTimestamp(T element, long recordTimestamp);
}
```

处理空闲数据源

由于数据产生的比较少，导致一段时间内没有数据产生，进而就没有水位线的生成，导致下游依赖水位线的一些操作就会出现问题。
比如某一个算子的上游有多个算子，水位线是取其上游两个算子的较小值，如果上游某一个算子因为缺少数据迟迟没有生成水位线，会导致下游没法触发计算。

通过 WatermarkStrategy#withIdleness() 方法允许在配置的时间内没有记录到达时将一个流标记为空闲。这样就意味着下游的数据不需要等待水位线的到来。当下次有水位线生成并发射到下游的时候，这个数据流重新变成活跃状态。

处理迟到事件

迟到事件是乱序事件的特例，和一般乱序事件不同的是它们的乱序程度超出了水位线的预计，导致窗口在它们到达之前已经关闭。

迟到事件出现时窗口已经关闭并产出了计算结果，处理的方法有：

1. 重新激活已经关闭的窗口并重新计算以修正结果。
2. 将迟到事件收集起来另外处理。
3. 将迟到事件视为错误消息并丢弃。

Side Output 机制：将迟到事件单独放入一个数据流，作为 window 计算结果的副产品，以便对其进行特殊处理。

Allowed Lateness 机制：允许设置一个最大迟到时长。Flink 会在窗口关闭后一直保存窗口的状态直至超过迟到时长，这期间的迟到事件不会被丢弃，而是会触发窗口重新计算。

DataStream 示例

1. 基于 flink-operations-playground 修改

演示环节

目录

Window TVF

窗口是处理无限流的核心。窗口将流分成有限大小的 "桶"，可以对其进行计算。

Flink提供了窗口表值函数（ table-valued functions，TVF），将表的元素划分为窗口，包括:

- Tumble Windows
- Hop Windows
- Cumulate Windows

TVF 基于原来的关系表增加了三列：window_start，window_end，window_time

TUMBLE

TUMBLE函数根据时间属性字段为关系的每一行分配一个窗口。

TUMBLE(TABLE data, DESCRIPTOR(timecol), size [, offset])

- data：表名
- timecol：时间列
- size：窗口大小

TUMBLE

```
Flink SQL> desc Bid;
+------------+-------------------------+------+-----+--------+--------------------------------+
|       name |                    type | null | key | extras |                      watermark |
+------------+-------------------------+------+-----+--------+--------------------------------+
|    bidtime | TIMESTAMP(3) *ROWTIME* | true |     |        | `bidtime` - INTERVAL '1' SECOND |
|      price |           DECIMAL(10, 2) | true |     |        |                                |
|       item |                   STRING | true |     |        |                                |
+------------+-------------------------+------+-----+--------+--------------------------------+
```

```
SELECT * FROM TABLE(
  TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES));
```

```
SELECT window_start, window_end, SUM(price)
 FROM TABLE(
  TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
 GROUP BY window_start, window_end;
```

HOP

HOP函数将元素分配给固定长度的窗口。有窗口大小、窗口滑动两个参数

HOP(TABLE data, DESCRIPTOR(timecol), slide, size [, offset])

- data：表名
- timecol：时间列
- slide：指定连续跳转窗口开始之间的持续时间
- size：窗口大小

HOP

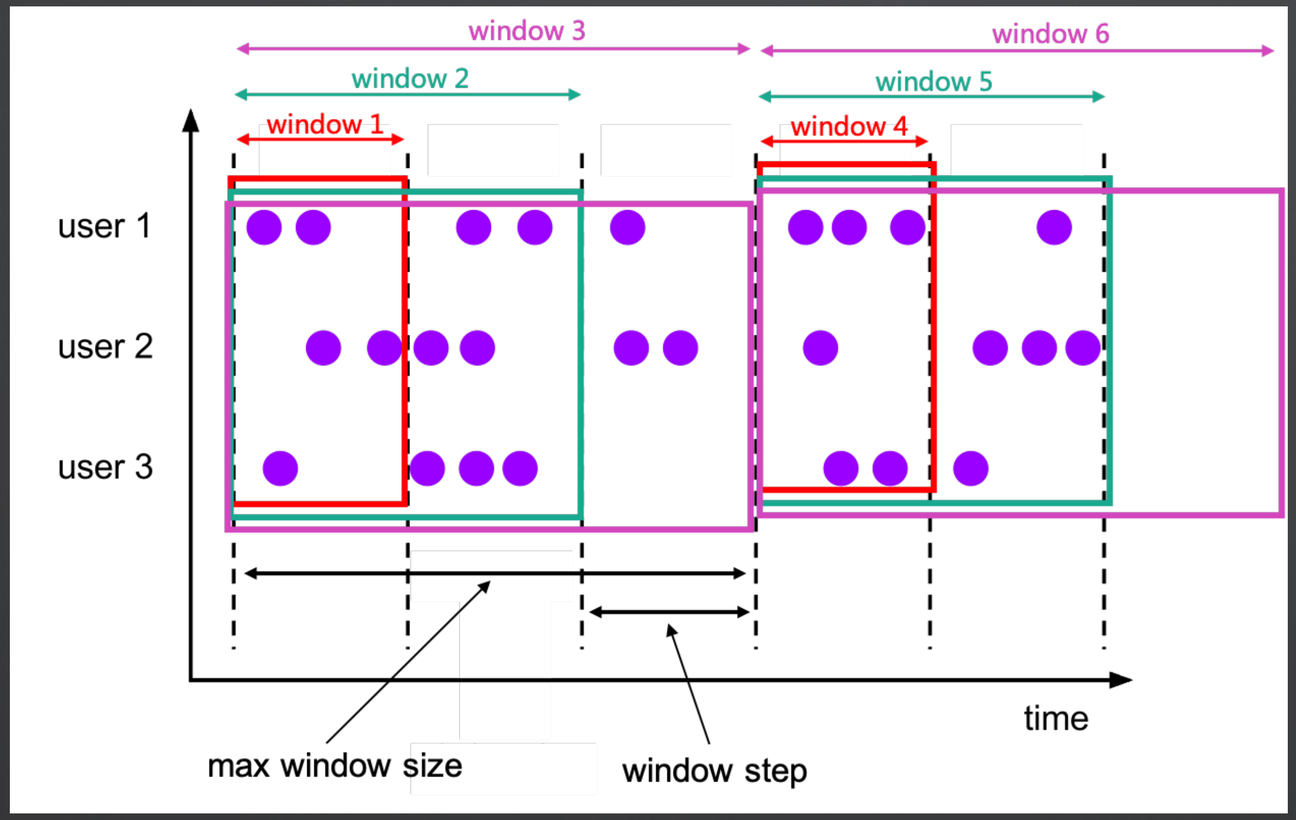
```
SELECT * FROM TABLE(
    HOP(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '5' MINUTES, INTERVAL '10' MINUTES));

SELECT window_start, window_end, SUM(price)
  FROM TABLE(
    HOP(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '5' MINUTES, INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
```

CUMULATE

CUMULATE函数将元素分配到窗口中，这些窗口覆盖了步长的初始区间内的行，并且每一步都扩展到多一个步长（保持窗口起点固定），直到最大窗口尺寸。

比如：统计截止到上一小时的当天PV



CUMULATE

CUMULATE(TABLE data, DESCRIPTOR(timecol), step, size)

- data：表名
- timecol：时间列
- step：增长的窗口大小
- size：累计窗口的最大，size 必须是 step 的倍数

```
SELECT * FROM TABLE(
   CUMULATE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '2' MINUTES, INTERVAL '10' MINUTES));

SELECT window_start, window_end, SUM(price)
  FROM TABLE(
   CUMULATE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '2' MINUTES, INTERVAL '10' MINUTES))
  GROUP BY window_start, window_end;
```

CUMULATE

统计 UV 例子

```
INSERT INTO cumulative_UV
SELECT WINDOW_end,COUNT(DISTINCT user_id) as UV
FROM Table(
    CUMULATE(Table user_behavior,DESCRIPTOR(ts),INTERVAL '10' MINUTES,INTERVAL '1' DAY))
)
GROUP BY WINDOW_start,WINDOW_end
```

Window TVF

1. 基于 Table Walkthrough 修改

演示环节

Top-N

Flink使用 OVER 窗口子句和过滤条件的组合来表达一个Top-N查询。

借助 OVER 窗口 PARTITION BY子句，Flink还支持每组Top-N

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     ROW_NUMBER() OVER ([PARTITION BY col1[, col2...]]
       ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
   FROM table_name)
WHERE rownum <= N [AND conditions]
```

Top-N

Flink使用 OVER 窗口子句和过滤条件的组合来表达一个Top-N查询。

借助 OVER 窗口 PARTITION BY子句，Flink还支持每组Top-N

ROW_NUMBER()：根据分区内行的顺序，为每一行分配一个唯一的、连续的数字

PARTITION BY col1[, col2...]：指定分区的列。每个分区将有一个Top-N的结果

ORDER BY col1 [asc|desc][, col2 [asc|desc] ...]：指定排序列。在不同的列上，排序方向可以是不同的

Top-N

每个类别中销量最大的前五个产品的实时情况

```
CREATE TABLE ShopSales (
  product_id   STRING,
  category     STRING,
  product_name STRING,
  sales        BIGINT
) WITH (...);

SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY category ORDER BY sales DESC) AS row_num
  FROM ShopSales)
WHERE row_num <= 5
```

Window Top-N

窗口Top-N是一个特殊的Top-N，它返回每个窗口和其他分区键的N个最小或最大值。

```
SELECT [column_list]
FROM (
   SELECT [column_list],
     ROW_NUMBER() OVER (PARTITION BY window_start, window_end [, col_key1...]
       ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
   FROM table_name) -- relation applied windowing TVF
WHERE rownum <= N [AND conditions]
```

Window Top-N

如何计算在每10分钟滚动窗口中销售额前3的供应商。

```
SELECT *
  FROM (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY window_start, window_end ORDER BY price DESC) as
rownum
    FROM (
      SELECT window_start, window_end, supplier_id, SUM(price) as price, COUNT(*) as cnt
      FROM TABLE(
        TUMBLE(TABLE Bid, DESCRIPTOR(bidtime), INTERVAL '10' MINUTES))
      GROUP BY window_start, window_end, supplier_id
    )
  ) WHERE rownum <= 3;
```

QA

极客时间 训练营

THANKS