1. 前言:

SparkSQL的前身是Shark，给熟悉RDBMS但又不理解MapReduce的技术人员提供快速上手的工具，但是Shark对Hive有太多依赖(如采用Hive的语法解析器、查询优化器等等),所以SparkSQL抛弃原有Shark的代码，汲取了Shark的一些优点，如内存列存储（In-Memory Columnar Storage）、Hive兼容性等，重新开发了SparkSQL代码.
不再受限于Hive，只是兼容Hive
而Hive on Spark是一个Hive的发展计划，该计划将Spark作为Hive的底层引擎之一，也就是说，Hive将不再受限于一个引擎，可以采用Map-Reduce、Tez、Spark等引擎

来自〈http://www.cnblogs.com/shishanyuan/p/4723604.html〉

> 可以这么说, sparkSQL做到了大部分HIVE的功能,但是比它快(因为重写了底层,而且用的是RDD),将操作写到代码里了
> 对SparkSQL来说,主要的工作就是写SQL,和写自定义函数了(类似于HIVE),
> 大致流程: 从数据源(HIVE,HDFS,文本等)读取数据,然后写SQL查询,(有业务处理就处理),复杂的就写自定义函数(继承UserDefinedFunction或者UserDefinedAggregateFunction)
> 查sql前当然要有数据和表,在代码里只能写临时表,要想创建永久表,还是得用hive那一套(相关语句可以直接运行也可以写代码里)

目前比较流行的是SparkSQL 连接Hive,这样可以操作Hive中的数据,(在spark1.2.1后,自带Hive,但是sparksql用到了hive的元数据,所以需要额外提供mysql(用来存储元数据,当然也有自带的叫derby),但局限性比较大,估计生产中还是会用外部Hive),这样就又得学Hive,暂时放弃

1.1 Spark SQL基础

使用spark SQL有两种方式，一种是作为分布式SQL引擎,此时只需要写SQL就可以进行计算。另一种是吃饭spark程序中通过领域API的形式来操作数据(被抽象为DateFrame)。

> 简单的说：一种是和SQL引擎去查数据(已经有数据了,一般和HIVE一起用)；一种是通过API的方式加载数据并操作(一般是从本地,HDFS,JDBC等加载文本数据)

2.为什么sparkSQL的性能得到提升

这个简单了解即可，

1. 内存列存储（In-Memory Columnar Storage）
2. 字节码生成技术
3. scala代码优化

3.sparkSQL组成

sparkSQL1.1总体上由四个模块组成：core、catalyst、hive、hive-Thriftserver：

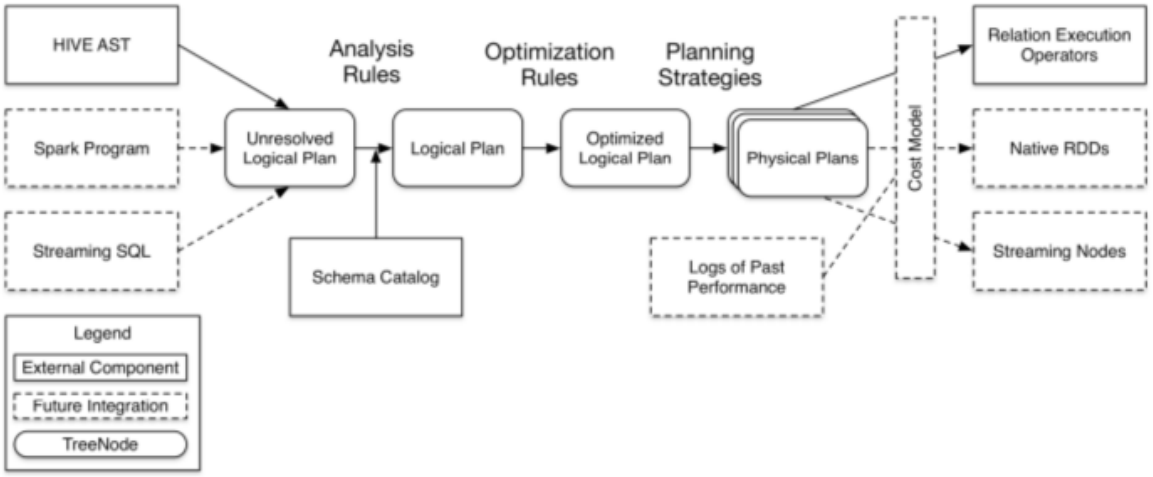
1. core 处理数据的输入输出，从不同的数据源获取数据（RDD、Parquet、json等），将查询结果输出成schemaRDD；

2. catalyst 处理查询语句的整个处理过程，包括解析、绑定、优化、物理计划等，说其是优化器，还不如说是查询引擎；

3. hive 对hive数据的处理

4. hive-ThriftServer 提供CLI和JDBC/ODBC接口

在这四个模块中，catalyst处于最核心的部分，其性能优劣将影响整体的性能。由于发展时间尚短，还有很多不足的地方，但其插件式的设计，为未来的发展留下了很大的空间。下面是catalyst的一个设计图：



其中虚线部分是以后版本要实现的功能，实线部分是已经实现的功能。从上图看，catalyst主要的实现组件有：

1.sqlParse，完成sql语句的语法解析功能，目前只提供了一个简单的sql解析器；

2. Analyzer，主要完成绑定工作，将不同来源的Unresolved LogicalPlan和数据元数据（如hive metastore、Schema catalog）进行绑定，生成resolved LogicalPlan；

3. optimizer对resolved LogicalPlan进行优化，生成optimized LogicalPlan；

4. Planner将LogicalPlan转换成PhysicalPlan；

5. CostModel，主要根据过去的性能统计数据，选择最佳的物理执行计划

这些组件的基本实现方法：

6. 先将sql语句通过解析生成Tree，然后在不同阶段使用不同的Rule应用到Tree上，通过转换完成各个组件的功能。

7. Analyzer使用Analysis Rules，配合数据元数据（如hive metastore、Schema catalog），完善Unresolved LogicalPlan的属性而转换成resolved LogicalPlan；

8. optimizer使用Optimization Rules，对resolved LogicalPlan进行合并、列裁剪、过滤器下推等优化作业而转换成optimized LogicalPlan；

9. Planner使用Planning Strategies，对optimized LogicalPlan进行转换，转换成可以执行的物理计划。

4.sparkSQL组件之解析

> 太复杂,以后看
> https://www.aboutyun.com/forum.php?mod=viewthread&tid=20910

5.sparkSQL 窗口函数

窗口函数是spark sql模块从1.4之后开始支持的，主要用于解决对一组数据进行操作，同时为每条数据返回单个结果，比如计算指定访问数据的均值、计算累进和或访问当前行之前行数据等，这些场景使用普通函数实现是比较困难的。数量不多,但功能强大.(其实窗口函数HIve就有了,语法基本一样)(数据源针对的是HIVE,需要打包到spark中测试)

Spark SQL支持三类窗口函数：排名函数、分析函数和聚合函数。以下汇总了Spark SQL支持的排名函数和分析函数。对于聚合函数来说，普通的聚合函数(类似sum,max)都可以作为窗口聚合函数使用

类别	SQL	DataFrame	
排名函数	rank	rank	为相同组的数据计算
排名函数	dense_rank	denseRank	为相同组内数据计算
排名函数	percent_rank	percentRank	该值的计算
排名函数	ntile	ntile	将组内数据排序然后
排名函数	row_number	rowNumber	
分析函数	cume_dist	cumeDist	
分析函数	lag	lag	用法：lag(input, [of
分析函数	lead	lead	用法：lead(input off

当一个函数被作为窗口函数使用时，需要为该窗口函数定义相关的窗口规范。窗口规范定义了哪些行会包括到给定输入行相关联的帧(frame)中。窗口规范包括三部分：

- 分区规范：定义哪些行属于相同分区，这样在对帧中数据排序和计算之前相同分区的数据就可以被收集到同一台机器上。如果没有指定分区规范，那么所有数据都会被收集到单个机器上处理。
- 排序规范：定义同一个分区中所有数据的排序方式，从而确定了给定行在他所属分区中的位置
- 帧规范：指定哪些行会被当前输入行的帧包括，通过其他行对于当前行的相对位置实现。

如果使用sql语句的话，PARTITION BY关键字用来为分区规范定义分区表达式、 ORDER BY关键字用来为排序规范定义排序表达式。格式：OVER (PARTITION BY ... ORDER BY ...)。

如果使用DataFrame API的话，API提供了函数来定义窗口规范。实例如下：

```
import org.apache.spark.sql.expressions.Window
val windowSpec = Window.partitionBy(...).orderBy(...)
```

为了分区和排序操作，需要定义帧的开始边界、结束边界和帧的类型，这也是一个帧规范的三部分。一共有五种边界：UNBOUNDED PRECEDING(分区第一行)，UNBOUNDED FOLLOWING(分区最后一行)，CURRENT ROW，<value> PRECEDING(当前行之前行)和<value> FOLLOWING(当前行之后行)。有两种帧类型：ROW帧和RANGE帧。

例子:

如果我们需要统计每个类别最畅销和次畅销的产品，首先需要基于产品的收入对相同类别的产品进行排名，然后基于排名取出最畅销和次畅销的产品。使用窗口函数实现的sql语句如下：

```
SELECT product, category, revenue
FROM (
   SELECT produce, category, revenue, dense_rank() OVER (PARTITION BY category
ORDER BY revenue DESC) as rank
   FROM productRevenue) tmp
WHERE rank <= 2
```

如果需要统计相同类别中每种产品与该类别中最畅销产品收入差距又该如何呢？首先为了计算差距，需要先找到每个类别中收入最高的产品

```
SELECT product, category, revenue, (max(revenue) OVER(PARTITION BY category
ORDER BY revenue DESC) - revenue) as revenue_diff
```

FROM produceRevenue

> https://blog.csdn.net/Shie_3/article/details/82890897
> https://blog.csdn.net/coding_hello/article/details/90664447
> https://www.cnblogs.com/abc8023/p/10910741.html