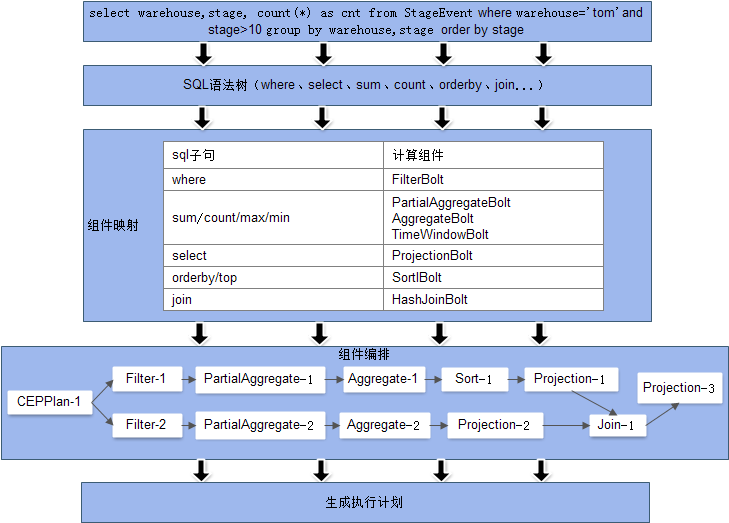
# EPL设计文档

# 创建执行计划



图中由上至下为构建执行计划的过程，总体分为如下过程:

1. 分析输入SQL语句，构建SQL语法树，生成SQL子句
2. 组件映射，把解析出的SQL子句映射到计算组件
3. 按照一定规则对已映射组件进行编排
4. 生成执行计划

## SQL解析

对输入的SQL，根据标准的SQL关键字来构建语法树，解析出SQL包含的select、where、groupby、orderby、join等子句以及对应的数据，如图中的SQL语句需要分析出如下数据：

|  |  |
| --- | --- |
| 子句 | 数据 |
| select | warehouse,stage, count(\*) as cnt |
| where | warehouse='tom'and stage>10 |
| groupby | warehouse ,stage |
| orderby | stage |
| 聚合函数 | count(\*) |

## 组件映射

组件是包含在EPL(Event Processing Language)框架中的运算单元，解析后的SQL子句可以映射为一个或多个组件。

EPL中包含的SQL运算组件：

* 1. where子句映射为EPL中的FilterBolt组件用于对流入数据的过滤
  2. sum/count/max/min等聚合函数组件映射为PartialAggregateBolt(用于计算单个聚合函数的值)、AggregateBolt(合并单个聚合函数值及计算符合聚合函数的值)和TimeWindowPartialAggregate(基于时间窗口的PartialAggregateBolt组件)
  3. select子句映射为ProjectionBolt，用于获取结果所需的字段或计算的函数值
  4. orderby子句映射为SortBolt，对流入的数据按指定的字段进行升序或降序排列。
  5. join映射为HashJoinBolt，实现对两个输入流的合并。

### Where子句的映射

Where子句映射为EPL中的过滤组件FilterBolt，对数据流进行过滤，如果符合where条件则通过到下面的组件，否则返回不继续计算。

例如Where子句：warehouse='tom' and stage>10，过滤组件会把该表达式构建为一棵计算树，这样任何进入该过滤组件的数据都会根据这棵树进行过滤。例如：FilterBolt组件会从流数据中获取warehouse字段的值，如果等于Tom，继续获取stage的值，如果大于10,则该数据可以通过，完成where子句的计算。



从where子句的计算过程可以看出，流中每组数据之间是没有关联关系的，即每组数据的过滤都可以独立完成，所以where子句可以分布在多个节点上完成，实现并行计算。

### 聚合函数的映射

类似于sum/count/max/min这样的聚合函数，它们的计算分解为PartialAggregateBolt和AggregateBolt两个组件完成。

对于单一聚合函数，如：sum(A)，可以把流入的数据分解到多个PartialAggregateBolt组件，每个组件完成一批数据的一部分计算，即完成sum(A)的部分和计算，随后就需要在AggregateBolt组件中把各个PartialAggregateBolt计算的值合并为一个值完成聚合函数计算。同样道理max/min会比较多个PartialAggregateBolt计算结果，合并为一个最大或最小值。

对于组合聚合函数表达式，如：sum(A)/count(A)+max(B)，会把该表达式分解为sum(A)、count(A)、max(B)，这样每个PartialAggregateBolt组件会根据流入的数据同时计算这三个聚合函数，在AggregateBolt组件中合并三个表达式的值，再计算组合表达式的值。如下图：



图中左面为聚合函数表达式对应的树结构，PartialAggregateBolt组件计算树的叶子节点sum(A)、count(A)、max(B)，这样可以利用多个PartialAggregateBolt并行计算单一聚合函数，同时PartialAggregateBolt用A1、A2、A3三个标识符分别代替聚合函数在树中的位置，并把该树结构放在AggregateBolt组件中。

当多个PartialAggregateBolt组件计算后，以{A1:sum(A),A2:count(A),A3:max(B)}格式，发送到AggregateBolt中，AggregateBolt首先合并各个PartialAggregateBolt发来的三个聚合函数的值，类似于计算单个聚合函数的方式，之后根据树中的标识(A1、A2、A3)替换为合并后的聚合函数值，最终后续遍历树，完成整棵树的计算，得出组合聚合函数表达式的值。

### Order by/Top子句的映射

Orderby子句根据指定的字段对输入的数据进行排序及取前Top个数据，该组件对收到的数据首先进行收集，比如10秒钟，之后对这段时间内的数据进行升序或降序排序，如果有Top操作，则按照降序排序，取前top个数据。

### Select子句的映射

Select子句从输入流中选择特定字段的值，从过滤、聚合、排序等组件处理之后的流中，选择部分或全部字段。

### Group By子句的映射

Groupby子句主要由聚合组件来计算。在聚合函数中每个聚合函数的计算都是和groupby子句的值进行关联的，即不同的分组对应不同的聚合函数值。当SQL语句中不包含groupby子句，则添加一个默认分组，此时全部的聚合函数值都计算都为一个值。

### Join子句的映射

Join子句完成对多个流的连接，连接的流可以是经过完整SQL处理的流或数据流。该组件针对订阅的两个流，根据join的连接条件进行流的匹配，如果在给定时间内流中数据匹配成功则继续处理join之后别的组件如：orderby、where等，如果无法匹配则会缓存一定的时间，如果过期，则丢弃这些流数据。

## 组件编排

上面阐述了SQL子句与计算组件之间的映射关系，组件间的关系如下所述：

1. 随机分发：随机派发流中的数据到下个组件的某个任务中。
2. 按字段分发：按照流中一个或多个字段来分组， 具有同样字段值的数据会被分到相同的组件实例中， 而不同的值则会被分配到不同的组件。 比如按照groupby字段进行分组。
3. 全局分发：一个或多个组件计算的数据值被分配到一个组件任务中。

在各个SQL子句根据规则生成不同的组件后，EPL按照如下的算法，把组件组合起来：

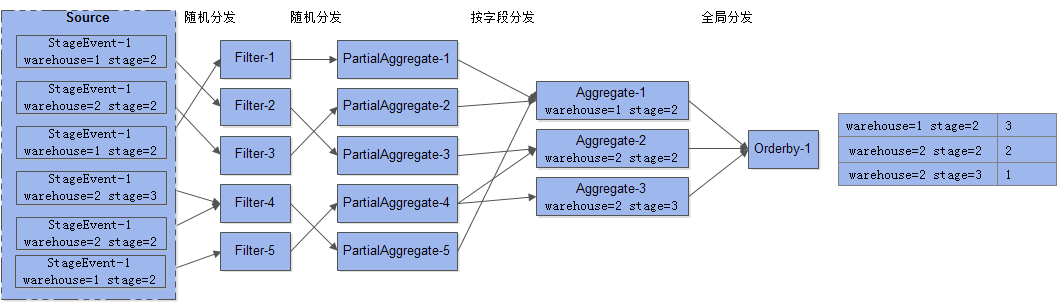
* 未包含Join语句：
  1. SQL语句是否包含Where子句，有则在执行计划中添加过滤组件，并且过滤组件与原始输入流的关系是随机分发。
  2. 是否包含聚合函数，有则添加PartialAggregate组件，并且与前一个组件的关系为随机分发，接着添加Aggreagte组件，与PartialAggregate的关系为按字段分发，如果包含groupby则按照该字段分发，如果不包含groupby，则按照系统添加的默认字段分发。
  3. 是否包含top子句，如果包含则添加sort组件，sort组件首先对数据进行排序，之后选择前top个，sort组件按照groupby字段分发方式与前面组件连接。由于top操作不需要进行总体排序，所以可以在多个节点进行计算取局部top，最后再在全局节点合并。
  4. 如果有top子句，则添加orderby组件，并且与top组件由全局分发方式连接。
  5. 如果没有top子句而包含orderby，则以全局分发方式添加orderby组件。
  6. 如果top和orderby都不存在，则以全局分发方式添加projection组件。
* 包含join组件：
  1. 如果join连接的流为SQL处理后的流，则连接的子SQL以未包含Join语句的方式添加到执行计划中。
  2. 如果为原始流，则直接添加到执行计划中。
  3. 处理完连接流后，在执行计划中添加join组件到执行计划，并且与连接的流以字段分组的方式关联起来，字段为join中判断流连接的条件。
  4. 类似与未包含join语句的方式，继续添加where、聚合函数、top、orderby等组件。

构建算法：

|  |
| --- |
| **contructPlan**(SqlStatement statement,CEPPlan plan){  if(**Retrieve**(statement) contain where clause){  FilterBolt filterBolt=**buildComponent**(“where”);  Add FilterBolt to plan with shuffle Grouping  **}**  if(**Retrieve**(statement) contain aggregate clause){  PartialAggregateBolt pritialAggregateBolt=**buildComponent**(“partial”);  If(filterBolt!=null){  Add pritialAggregateBolt to filterBolt with shuffle Grouping  }else{  Add pritialAggregateBolt to plan with shuffle Grouping  }  AggregateBolt aggregateBolt= **buildComponent**(“aggregate”);  Add aggregateBolt to pritialAggregateBolt with field Grouping  }  if(**Retrieve**(statement) contain orderby clause){  if(**Retrieve**(statement) contain top clause){  SortBolt sortTopBolt= **buildComponent**(“sort/top”);  Add sortTopBolt to AggregateBolt with field Grouping  SortBolt sortBolt= **buildComponent**(“sort”);  Add sortBolt to sortTopBolt with global Grouping  }else{  SortBolt sortBolt= **buildComponent**(“sort”);  Add sortBolt to AggregateBolt with global Grouping  }  }else{  ProjectBolt projectBolt= **buildComponent**(“select”);  Add projectBolt to AggregateBolt with global Grouping  }  return plan;  }  **buildComponent**(String boltType){  return BoltFactory.build(boltType);  }  **contructJoinPlan**(SqlStatement statement,CEPPlan plan){  CEPPlan plan1=**contructPlan(get(left sql clause),plan);**  CEPPlan plan2=**contructPlan(get(left sql clause),plan);**  HashJoinBolt joinBolt= **buildComponent**(“join”);  Add joinBolt to plan with shuffle Grouping  Add plan1 to joinBolt with field Grouping  Add plan2 to joinBolt with field Grouping  Return plan;  } |

## 生成执行计划

通过解析、任务分解、任务编排步骤后，整个SQL语句形成一条以不同关联方式串联起来的计算组件链条。如下图所示为例子中构建完成的执行计划：



# 生成Topology

流式计算框架以Topology为单位提交任务，一个Topology由一个或多个执行计划构成，如下图所示：



1. Topology基本结构：

Topology以列表方式存储全部执行计划，每条执行计划链对应一条提交的sql语句。

1. Topology提交执行：

迭代topology中的执行计划，并把plan中每个组件都及组件间关联关系设置到storm中。如下：

|  |
| --- |
| **submitTopology(){**  for each plan in topology  Recursive each component in plan  Stormbuilder.setBolt(component.getName(),component.getBolt())  Stormbuilder.setGrouping(component.getGrouping());  Stormbuilder.submit();  **}** |