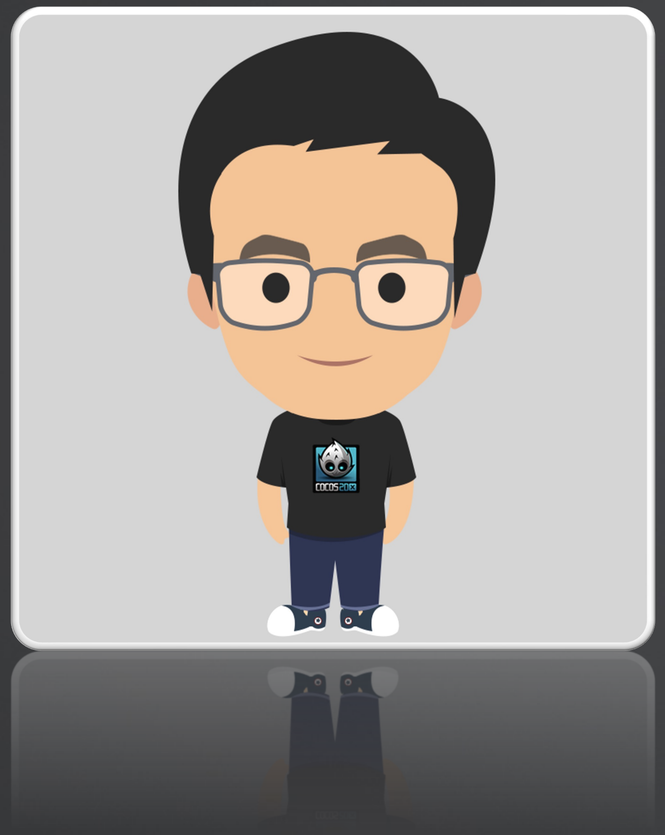
FlinkSQL 实战

讲师/ 王超

讲师介绍



- 多年大数据工作经验，历任多家公司大数据技术 Leader
- 先后出版《大数据技入门到商业实践》及《Flink 与 Kylin 深度实践》等多本书籍
- 课程内容通俗移动，为中国移动、中国联通等多家企业组织技术内训，受到企业一致好评

目录

1 批处理以及流处理技术发展

2 FlinkSQL 流批一体统一处理架构思想设计

3 Flink Table 与 SQL 编程详解

4 Flink SQL 函数解析

目录

1 批处理以及流处理技术发展

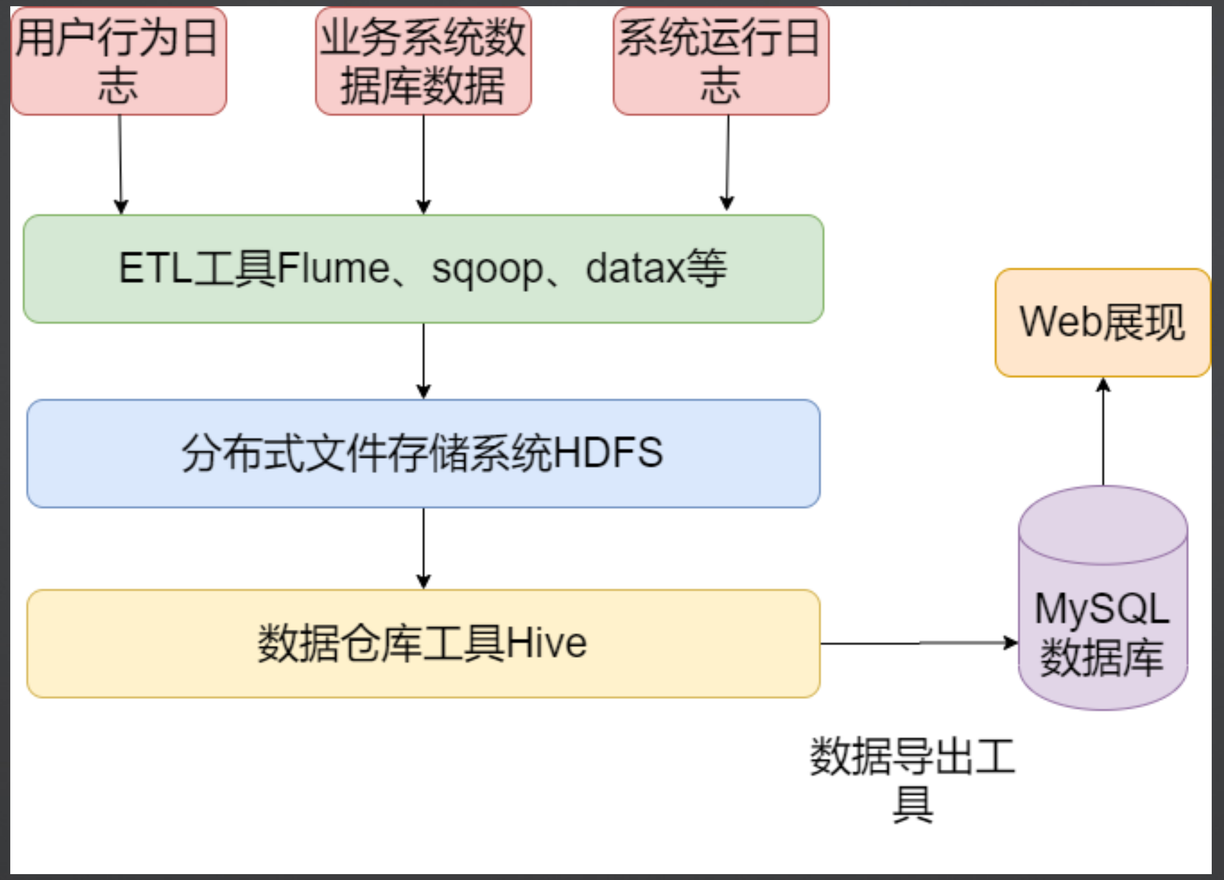
2 FlinkSQL 流批一体统一处理架构思想设计

3 Flink Table 与 SQL 编程详解

4 Flink SQL 函数解析

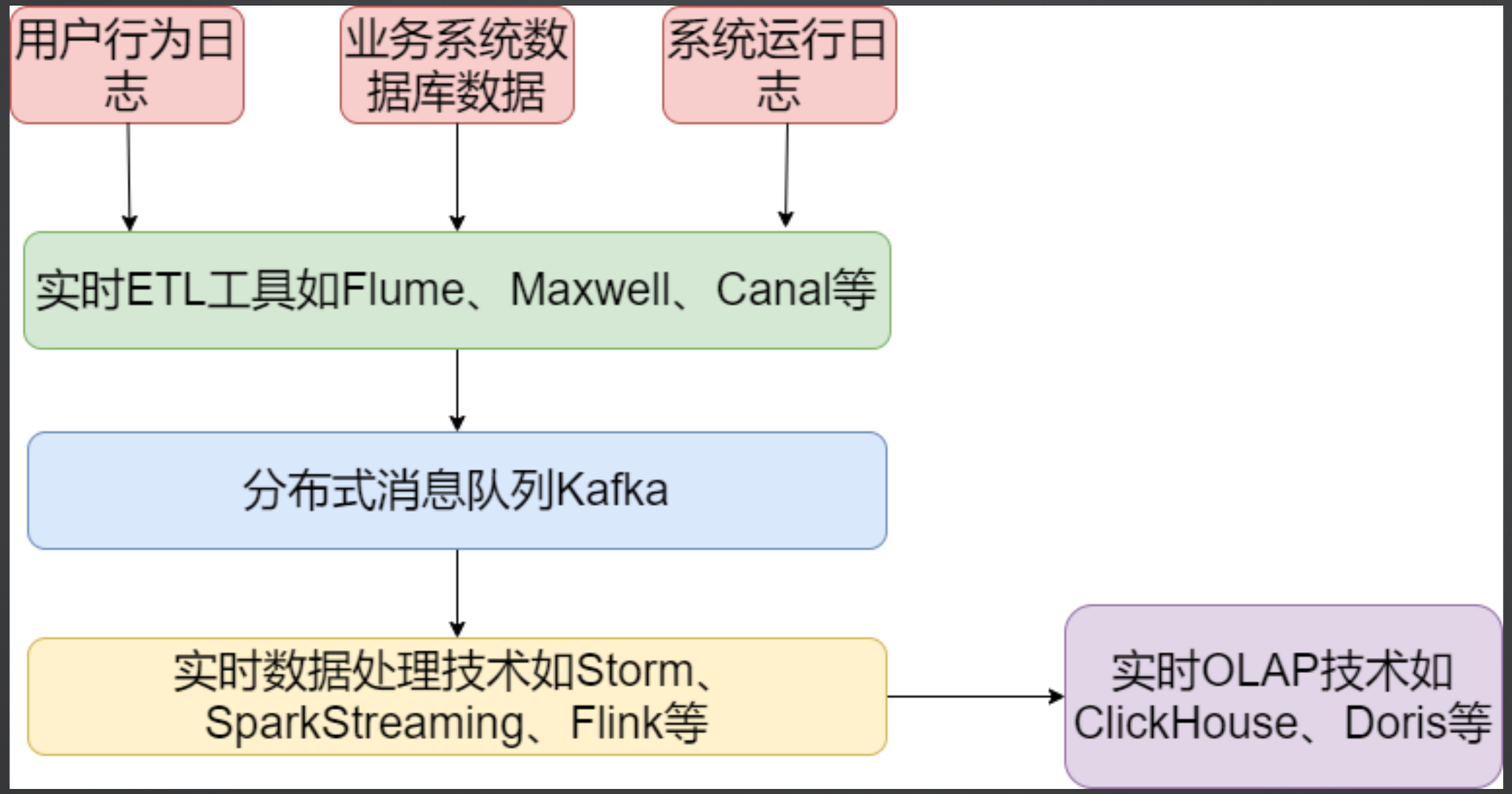
批处理以及流处理技术发展

- 批量数据处理发展过程



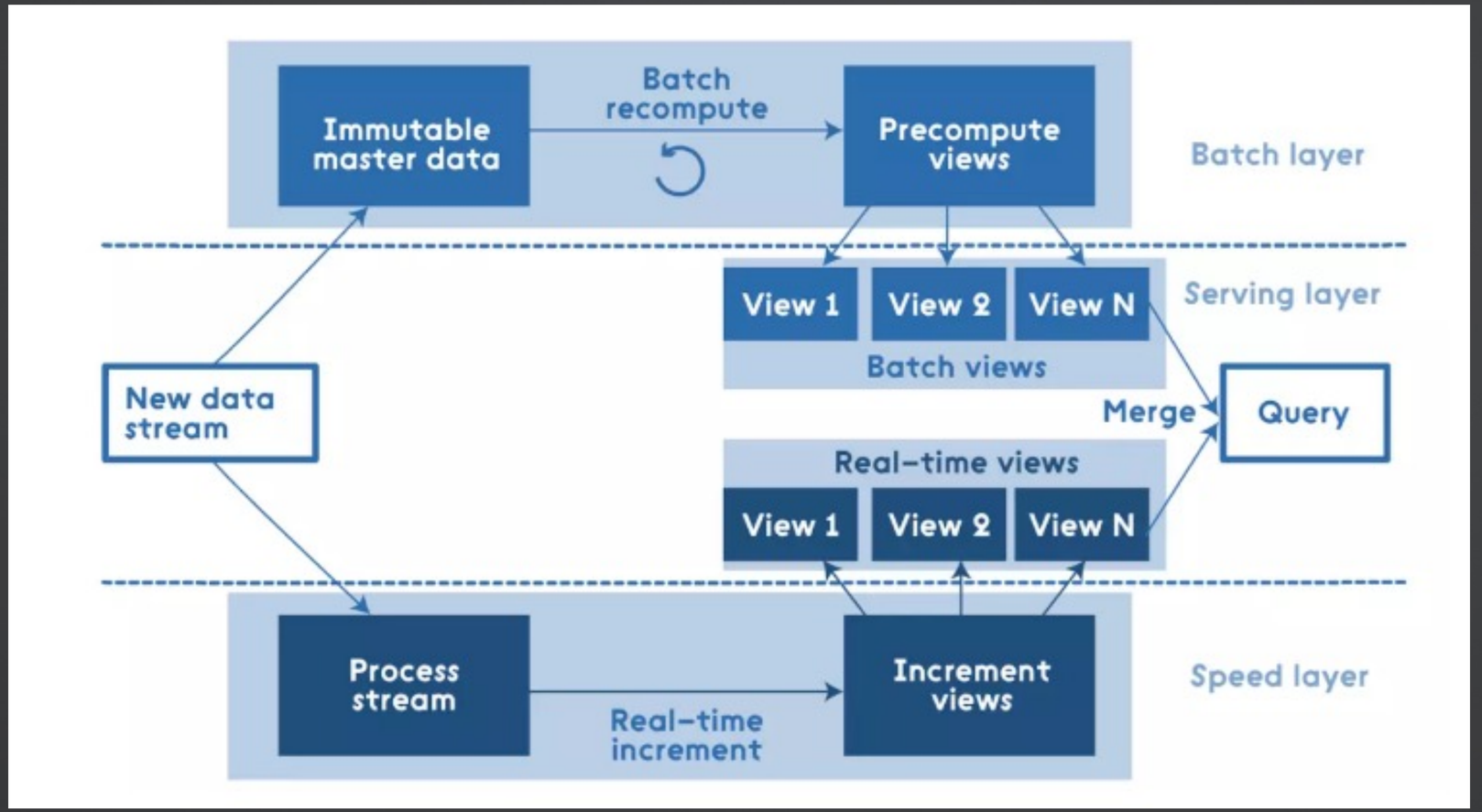
批处理以及流处理技术发展

- 流式数据处理发展过程



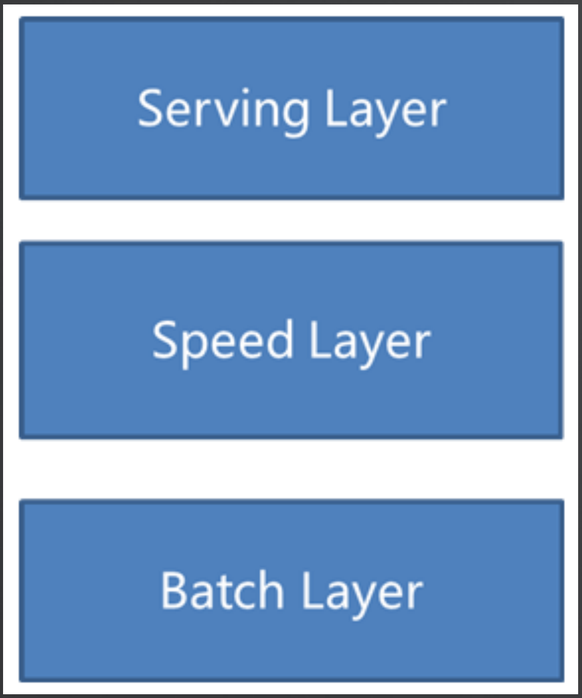
批处理以及流处理技术发展

- Lambda 架构



批处理以及流处理技术发展

- Lambda 架构三层划分

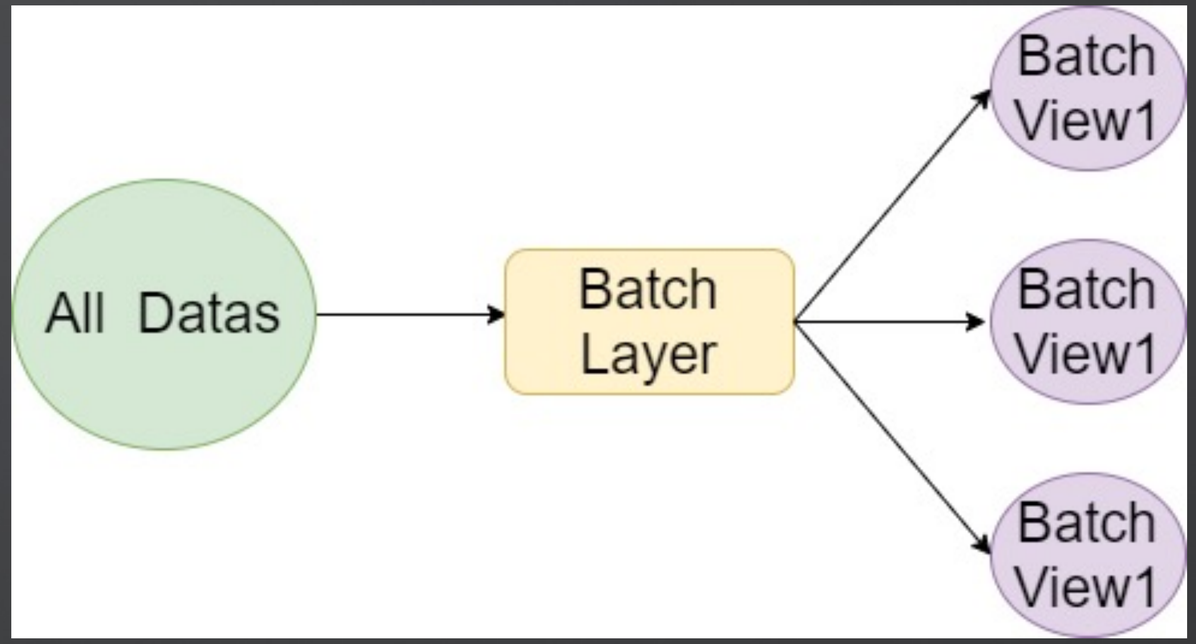


- 如何实现实时在任意大数据集上进行查询？大量数据再加上实时计算，这个难度就比较大了。
- Lambda 架构通过三层架构来解决该问题：Batch Layer、Speed Layer 和 Serving Layer。

批处理以及流处理技术发展

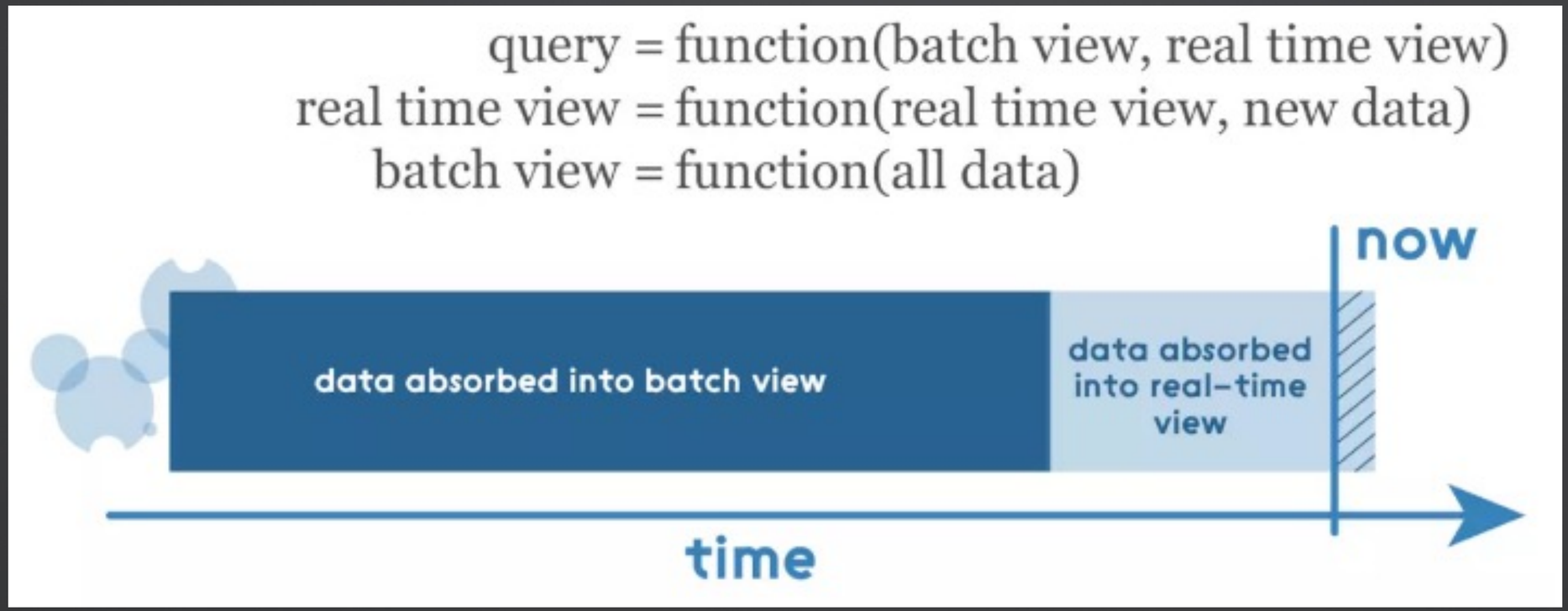
- Lambda 架构三层划分

- Batch Layer：主要用于实现对历史数据计算结果的保存，每天计算的结果都保存成为一个 Batch View，然后通过对 Batch View 的计算，实现历史数据的计算。



批处理以及流处理技术发展

- Speed Layer



批处理以及流处理技术发展

- Speed Layer

Batch Layer 可以很好的处理离线数据，但有很多场景数据实时生成，并且需要实时查询处理。Speed Layer 正是用来处理增量的实时数据。

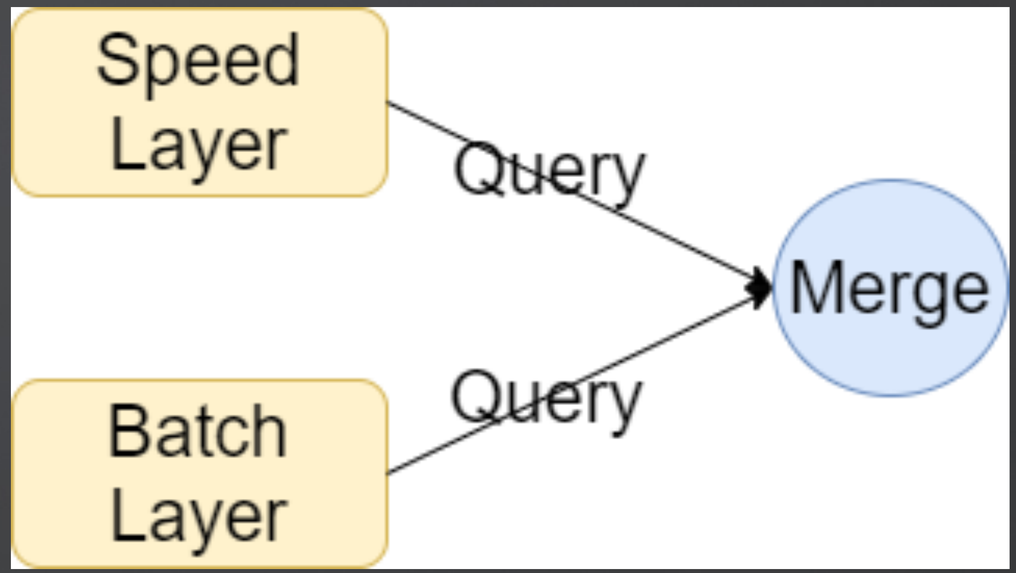
Speed Layer 和 Batch Layer 比较类似，对数据进行计算并生成 data View，其主要区别在于：

- Speed Layer 处理的数据是最近的增量数据流，Batch Layer 处理的是全体数据集。
- Speed Layer 为了效率，接收到新数据时不断更新 Realtime View，而 Batch Layer 根据全体离线数据集直接得到 Batch View。Speed Layer 是一种增量计算，而非重新计算（recomputation）。
- Speed Layer 因为采用增量计算，所以延迟小，而 Batch Layer 是全数据集的计算，耗时比较长。

批处理以及流处理技术发展

- Serving Layer

- Lambda 架构的 Serving Layer 用于响应用户的查询请求，合并 Batch View 和 Realtime View 中的结果数据集到最终的数据集。



目录

1 批处理以及流处理技术发展

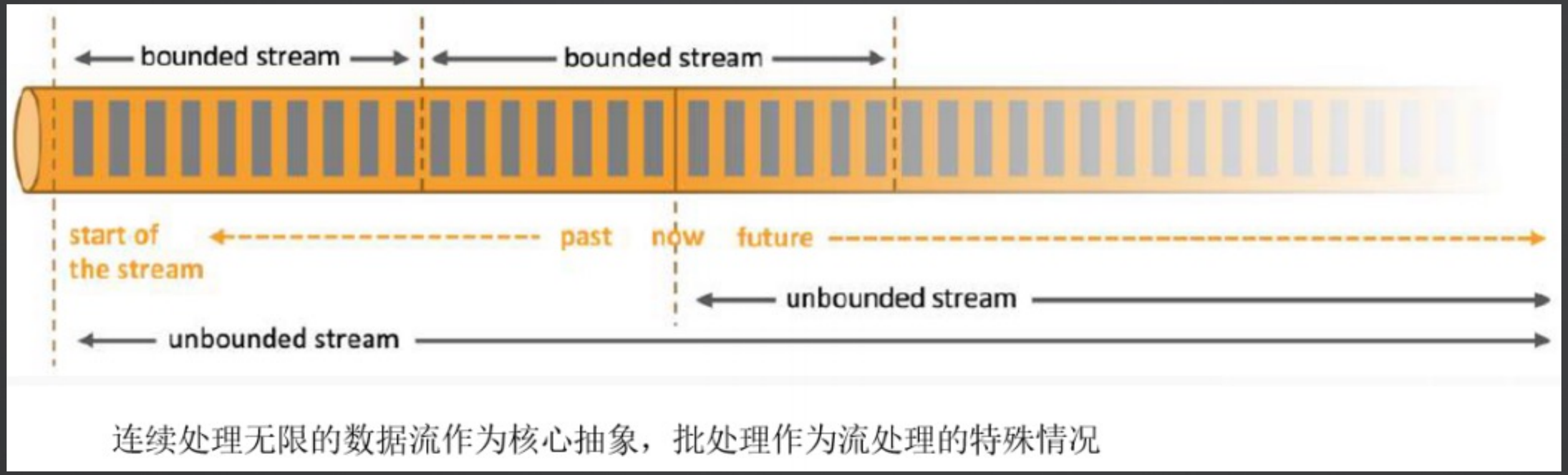
2 FlinkSQL 流批一体统一处理架构思想设计

3 Flink Table 与 SQL 编程详解

4 Flink SQL 函数解析

FlinkSQL 流批一体统一处理架构思想设计

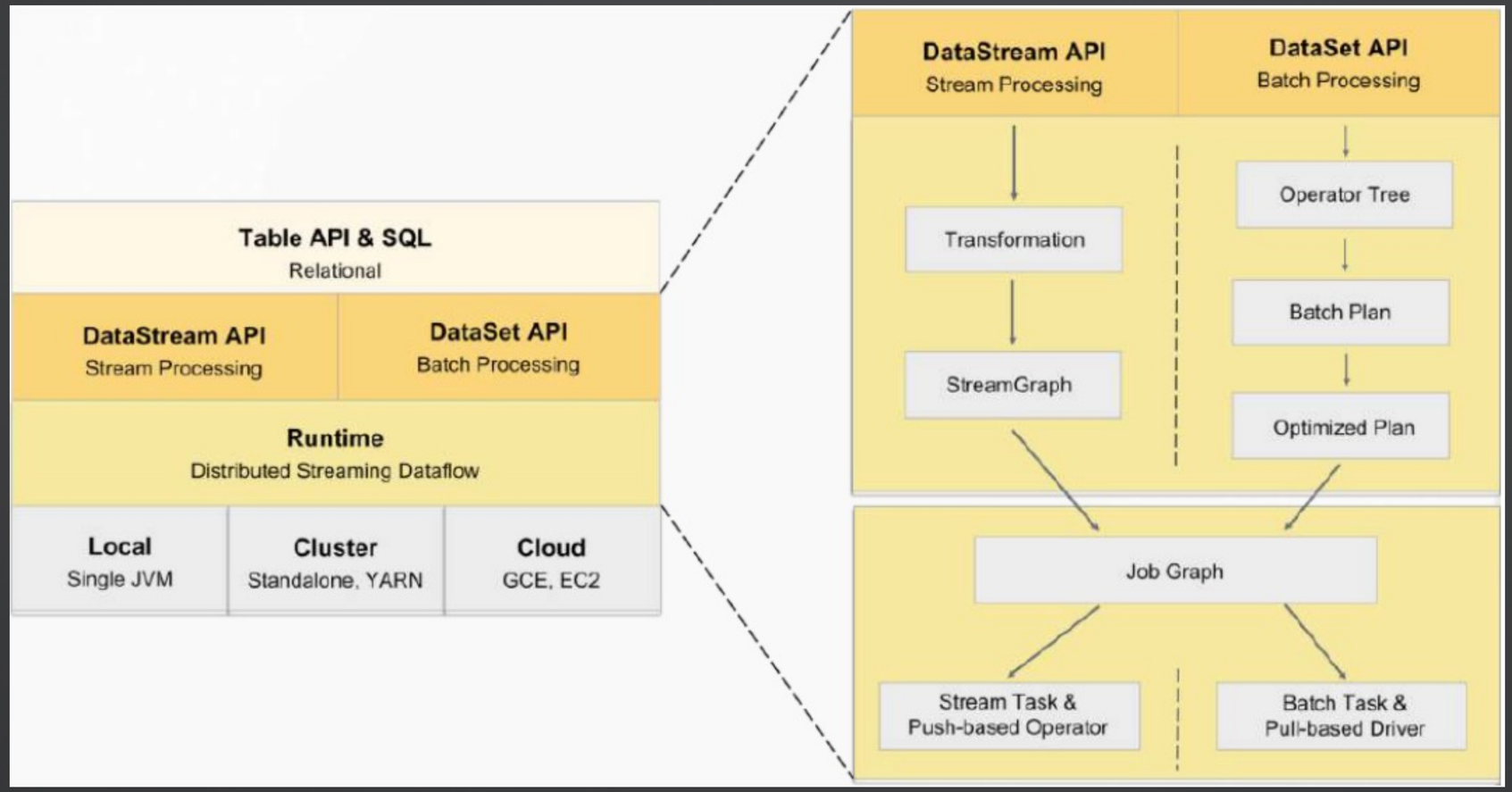
- 有界流和无界流：在 Flink 中，批处理是流处理的一个特例



连续处理无限的数据流作为核心抽象，批处理作为流处理的特殊情况

FlinkSQL 流批一体统一处理架构思想设计

- Flink 老版本架构问题



FlinkSQL 流批一体统一处理架构思想设计

从 Flink 用户角度（企业开发人员）

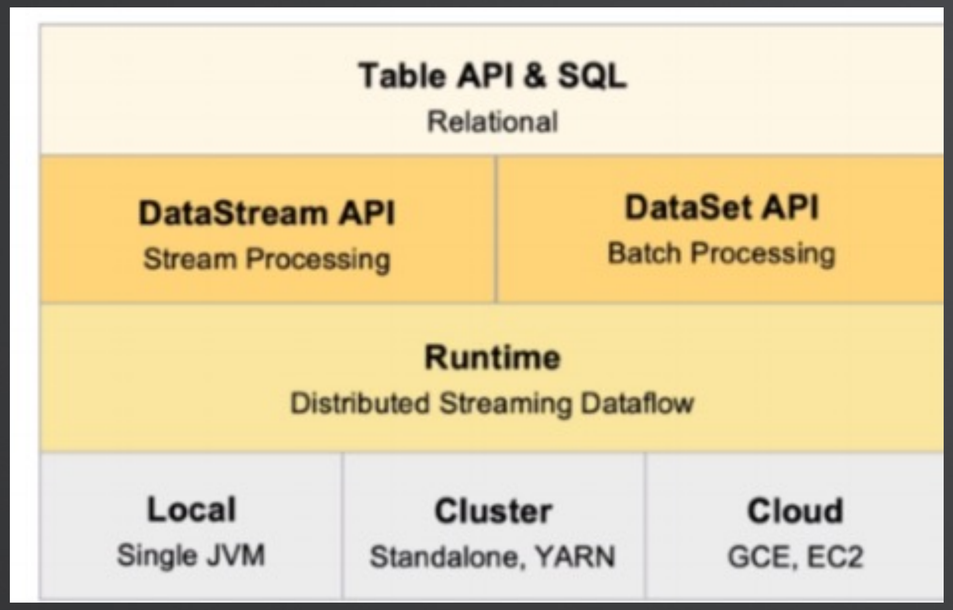
- 开发的时候，Flink SQL 支持的不好，就需要在两个底层 API 中进行选择，甚至维护两套代码。
- 不同的语义、不同的 connector 支持、不同的错误恢复策略等。
- Table API 也会受不同的底层 API、不同的 connector 等问题的影响。

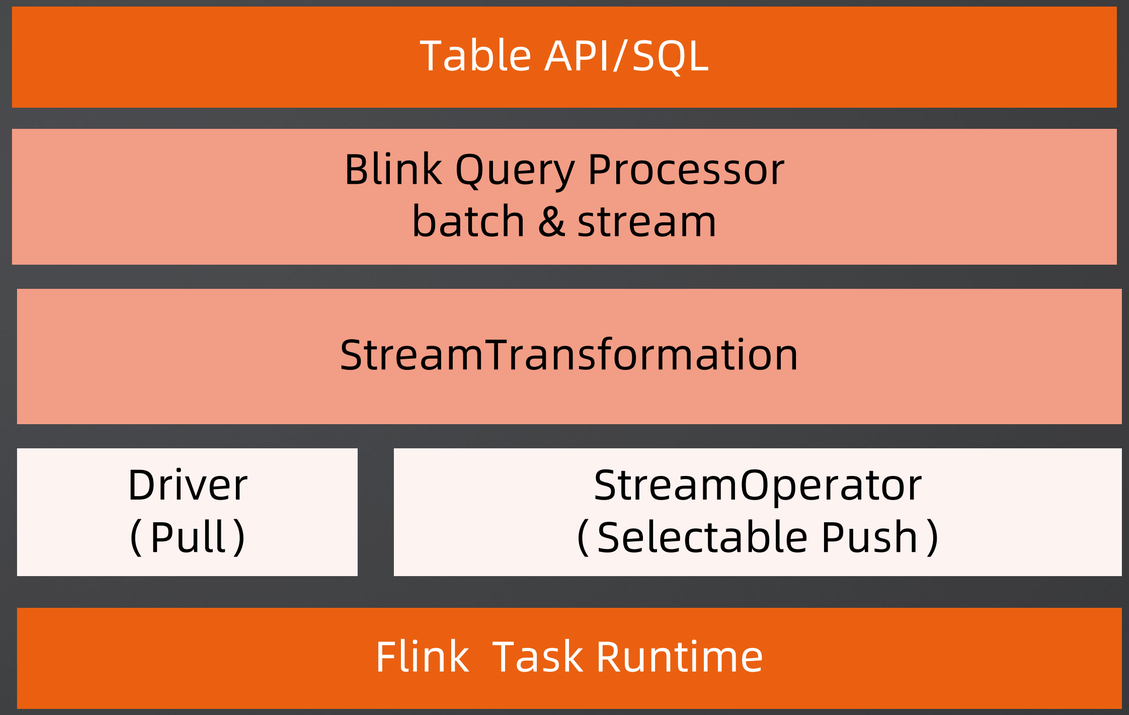
从 Flink 开发者角度（Flink 社区人员）

- 不同的翻译流程、不同的算子实现、不同的 Task 执行。
- 代码难以复用。
- 两条独力的技术栈需要更多人力，功能开发变慢、性能提升变难、bug 变多。

FlinkSQL 流批一体统一处理架构思想设计

- Flink 新版本架构设计

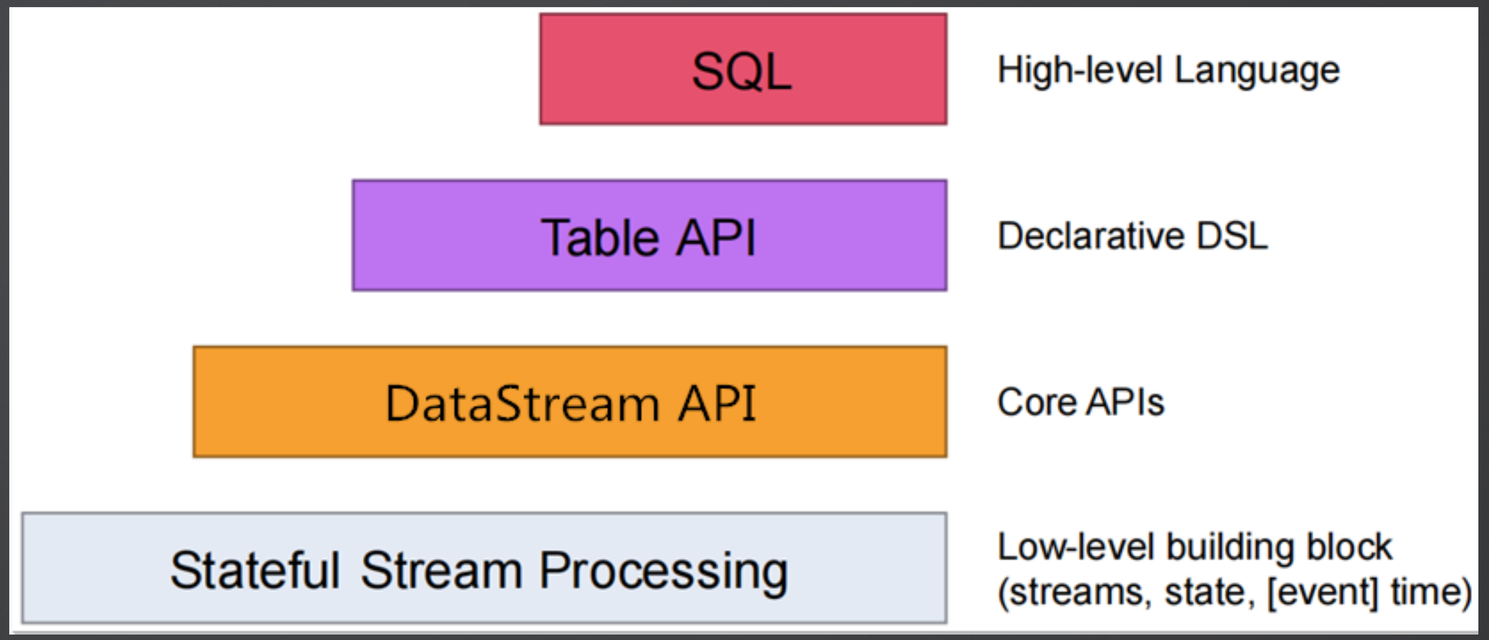




FlinkSQL 流批一体统一处理架构思想设计

- API 分层

- Flink1.12 之前的版本，Table API 和 SQL 处于活跃开发阶段，并没有实现流批统一的所有特性，所以使用的时候需要慎重。从 Flink1.12 开始，Table API 和 SQL 就已经成熟了，可以在生产上放心使用。



FlinkSQL 流批一体统一处理架构思想设计

- Table API 是一种关系型 API，类 SQL 的 API，用户可以像操作表一样地操作数据， 非常的直观和方便。
- SQL 作为一个"人所皆知"的语言，如果一个引擎提供 SQL，它将很容易被人们接受。这已经是业界很常见的现象了。
- Table & SQL API 还有另一个职责，就是负责流处理和批处理统一的 API 层。
- Flink 的 Table API 和 SQL 是流批统一的 API。 这意味着 Table API & SQL 在无论有限的批式输入还是无限的流式输入下，都具有相同的语义。 因为传统的关系代数以及 SQL 最开始都是为了批式处理而设计的， 关系型查询在流式场景下不如在批式场景下容易懂。

FlinkSQL 流批一体统一处理架构思想设计

- 动态表的概念
- SQL 和关系代数在设计时并未考虑流数据。因此，在关系代数和 SQL 之间几乎没有概念上的差异。
- 关系代数（主要就是指关系型数据库中的表）和 SQL，主要就是针对批处理的，这和流处理有天生的隔阂。

关系代数 / SQL	流处理
关系(或表)是有界(多)元组集合。	流是一个无限元组序列。
对批数据(例如关系数据库中的表)执行的查询可以访问完整的输入数据。	流式查询在启动时不能访问所有数据，必须"等待"数据流入。
批处理查询在产生固定大小的结果后终止。	流查询不断地根据接收到的记录更新其结果，并且始终不会结束。

FlinkSQL 流批一体统一处理架构思想设计

- 动态表的概念

尽管存在这些差异，但是使用关系查询和 SQL 处理流并不是不可能的。高级关系数据库系统提供了一个称为 物化视图(Materialized Views) 的特性。物化视图被定义为一条 SQL 查询，就像常规的虚拟视图一样。与虚拟视图相反，物化视图缓存查询的结果，因此在访问视图时不需要对查询进行计算。

缓存的一个常见难题是防止缓存为过期的结果提供服务。当其定义查询的基表被修改时，物化视图将过期。 即时视图维护(Eager View Maintenance) 是一种一旦更新了物化视图的基表就立即更新视图的技术。

如果我们考虑以下问题，那么即时视图维护和流上的 SQL 查询之间的联系就会变得显而易见：

- 数据库表是 INSERT、UPDATE 和 DELETE DML 语句的 stream 的结果，通常称为 changelog stream 。
- 物化视图被定义为一条 SQL 查询。为了更新视图，查询不断地处理视图的基本关系的changelog 流。
- 物化视图是流式 SQL 查询的结果。

FlinkSQL 流批一体统一处理架构思想设计

- 动态表与连续查询（Continuous Query）

动态表 是 Flink 的支持流数据的 Table API 和 SQL 的核心概念。与表示批处理数据的静态表不同，动态表是随时间变化的。可以像查询静态批处理表一样查询它们。查询动态表将生成一个 连续查询 。一个连续查询永远不会终止，结果会生成一个动态表。查询不断更新其（动态）结果表，以反映其（动态）输入表上的更改。本质上，动态表上的连续查询非常类似于定义物化视图的查询。

需要注意的是，连续查询的结果在语义上总是等价于以批处理模式在输入表快照上执行的相同查询的结果。

流、动态表和连续查询之间的关系

FlinkSQL 流批一体统一处理架构思想设计

- 动态表与连续查询（Continuous Query）

在数据流上执行关系查询时，数据流与动态表的转换关系图的主要步骤如下：

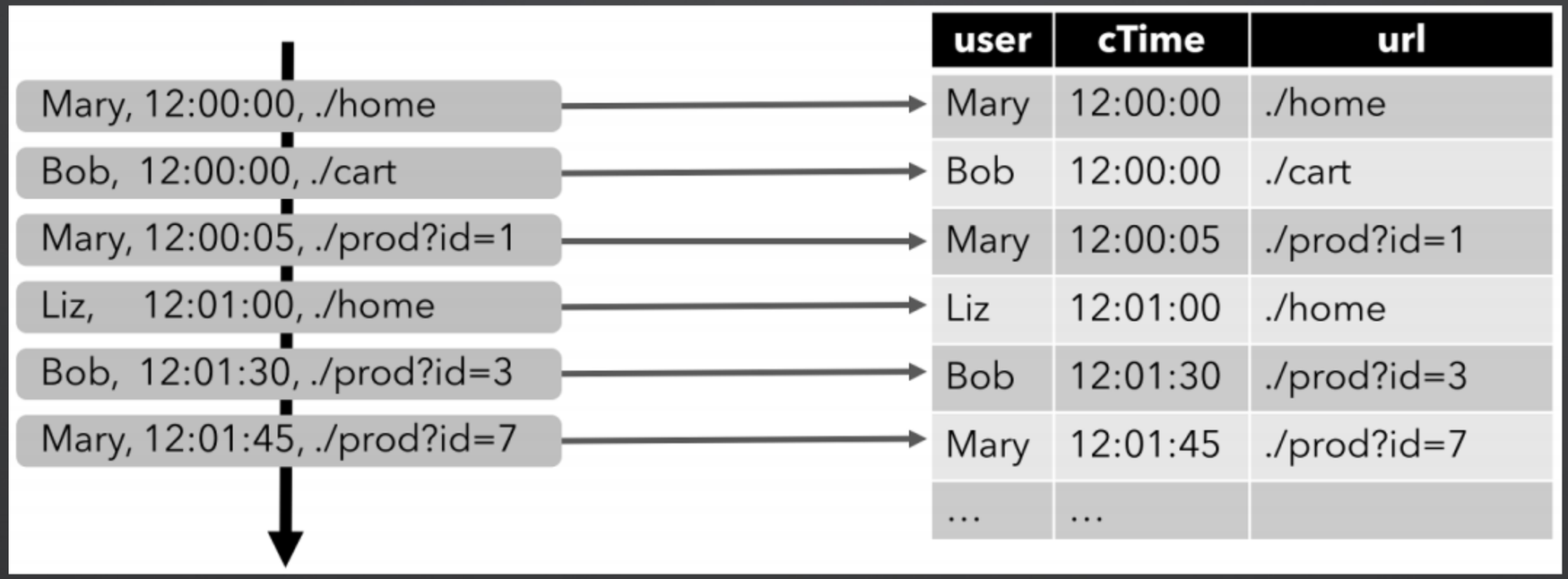
1. 将流转换为动态表。
2. 在动态表上计算一个连续查询，生成一个新的动态表。
3. 生成的动态表被转换回流。

流、动态表和连续查询之间的关系

FlinkSQL 流批一体统一处理架构思想设计

动态表详解

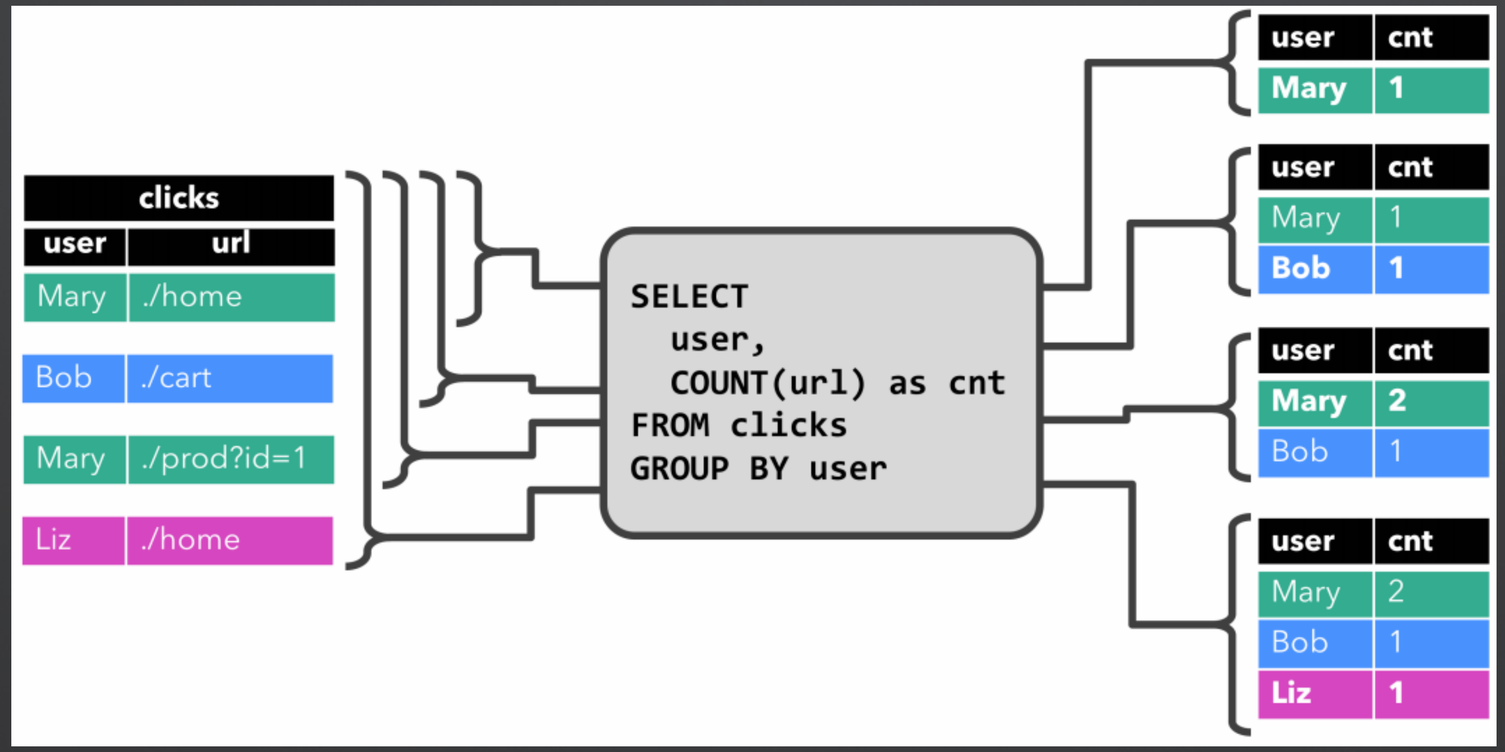
- 定义动态表（点击事件流）：为了执行关系查询，首先得把数据流转换为动态表。下图左侧为点击流，右侧为动态表，流上的新增事件都会对应动态表上的 insert 操作。



FlinkSQL 流批一体统一处理架构思想设计

动态表详解

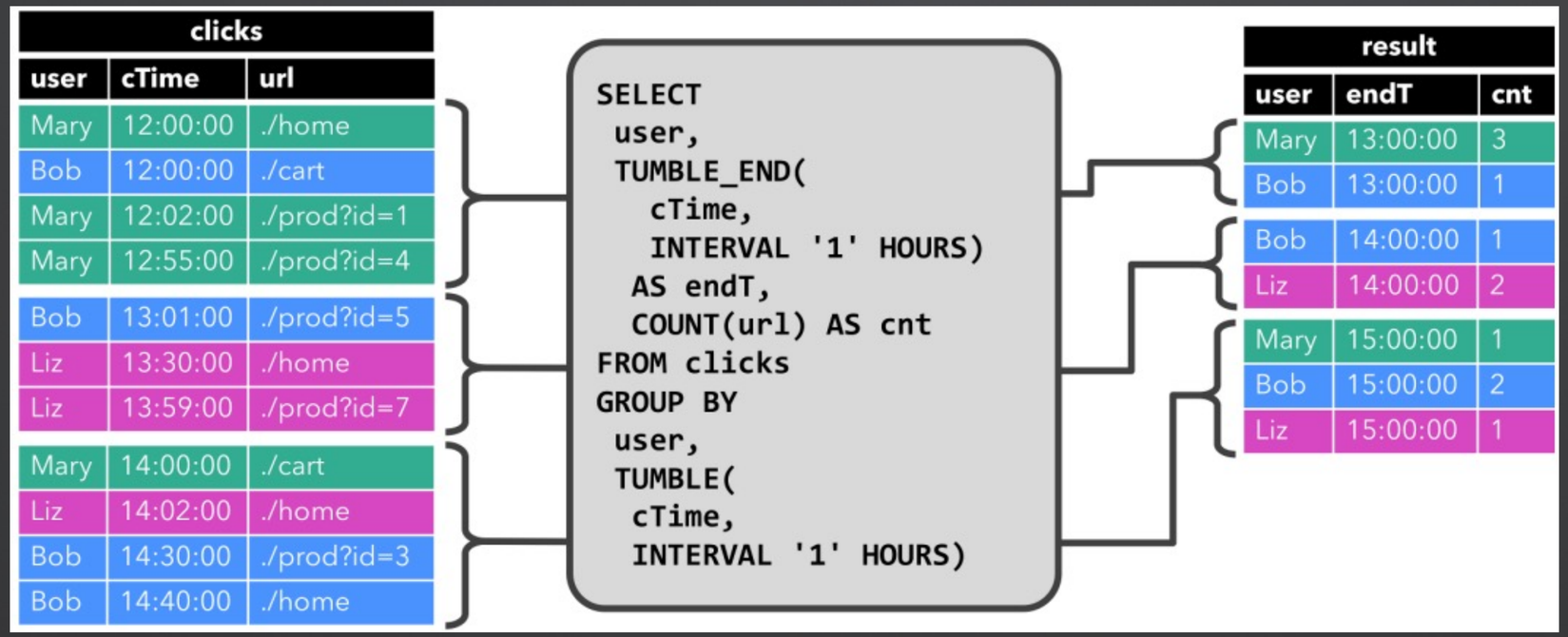
- 连续查询：接下来，我们在动态表上执行连续查询生成一个新的动态表（结果表），连续查询不会停止，它会根据输入表新数据的到来不断查询计算并更新结果表。我们在 click 动态表上执行了group by count 聚合查询，随着时间推移，右边动态结果表随着左测输入表每条数据的变化而变化。



FlinkSQL 流批一体统一处理架构思想设计

动态表详解

- 连续查询：我们在 click 动态表上执行了 group by count 聚合，另外还加入了一个翻滚窗口，统计1小时翻滚窗口内每个用户的访问次数。随着时间推移，右边动态结果表随着左测输入表数据的变化而变化，但是每个窗口的结果是独立的，且计算是在每个窗口结束时才触发的。



FlinkSQL 流批一体统一处理架构思想设计

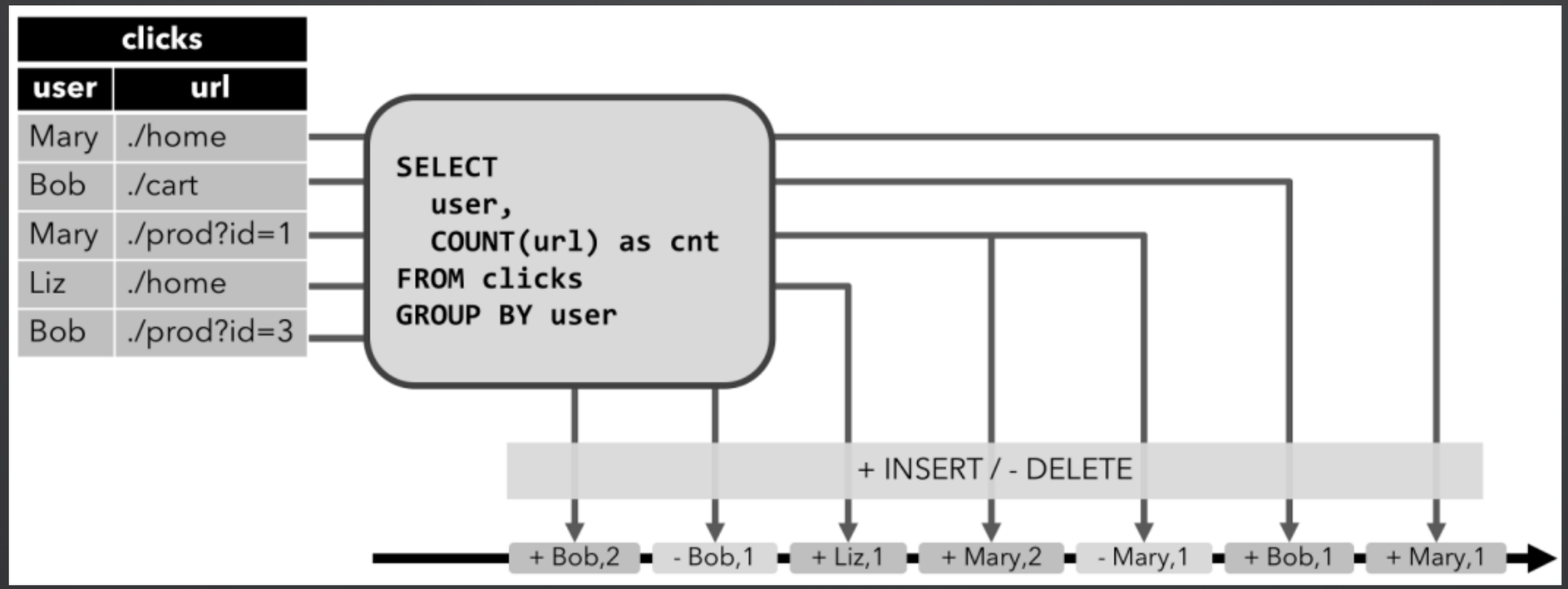
动态表详解：动态表转为数据流

- 与常规数据库表一样，动态表可以通过插入（Insert）、更新（Update）和删除（Delete） 更改，进行持续的修改。将动态表转换为流或将其写入外部系统时，需要对这些更改进行编码。Flink 的 Table API 和 SQL 支持三种方式对动态表的更改进行编码。
- 仅追加流（Append-only stream，即insert-only）
- 仅通过插入 INSERT 更改来修改的动态表，可以直接转换为“仅追加”流。这个流中发出的数据就是动态表中新增的每一个事件。

FlinkSQL 流批一体统一处理架构思想设计

动态表详解：撤回流（Retract stream）

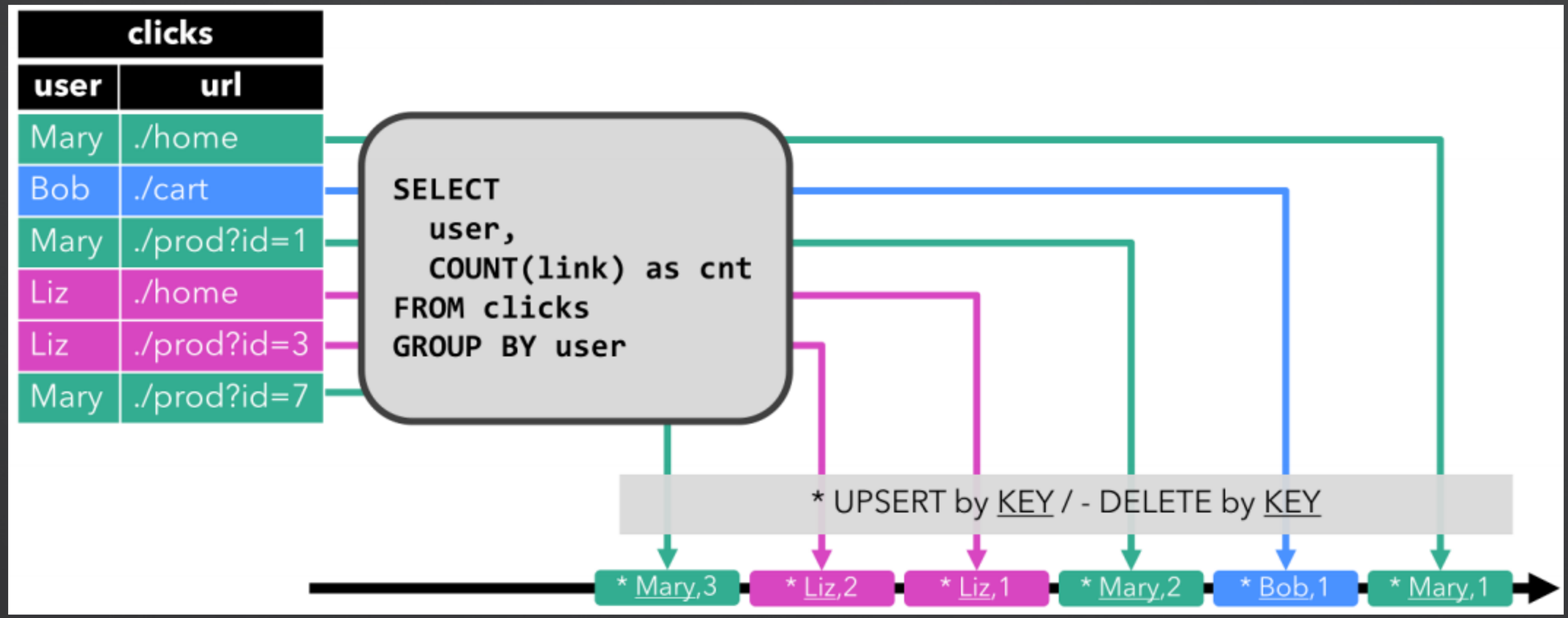
- 插入、更新、删除都支持的动态表会转换为撤回流。
- 撤回流包含两类消息：添加（add）消息和撤回（retract）消息。
- 动态表通过将 INSERT 编码为 add 消息、DELETE 编码为 retract 消息、UPDATE 编码为被更改行（更改前）的 retract 消息和更新后行（新行）的 add 消息，转换为 retract 流。



FlinkSQL 流批一体统一处理架构思想设计

动态表详解：更新插入流（Upsert 流）

- Upsert 流包含两种类型的消息：upsert 消息和 delete 消息。转换为 upsert 流的动态表， 需要有唯一的键（key）。通过将 INSERT 和 UPDATE 更改编码为 upsert 消息，将 DELETE 更改编码为 DELETE 消息， 就可以将具有唯一键（Unique Key）的动态表转换为流。



FlinkSQL 流批一体统一处理架构思想设计

动态表详解

查询限制

- 对无界数据流进行连续的查询会有一些限制，主要是如下两个方面：状态大小限制、计算成本更新限制。

状态大小限制

- 无界数据流上的连续查询需要运行数周或者数月甚至更长，因此连续查询处理的数据量可能会很大。例如，前面计算用户访问量的例子中，需要维护用户访问量的计数状态，如果只考虑已注册用户则状态不会太大，如果为每个非注册用户分配唯一的用户名，则需要维护非常大的状态，随着时间推移就可能导致查询失败。

```
SELECT user, COUNT(url)
FROM clicks
GROUP BY user;
```

FlinkSQL 流批一体统一处理架构思想设计

动态表详解

查询限制

- 对无界数据流进行连续的查询会有一些限制，主要是如下两个方面：状态大小限制、计算成本更新限制。

计算更新成本限制

- 某些查询即使只添加或更新了一条输入记录，也需要重新计算和更新大部分发出的结果行。这样的查询不太适合作为连续查询执行。
- 例如下面的例子，它根据最后一次点击的时间为每个用户计算一个 RANK。 一旦 clicks 表收到新用户，用户的 lastAction 就会更新并计算新的排名。 但是，由于两行不能具有相同的排名，所有排名较低的行也需要更新。

```
SELECT user, RANK() OVER (ORDER BY lastAction)
FROM (
 SELECT user, MAX(cTime) AS lastAction FROM clicks GROUP BY user
);
```

目录

1 批处理以及流处理技术发展

2 FlinkSQL 流批一体统一处理架构思想设计

3 Flink Table 与 SQL 编程详解

4 Flink SQL 函数解析

Flink Table 编程详解

- 依赖管理

- flink-table-common：这个包中主要是包含 Flink Planner和 Blink Planner一些共用的代码。
- flink-table-api-java：这部分是用户编程使用的 API，包含了大部分的API。
- flink-table-api-scala：这里只是非常薄的一层，仅和 Table API的 Expression 和 DSL 相关。
- flink-table-planner： planner 计划器，是 table API 最主要的部分，提供了运行时环境和生成程序执行计划的 planner。
- flink-table-api-java-bridge：桥接器，主要用于实现通过java代码开发FlinkSQL程序

Flink Table 编程详解

创建 TableEnvironment：TableEnvironment 是FlinkSQL的核心类，主要用于实现FlinkSQL的编程操作

TableEnvironment 的功能

- 注册 catalog（可以理解为数据系统实例，比如某个 HBase 集群，某个 MySQL 服务器）；
- 在 catalog 注册库和表；
- 加载插件模块；
- 执行 SQL 查询；
- 注册 UDF；
- DataStream 和 Table 互转（仅仅在 StreamExecutionEnvironment 下）。

Flink Table 编程详解

两种 TableEnvironment

第一种：TableEnvironment 主要使用这种方式来获取TableEnvironment

- 标准的做法是创建 TableEnvironment，在创建 TableEnv 的时候，可以传入一个 EnvironmentSettings 或者TableConfig 参数，用来配置 TableEnvironment 的一些特性。

```
EnvironmentSettings settings = EnvironmentSettings
 .newInstance()
 .useBlinkPlanner()//Flink1.14开始就删除了其他的执行器了，只保留了BlinkPlanner
 .inStreamingMode()//默认就是StreamingMode
 //.inBatchMode()
 .build();
TableEnvironment tEnv = TableEnvironment.create(settings);
```

注意：Flink1.14 开始删除了其他的执行器了，只保留了 BlinkPlanner。

Flink Table 编程详解

两种 TableEnvironment

第二种：StreamExecutionEnvironment 了解一下即可

如果要混用 DataStream 和 Table API/SQL，可以使用 StreamTableEnvironment。

```
//1. 获取Stream执行环境
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
//2. 创建表执行环境
StreamTableEnvironment tEnv = StreamTableEnvironment.create(env);
```

TableEnvironment 和 StreamExecutionEnvironment 二选一即可。

Flink Table 编程详解

创建表和注册表

有了 TableEnvironment，我们接下来就需要创建表并注册表，注册表是可选的。

创建表的两种方式

创建表是为了在表上执行 TableAPI 或者 SQL 查询，可以通过如下两种方式创建表：

创建表的方式	说明	代码示例
虚拟表	从现有的table创建表，通常是从Table API&SQL查询结果创建表，这种一般是中间表	tEnv.createTemporaryView("sourceTable", projTable);
常规表	另外一种更实用的方式是通过外部数据创建表，例如文件, 数据库表, 或者消息队列，这种一般是创建输入表或者输出表，采用connector方式创建	tableEnv.createTable("SourceTableA", sourceDescriptor); tableEnv.createTemporaryTable("SourceTableB", sourceDescriptor);

Flink Table 编程详解

内置 Connectors

- Flink 的 Table API & SQL 通过 Connectors 连接外部系统，并执行批/流方式的读写操作。Connectors 提供了丰富的外部系统连接器，根据 source 和 sink 的类型，它们支持不同的格式，例如 CSV、Avro、Parquet 或 ORC。
- 注意：如果作为 sink 还要注意支持的输出模式（Append/Retract/Upsert）【官网有标记】

附带的连接器

连接器可以和多种多样的第三方系统进行交互。目前支持以下系统:

- Apache Kafka (source/sink)
- Apache Cassandra (sink)
- Amazon Kinesis Streams (source/sink)
- Elasticsearch (sink)
- FileSystem (sink)
- RabbitMQ (source/sink)
- Google PubSub (source/sink)
- Hybrid Source (source)
- Apache NiFi (source/sink)
- Apache Pulsar (source)
- JDBC (sink)

目录

1 批处理以及流处理技术发展

2 FlinkSQL 流批一体统一处理架构思想设计

3 Flink Table 与 SQL 编程详解

4 Flink SQL 函数解析

FlinkSQL 函数详解

内置 函数

Flink SQL当中有大量的内置函数，可以给我们数据处理带来极大的便利

https://nightlies.apache.org/flink/flink-docs-release-1.15/docs/dev/table/functions/systemfunctions/

FlinkSQL 函数详解

自定义函数

除了内置函数之外，FlinkSQL还支持自定义函数，我们可以通过自定义函数来扩展函数的使用

FlinkSQL当中自定义函数主要分为四大类：

ScalarFunction：标量函数

标量函数

特点：每次只接收一行的数据，输出结果也是 1 行 1 列

典型的标量函数如：upper(str), lower(str), abs(salary)

TableFunction：表生成函数

表生成函数

特点：运行时每接收一行数据（一个或多个字段），能产出多行、 多列的结果

典型的如：explode( ), unnest ( )

FlinkSQL 函数详解

自定义函数

除了内置函数之外，FlinkSQL还支持自定义函数，我们可以通过自定义函数来扩展函数的使用

FlinkSQL当中自定义函数主要分为三大类：

AggregateFunction：聚合函数

聚合函数

特点：对输入的数据行（一组）进行持续的聚合，最终对每组数据输出一行（多列）结果

典型的如：sum( ), max( )

TableAggregateFunction：表聚合函数

聚合函数

特点：对输入的数据行（一组）进行持续的聚合，最终对每组数据输出一行或多行（多列）结果

极客时间 训练营

THANKS