Flink Table API and SQL

# 切入点

对于不同的数据（流数据和批数据），flink提供了不同的切入点

## 批数据

|  |
| --- |
| ExecutionEnvironment env **=** ExecutionEnvironment**.**getExecutionEnvironment**();**  BatchTableEnvironment tableEnv **=** TableEnvironment**.**getTableEnvironment**(**env**);** |

## 流数据

|  |
| --- |
| StreamExecutionEnvironment env **=** StreamExecutionEnvironment**.**getExecutionEnvironment**();**  StreamTableEnvironment tableEnv **=** TableEnvironment**.**getTableEnvironment**(**env**);** |

# 表注册

tableEnvironments 有一个内部的catalog来记录注册的表（表名唯一），注册后，可以通过唯一的表名来访问

类似于spark的createOrReplaceTempView, flink中也有注册表的接口

4种不同数据注册的方式

## Dataset类型

|  |
| --- |
| tableEnv**.**registerDataSet**(**"Customers"**,** cust**)**  tableEnv**.**registerDataSet**(**"Orders"**,** ord**,** "user, product, amount"**);** |

## datastream类型

|  |
| --- |
| tableEnv.registerDataStream("Customers", cust)  tableEnv.registerDataStream("Orders", ord, "user, product, amount"); |

## table类型

|  |
| --- |
| Table custT **=** tableEnv  **.**toTable**(**custDs**,** "name, zipcode"**)**  **.**where**(**"zipcode = '12345'"**)**  **.**select**(**"name"**)**  tableEnv**.**registerTable**(**"custNames"**,** custT**)** |

以上有错误！

Dataset转table使用以下方式：

Table custT = tableEnv.fromDataSet(custDs);

## 外部数据（tableSource）

|  |
| --- |
| TableSource custTS **=** **new** **CsvTableSource(**"/path/to/file"**,** **...)**  tableEnv**.**registerTableSource**(**"Customers"**,** custTS**)** |

**Available Table Sources**

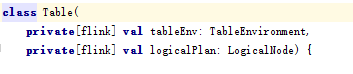
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classname**. | **Maven dependency** | **Batch?** | **Streaming?** | **Description** |
| CsvTableSouce | flink-table | Y | Y | A simple source for CSV files |
| Kafka08JsonTableSource | flink-connector-kafka-0.8 | N | Y | A Kafka 0.8 source for JSON data |
| Kafka09JsonTableSource | flink-connector-kafka-0.9 | N | Y | A Kafka 0.9 source for JSON data |

# Table api

Flink



Dataset/Datastream和table可以互相转化



Table操作时lazy的，生成logicalPlan

在我们从DataSet或者DataStream创建了Table之后，可以利用类似于filter, join, 或者 select关系型转化操作来转化为一个新的Table对象。而从内部实现上来说，所有应用于Table的转化操作都会变成一棵逻辑表操作树，在Table对象被转化回DataSet或者DataStream之后，专门的转化器会将这棵逻辑操作符树转化为对等的DataSet或者DataStream操作符。譬如'location.like("room%")这样的表达式会经由代码生成编译为Flink中的函数。

|  |  |
| --- | --- |
| As | 重命名 |
| distinct | 去重 |
| groupBy | tab.groupBy("key").select("key, value.avg") |
| intersect | left.intersect(right) return相同元素 |
| join |  |
| leftOuterJoin |  |
| rightOuterJoin |  |
| fullOuterJoin |  |
| limit |  |
| minus,minusAll | 左边减去右边table的元素 |
| union,unionAll | 合并去重，合并不去重 |
| orderby |  |

E.g:

Table wordCounts = table

.groupBy("word")

.select("word, count.sum as count");

完整的api：

<https://ci.apache.org/projects/flink/flink-docs-release-1.2/api/java/org/apache/flink/api/table/Table.html>

流数据的api仅支持selection,projection,and union

Table API Stream Aggregations

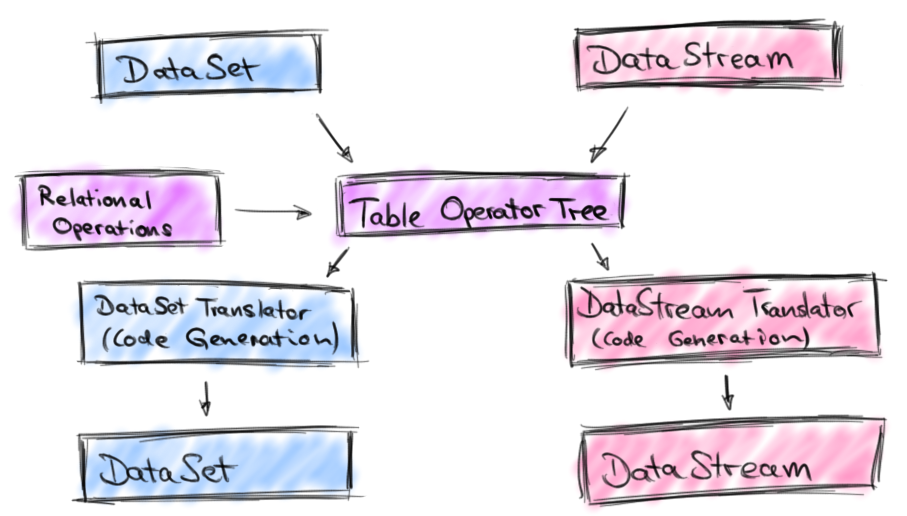
<https://issues.apache.org/jira/browse/FLINK-4557>

# SQL

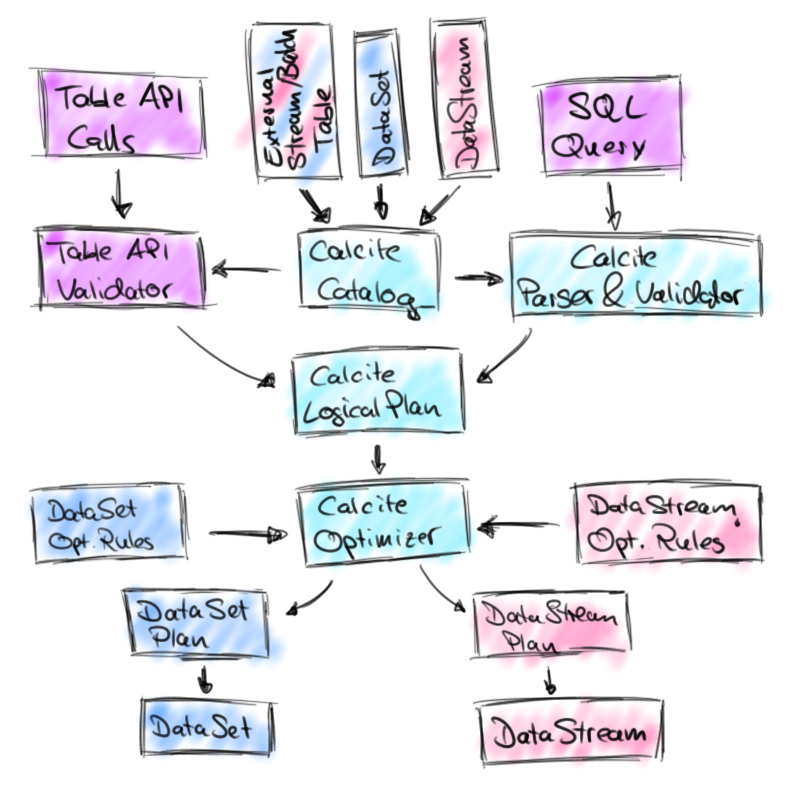
e.g: Table result = tableEnv.sql(**"SELECT count(name) FROM student"**);

Table, dataset, datastream, external tablesource 必须先在TableEnvironment中注册才能使用sql查询

老架构



新架构



新的架构主要是将Table API与SQL集成起来，用这两货的API构建的查询最终都会转化到Calcite的所谓的logicl plans表述。转化之后的流查询与批查询基本上差不多，然后Calcite的优化器会基于转化和优化规则来优化这些logical plans，针对数据源（流还是静态数据）的不同我们会应用不同的规则。最后，经过优化的logical plan会转化为一个普通的Flink DataStream或者DataSet对象，即还是利用代码生成来将关系型表达式编译为Flink的函数。

## 批数据

ExecutionEnvironment env **=** ExecutionEnvironment**.**getExecutionEnvironment**();**

BatchTableEnvironment tableEnv **=** TableEnvironment**.**getTableEnvironment**(**env**);**

*// read a DataSet from an external source*

DataSet**<**Tuple3**<**Long**,** String**,** Integer**>>** ds **=** env**.**readCsvFile**(...);**

*// register the DataSet as table "Orders"*

tableEnv**.**registerDataSet**(**"Orders"**,** ds**,** "user, product, amount"**);**

*// run a SQL query on the Table and retrieve the result as a new Table*

Table result **=** tableEnv**.**sql**(**

"SELECT SUM(amount) FROM Orders WHERE product LIKE '%Rubber%'"**);**

限制：

当前版本支持

selection (filter),

projection,

inner equi-joins,

grouping,

non-distinct aggregates

sorting on batch tables.

Among others, the following SQL features are not supported, yet:

* Distinct aggregates (例如COUNT(DISTINCT name))
* 非等值join连接 和笛卡尔积
* Grouping sets语法
* 不支持order by fetch（limit X）

## 流数据（SQL支持重点）

和批数据的处理方式类似

|  |
| --- |
| StreamExecutionEnvironment env **=** StreamExecutionEnvironment**.**getExecutionEnvironment**();**  StreamTableEnvironment tableEnv **=** TableEnvironment**.**getTableEnvironment**(**env**);**  *// ingest a DataStream from an external source*  DataStream**<**Tuple3**<**Long**,** String**,** Integer**>>** ds **=** env**.**addSource**(...);**  *// register the DataStream as table "Orders"*  tableEnv**.**registerDataStream**(**"Orders"**,** ds**,** "user, product, amount"**);**  *// run a SQL query on the Table and retrieve the result as a new Table*  Table result **=** tableEnv**.**sql**(**  "SELECT product, amount FROM Orders WHERE product LIKE '%Rubber%'"**);** |

限制：仅支持select,from,where 和union语句，不支持aggregate和join

以上限制均基于Flink v1.2 doc

<https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/table_api.html>

Stream Processing for Everyone with SQL and Apache Flink

<https://flink.apache.org/news/2016/05/24/stream-sql.html>

val roomSensors: Table = tableEnv.sql(

"SELECT STREAM time, location AS room, (tempF - 32) \* 0.556 AS tempC " +

"FROM sensorData " +

"WHERE location LIKE 'room%'"

)

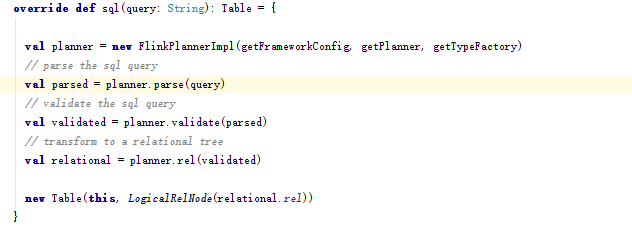
# Calcite

类似于spark 的catalyst，flink 用calcite来进行sql的解析

flink借助Apache Calcite做了sql解析、逻辑树生成的过程，得到Calcite的RelRoot类，生成flink的Table，Table里的执行计划会转化成DataSet的计算，经历物理执行计划优化等步骤。

## 执行过程

类比Spark SQL，Calcite代替了大部分Spark SQL Catalyst的工作。两者最终是计算一颗逻辑执行计划树



1. TableEnvironment.sql()为调用入口
2. 类似Calcite的PlannerImpl，flink实现了个FlinkPlannerImpl，执行parse(sql)，validate(sqlNode)，rel(sqlNode)操作
3. 生成Table

## Stream // todo

与relation sql的区别：

Relation SQL：历史数据

Stream SQL：现在+未来的数据

Calcite中的streamSQL 语法是基于SQL的扩展的

Window