hive、hive on spark 、sparkSQL区别和联系

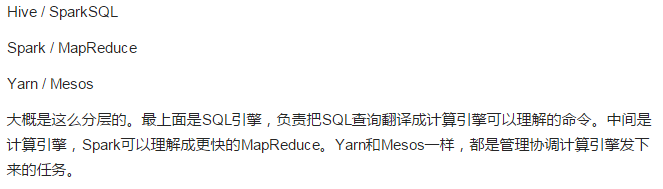
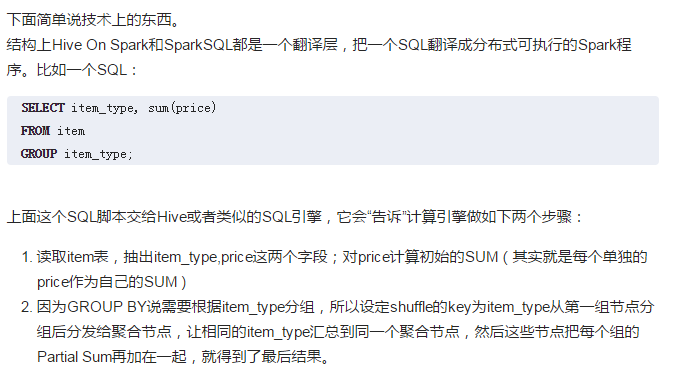
hive是基于Hadoop的一个[数据仓库](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=276841&ss_c=ssc.citiao.link)工具，可以将结构化的[数据文件](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=400311&ss_c=ssc.citiao.link)映射为一张数据库表，并提供简单的sql查询功能，可以将[sql语句](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=12370852&ss_c=ssc.citiao.link)转换为MapReduce任务进行运行。其优点是学习成本低，可以通过类SQL语句快速实现简单的MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。

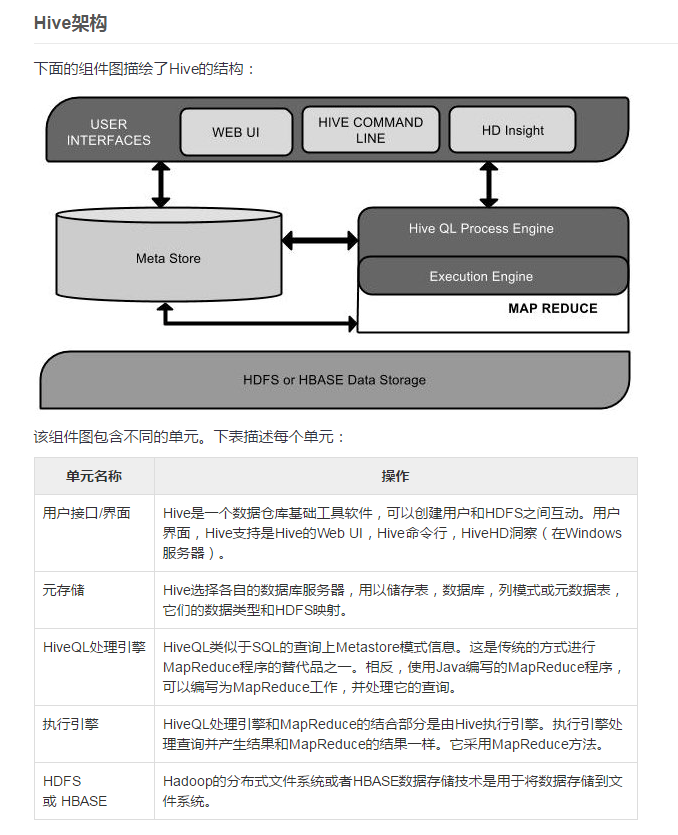
hive 是目前大数据领域。事实上的SQL标准。其底层默认是是基于MapReduce实现的，但是由于MapReduce速度实在太慢，因此这两年，陆续推出了新的SQL查询引擎。包括SparkSQL，Hive On Spark。

Spark SQL与Hive On Spark 是不一样的。Spark SQL是Spark自己研究出来的针对各种数据源，包括hive、JSon、JDBC、RDD等都可以执行查询的，一套基于Spark计算引擎的查询引擎。它是Spark的一个项目，只不过提供了针对hive执行查询的功能而已。

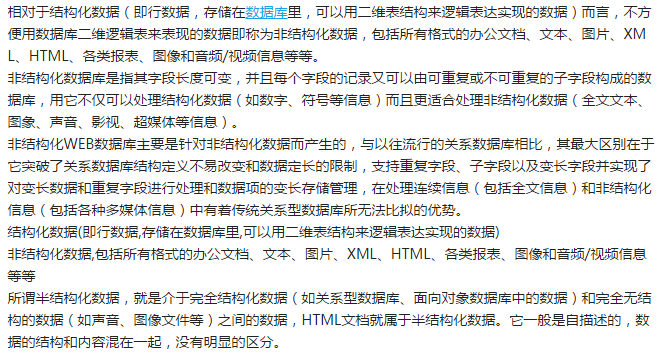
Hiev on Spark 是hive的一个项目，它是指，不通过MapReduce作为唯一的查询引擎，而是将Spark作为底层的查询引擎。Hive on Spark只适用于Hive







结构化、半结构化、非结构化数据:



**1、SparkSQL的发展历程**

**1.1 Hive and Shark**

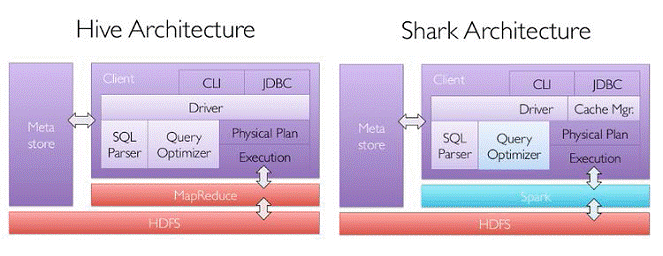
SparkSQL的前身是Shark，给熟悉RDBMS但又不理解MapReduce的技术人员提供快速上手的工具，Hive应运而生，它是当时唯一运行在Hadoop上的SQL-on-Hadoop工具。但是MapReduce计算过程中大量的中间磁盘落地过程消耗了大量的I/O，降低的运行效率，为了提高SQL-on-Hadoop的效率，大量的SQL-on-Hadoop工具开始产生，其中表现较为突出的是：

 MapR的Drill

 Cloudera的Impala

 Shark

其中Shark是伯克利实验室Spark生态环境的组件之一，它修改了下图所示的右下角的内存管理、物理计划、执行三个模块，并使之能运行在Spark引擎上，从而使得SQL查询的速度得到10-100倍的提升。

[](http://images0.cnblogs.com/blog/107289/201508/121050130982649.gif)

**1.2 Shark和SparkSQL**

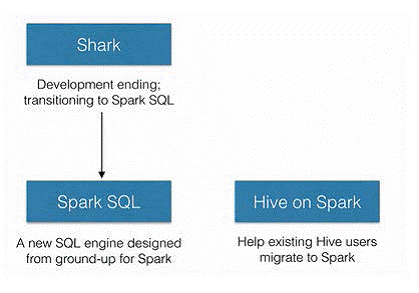
但是，随着Spark的发展，对于野心勃勃的Spark团队来说，Shark对于Hive的太多依赖（如采用Hive的语法解析器、查询优化器等等），制约了Spark的One Stack Rule Them All的既定方针，制约了Spark各个组件的相互集成，所以提出了SparkSQL项目。SparkSQL抛弃原有Shark的代码，汲取了Shark的一些优点，如内存列存储（In-Memory Columnar Storage）、Hive兼容性等，重新开发了SparkSQL代码；由于摆脱了对Hive的依赖性，SparkSQL无论在数据兼容、性能优化、组件扩展方面都得到了极大的方便，真可谓“退一步，海阔天空”。

**数据兼容方面**  不但兼容Hive，还可以从RDD、parquet文件、JSON文件中获取数据，未来版本甚至支持获取RDBMS数据以及cassandra等NOSQL数据；

**性能优化方面**  除了采取In-Memory Columnar Storage、byte-code generation等优化技术外、将会引进Cost Model对查询进行动态评估、获取最佳物理计划等等；

**组件扩展方面**  无论是SQL的语法解析器、分析器还是优化器都可以重新定义，进行扩展。

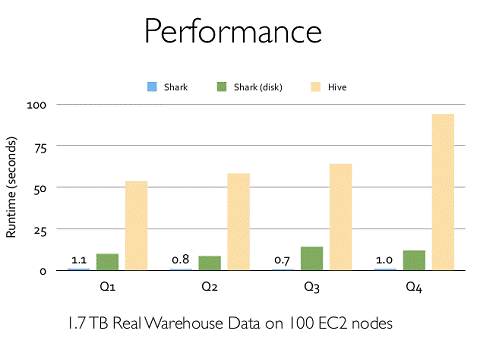
2014年6月1日Shark项目和SparkSQL项目的主持人Reynold Xin宣布：停止对Shark的开发，团队将所有资源放SparkSQL项目上，至此，Shark的发展画上了句话，但也因此发展出两个直线：SparkSQL和Hive on Spark。

[](http://images0.cnblogs.com/blog/107289/201508/121050170513975.gif)

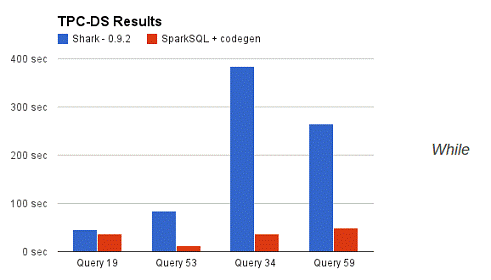
其中SparkSQL作为Spark生态的一员继续发展，而不再受限于Hive，只是兼容Hive；而Hive on Spark是一个Hive的发展计划，该计划将Spark作为Hive的底层引擎之一，也就是说，Hive将不再受限于一个引擎，可以采用Map-Reduce、Tez、Spark等引擎。

**1.3 SparkSQL的性能**

Shark的出现，使得SQL-on-Hadoop的性能比Hive有了10-100倍的提高：

[](http://images0.cnblogs.com/blog/107289/201508/121050191299372.gif)

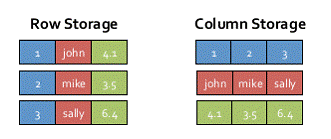
那么，摆脱了Hive的限制，SparkSQL的性能又有怎么样的表现呢？虽然没有Shark相对于Hive那样瞩目地性能提升，但也表现得非常优异：

[](http://images0.cnblogs.com/blog/107289/201508/121050198485987.gif)

为什么SparkSQL的性能会得到怎么大的提升呢？主要SparkSQL在下面几点做了优化：

**A：内存列存储（In-Memory Columnar Storage）**

SparkSQL的表数据在内存中存储不是采用原生态的JVM对象存储方式，而是采用内存列存储，如下图所示。

[](http://images0.cnblogs.com/blog/107289/201508/121050221456085.gif)

该存储方式无论在空间占用量和读取吞吐率上都占有很大优势。

对于原生态的JVM对象存储方式，每个对象通常要增加12-16字节的额外开销，对于一个270MB的TPC-H lineitem table数据，使用这种方式读入内存，要使用970MB左右的内存空间（通常是2～5倍于原生数据空间）；另外，使用这种方式，每个数据记录产生一个JVM对象，如果是大小为200B的数据记录，32G的堆栈将产生1.6亿个对象，这么多的对象，对于GC来说，可能要消耗几分钟的时间来处理（JVM的垃圾收集时间与堆栈中的对象数量呈线性相关）。显然这种内存存储方式对于基于内存计算的Spark来说，很昂贵也负担不起。

对于内存列存储来说，将所有原生数据类型的列采用原生数组来存储，将Hive支持的复杂数据类型（如array、map等）先序化后并接成一个字节数组来存储。这样，每个列创建一个JVM对象，从而导致可以快速的GC和紧凑的数据存储；额外的，还可以使用低廉CPU开销的高效压缩方法（如字典编码、行长度编码等压缩方法）降低内存开销；更有趣的是，对于分析查询中频繁使用的聚合特定列，性能会得到很大的提高，原因就是这些列的数据放在一起，更容易读入内存进行计算。

**B：字节码生成技术（bytecode generation，即CG）**

在数据库查询中有一个昂贵的操作是查询语句中的表达式，主要是由于JVM的内存模型引起的。比如如下一个查询：

*SELECT a + b FROM table*

在这个查询里，如果采用通用的SQL语法途径去处理，会先生成一个表达式树（有两个节点的Add树，参考后面章节），在物理处理这个表达式树的时候，将会如图所示的7个步骤：

1.  调用虚函数Add.eval()，需要确认Add两边的数据类型

2.  调用虚函数a.eval()，需要确认a的数据类型

3.  确定a的数据类型是Int，装箱

4.  调用虚函数b.eval()，需要确认b的数据类型

5.  确定b的数据类型是Int，装箱

6.  调用Int类型的Add

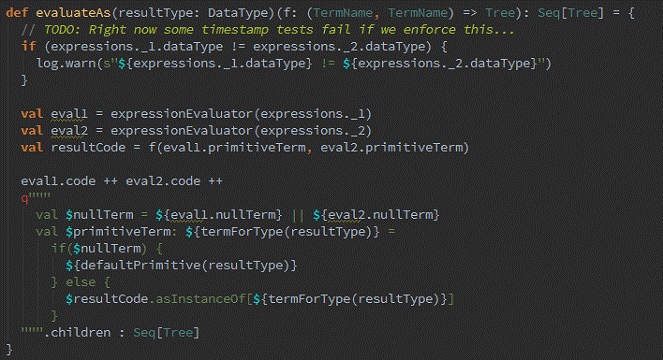
7.  返回装箱后的计算结果

其中多次涉及到虚函数的调用，虚函数的调用会打断CPU的正常流水线处理，减缓执行。

Spark1.1.0在catalyst模块的expressions增加了codegen模块，如果使用动态字节码生成技术（配置spark.sql.codegen参数），SparkSQL在执行物理计划的时候，对匹配的表达式采用特定的代码，动态编译，然后运行。如上例子，匹配到Add方法：

[clip_image012](http://images0.cnblogs.com/blog/107289/201508/121050241766498.gif)

然后，通过调用，最终调用：

[](http://images0.cnblogs.com/blog/107289/201508/121050258793711.gif)

最终实现效果类似如下伪代码：

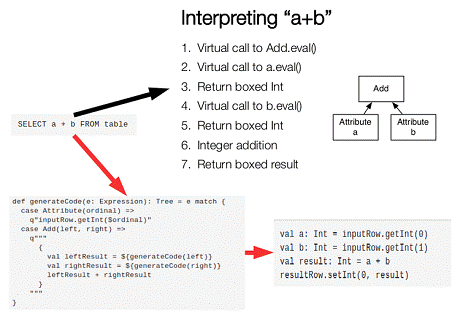
*val a: Int = inputRow.getInt(0)*

*val b: Int = inputRow.getInt(1)*

*val result: Int = a + b*

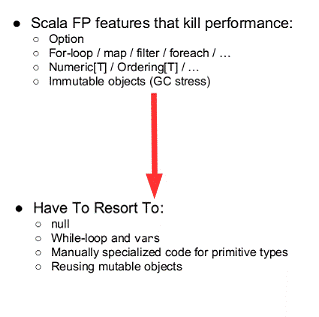
*resultRow.setInt(0, result)*

对于Spark1.1.0，对SQL表达式都作了CG优化，具体可以参看codegen模块。CG优化的实现主要还是依靠scala2.10的运行时放射机制（runtime reflection）。对于SQL查询的CG优化，可以简单地用下图来表示：

[](http://images0.cnblogs.com/blog/107289/201508/121050318175164.gif)

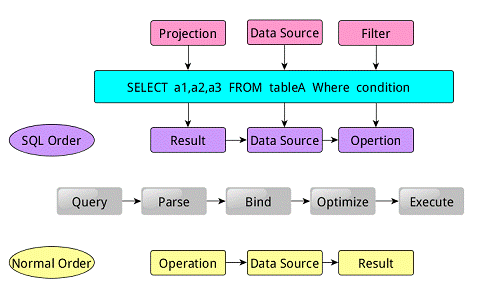
**C：Scala代码优化**

另外，SparkSQL在使用Scala编写代码的时候，尽量避免低效的、容易GC的代码；尽管增加了编写代码的难度，但对于用户来说，还是使用统一的接口，没受到使用上的困难。下图是一个Scala代码优化的示意图：

[](http://images0.cnblogs.com/blog/107289/201508/121050333958648.gif)

**2、 SparkSQL运行架构**

类似于关系型数据库，SparkSQL也是语句也是由Projection（a1，a2，a3）、Data Source（tableA）、Filter（condition）组成，分别对应sql查询过程中的Result、Data Source、Operation，也就是说SQL语句按Result-->Data Source-->Operation的次序来描述的。

[](http://images0.cnblogs.com/blog/107289/201508/121050357706489.gif)

当执行SparkSQL语句的顺序为：

1.对读入的SQL语句进行解析（Parse），分辨出SQL语句中哪些词是关键词（如SELECT、FROM、WHERE），哪些是表达式、哪些是Projection、哪些是Data Source等，从而判断SQL语句是否规范；

2.将SQL语句和数据库的数据字典（列、表、视图等等）进行绑定（Bind），如果相关的Projection、Data Source等都是存在的话，就表示这个SQL语句是可以执行的；

3.一般的数据库会提供几个执行计划，这些计划一般都有运行统计数据，数据库会在这些计划中选择一个最优计划（Optimize）；

4.计划执行（Execute），按Operation-->Data Source-->Result的次序来进行的，在执行过程有时候甚至不需要读取物理表就可以返回结果，比如重新运行刚运行过的SQL语句，可能直接从数据库的缓冲池中获取返回结果。

**2.1 Tree和Rule**

SparkSQL对SQL语句的处理和关系型数据库对SQL语句的处理采用了类似的方法，首先会将SQL语句进行解析（Parse），然后形成一个Tree，在后续的如绑定、优化等处理过程都是对Tree的操作，而操作的方法是采用Rule，通过模式匹配，对不同类型的节点采用不同的操作。在整个sql语句的处理过程中，Tree和Rule相互配合，完成了解析、绑定（在SparkSQL中称为Analysis）、优化、物理计划等过程，最终生成可以执行的物理计划。

**2.1.1 Tree**

  Tree的相关代码定义在sql/catalyst/src/main/scala/org/apache/spark/sql/catalyst/trees

  Logical Plans、Expressions、Physical Operators都可以使用Tree表示

  Tree的具体操作是通过TreeNode来实现的

  SparkSQL定义了catalyst.trees的日志，通过这个日志可以形象的表示出树的结构

  TreeNode可以使用scala的集合操作方法（如foreach, map, flatMap, collect等）进行操作

  有了TreeNode，通过Tree中各个TreeNode之间的关系，可以对Tree进行遍历操作，如使用transformDown、transformUp将Rule应用到给定的树段，然后用结果替代旧的树段；也可以使用transformChildrenDown、transformChildrenUp对一个给定的节点进行操作，通过迭代将Rule应用到该节点以及子节点。

  TreeNode可以细分成三种类型的Node：

  UnaryNode 一元节点，即只有一个子节点。如Limit、Filter操作

  BinaryNode 二元节点，即有左右子节点的二叉节点。如Jion、Union操作

  LeafNode 叶子节点，没有子节点的节点。主要用户命令类操作，如SetCommand

**2.1.2 Rule**

  Rule的相关代码定义在sql/catalyst/src/main/scala/org/apache/spark/sql/catalyst/rules

  Rule在SparkSQL的Analyzer、Optimizer、SparkPlan等各个组件中都有应用到

  Rule是一个抽象类，具体的Rule实现是通过RuleExecutor完成

  Rule通过定义batch和batchs，可以简便的、模块化地对Tree进行transform操作

  Rule通过定义Once和FixedPoint，可以对Tree进行一次操作或多次操作（如对某些Tree进行多次迭代操作的时候，达到FixedPoint次数迭代或达到前后两次的树结构没变化才停止操作，具体参看RuleExecutor.apply）

**2.2 sqlContext和hiveContext的运行过程**

SparkSQL有两个分支，sqlContext和hiveContext，sqlContext现在只支持SQL语法解析器（SQL-92语法）；hiveContext现在支持SQL语法解析器和hivesql语法解析器，默认为hiveSQL语法解析器，用户可以通过配置切换成SQL语法解析器，来运行hiveSQL不支持的语法，

**2.2.1 sqlContext的运行过程**

sqlContext总的一个过程如下图所示：

1.SQL语句经过SqlParse解析成UnresolvedLogicalPlan；

2.使用analyzer结合数据数据字典（catalog）进行绑定，生成resolvedLogicalPlan；

3.使用optimizer对resolvedLogicalPlan进行优化，生成optimizedLogicalPlan；

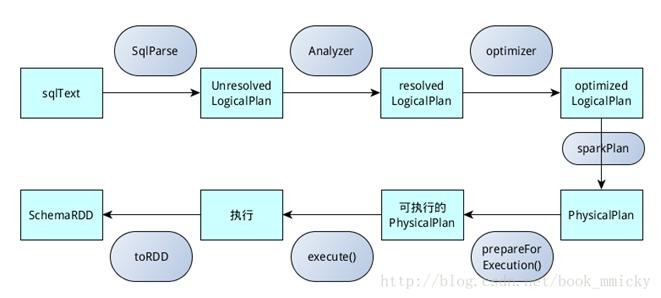
4.使用SparkPlan将LogicalPlan转换成PhysicalPlan；

5.使用prepareForExecution()将PhysicalPlan转换成可执行物理计划；

6.使用execute()执行可执行物理计划；

7.生成SchemaRDD。

在整个运行过程中涉及到多个SparkSQL的组件，如SqlParse、analyzer、optimizer、SparkPlan等等

[](http://images0.cnblogs.com/blog/107289/201508/121050368954046.jpg)

**2.2.2hiveContext的运行过程**

hiveContext总的一个过程如下图所示：

1.SQL语句经过HiveQl.parseSql解析成Unresolved LogicalPlan，在这个解析过程中对hiveql语句使用getAst()获取AST树，然后再进行解析；

2.使用analyzer结合数据hive源数据Metastore（新的catalog）进行绑定，生成resolved LogicalPlan；

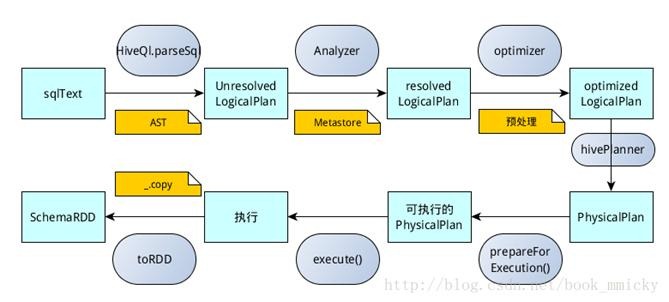
3.使用optimizer对resolved LogicalPlan进行优化，生成optimized LogicalPlan，优化前使用了ExtractPythonUdfs(catalog.PreInsertionCasts(catalog.CreateTables(analyzed)))进行预处理；

4.使用hivePlanner将LogicalPlan转换成PhysicalPlan；

5.使用prepareForExecution()将PhysicalPlan转换成可执行物理计划；

6.使用execute()执行可执行物理计划；

7.执行后，使用map(\_.copy)将结果导入SchemaRDD。

[](http://images0.cnblogs.com/blog/107289/201508/121050400209257.jpg)

**2.3 catalyst优化器**

SparkSQL1.1总体上由四个模块组成：core、catalyst、hive、hive-Thriftserver：

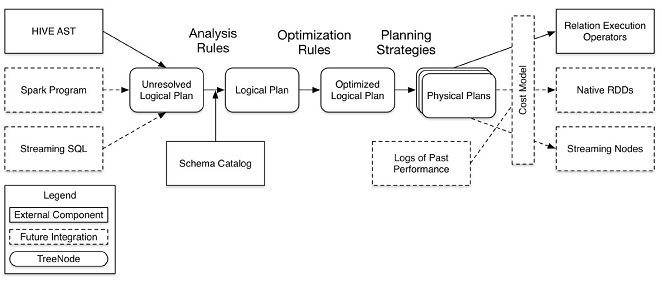
  core处理数据的输入输出，从不同的数据源获取数据（RDD、Parquet、json等），将查询结果输出成schemaRDD；

  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；

  hive对hive数据的处理

  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：

[](http://images0.cnblogs.com/blog/107289/201508/121050438329327.gif)

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：

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；

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；

optimizer对resolved LogicalPlan进行优化，生成optimized LogicalPlan；

 Planner将LogicalPlan转换成PhysicalPlan；

 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：

 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。

 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；

 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；

 Planner使用Planning Strategies，对optimized LogicalPlan

**3、SparkSQL CLI**

CLI（Command-Line Interface，命令行界面）是指可在用户提示符下键入可执行指令的界面，它通常不支持鼠标，用户通过键盘输入指令，计算机接收到指令后予以执行。Spark CLI指的是使用命令界面直接输入SQL命令，然后发送到Spark集群进行执行，在界面中显示运行过程和最终的结果。

Spark1.1相较于Spark1.0最大的差别就在于Spark1.1增加了Spark SQL CLI和ThriftServer，使得Hive用户还有用惯了命令行的RDBMS数据库管理员较容易地上手，真正意义上进入了SQL时代。

**【注】**Spark CLI和Spark Thrift Server实验环境为第二课《Spark编译与部署（下）--Spark编译安装》所搭建

**3.1**  **运行环境说明**

**3.1.1 硬软件环境**

  主机操作系统：Windows 64位，双核4线程，主频2.2G，10G内存

  虚拟软件：VMware® Workstation 9.0.0 build-812388

  虚拟机操作系统：CentOS 64位，单核

  虚拟机运行环境：

  JDK：1.7.0\_55 64位

  Hadoop：2.2.0（需要编译为64位）

  Scala：2.11.4

  Spark：1.1.0（需要编译）

  Hive：0.13.1

**3.1.2 机器网络环境**

集群包含三个节点，节点之间可以免密码SSH访问，节点IP地址和主机名分布如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **IP地址** | **机器名** | **类型** | **核数/内存** | **用户名** | **目录** |
| 1 | 192.168.0.61 | hadoop1 | NN/DN/RM  Master/Worker | 1核/3G | hadoop | /app 程序所在路径  /app/scala-...  /app/hadoop  /app/complied |
| 2 | 192.168.0.62 | hadoop2 | DN/NM/Worker | 1核/2G | hadoop |
| 3 | 192.168.0.63 | hadoop3 | DN/NM/Worker | 1核/2G | hadoop |

**3.2 配置并启动**

**3.2.1 创建并配置hive-site.xml**

在运行Spark SQL CLI中需要使用到Hive Metastore，故需要在Spark中添加其uris。具体方法是在SPARK\_HOME/conf目录下创建hive-site.xml文件，然后在该配置文件中，添加hive.metastore.uris属性，具体如下：

*<configuration>*

*<property>*

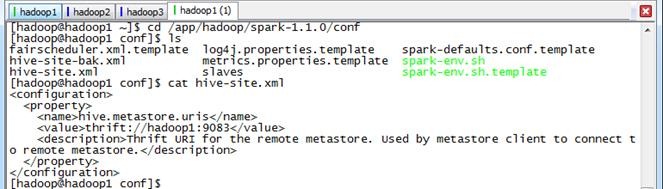
*<name>hive.metastore.uris</name>*

*<value>thrift://hadoop1:9083</value>*

*<description>Thrift URI for the remote metastore. Used by metastore client to connect to remote metastore.</description>*

*</property>*

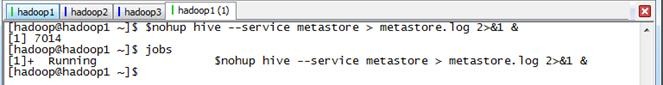
*</configuration>*

[](http://images0.cnblogs.com/blog/107289/201508/121050450047167.jpg)

**3.2.2 启动Hive**

在使用Spark SQL CLI之前需要启动Hive Metastore（如果数据存放在HDFS文件系统，还需要启动Hadoop的HDFS），使用如下命令可以使Hive Metastore启动后运行在后台，可以通过jobs查询：

*$nohup hive --service metastore > metastore.log 2>&1 &*

[](http://images0.cnblogs.com/blog/107289/201508/121050495046393.jpg)

**3.2.3 启动Spark集群和Spark SQL CLI**

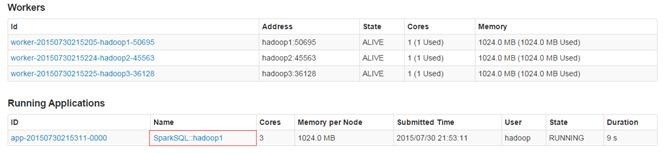
通过如下命令启动Spark集群和Spark SQL CLI：

*$cd /app/hadoop/spark-1.1.0*

*$sbin/start-all.sh*

*$bin/spark-sql --master spark://hadoop1:7077 --executor-memory 1g*

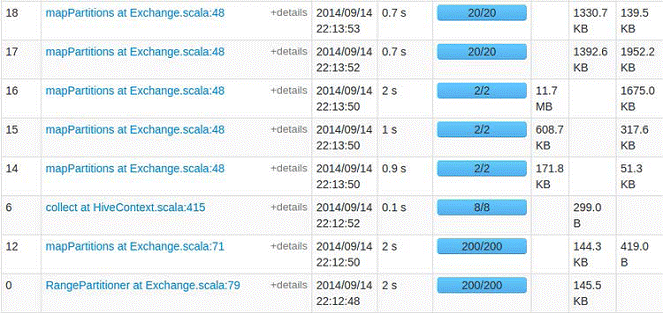
在集群监控页面可以看到启动了SparkSQL应用程序：

[](http://images0.cnblogs.com/blog/107289/201508/121050510359593.jpg)

这时就可以使用HQL语句对Hive数据进行查询，另外可以使用COMMAND，如使用set进行设置参数：默认情况下，SparkSQL Shuffle的时候是200个partition，可以使用如下命令修改该参数：

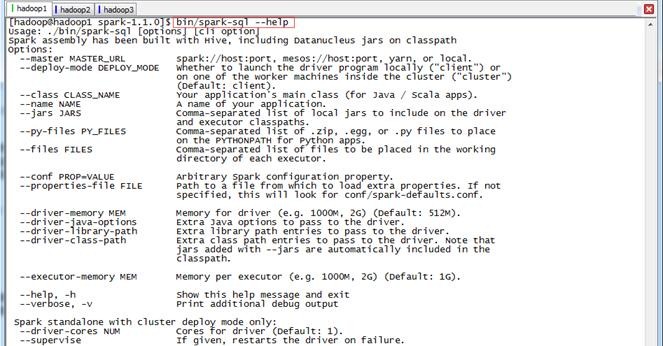
*SET spark.sql.shuffle.partitions=20;*

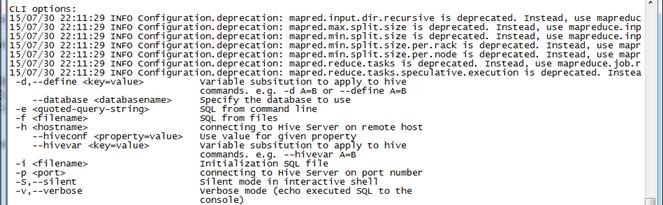
运行同一个查询语句，参数改变后，Task（partition）的数量就由200变成了20。

[](http://images0.cnblogs.com/blog/107289/201508/121050523018405.gif)

**3.2.4 命令参数**

通过bin/spark-sql --help可以查看CLI命令参数：

[](http://images0.cnblogs.com/blog/107289/201508/121050539733862.jpg)

[](http://images0.cnblogs.com/blog/107289/201508/121050578328916.jpg)

其中[options] 是CLI启动一个SparkSQL应用程序的参数，如果不设置--master的话，将在启动spark-sql的机器以local方式运行，只能通过http://机器名:4040进行监控；这部分参数，可以参照Spark1.0.0 应用程序部署工具spark-submit 的参数。

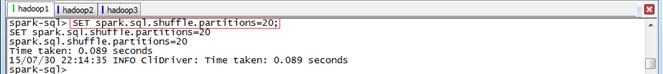
[cli option]是CLI的参数，通过这些参数CLI可以直接运行SQL文件、进入命令行运行SQL命令等等，类似以前的Shark的用法。需要注意的是CLI不是使用JDBC连接，所以不能连接到ThriftServer；但可以配置conf/hive-site.xml连接到Hive的Metastore，然后对Hive数据进行查询。

**3.3 实战Spark SQL CLI**

**3.3.1 获取订单每年的销售单数、销售总额**

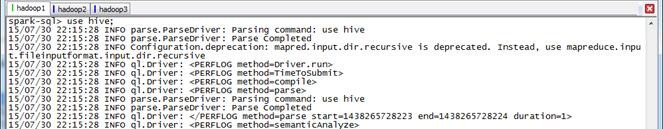
第一步   设置任务个数，在这里修改为20个

*spark-sql>SET spark.sql.shuffle.partitions=20;*

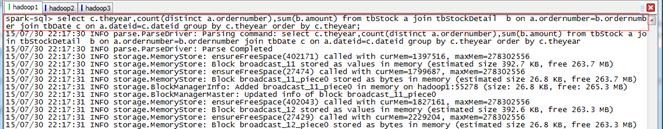
*[](http://images0.cnblogs.com/blog/107289/201508/121051022233941.jpg)*

第二步   运行SQL语句

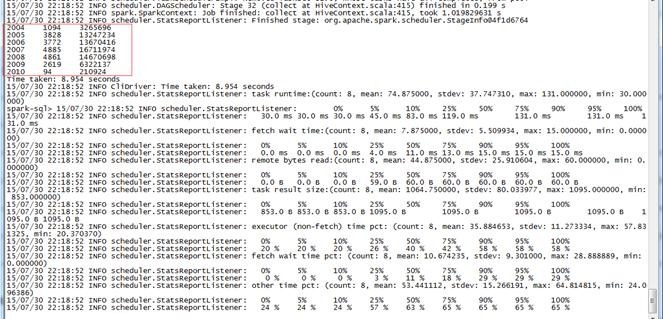
*spark-sql>use hive;*

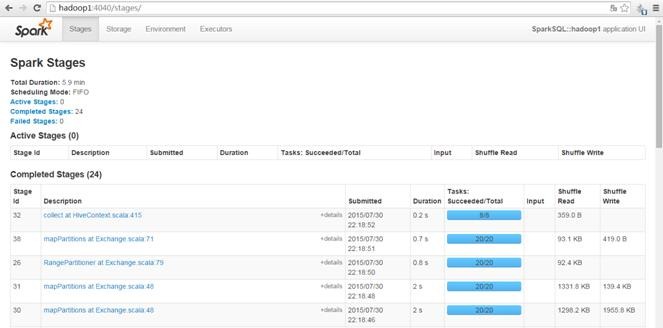
*[](http://images0.cnblogs.com/blog/107289/201508/121051048487239.jpg)*

*spark-sql>select c.theyear,count(distinct a.ordernumber),sum(b.amount) from tbStock a join tbStockDetail  b on a.ordernumber=b.ordernumber join tbDate c on a.dateid=c.dateid group by c.theyear order by c.theyear;*

[](http://images0.cnblogs.com/blog/107289/201508/121051137707947.jpg)

第三步   查看运行结果

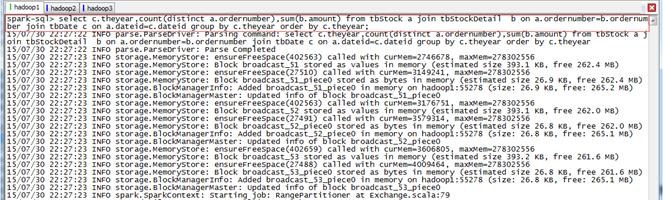
[](http://images0.cnblogs.com/blog/107289/201508/121051183796671.jpg)

[](http://images0.cnblogs.com/blog/107289/201508/121051224898183.jpg)

**3.3.2 计算所有订单每年的总金额**

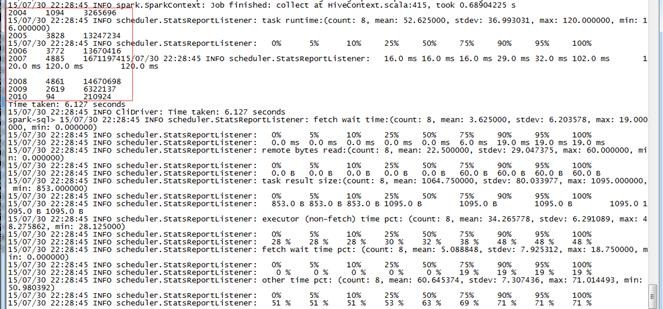
第一步   执行SQL语句

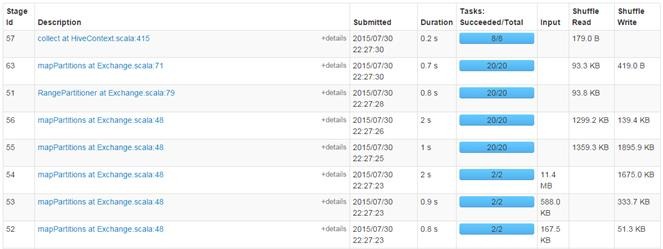
*spark-sql>select c.theyear,count(distinct a.ordernumber),sum(b.amount) from tbStock a join tbStockDetail  b on a.ordernumber=b.ordernumber join tbDate c on a.dateid=c.dateid group by c.theyear order by c.theyear;*

*[](http://images0.cnblogs.com/blog/107289/201508/121051286295407.jpg)*

第二步   执行结果

使用CLI执行结果如下：

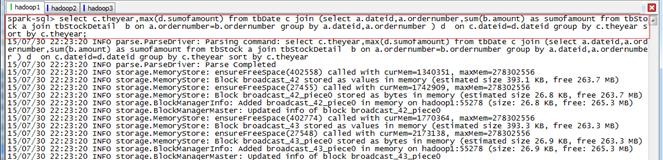
[](http://images0.cnblogs.com/blog/107289/201508/121051310825692.jpg)

[](http://images0.cnblogs.com/blog/107289/201508/121051334732061.jpg)

**3.3.3 计算所有订单每年最大金额订单的销售额**

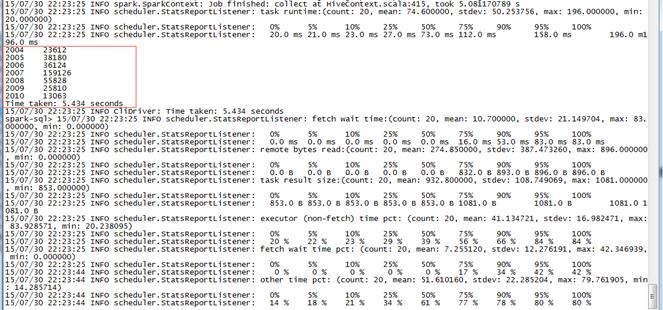
第一步   执行SQL语句

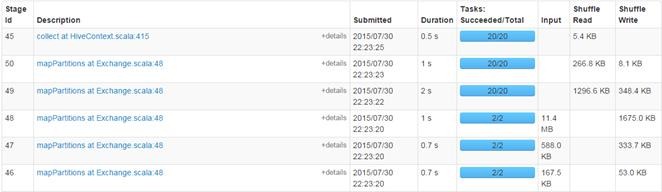
*spark-sql>select c.theyear,max(d.sumofamount) from tbDate c join (select a.dateid,a.ordernumber,sum(b.amount) as sumofamount from tbStock a join tbStockDetail  b on a.ordernumber=b.ordernumber group by a.dateid,a.ordernumber ) d  on c.dateid=d.dateid group by c.theyear sort by c.theyear;*

*[](http://images0.cnblogs.com/blog/107289/201508/121051389423357.jpg)*

第二步   执行结果

使用CLI执行结果如下：

[](http://images0.cnblogs.com/blog/107289/201508/121051405206841.jpg)

[](http://images0.cnblogs.com/blog/107289/201508/121051438795281.jpg)

**4、Spark Thrift Server**

ThriftServer是一个JDBC/ODBC接口，用户可以通过JDBC/ODBC连接ThriftServer来访问SparkSQL的数据。ThriftServer在启动的时候，会启动了一个SparkSQL的应用程序，而通过JDBC/ODBC连接进来的客户端共同分享这个SparkSQL应用程序的资源，也就是说不同的用户之间可以共享数据；ThriftServer启动时还开启一个侦听器，等待JDBC客户端的连接和提交查询。所以，在配置ThriftServer的时候，至少要配置ThriftServer的主机名和端口，如果要使用Hive数据的话，还要提供Hive Metastore的uris。

**【注】**Spark CLI和Spark Thrift Server实验环境为第二课《Spark编译与部署（下）--Spark编译安装》所搭建

**4.1 配置并启动**

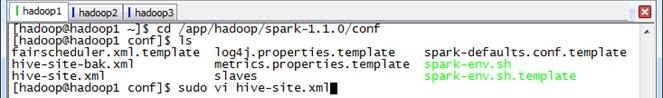
**4.1.1 创建并配置hive-site.xml**

第一步   创建hive-site.xml配置文件

在$SPARK\_HOME/conf目录下修改hive-site.xml配置文件（如果在Spark SQL CLI中已经添加，可以省略）：

*$cd /app/hadoop/spark-1.1.0/conf*

*$sudo vi hive-site.xml*

*[](http://images0.cnblogs.com/blog/107289/201508/121051479421808.jpg)*

第二步   修改配置文件

设置hadoop1为Metastore服务器，hadoop2为Thrift Server服务器，配置内容如下：

*<configuration>*

*<property>*

*<name>hive.metastore.uris</name>*

*<value>thrift://hadoop1:9083</value>*

*<description>Thrift URI for the remote metastore. Used by metastore client to connect to remote metastore.</description>*

*</property>*

*<property>*

*<name>hive.server2.thrift.min.worker.threads</name>*

*<value>5</value>*

*<description>Minimum number of Thrift worker threads</description>*

*</property>*

*<property>*

*<name>hive.server2.thrift.max.worker.threads</name>*

*<value>500</value>*

*<description>Maximum number of Thrift worker threads</description>*

*</property>*

*<property>*

*<name>hive.server2.thrift.port</name>*

*<value>10000</value>*

*<description>Port number of HiveServer2 Thrift interface. Can be overridden by setting $HIVE\_SERVER2\_THRIFT\_PORT</description>*

*</property>*

*<property>*

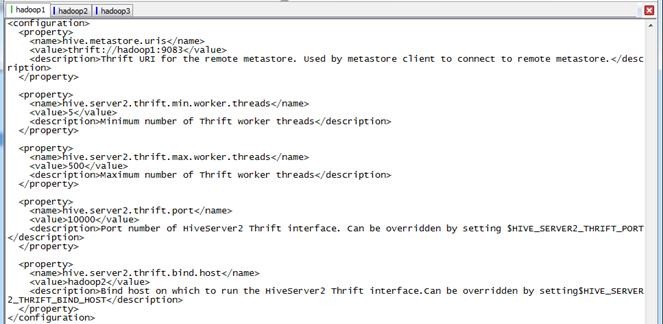
*<name>hive.server2.thrift.bind.host</name>*

*<value>hadoop2</value>*

*<description>Bind host on which to run the HiveServer2 Thrift interface.Can be overridden by setting$HIVE\_SERVER2\_THRIFT\_BIND\_HOST</description>*

*</property>*

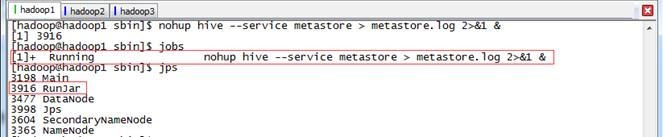
*</configuration>*

[](http://images0.cnblogs.com/blog/107289/201508/121051492233849.jpg)

**4.1.2 启动Hive**

在hadoop1节点中，在后台启动Hive Metastore（如果数据存放在HDFS文件系统，还需要启动Hadoop的HDFS）：

*$nohup hive --service metastore > metastore.log 2>&1 &*

[](http://images0.cnblogs.com/blog/107289/201508/121051587709199.jpg)

**4.1.3 启动Spark集群和Thrift Server**

在hadoop1节点启动Spark集群

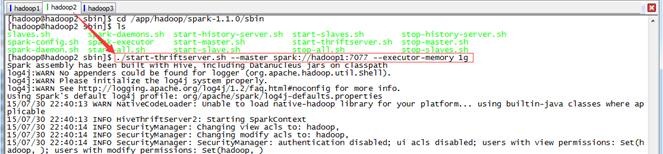
*$cd /app/hadoop/spark-1.1.0/sbin*

*$./start-all.sh*

在hadoop2节点上进入SPARK\_HOME/sbin目录，使用如下命令启动Thrift Server

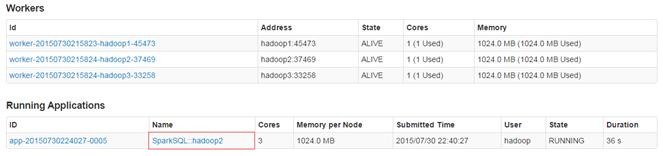
*$cd /app/hadoop/spark-1.1.0/sbin*

*$./start-thriftserver.sh --master spark://hadoop1:7077 --executor-memory 1g*

[](http://images0.cnblogs.com/blog/107289/201508/121052016924940.jpg)

**注意**：Thrift Server需要按照配置在hadoop2启动！

在集群监控页面可以看到启动了SparkSQL应用程序：

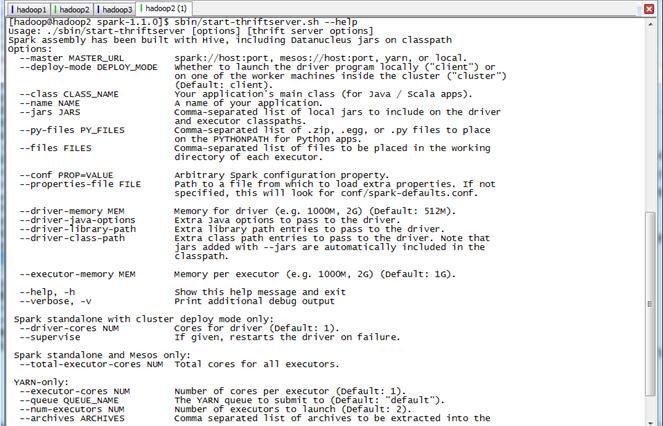
[](http://images0.cnblogs.com/blog/107289/201508/121052034263910.jpg)

**4.1.4 命令参数**

使用sbin/start-thriftserver.sh --help可以查看ThriftServer的命令参数：

*$sbin/start-thriftserver.sh --help Usage: ./sbin/start-thriftserver [options] [thrift server options]*

*Thrift server options: Use value for given property*

[](http://images0.cnblogs.com/blog/107289/201508/121052126456060.jpg)

[clip_image074](http://images0.cnblogs.com/blog/107289/201508/121052157236286.jpg)

其中[options] 是Thrift Server启动一个SparkSQL应用程序的参数，如果不设置--master的话，将在启动Thrift Server的机器以local方式运行，只能通过http://机器名:4040进行监控；这部分参数，可以参照Spark1.0.0 应用程序部署工具spark-submit 的参数。在集群中提供Thrift Server的话，一定要配置master、executor-memory等参数。

[thrift server options]是Thrift Server的参数，可以使用-dproperty=value的格式来定义；在实际应用上，因为参数比较多，通常使用conf/hive-site.xml配置。

**4.2 实战Thrift Server**

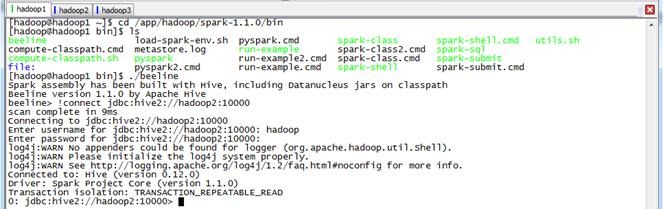
**4.2.1 远程客户端连接**

可以在任意节点启动bin/beeline，用!connect jdbc:hive2://hadoop2:10000连接ThriftServer，因为没有采用权限管理，所以用户名用运行bin/beeline的用户hadoop，密码为空：

*$cd /app/hadoop/spark-1.1.0/bin*

*$./beeline*

*beeline>!connect jdbc:hive2://hadoop2:10000*

*[](http://images0.cnblogs.com/blog/107289/201508/121052175823686.jpg)*

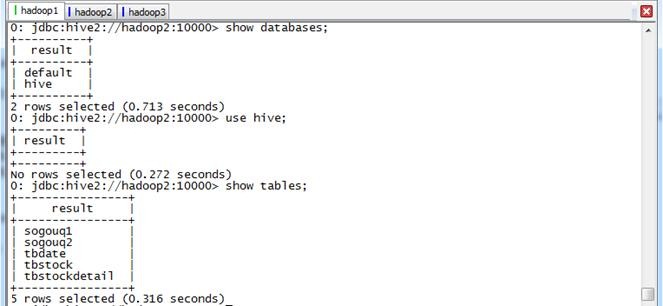
**4.2.2 基本操作**

第一步   显示hive数据库所有表

*beeline>show database;*

*beeline>use hive;*

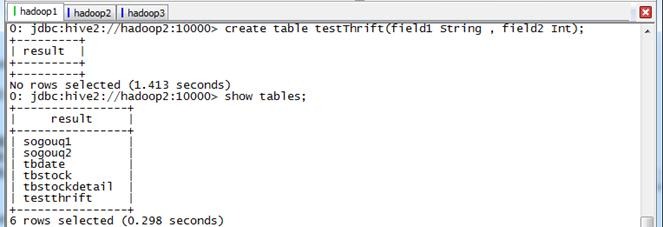
*beeline>show tables;*

*[](http://images0.cnblogs.com/blog/107289/201508/121052188957484.jpg)*

第二步   创建表testThrift

*beeline>create table testThrift(field1 String , field2 Int);*

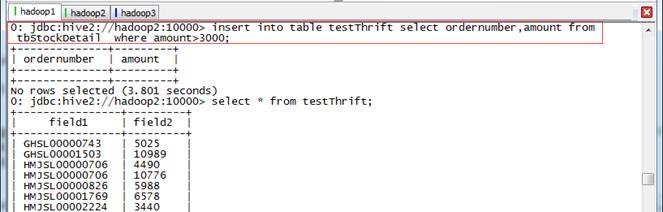
*beeline>show tables;*

[](http://images0.cnblogs.com/blog/107289/201508/121052212071812.jpg)

第三步   把tbStockDetail表中金额大于3000插入到testThrift表中

*beeline>insert into table testThrift select ordernumber,amount from tbStockDetail  where amount>3000;*

*beeline>select \* from testThrift;*

[](http://images0.cnblogs.com/blog/107289/201508/121052222397396.jpg)

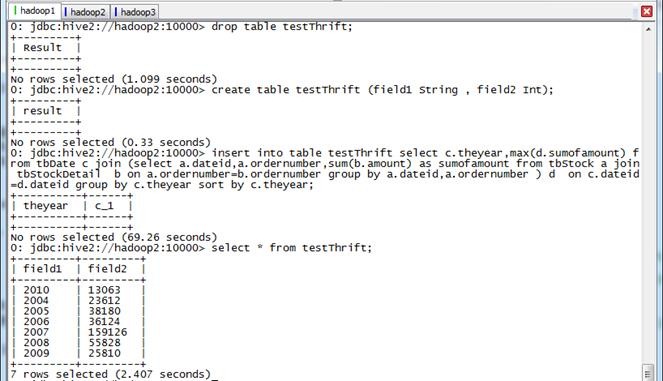
第四步   重新创建testThrift表中，把年度最大订单插入该表中

*beeline>drop table testThrift;*

*beeline>create table testThrift (field1 String , field2 Int);*

*beeline>insert into table testThrift select c.theyear,max(d.sumofamount) from tbDate c join (select a.dateid,a.ordernumber,sum(b.amount) as sumofamount from tbStock a join tbStockDetail  b on a.ordernumber=b.ordernumber group by a.dateid,a.ordernumber ) d  on c.dateid=d.dateid group by c.theyear sort by c.theyear;*

*beeline>select \* from testThrift;*

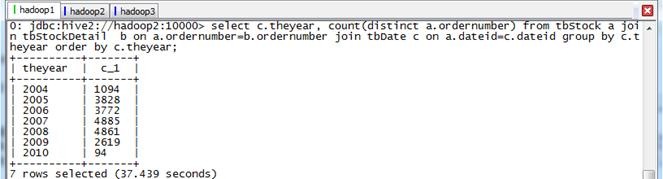
*[](http://images0.cnblogs.com/blog/107289/201508/121052239261380.jpg)*

**4.2.3 计算所有订单每年的订单数**

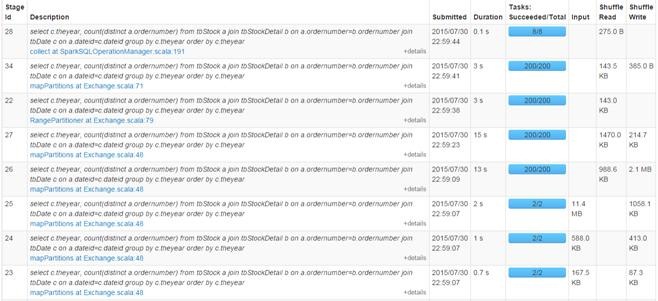
第一步   执行SQL语句

*spark-sql>select c.theyear, count(distinct a.ordernumber) from tbStock a join tbStockDetail  b on a.ordernumber=b.ordernumber join tbDate c on a.dateid=c.dateid group by c.theyear order by c.theyear;*

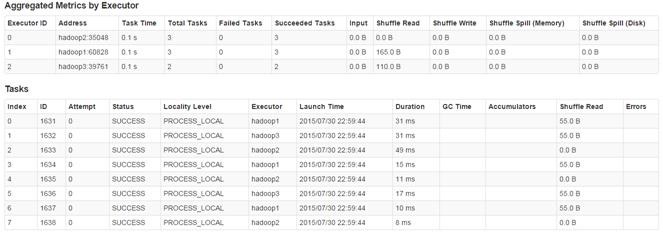
第二步   执行结果

[](http://images0.cnblogs.com/blog/107289/201508/121052298488604.jpg)

Stage监控页面：

[](http://images0.cnblogs.com/blog/107289/201508/121052344428801.jpg)

查看Details for Stage 28

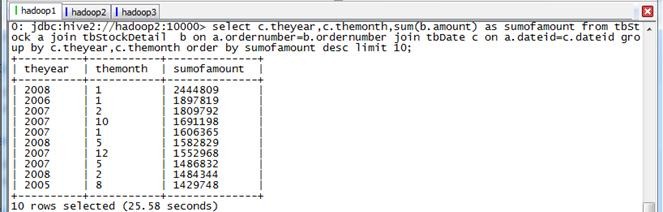
[](http://images0.cnblogs.com/blog/107289/201508/121052359103787.jpg)

**4.2.4 计算所有订单月销售额前十名**

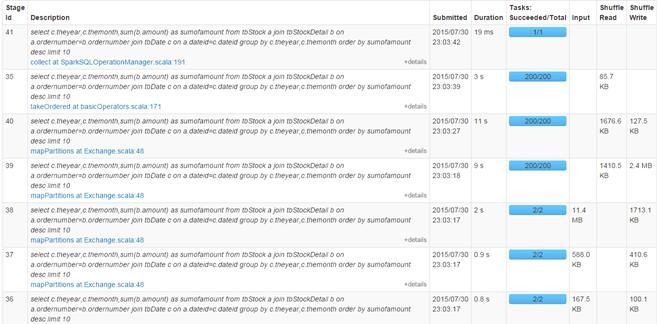
第一步   执行SQL语句

*spark-sql>select c.theyear,c.themonth,sum(b.amount) as sumofamount from tbStock a join tbStockDetail  b on a.ordernumber=b.ordernumber join tbDate c on a.dateid=c.dateid group by c.theyear,c.themonth order by sumofamount desc limit 10;*

第二步   执行结果

[](http://images0.cnblogs.com/blog/107289/201508/121052368642629.jpg)

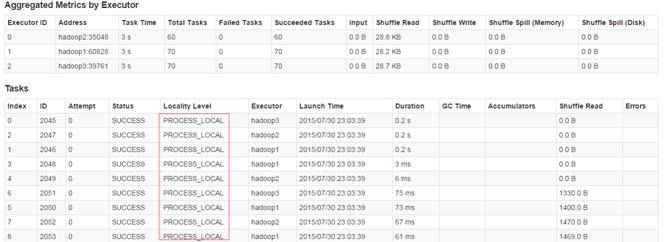
Stage监控页面：

[](http://images0.cnblogs.com/blog/107289/201508/121052394732698.jpg)

在其第一个Task中，从本地读入数据

[](http://images0.cnblogs.com/blog/107289/201508/121052417078282.jpg)

在后面的Task是从内存中获取数据

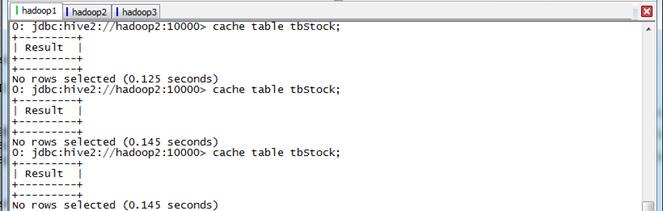
[](http://images0.cnblogs.com/blog/107289/201508/121102481458644.jpg)

**4.2.5 缓存表数据**

第一步   缓存数据

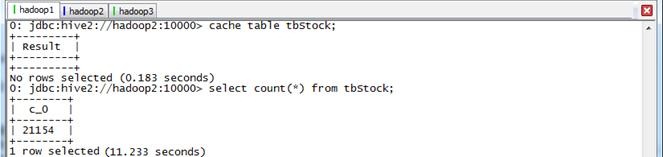
*beeline>cache table tbStock;*

*beeline>select count(\*) from tbStock;*

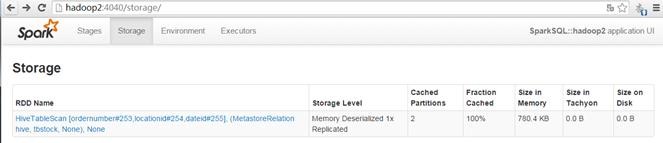
*[](http://images0.cnblogs.com/blog/107289/201508/121102492231216.jpg)*

第二步   运行4.2.4中的“计算所有订单月销售额前十名”

*beeline>select count(\*) from tbStock;*

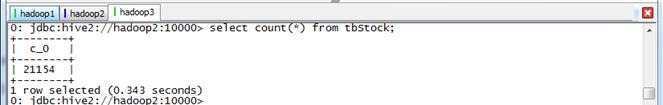
[](http://images0.cnblogs.com/blog/107289/201508/121102501769057.jpg)

本次计算划给11.233秒，查看webUI，数据已经缓存，缓存率为100%：

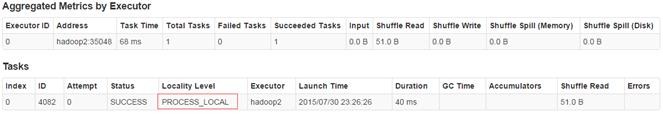
[](http://images0.cnblogs.com/blog/107289/201508/121102512071344.jpg)

第三步   在另外节点再次运行

在hadoop3节点启动bin/beeline，用!connect jdbc:hive2://hadoop2:10000连接ThriftServer，然后直接运行对tbStock计数（注意没有进行数据库的切换）：

[](http://images0.cnblogs.com/blog/107289/201508/121102520678213.jpg)

用时0.343秒，再查看webUI中的stage：

[](http://images0.cnblogs.com/blog/107289/201508/121102531148027.jpg)

Locality Level是PROCESS，显然是使用了缓存表。

从上可以看出，ThriftServer可以连接多个JDBC/ODBC客户端，并相互之间可以共享数据。顺便提一句，ThriftServer启动后处于监听状态，用户可以使用ctrl+c退出ThriftServer；而beeline的退出使用!q命令。

**4.2.6 在IDEA中JDBC访问**

有了ThriftServer，开发人员可以非常方便的使用JDBC/ODBC来访问SparkSQL。下面是一个scala代码，查询表tbStockDetail，返回amount>3000的单据号和交易金额：

第一步   在IDEA创建class6包和类JDBCofSparkSQL

参见《Spark编程模型（下）--IDEA搭建及实战》在IDEA中创建class6包并新建类JDBCofSparkSQL。该类中查询tbStockDetail金额大于3000的订单：

*package class6*

*import java.sql.DriverManager*

*object JDBCofSparkSQL {*

*def main(args: Array[String]) {*

*Class.forName("org.apache.hive.jdbc.HiveDriver")*

*val conn = DriverManager.getConnection("jdbc:hive2://hadoop2:10000/hive", "hadoop", "")*

*try {*

*val statement = conn.createStatement*

*val rs = statement.executeQuery("select ordernumber,amount from tbStockDetail  where amount>3000")*

*while (rs.next) {*

*val ordernumber = rs.getString("ordernumber")*

*val amount = rs.getString("amount")*

*println("ordernumber = %s, amount = %s".format(ordernumber, amount))*

*}*

*} catch {*

*case e: Exception => e.printStackTrace*

*}*

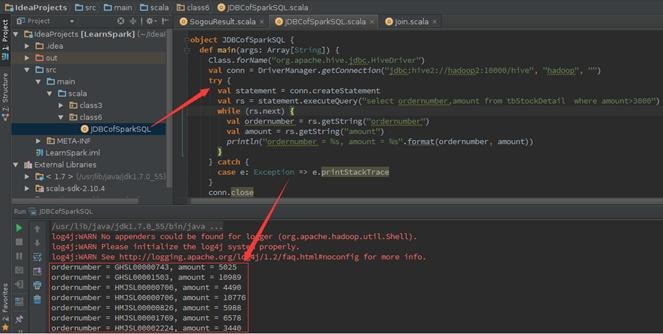
*conn.close*

*}*

*}*

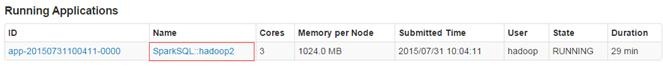
第二步   查看运行结果

在IDEA中可以观察到，在运行日志窗口中没有运行过程的日志，只显示查询结果

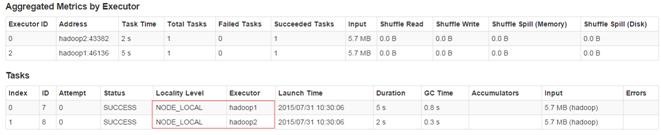
[](http://images0.cnblogs.com/blog/107289/201508/121102542395584.jpg)

第三步   查看监控结果

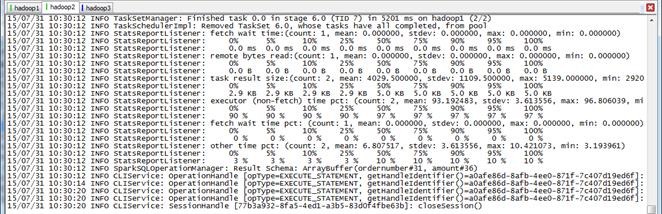
从Spark监控界面中观察到，该Job有一个编号为6的Stage，该Stage有2个Task，分别运行在hadoop1和hadoop2节点，获取数据为NODE\_LOCAL方式。

[](http://images0.cnblogs.com/blog/107289/201508/121102573641796.jpg)

[clip_image114](http://images0.cnblogs.com/blog/107289/201508/121102592545250.jpg)

[](http://images0.cnblogs.com/blog/107289/201508/121103009898921.jpg)

在hadoop2中观察Thrift Server运行日志如下：

[](http://images0.cnblogs.com/blog/107289/201508/121103030512091.jpg)