实时流处理引擎Slipstream

范 颖 捷 | 2018年7月

TRANSWARP
星 环 科 技

TRANSWARP
星 环 科 技

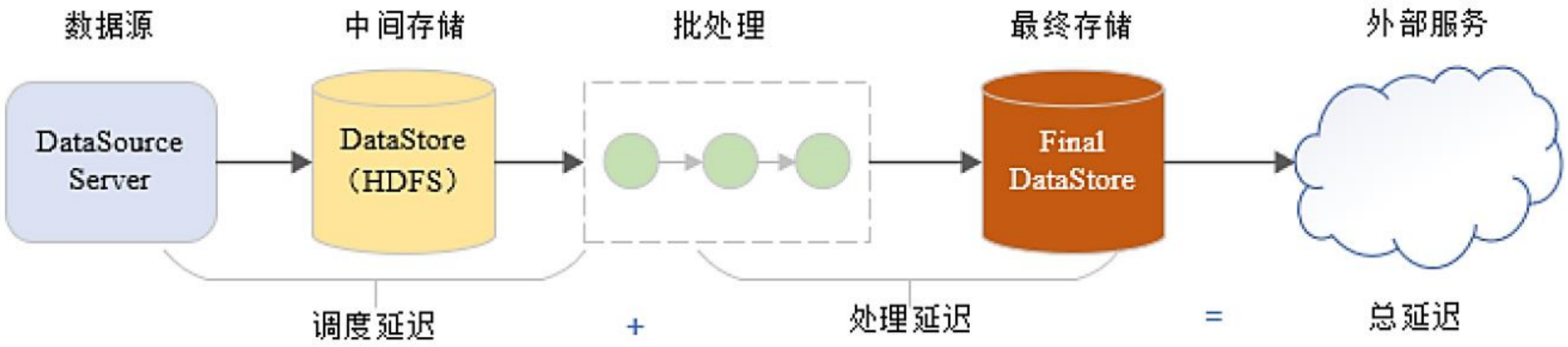
目录》
CONTENTS

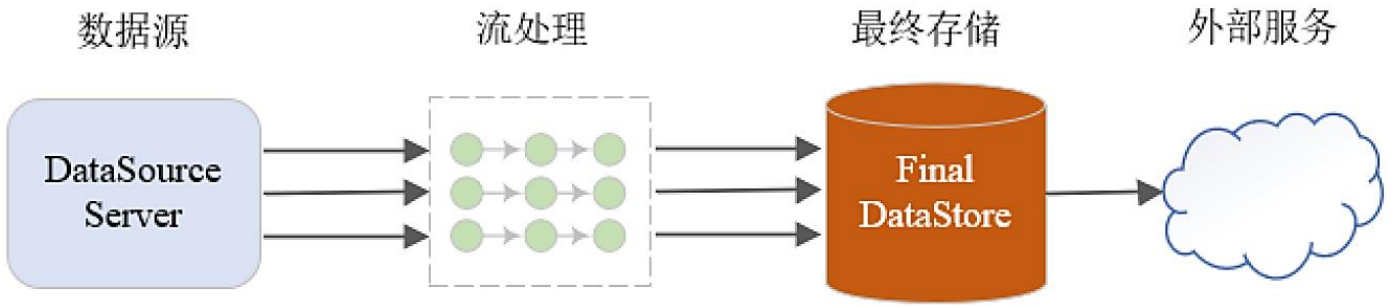
1
chapter

Slipstream基础

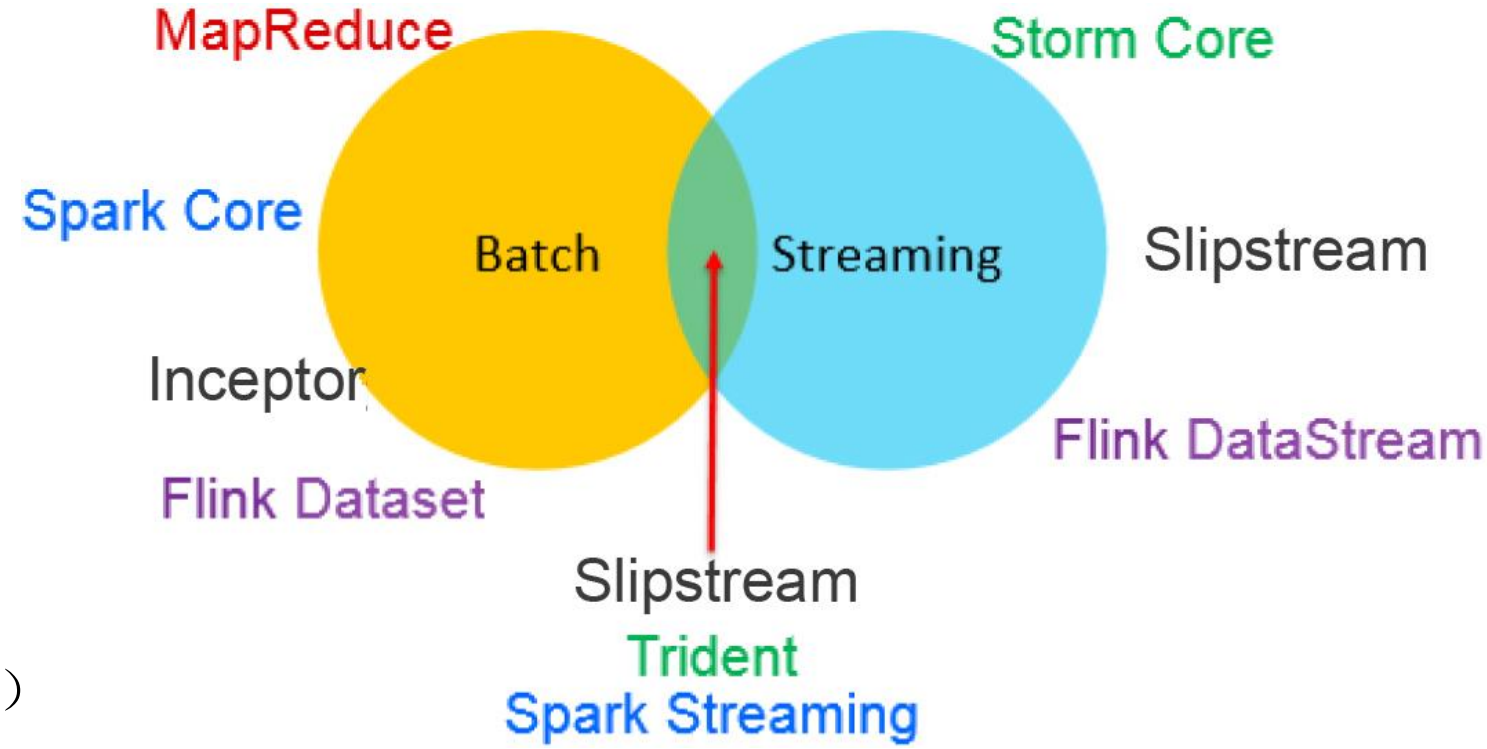
- ✓ 背景介绍
- ✓ Slipstream简介
- ✓ Slipstream基本概念
- ✓ StreamSQL vs. 普通SQL

- 批处理
  - 批量处理
  - 调度延时
  - 处理延时
- 流处理
  - 流式处理
  - 低延时

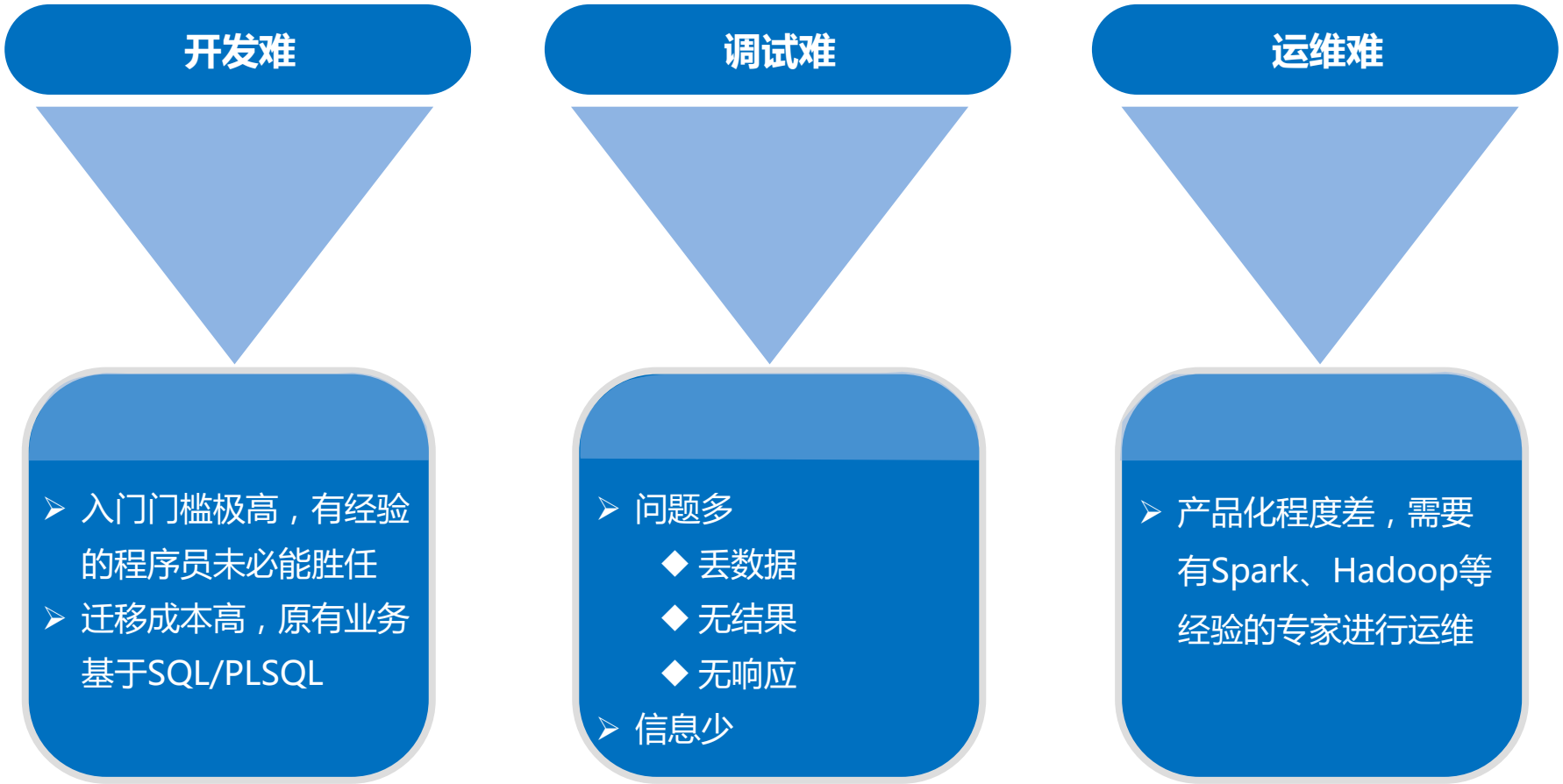




- 批处理计算框架
  - MapReduce
  - Spark Core
  - Inceptor
  - Flink Dataset
- 流式计算框架
  - 基于事件驱动
    - Storm Core
    - Slipstream
    - Flink DataStream
  - 基于微型批处理（Micro-batch）
    - Spark Streaming
    - Slipstream
    - Storm Trident



- 流式计算框架存在的问题



➢ 定位

- 融合事件驱动与微批处理的实时流计算引擎
- 分布式流式SQL引擎

➢ 特点

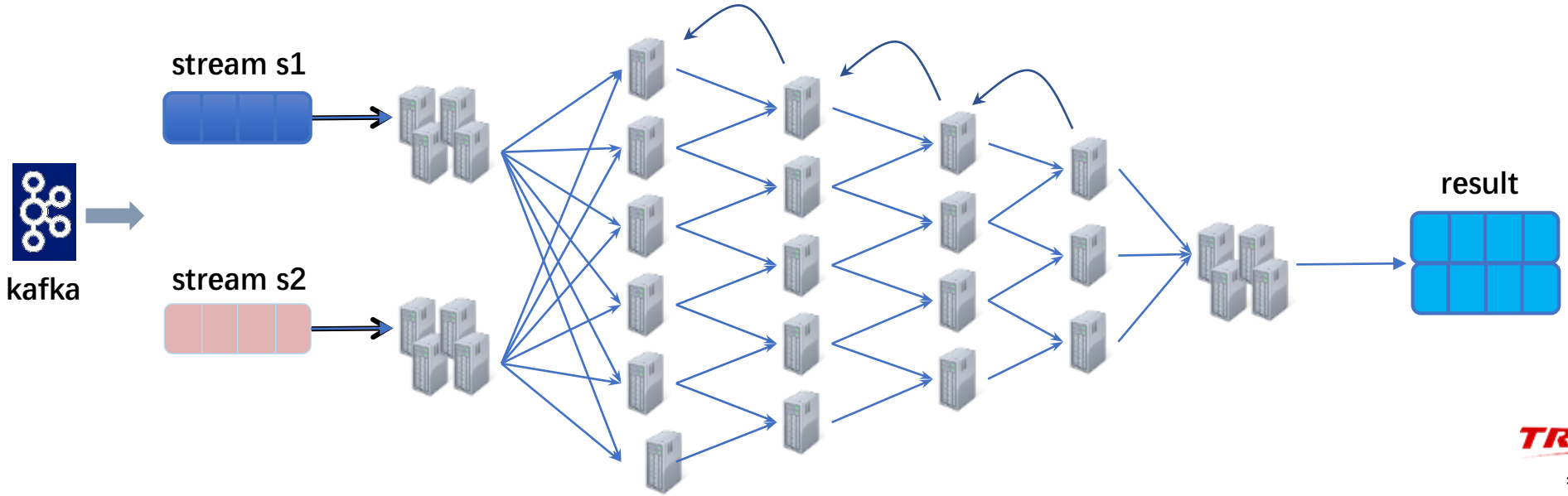
- 微批模式和事件驱动模式的一体化
  - 两种流处理模式可从容应对不同延时级别的业务场景，微批→秒级，事件驱动→毫秒级
  - 在同一套系统里，可以根据业务需求，通过参数配置和SQL改写，灵活切换两种流处理模式
- 支持分布式流式SQL（StreamSQL）
  - 支持99%的ANSI SQL 2003，95%的PL/SQL Oracle与PL/SQL DB2
  - 支持通过SQL创建并操作流，并支持在流上做存储过程等复杂逻辑处理
  - 降低流应用开发门槛，提高流应用开发效率
  - 帮助用户低成本将传统业务逻辑变成流应用

➢ 特点

- 强大的优化器提升性能
  - 包含3级优化器：基于规则的优化器、基于代价的优化器、代码生成器
  - 在一些条件下，采用StreamSQL可以获得比编程方式更高的性能，因为Slipstream做了特殊优化，编程无法轻易实现，例如：增加迭代框架
- 极高的易用性
  - 只要有编写普通SQL的经验，就可以写出高效、稳定、安全的流处理应用
- 产品化程度高
  - StreamSQL作为一个通用的接口显著提升了产品化程度
- 迁移成本低
  - 用户原有的业务逻辑很多是通过SQL实现的，系统迁移时只需修改SQL，迁移成本大大降低

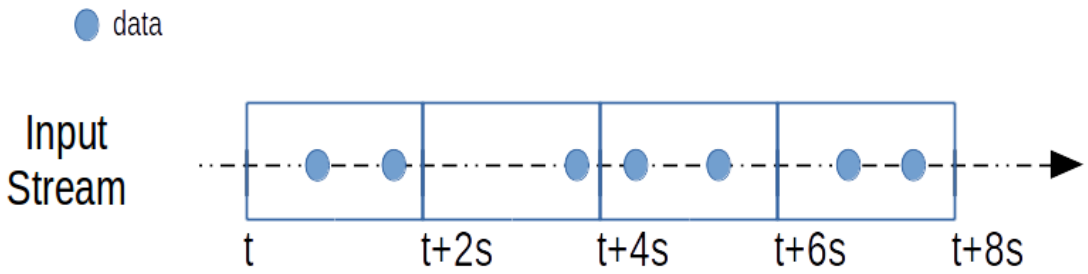
Slipstream的三个核心概念

- Stream（数据流）
- StreamJob（流任务）
  - 对一个或多个Stream进行计算，并将结果写入一张表的任务
- Application（流应用）
  - 一组业务逻辑相关的StreamJob的集合



➢ Stream（数据流）

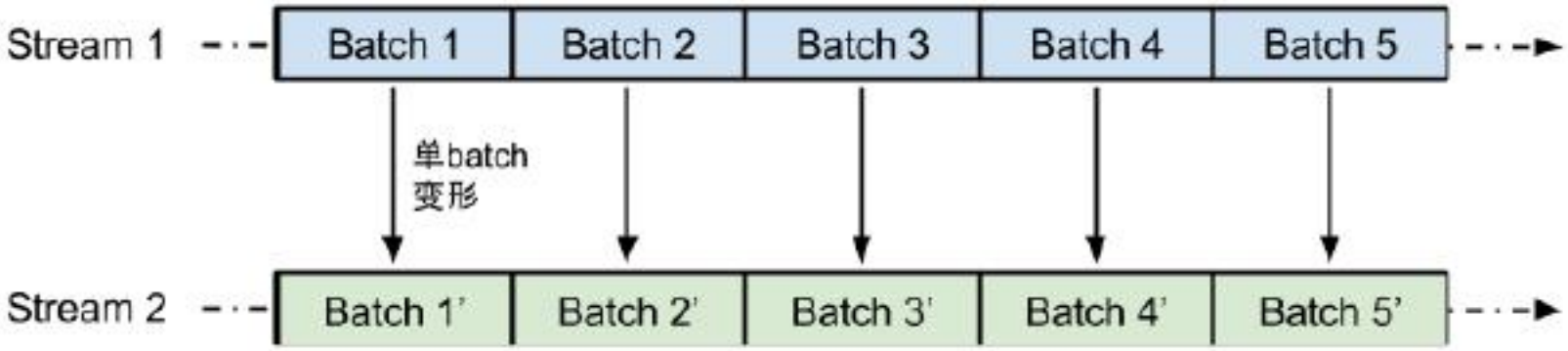
- Input Stream（输入流）
  - 定义：直接接收数据源传来的数据而形成的Stream
  - 功能：定义了如何从数据源读取数据，如：Kafka、Socket
  - 微批（Micro-batch）模式：将Input Stream按时间划分成若干小数据块（Batch）来处理，即在由若干单位时间组成的时间间隔内，将接收的数据放到一个Batch中（Batch的时间长度称为Batch Duration）
  - 事件驱动（Event-driven）模式：以单条数据被Input Stream接收为事件，逐条读取并处理



- Stream（数据流）
  - Derived Stream（衍生流）
    - Derived Stream由StreamSQL语句对已有的Stream变形（Transform）得来
    - Stream变形通常由CSAS（Create Stream As Select）语句完成
    - 微批模式

      ①含义：Stream变形是从已有Batch计算得到新Batch的过程

      ②单Batch变形：对Stream中单个Batch进行计算得到新Batch的过程



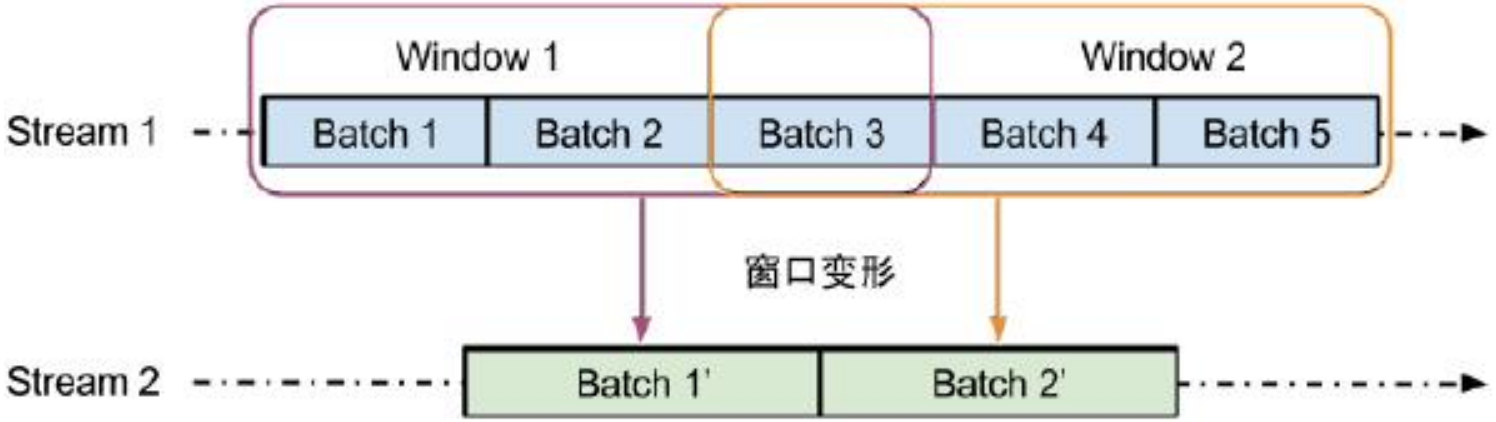
- Stream（数据流）
  - Derived Stream（衍生流）
    - 微批模式

③窗口变形（多Batch变形）

＊对一个时间窗口（Window）内的多个Batch进行计算得到新Batch的过程

＊Window Stream：通过窗口变形得到的Derived Stream

＊两个重要参数：Length和Slide，Length是窗口持续时间，Slide是两个相邻窗口之间的间隔时间，Length和Slide必须是Batch Duration的倍数



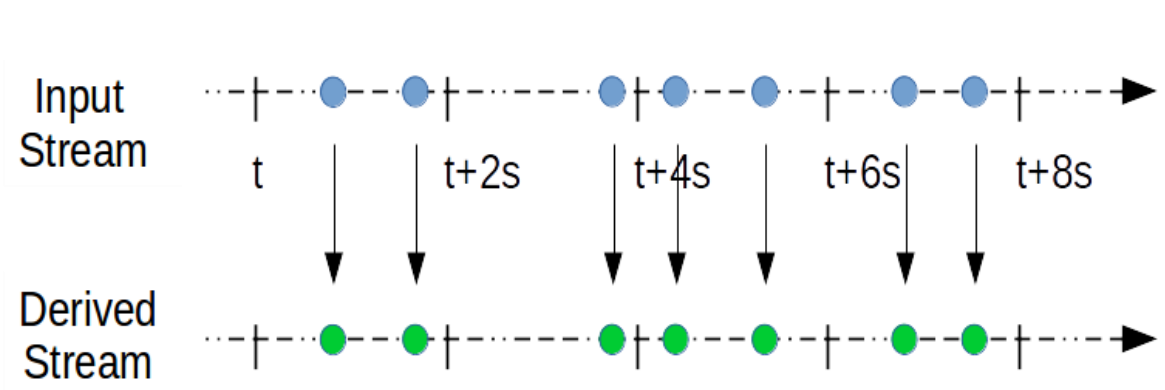
➢ Stream（数据流）

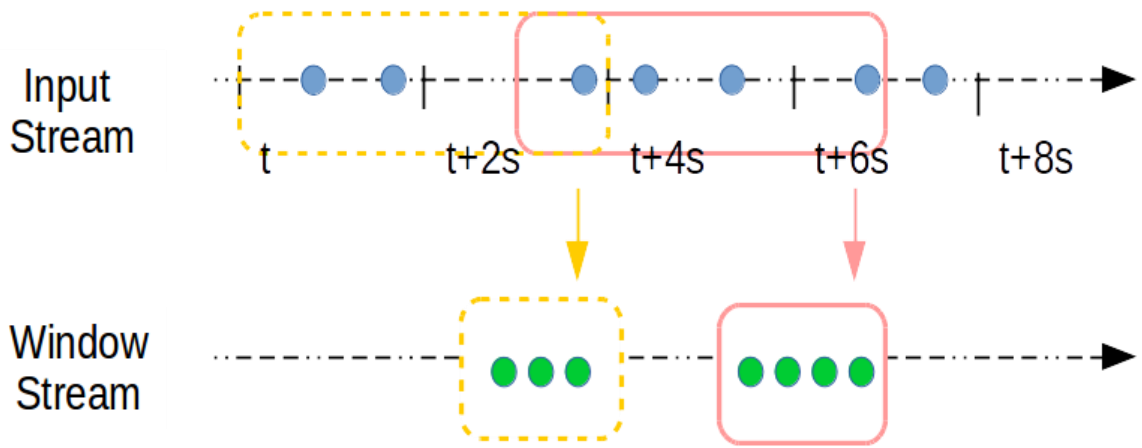
- Derived Stream（衍生流）
  - 事件驱动模式

①含义：每得到一条数据就对其进行变形，得到Derived Stream

②单数据变形：对Stream中单条数据进行计算得到新数据的过程

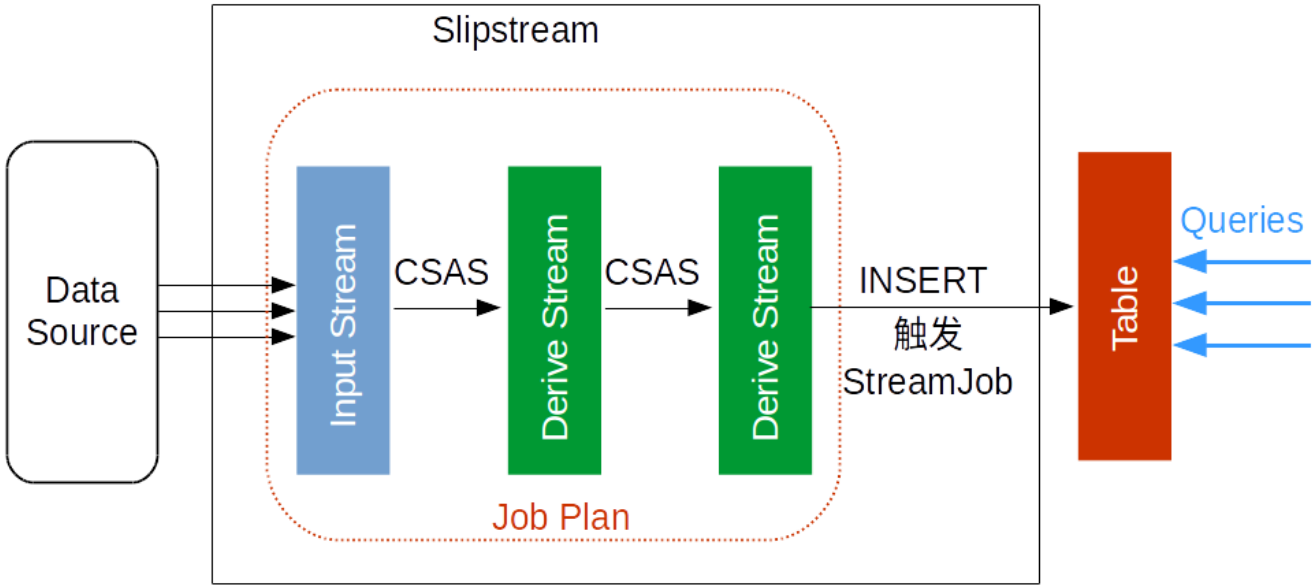
③窗口变形（多数据变形）：对一个时间窗口（Window）内的多条数据进行计算得到新数据的过程





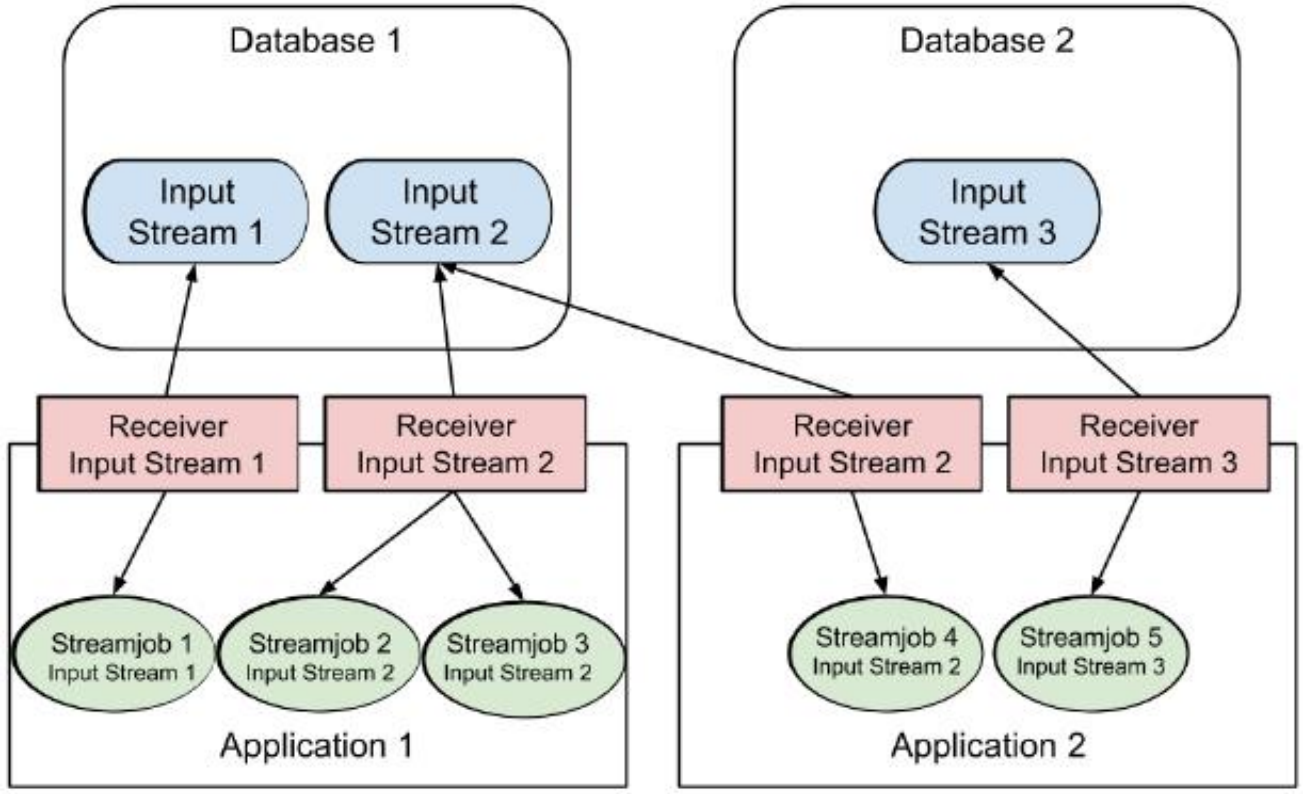
StreamJob（流任务）

- StreamSQL中的Stream是静态的，它仅仅描述了如何对数据源传来的数据进行接收和变形的计划，但并不执行这些计划
- 要让StreamSQL执行计划，需要有相应的Action操作来触发StreamJob
- 一个StreamJob启动时，StreamSQL会为每一个Input Stream启动一组称为Receiver的任务来接收数据，接收来的数据经过一系列Derived Stream的变形最终被插入一张表，供用户查询



➢ Application（流应用）

- Application是一组业务逻辑相关的StreamJob集合
- 合理使用Application划分StreamJob可以实现资源的共享和隔离
  - 资源共享：Application内使用同一个Input Stream的StreamJob共享一组Receiver
  - 资源隔离：不同Application中的StreamJob若使用同一个Input Stream，则每个Application都为这个InputStream启动一组Receiver



DML语句的运行机制不同

- 普通SQL：阻塞式运行
  - 提交SQL后，用户需等待SQL执行结束，期间命令被持续阻塞，无法执行其他命令
- StreamSQL：背景运行
  - 计算任务持续在后台运行
  - 执行StreamSQL的DML语句会立即返回结果

查询结果的输出不同

- 普通SQL：查询结果或者显示在Console，或者通过JDBC读取
- StreamSQL：用户必须显式地指定查询结果输出到某个地方
  - 后台持续运行的SQL无法直接跟Console交互
  - 查询结果通常会插入到表中，如：Insert Into result_table Select ...

2
chapter

StreamSQL DDL

- ✓ Stream管理
- ✓ StreamJob管理
- ✓ Application管理

Stream管理操作

- Create Stream：创建Input Stream
- Create Stream As Select：创建Derived Stream
- Show Streams：列出所有的Stream
- Describe Stream：查看Stream信息
- Alter Stream：修改Input Stream
- Drop Stream：删除Input Stream

➢ 创建Input Stream：完整语法

```
CREATE STREAM <stream_name>(<col_name> <data_type>, <col_name> <data_type>, ...)
   [ROW FORMAT DELIMITED FIELDS TERMINATED BY '<delimiter1>‘
      COLLECTION ITEMS TERMINATED BY '<delimiter2>'
      MAP KEYS TERMINATED BY '<delimiter3>' ] ①
   TBLPROPERTIES(["topic"="<topic_name>",] ②
      ["source"="kafka",] ③
      ["kafka.zookeeper"="<ip:port>",] ④
      ["kafka.broker.list"="<ip:port>",] ⑤
      ["partitions"="<int>,<int>, ...",] ⑥
      ["transwarp.consumer.security.protocol"="<protocol>",] ⑦
      ["transwarp.consumer.sasl.kerberos.service.name"="service_name",] ⑦
      ["transwarp.consumer.sasl.jaas.config"="jass_string",] ⑦
      ["batchduration"="<int>",] ⑧
      ["<key>"="<value>",] ⑨
      ["transwarp.consumer.<property_key>"="kafka_consumer_property_value",] ⑩
      ["transwarp.producer.<property_key>"="kafka_producer_property_value",] ⑪
      ...);
```

➢ 创建Input Stream：完整语法

① <delimiterN>指定数据字段间的分隔符，对于简单类型默认是“,”。

② <topic_name>指定Input Stream指向的Kafka Topic，默认值是database_name.table_name。

③ <source>指定数据源来源，默认为Kafka。

④ <ip:port>指定TDH集群中的一个Zookeeper节点的ip和端口号，除非另外设置，Zookeeper的端口号为2181，默认为当前集群的ZK的地址和端口号。建议在创建Stream时设置此项。

⑤ <ip:port>指定集群中的任意一个Kafka Broker的ip和端口号，默认为9092。

⑥ 可选项。从Topic中指定的Partition上获取流数据。注意，指定Partition属性时，需要同时指定kafka.broker.list属性。

⑦ 这三个选项在安全模式下以Kafka为源时需要使用，细节参考安全模式。

⑧ 设置Batch的长度，整数，单位为毫秒。该属性只能在事件时间模式下使用，且只对微批模式的流处理适用。

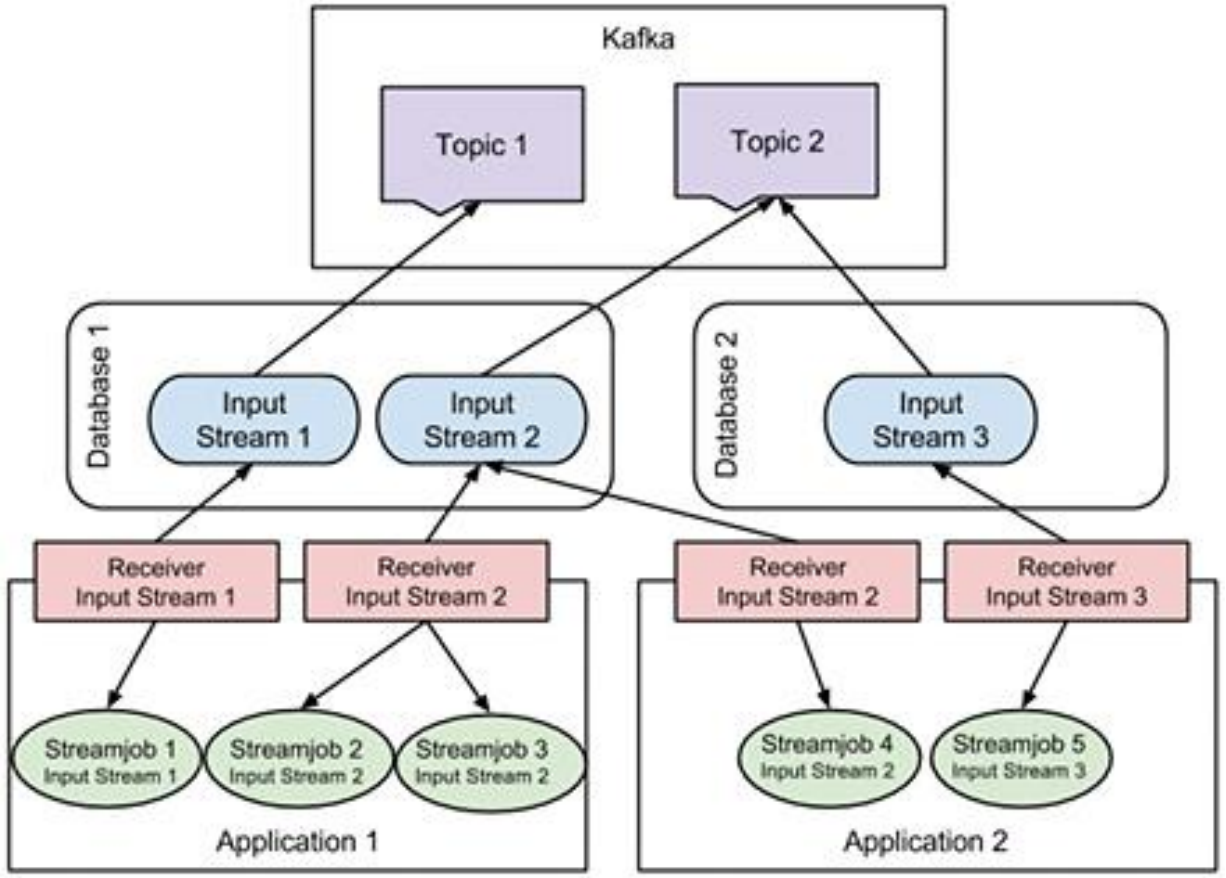
➢ 创建Input Stream：完整语法

⑨ TBLPROPERTIES中还可以包含任意自定义的键值对，用于存放用户自定义的流属性。

⑩ 指定StreamJob启动Kafka Consumer使用的属性值。property_key为Kafka Consumer所支持的某个配置属性名，例如，想配置Consumer能接收的消息的大小，可指定"transwarp.consumer.fetch.max.bytes"="1000000"。

⑪ 当Stream作为输出流时，可通过此配置项来设定StreamJob启动Kafka Producer时使用的属性值。property_key为Kafka producer所支持的某个配置属性名，例如，想配置Producer发送消息的ack模式，可指定"transwarp.producer.acks"="0"。

- 创建Input Stream：数据源为Kafka
  - 发布给Kafka Topic的消息通过Input Stream进入Slipstream
  - 一个Input Stream只能接收一个Topic中的消息，多个Input Stream可以指向同一个Topic



➢ 创建Input Stream：数据源为Kafka

- 创建从指定Topic中接收数据的Stream

```
CREATE STREAM demo(id INT, letter STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
   TBLPROPERTIES(
      "topic" = "demo",
      "kafka.zookeeper" = "172.16.1.128:2181",
      "kafka.broker.list" = "172.16.1.128:9092"
   );
```

- 创建从指定Partition中接收数据的Stream

```
CREATE STREAM demo(id INT, letter STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
   TBLPROPERTIES(
      "topic" = "demo",
      "partitions" = "0,3",
      "kafka.zookeeper" = "172.16.1.128:2181",
      "kafka.broker.list" = "172.16.1.128:9092"
   );
```

- 创建Input Stream：数据源为Socket

```
CREATE STREAM socket_stream(id INT, name STRING, sex BOOLEAN)
   TBLPROPERTIES(
      "source" = "socket",
      "host" = "172.16.1.182",
      "port" = "8888"
   );
```

- 创建Input Stream：复杂类型Struct

```
/* Kafka数据格式：1,5|five   2,9|nine */
CREATE STREAM struct_stream (id INT, s1 STRUCT<a:INT, b:STRING>)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY ',‘
   COLLECTION ITEMS TERMINATED BY '|';
```

- 创建Input Stream：复杂类型Array

```
/* Kafka数据格式：1,1|2|3|4   2,2|4|6|8 */
CREATE STREAM array_stream (id INT, a1 ARRAY<STRING>)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
   COLLECTION ITEMS TERMINATED BY '|';
```

- 创建Input Stream：复杂类型Map

```
/* Kafka数据格式：1,job:800|team:60   2,job:60|team:80 */
CREATE STREAM map_stream (id INT, m1 MAP<STRING,INT>)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY ',‘
   COLLECTION ITEMS TERMINATED BY '|'  MAP KEYS TERMINATED BY ':';
```

- 创建Input Stream：复杂类型Timestamp

```
/* Timestamp默认格式：yyyy-MM-dd HH:mm:SS
   Kafka数据格式：1,2015-01-15 00:16:10   2,2077-11-15 23:16:10 */
CREATE STREAM student_stream (id INT, t1 TIMESTAMP);
```

➢ 创建Derived Stream：数据源为Stream

- 使用已有的Stream创建Derived Stream

```
CREATE STREAM <stream_name> AS SELECT <select_statement>;
/* Filter变换 */
CREATE STREAM female_name_stream AS SELECT name FROM employee_stream
   WHERE sex=true;
/* Window变换 */
CREATE STREAM employee_window_stream AS SELECT * FROM employee_stream
   STREAMWINDOW w1  AS(length '12' second slide '4' second);
```

➢ 列出所有Stream与查看Stream信息

```
SHOW STREAMS;
DESCRIBE|DESC [EXTENDED|FORMATTED] STREAM <stream_name>
```

➢ 修改Input Stream：重命名

```
ALTER STREAM <old_name> RENAME TO <new_name>;
```

2.1 Stream管理

- 修改Input Stream：修改属性

```
/* 修改任意流属性，包括Kafka Topic、Zookeeper、Broker List，以及自定义流属性 */
ALTER STREAM <stream_name> SET TBLPROPERTIES ("key"="value");
```

- 修改Input Stream：增加列

```
ALTER STREAM <stream_name> ADD COLUMNS(name type[, name1 type1 ...]);
```

- 修改Input Stream：替换列

```
ALTER STREAM <stream_name> REPLACE COLUMNS(name type[, name1 type1 ...]);
```

- 删除Input Stream

```
DROP STREAM <stream_name>;
```

2.2 StreamJob管理

- StreamJob是触发StreamSQL执行的Action，一般具有插入结果表语义
- StreamJob主要存储StreamJob Level的配置参数，以及对应的SQL
- StreamJob管理操作
  - Create StreamJob：创建StreamJob
  - Show StreamJobs：列出所有持久化的StreamJob
  - Describe StreamJob：查看StreamJob信息
  - Alter StreamJob：修改StreamJob
  - Drop StreamJob：删除StreamJob

2.2 StreamJob管理

- 创建StreamJob

```
/* <sql_action>为具有Action语义的SQL语句 */
CREATE STREAMJOB <streamjob_name> AS ("<sql_action>")
   JOBPROPERTIES(["key"="value"][,"key"="value"...]);
```

- 查看StreamJob信息

```
DESCRIBE|DESC STREAMJOB <streamjob_name>;
```

- 列出所有持久化的StreamJob

```
SHOW STREAMJOBS;
```

- 修改StreamJob

```
ALTER STREAMJOB <streamjob_name> SET APPPROPERTIES("key"="value");
```

- 删除StreamJob

```
DROP STREAMJOB <streamjob_name>;
```

➢ StreamJob运行时管理

- StreamJob作为StreamSQL运行时的基本单元，也是实时监控的基本单元

```
/* 启动当前Application下的所有StreamJob */
START STREAMJOBS;
/* 启动当前Application下的某个StreamJob */
START STREAMJOB <s1>;
/* 查看当前Application下的所有StreamJob */
LIST STREAMJOBS;
/* 查看当前Application下的某个StreamJob的详细信息 */
LIST STREAMJOB <sid>;
/* 停止当前Application下的所有StreamJob */
STOP STREAMJOBS;
/* 停止当前Application下的某个StreamJob */
STOP STREAMJOB <s1>;
```

2.3 Application管理

- Application主要用于运行时的隔离和权限验证，在静态时只是一个逻辑概念，一般只用于参数配置
- 在测试时，用户登录Shell（如Beeline），在自己建的Application下编写业务逻辑
- 当业务需要上线时，可以用一个Prod Id，具有严格的权限控制，并对应一个Application，将自己的SQL脚本在对应的Application下启动就可以
- Application管理操作
  - Create Application：创建Application
  - Show Applications：列出所有Application
  - Describe Application：查看Application信息
  - Alter Application：修改Application
  - Drop Application：删除Application

2.3 Application管理

- 创建Applicaiton

```
CREATE APPLICATION <app_name>
   WITH APPPROPERTIES(["key"="value"][,"key"="value"...]);
```

- 查看Applicaiton信息

```
DESCRIBE|DESC APPLICATION <app_name>;
```

- 列出所有Applicaiton

```
SHOW APPLICATIONS;
```

- 修改Applicaiton

```
ALTER APPLICATION <app_name> SET APPPROPERTIES("key"="value");
```

- 删除Applicaiton

```
DROP APPLICATION <app_name>;
```

➢ Applicaiton运行时管理

- 用户运行任何StreamJob之前需要进入对应的Application
- 默认情况下，当前的Application是Default

```
/* 进入Application */
USE APPLICATION app1;

/* 显示正在运行的Application */
LIST APPLICATIONS;
```

3
chapter

StreamSQL DML

- ✓ StreamSQL窗口
- ✓ StreamSQL输出方式

- 流应用通常会对一个窗口（时间间隔）内的数据做多表关联、聚合或统计
- 非窗口计算
  - 默认以系统时间为基准，每隔一段时间（微批）或每产生一条数据（事件）触发一次计算
- 窗口计算：时间作为窗口切分单位
  - 事件时间切分：在SQL中指定数据流中的某个时间字段，以该时间字段为基准切分窗口
  - 系统时间切分：在SQL中不指定时间字段，以系统时间为基准切分窗口
- 窗口类型
  - 滑动窗口
    - 定义：按一定时间间隔向未来滑动的长度固定的窗口
    - 特点：前后窗口之间有重叠
    - 示例：对于窗口长度为2s，滑动间隔为1s的滑动窗口，[0s,2s)为第一个窗口，[1s,3s)为第二个窗口，[2s,4s)为第三个窗口，依次类推

➢ 窗口类型

- 滑动窗口
  - 特例：无限滑动窗口，即按照滑动间隔，每滑动一次触发一次计算，但每次计算都包括所有以前窗口的计算值，例如：统计网页访问次数，每隔一段时间输出当时的累计值
- 跳动窗口
  - 定义：当窗口长度和滑动间隔相同，滑动窗口就退化为跳动窗口
  - 特点：前后窗口之间无重叠，跳动窗口是滑动窗口Length = Slide的特例
  - 示例：以2s为间隔的跳动窗口，[0s,2s)为第一个区间，[2s,4s)为第二个区间，依次类推

➢ 窗口切分方式

- 系统时间切分：以流处理引擎的系统时间为基准切分窗口

```
CREATE STREAM s1(id INT, name STRING, ts TIMESTAMP);
INSERT INTO t1 SELECT * FROM s1 STREAMWINDOW w1 AS(LENGTH '2' SECOND SLIDE '1' SECOND);
```

➢ 窗口切分方式

- 事件时间切分：以数据流中的指定时间字段为基准切分窗口

```
/* 以时间字段ts为基准切分窗口，窗口宽度为2秒，滑动间隔为1秒 */
SET streamsql.use.eventtime=true;
CREATE STREAM s1(id INT, name STRING, ts TIMESTAMP)
   tblproperties("timefield"="ts");
INSERT INTO t1 SELECT * FROM s1 STREAMWINDOW w1 AS(LENGTH '2' SECOND SLIDE '1' SECOND);
```

- 事件时间切分：自定义时间格式（默认时间格式为yyyy-MM-dd HH:mm:ss）

```
/* 以时间字段ts为基准切分窗口，窗口宽度为2分钟，滑动间隔为1分钟，
   指定格式为 “yyyy-MM-dd HH:mm”，不包含秒级信息 */
CREATE STREAM s1(id INT, name STRING, ts TIMESTAMP)
   TBLPROPERTIES("timefield"="ts", "timeformat"="yyyy-MM-dd HH:mm");
INSERT INTO t1 SELECT * FROM s1 STREAMWINDOW w1 AS(LENGTH '2' MINUTE SLIDE '1' MINUTE);
```

➢ 窗口切分方式

- 事件时间切分：支持String类型

```
CREATE STREAM s1(id INT, name STRING, ts STRING)
  TBLPROPERTIES("timefield"="ts","timeformat"="MM/dd/yyyy HH:mm:ss");
```

- 事件时间切分与系统时间切分的区别
  - 默认的窗口切分方式为系统时间
  - 优先级：前者高于后者
  - 灵活性：前者更灵活，不同窗口可以指定不同的时间字段
  - 格式支持：前者可以满足多种时间格式

➢ 输出到流

- StreamSQL支持将查询结果输出到另一个流
- 方式一：将一个流原样输出到另一个流

```
CREATE STREAM s1(id INT, name STRING);
CREATE STREAM s2(id INT, name STRING);
INSERT INTO s2 SELECT * FROM s1;
```

- 方式二：将一个流变换为另一个流

```
/* Filter变换 */
CREATE STREAM female_name_stream
   AS SELECT name FROM employee_stream WHERE sex==true;
/* Window变换 */
CREATE STREAM employee_window_stream
   AS SELECT * FROM employee_stream
   STREAMWINDOW w1 AS(LENGTH '12' SECOND SLIDE '4' SECOND);
```

3.2 StreamSQL输出方式

➢ 输出到Hyperbase

```
/* 创建输入流 */
CREATE STREAM employee_stream(id INT, name STRING, sex BOOLEAN);

/* 创建Hyperbase表 */
CREATE TABLE hyper(row_key data_type, col_name1 data_type, col_name2, data_type, ...)
   STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler|
      io.transwarp.hyperbase.HyperbaseStorageHandler‘
      WITH SERDEPROPERTIES("hbase.columns.mapping"=":row_key_column,
      column_family:column_qualifer_1, column_family:column_qualifier_2, ...")
  TBLPROPERTIES ("hbase.table.name" ="hbase_table_name");

/* 将流转换后插入Hyperbase表 */
 INSERT INTO hyper SELECT * FROM employee_stream WHERE sex=true;
```

3.2 StreamSQL输出方式

- 输出到Kafka

```
/* 创建两个Kafka Topic输入流 */
CREATE STREAM s1(id INT, letter STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
   TBLPROPERTIES(
      "topic"="s1",
      "kafka.zookeeper"="172.16.1.128:2181");
CREATE STREAM s2(id INT, letter STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
   TBLPROPERTIES(
      "topic"="s2",
      "kafka.zookeeper"="172.16.1.128:2181");

/* 将s1输出到s2 */
INSERT INTO s2 SELECT * FROM s1;
```

4
chapter

事件驱动的流处理

- ✓ 基本概念
- ✓ 用法举例

含义

- 以单条数据被Input Stream接收为事件，逐条读取数据，并立刻加工处理，最后输出
- 来一条处理一条

特点

- 相比微批模式（TDH 4.8之前），事件驱动模式的延迟更低，在延迟敏感的场景中表现更佳

开启事件驱动模式

- 第一步：让Slipstream引擎运行在事件驱动模式下
  - 在Transwarp Manager中配置系统参数：NGMR_ENGINE_MODE = morphling
  - morphling代表事件驱动模式（TDH 5.1之后的默认值），mapred代表微批（TDH 5.1之前的默认值）
- 第二步：让StreamJob运行在事件驱动模式下
  - TDH 5.1之前，在启动StreamJob之前，配置Job参数：streamsql.use.eventmode = true
  - TDH 5.1之后，如果当前引擎是morphling，则默认开启该参数，否则默认关闭

➢ 将流表中的数据写入Inceptor普通表

```
/* 开启Job的事件驱动模式 */
SET streamsql.use.eventmode=true;

/* 创建Kafka输入流 */
CREATE STREAM s1(score INT, name STRING)
   TBLPROPERTIES(
      "topic"="tps1",
      "kafka.zookeeper"="172.16.1.128:2181",
      "kafka.broker.list"="“172.16.1.128:9092");

/* 创建Inceptor表 */
CREATE TABLE t1(score INT, name STRING);

/* 将流表中的数据写入Inceptor表 */
INSERT INTO t1 SELECT * FROM s1;
```

➢ 在Window Stream上做聚合后写入Inceptor普通表

```
/* 开启Job的事件驱动模式 */
SET streamsql.use.eventmode=true;
/* 开启窗口的事件时间切分模式 */
SET streamsql.use.eventtime=true;
/* 创建Window Stream */
CREATE STREAM s1(name STRING, score INT, ts STRING)
   TBLPROPERTIES("topic"="tps1",
      "kafka.zookeeper"="tw-node127:2181",
      "kafka.broker.list"="tw-node127:9092",
      "timefield"="ts“,
      "timeformat"="yyyy-MM-dd HH:mm:ss",
      "use.lowlevel.consumer"="true");
CREATE STREAM ws1 AS SELECCT * FROM s1 STREAMWINDOW (LENGTH ‘4’ SECOND SLIDE ‘2’ SECOND);
/* 先对Window Stream做聚合，再写入Inceptor表 */
CREATE TABLE t1(score INT, name STRING);
INSERT INTO t1 SELECT name, SUM(score) FROM ws1 GROUP BY name;
```

5
chapter

实战案例

- ✓ 案例描述
- ✓ 解决方法

➢ 业务需求

- 为了及时了解商品热度，某电商管理者希望对网站商品的访问量进行统计分析，统计范围为过去5分钟，并以1分钟为周期进行持续更新

➢ 样例数据

- 前端实时收集的数据

```
/* 数据格式：访问者IP, 访问的商品网页, 访问时间 */
27.0.1.125,www.taobao.com/xx1.html,2016-8-14 20:43:31.213
72.21.203.5,www.taobao.com/xx2.html,2016-8-14 20:45:31.132
207.241.224.2,www.taobao.com/xx3.html,2016-8-14 20:46:15.540
12.129.206.133,www.taobao.com/xx3.html,2016-8-14 20:47:21.332
208.111.148.7,www.taobao.com/xx1.html,2016-8-14 20:50:31.876
```

➢ 解决方法

- Step1：通过Kafka消息队列收集数据

```
# ./kafka-topics.sh –create
      --zookeeper tdh-71:2181,tdh-72:2181,tdh-73:2181
      --topic accesslog --partition 3 --replication-factor 1
```

- Step2：创建以Kafka为数据源的Input Stream

```
CREATE STREAM accesslog (
   ip STRING,
   url STRING,
   time TIMESTAMP)
   ROW FORMAT delimited FIELDS TERMINATED BY ','
   TBLPROPERTIES(
      "topic" = "accesslog",
      "kafka.zookeeper"="tdh-71:2181,tdh-72:2181,tdh-73:2181",
      "kafka.broker.list"="tdh-71:9092,tdh-72:9092,tdh-73:9092");
```

➢ 解决方法

- Step3：根据业务创建长度为5min，滑动步长为1min的时间窗口

```
CREATE STREAM waccesslog AS
   SELECT * FROM accesslog STREAMWINDOW sw AS (LENGTH '5' MINUTE SLIDE '1' MINUTE);
```

- Step4：创建结果表来存储流处理后的数据

```
CREATE TABLE result(rowkey STRING, url STRING, count INT) STORED AS HYPERDRIVE;
```

- Step5：创建StreamJob，定义并启动流处理任务

```
CREATE STREAMJOB hotcount AS (
   "INSERT INTO result SELECT uniq(), url, COUNT(*) FROM waccesslog GROUP BY url");
START STREAMJOB hotcount;
```

Q&A

TRANSWARP
星 环 科 技