聚合查询支持表达式

目标

简单说明

现有聚合查询支持每个ResultColumn的Expression有且仅有一个聚合函数的情况，比如

```
select count(a),count(a),sum(b) from root.sg
```

表达式是不支持的，比如

```
select count(a)+count(b), sum(b)+1 from root.sg
```

我们的目标就是让单个聚合查询的结果可以视作常数(一般只有数值才需要表达式计算)，然后计算表达式。见实现示例。

实现示例

- sum(a),sum(b)原始值

```
IoTDB> select sum(a),sum(b) from root.sg
+--------------+--------------+
|sum(root.sg.a)|sum(root.sg.b)|
+--------------+--------------+
|          11.0|           9.0|
+--------------+--------------+
Total line number = 1
It costs 0.008s
```

- count(a),count(b)原始值

```
IoTDB> select count(a),count(b) from root.sg
+---------------+---------------+
|count(root.sg.a)|count(root.sg.b)|
+---------------+---------------+
|              4|              3|
+---------------+---------------+
Total line number = 1
It costs 0.008s
```

- select count(a)+count(b) from root.sg

```
IoTDB> select count(a)+count(b) from root.sg
+-----------------------------------+
|count(root.sg.a) + count(root.sg.b)|
+-----------------------------------+
|                                7.0|
+-----------------------------------+
Total line number = 1
It costs 0.006s
```

- select count(a)+count(b), count(a)+count(b) from root.sg

```
IoTDB> select count(a)+count(b),count(a)+count(b) from root.sg
+-----------------------------------+-----------------------------------+
|count(root.sg.a) + count(root.sg.b)|count(root.sg.a) + count(root.sg.b)|
+-----------------------------------+-----------------------------------+
|                              7.0|                               7.0|
+-----------------------------------+-----------------------------------+
Total line number = 1
It costs 0.007s
```

- select sum(a)/count(b), sum(b)+sum(a) from root.sg

```
IoTDB> select sum(a)/count(b), sum(b)+sum(a) from root.sg
+------------------------------+----------------------------+
|sum(root.sg.a) / count(root.sg.b)|sum(root.sg.b) + sum(root.sg.a)|
+------------------------------+----------------------------+
|              3.6666666666666665|                        20.0|
+------------------------------+----------------------------+
Total line number = 1
It costs 0.010s
```

- select count(a)+1-2, count(b)*5 from root.sg

```
IoTDB> select count(a)+1-2, count(b)*5 from root.sg
+-------------------------+-------------------+
|(count(root.sg.a) + 1) - 2|count(root.sg.b) * 5|
+-------------------------+-------------------+
|                      3.0|               15.0|
+-------------------------+-------------------+
Total line number = 1
It costs 0.015s
```

- select count(a)+count(b), count(b) from root.sg

```
IoTDB> select count(a)+count(b), count(b) from root.sg
+-----------------------------------+---------------+
|count(root.sg.a) + count(root.sg.b)|count(root.sg.b)|
+-----------------------------------+---------------+
|                                7.0|              3|
+-----------------------------------+---------------+
Total line number = 1
It costs 4.121s
```

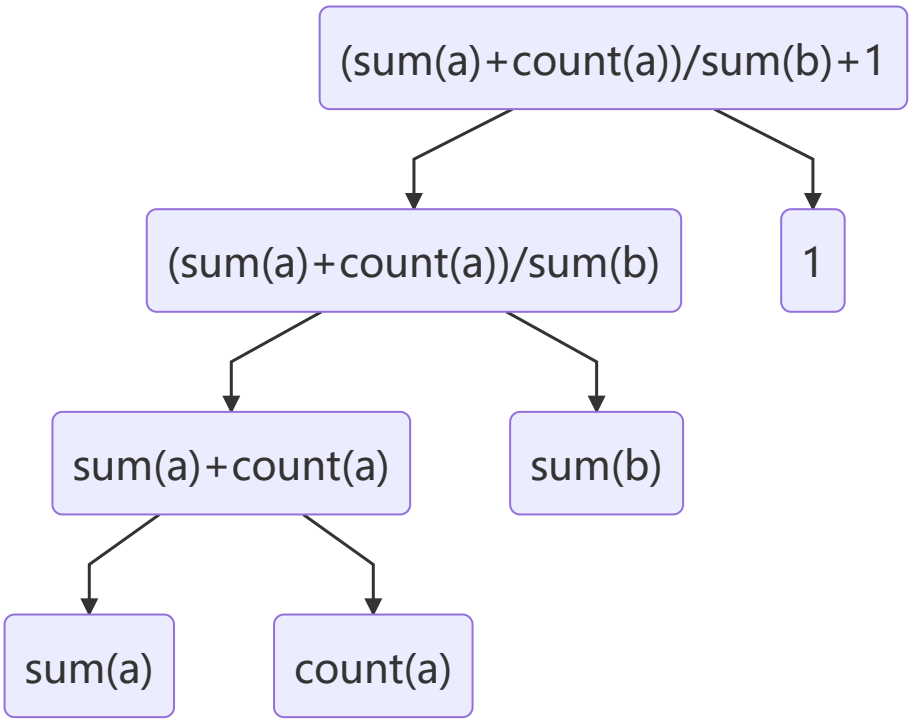
设计思想

Expression

相关类位置：org.apache.iotdb.db.query.expression

```
select (sum(a)+count(a))/sum(b)+1,sum(a),sum(b) from root.sg
```

对于上述表达式, (sum(a)+count(a))/sum(b)+1, sum(a), sum(b)对应三个ResultColumn，ResultColumn会对应一个Expression。IoTDBSqlVisitor会解析表达式，并递归将表达式拆解为多叉树的结构，以(sum(a)+count(a))/sum(b)+1为例，最后Expression结构如下



可以看到，多叉树的叶子节点为常数或者聚合函数。常数无须额外计算，我们只需要拿到每个聚合函数的结果，然后自底向上计算表达式结果即可。

内嵌聚合查询

结合Expression部分，要拿到聚合函数的结果，我们可以复用已经实现的聚合查询的逻辑。

原始聚合查询入口如下

- 首先通过AggregationQueryOperator生成AggregationPlan，这一步主要需要的来自SelectComponent的输入如下

```
rawDataQueryPlan.setPaths(selectComponent.getPaths());
rawDataQueryPlan.setResultColumns(selectComponent.getResultColumns());
rawDataQueryPlan.setEnableTracing(enableTracing);
rawDataQueryPlan.setDataTypes(generator.getSeriesTypes(selectComponent.getPaths(
)));
```

```
select count(a),count(a),sum(b),sum(a) from root.sg
对于上述语句就有
paths->[root.sg.a,root.sg.a,root.sg.b,root.sg.a]
resultColumns->[count(root.sg.a),count(root.sg.a),sum(root.sg.b),sum(root.sg.a)]
```

- 拿到AggregationPlan，通过QueryRouter.aggregate()会得到SingleDataset，其中dataSet的record包含我们想要的聚合查询结果

因此我们需要从UDAF中拿到生成内嵌聚合查询的paths / resultColulmns / enableTracing / DataTypes

```
select (sum(a)+count(a))/sum(b)+1,sum(a),sum(b) from root.sg
对于上述语句，我们希望得到的，传给innerAggregationPlan的paths和resultColumns如下
paths->[root.sg.a,root.sg.a,root.sg.b,root.sg.a,root.sg.b]
resultColumns->
[sum(root.sg.a),count(root.sg.a),sum(root.sg.b),sum(root.sg.a),sum(root.sg.b)]

可以认为我们其实是先把语句拆成了
select sum(a),count(a),sum(b),sum(a),sum(b) from root.sg
```

计算表达式

在生成innerAggregationPlan的时候，维护了如下map结构，通过这个map，我们可以得到表达式中每一个聚合查询的值

```
private Map<Expression, Integer> expressionToInnerResultIndexMap;
```

拿到聚合查询结果之后，我们通过UDAFQueryExecutor.calcUDAFExpression()方法计算原始UDAFPlan每一个ResultColumn的结果，使用递归计算即可。

相关类

UDAFQueryOperator

位置：org.apache.iotdb.db.qp.logical.crud.UDAFQueryOperator

关键属性

```
// 通过原始resultColumns，获取内嵌聚合查询的resultColumns
private ArrayList<ResultColumn> innerResultColumnsCache;

private ArrayList<PartialPath> innerPathsCache;

private ArrayList<String> innerAggregationsCache;

private Map<Expression, Integer> expressionToInnerResultIndexMap = new HashMap<>
();
```

UDAFPlan

位置：org.apache.iotdb.db.qp.physical.crud.UDAFPlan

这个类主要重写了deduplicate()方法，参照了UDTFPlan，主要是维护pathToIndex，跟原有逻辑一致。

UDAFQueryExecutor

位置：org.apache.iotdb.db.query.executor.UDAFQueryExecutor

关键方法

```
// 通过innerAggregationPlan返回的innerDataSet构建真正的dataSet
public SingleDataSet convertInnerAggregationDataset(SingleDataSet
singleDataSet);

// 递归计算表达式结果
private Pair<TSDataType, Object> calcUDAFExpression(Expression expression);
```