【Flink SQL引擎】： Calcite 功能简析及在 Flink 的应用

文章目录

1. Calcite 是什么?

- Apache Calcite 是一个动态数据的管理框架，可以用来构建数据库系统的语法解析模块
- 不包含数据存储、数据处理等功能
- 可以通过编写 Adaptor 来扩展功能，以支持不同的数据处理平台
- Flink SQL 使用并对其扩展以支持 SQL 语句的解析和验证

2. Calcite 谁在用?

下图是一张官方提供的生态系统图，可以看到大名鼎鼎的 Hive、Flink、Druid 以及 Spark、ES 等都可以被纳入 Calcite 生态圈。

图1. Calcite生态圈

3. 概念解析

关系代数	Relational algebra	`RelNode`
行表达式	Row expressions	`RexNode`
特征	Traits	`RelTrait`
转化特征	Conventions	`Convention`
规则	Rules	`RelOptRule`
规划器	Planners	`RelOptPlanner`
程序	Programs	`Program`

图2. Calcite 概念

1. **关系代数（Relational algebra）**：即关系表达式。它们通常以动词命名，例如

Sort, Join, Project, Filter, Scan, Sample.

2. **行表达式（Row expressions）**：例如 RexLiteral (常量), RexVariable (变量), RexCall (调用) 等，例如投影列表（Project）、过滤规则列表（Filter）、JOIN 条件列表和 ORDER BY 列表、WINDOW 表达式、函数调用等。使用 RexBuilder 来构建行表达式。
3. **表达式有各种特征（Trait）**：使用 Trait 的 satisfies() 方法来测试某个表达式是否符合某 Trait 或 Convention.
4. **转化特征（Convention）**：属于 Trait 的子类，用于转化 RelNode 到具体平台实现（可以将下文提到的 Planner 注册到 Convention 中）. 例如 JdbcConvention，FlinkConventions.DATASTREAM 等。同一个关系表达式的输入必须来自单个数据源，各表达式之间通过 Converter 生成的 Bridge 来连接。
5. **规则（Rules）**：用于将一个表达式转换（Transform）为另一个表达式。它有一个由 RelOptRuleOperand 组成的列表来决定是否可将规则应用于树的某部分。
6. **规划器（Planner）**：即请求优化器，它可以根据一系列规则和成本模型（例如基于成本的优化模型 VolcanoPlanner、启发式优化模型 HepPlanner）来将一个表达式转为语义等价（但效率更优）的另一个表达式。

4. 整体模块和处理流程

- **Catalog** – 定义元数据和命名空间，包含 Schema（库）、Table（表）、RelDataType（类型信息）

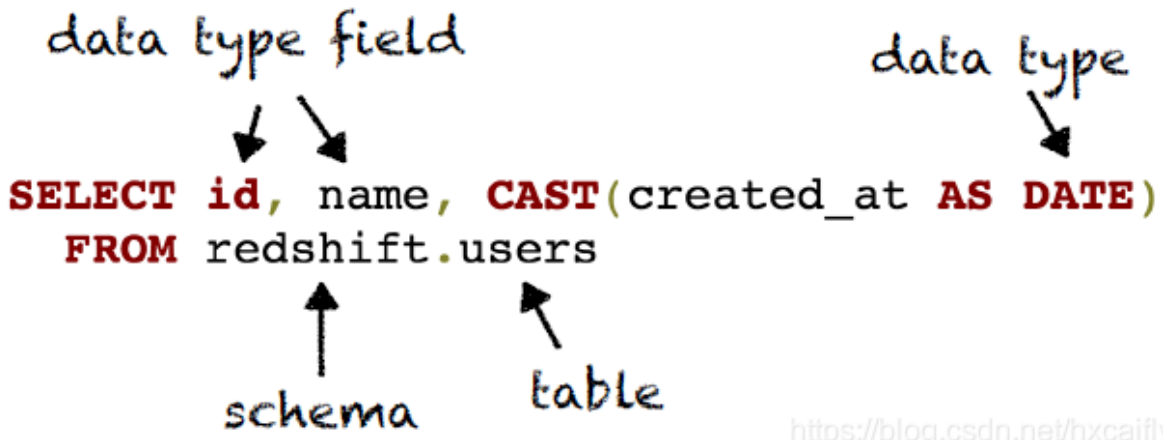

图3. Catalog 说明

- **SQL Parser** – 将用户编写的 SQL 语句转为 SqlNode 构成的抽象语法树（AST）
  - 通过 JavaCC 模版生成 LL(k) 语法分析器，主模版是 Parser.jj；可对其进行扩展

- 负责处理各个 Token，逐步生成一棵 SqlNode 组成的 AST

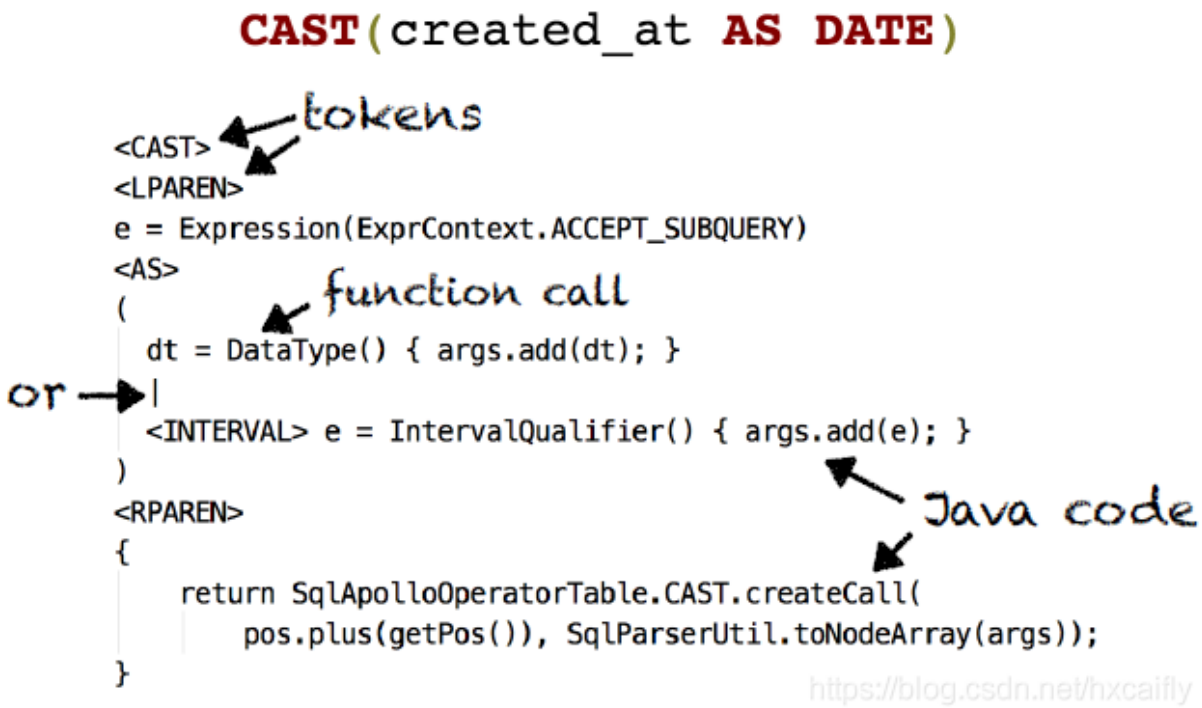

图4. Parser 模板

- **SQL Validator** – 使用 Catalog 中的元数据检验上述 SqlNode AST 并生成 RelNode 组成的 AST
- **Query Optimizer** – 将 RelNode AST 转为逻辑计划，然后优化它，最终转为实际执行方案。以下是一些常见的优化规则（Rules）：

1. 移除未使用的字段
2. 合并多个投影（projection）列表
3. 使用 JOIN 来代替子查询
4. 对 JOIN 列表重排序
5. 下推（push down）投影项
6. 下推过滤条件

推荐阅读：
1 . 是时候学习真正的 spark 技术了
2. 一文搞懂spark sql中的CBO（基于代价的优化）

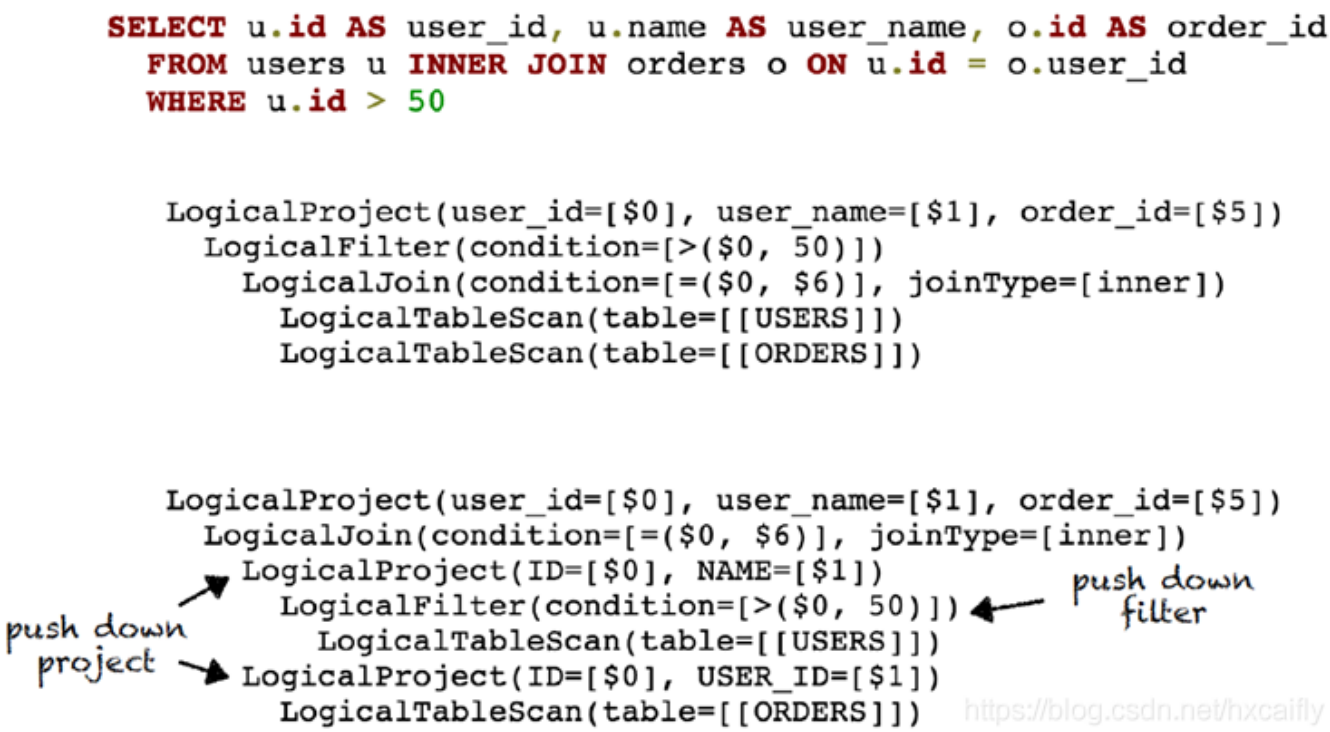

图5. 优化规则示例： 下推

整体而言，Calcite 处理流程整体可以分为 Parse（语法和语义解析，生成 SqlNode 树）、Validate（验证各对象是否已在 Catalog 中注册）、Optimize（优化、生成 RelNode 树以及物理执行计划）、Execute（具体执行）四个阶段。

5. 流处理语句支持现状

Calcite 支持部分 SQL 流处理语句，也提供了对 Tumbling / Hopping / Sliding / Cascading 等类型 Window 的支持，而 Flink 则把 Window 分为 Tumbling、Sliding (Hopping in SQL)、Session、Global 等类型，与 Calcite 提供的并不完全一致。

目前 Calcite 流处理语句已实现对 SELECT, WHERE, GROUP BY, HAVING, UNION ALL, ORDER BY 以及 FLOOR, CEIL 函数的支持。

6. Flink 与 Calcite

下图是 Flink 系统结构，其中 Table API 与 SQL 模块以 Calcite 为核心，大量用到 Calcite 的各种类和方法。

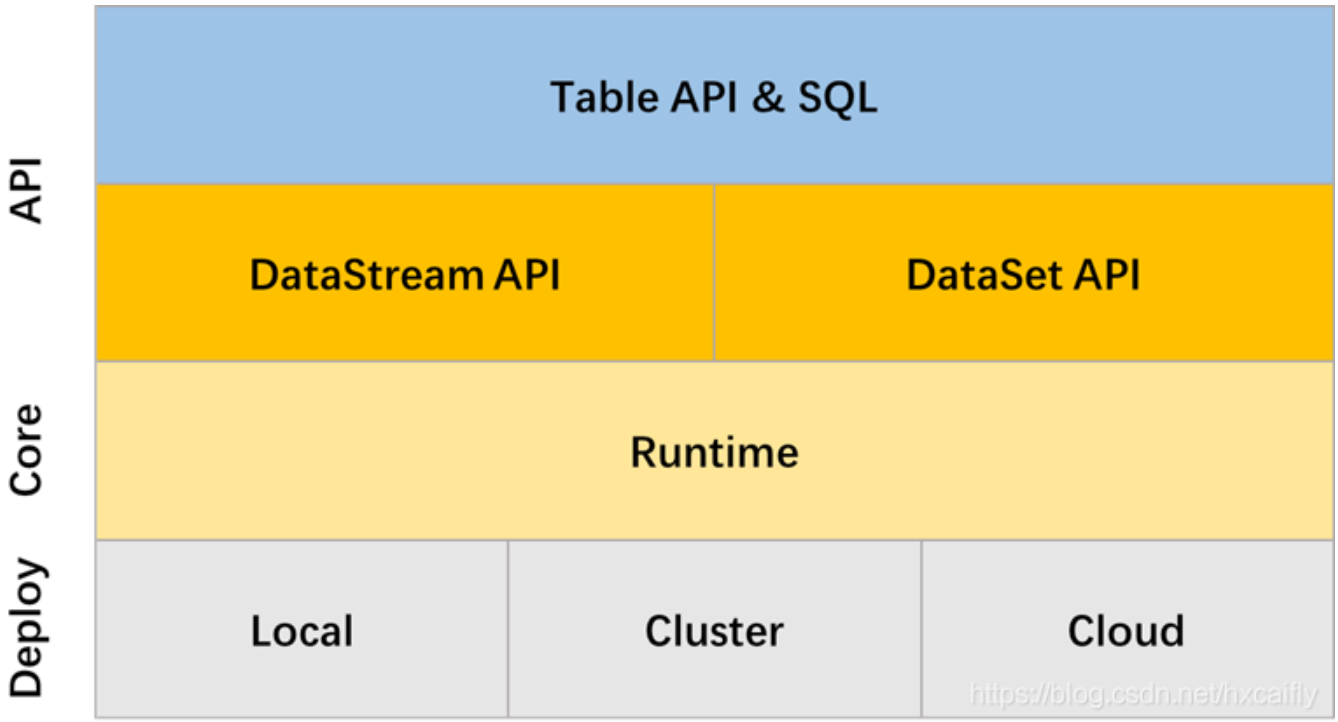

图6. Flink 系统结构

下图是 Flink Table 模块的内部表示。

可以看到它以 Calcite Catalog 为核心，上面承载了 Table API 和 SQL API 两套表达方式，最后殊途同归，统一生成为 Calcite Logical Plan（SqlNode 树）；随后验证、优化为 RelNode 树，最终通过 Rules（规则）和 Convention（转化特征）生成具体的 DataSet Plan（批处理）或 DataStream Plan（流处理），即 Flink 算子构成的处理逻辑。

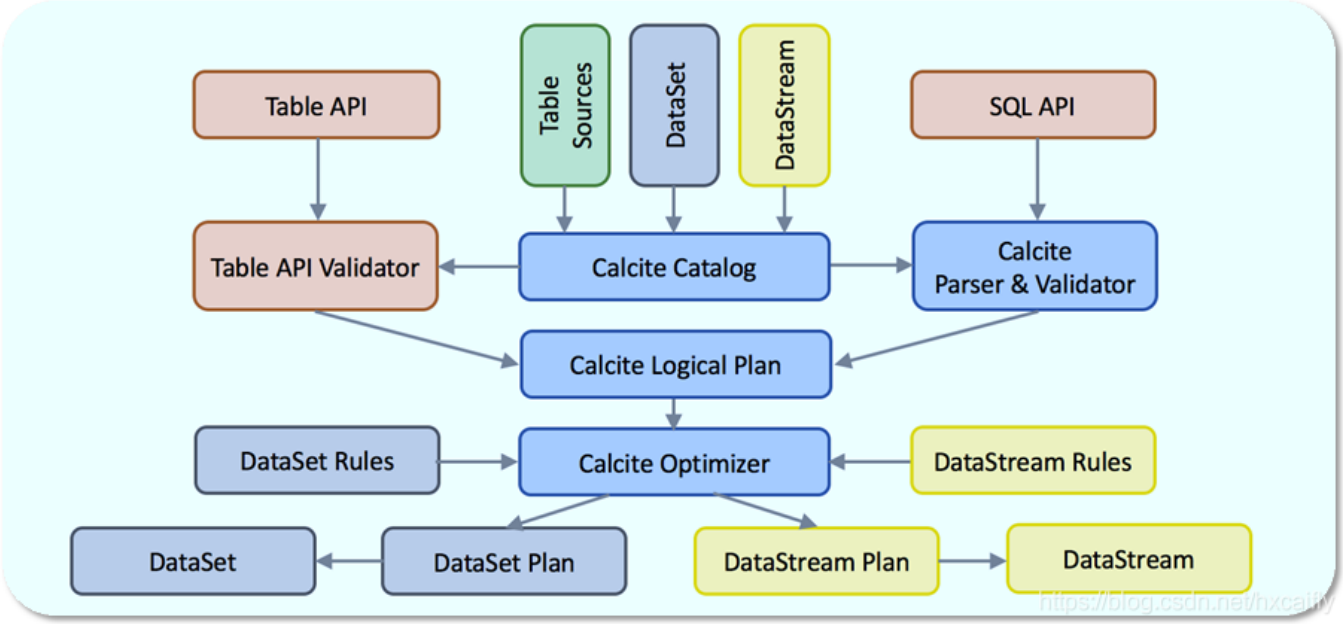

图7. Flink Table 模块功能图

下图是 SQL 和 Table API 两种表达形式的处理逻辑，上下两种是等价的：

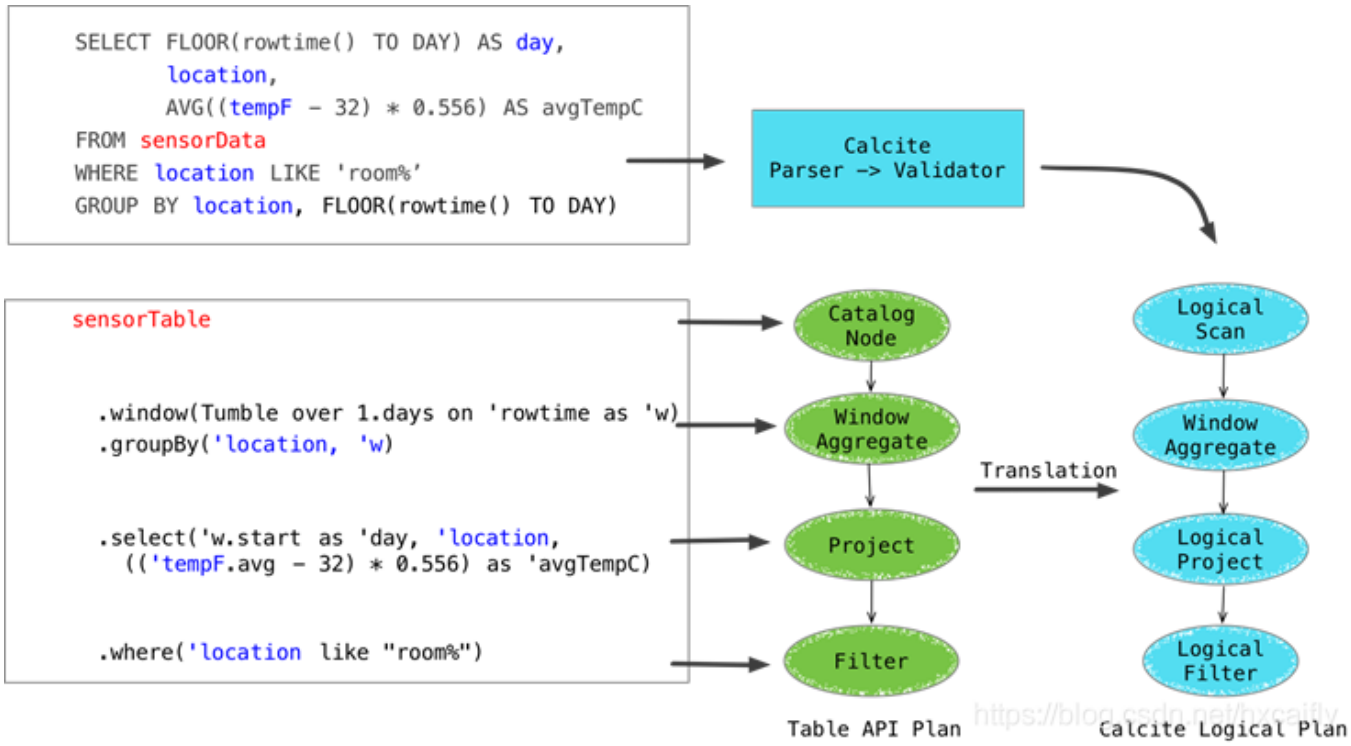

图8. Flink Table API / SQL 转换流程示例

总而言之，Table / SQL API 的编程框架如下：

1. 通过 TableEnvironment 配置 CalciteConfig 对象，自动设置 SQL & Table API 默认处理参数。
2. 使用 registerTableSource() 来将一个 TableSource 注册到 rootSchema. 后续可以通过 scan() 获取此 Table 并调用各种 Table API 进行处理。
3. 接下可以调用 sqlQuery() 和 sqlUpdate() 方法来使用 SQL 语句进行数据处理。

运行时 Demo

下面的案例展示了对一句 SQL 查询的中间和最终处理结果：

```
SELECT stateCode, county, SUM(point_granularity) FROM source WHERE stateCode = 'FL' GROUP BY stateCode, county

== Abstract Syntax Tree ==
LogicalAggregate(group=[{0, 1}], EXPR$2=[SUM($2)])
  LogicalProject(stateCode=[$1], county=[$2], point_granularity=[$17])
    LogicalFilter(condition=[=($1, _UTF-16LE'FL')])
      LogicalTableScan(table=[[source]])


== Optimized Logical Plan ==
DataStreamGroupAggregate(groupBy=[stateCode, county], select=[stateCode, county, SUM(point_granularity) AS EXPR$2])
  DataStreamCalc(select=[CAST(_UTF-16LE'FL') AS stateCode, county, point_granularity], where=[=(stateCode, _UTF-16LE'FL')])
    StreamTableSourceScan(table=[[source]], fields=[county, point_granularity, stateCode], source=[CsvTableSource(read fields: county, point_granularity, stateCode)])


== Physical Execution Plan ==
Stage 1 : Data Source
    content : collect elements with CollectionInputFormat

    Stage 2 : Operator
        content : Split Reader: Custom File source
        ship_strategy : REBALANCE

        Stage 3 : Operator
            content : Map
            ship_strategy : FORWARD

            Stage 4 : Operator
                content : where: (=(stateCode, _UTF-16LE'FL')), select: (CAST(_UTF-16LE'FL') AS stateCode, county, point_granularity)
                ship_strategy : FORWARD

                Stage 6 : Operator
                    content : groupBy: (stateCode, county), select: (stateCode, county, SUM(point_granularity) AS EXPR$2)
                    ship_strategy : HASH
```


图9. SQL 语句、生成的 AST、优化后的逻辑计划、最终物理计划