

# SparkSQL 简介

# 目 录

<b>1 SPARKSQL的发展历程.....</b>	<b>3</b>
1.1 HIVE AND SHARK.....	3
1.2 SHARK和SPARKSQL.....	3
1.3 SPARKSQL的性能.....	4
<b>2 SPARKSQL运行架构.....</b>	<b>8</b>
2.1 TREE和RULE .....	9
2.1.1 Tree .....	9
2.1.2 Rule.....	10
2.2 SQLCONTEXT和HIVECONTEXT的运行过程 .....	10
2.2.1 sqlContext的运行过程.....	10
2.2.2 hiveContext的运行过程.....	11
2.3 CATALYST优化器 .....	12
<b>3 SPARKSQL CLI.....</b>	<b>13</b>
3.1 运行环境说明 .....	13
3.1.1 硬软件环境.....	13
3.1.2 机器网络环境.....	14
3.2 配置并启动 .....	14
3.2.1 创建并配置hive-site.xml .....	14
3.2.2 启动Hive.....	15
3.2.3 启动Spark集群和Spark SQL CLI .....	15
3.2.4 命令参数.....	16
3.3 实战SPARK SQL CLI .....	17
3.3.1 获取订单每年的销售单数、销售总额.....	17
3.3.2 计算所有订单每年的总金额.....	18
3.3.3 计算所有订单每年最大金额订单的销售额 .....	20
<b>4 SPARK THRIFT SERVER.....</b>	<b>21</b>
4.1 配置并启动 .....	21
4.1.1 创建并配置hive-site.xml .....	21
4.1.2 启动Hive.....	23
4.1.3 启动Spark集群和Thrift Server.....	23
4.1.4 命令参数.....	24
4.2 实战THRIFT SERVER .....	25
4.2.1 远程客户端连接.....	25
4.2.2 基本操作.....	26
4.2.3 计算所有订单每年的订单数 .....	28
4.2.4 计算所有订单月销售额前十名 .....	29
4.2.5 缓存表数据.....	31
4.2.6 在IDEA中JDBC访问.....	32

# SparkSQL 简介

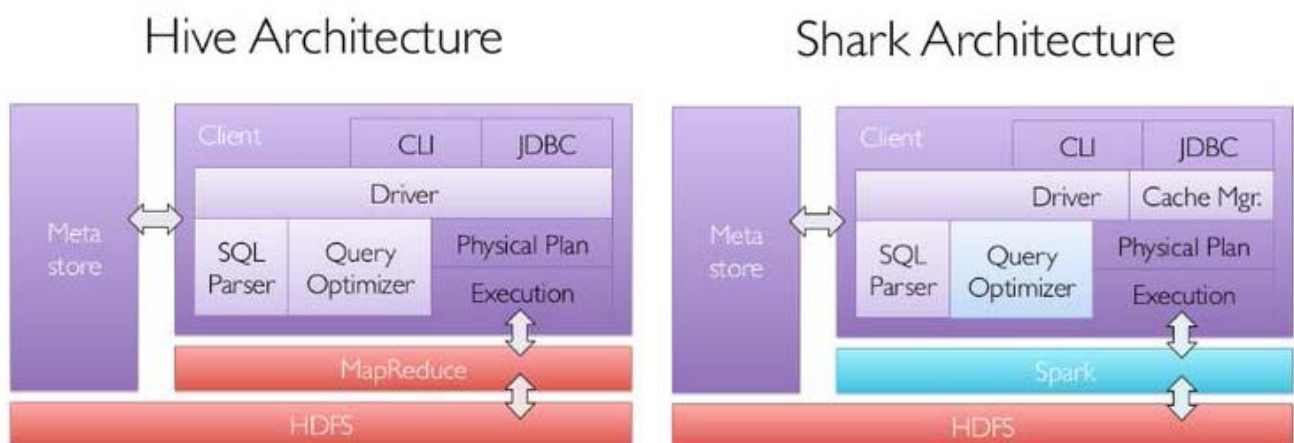
## 1 SparkSQL 的发展历程

### 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。



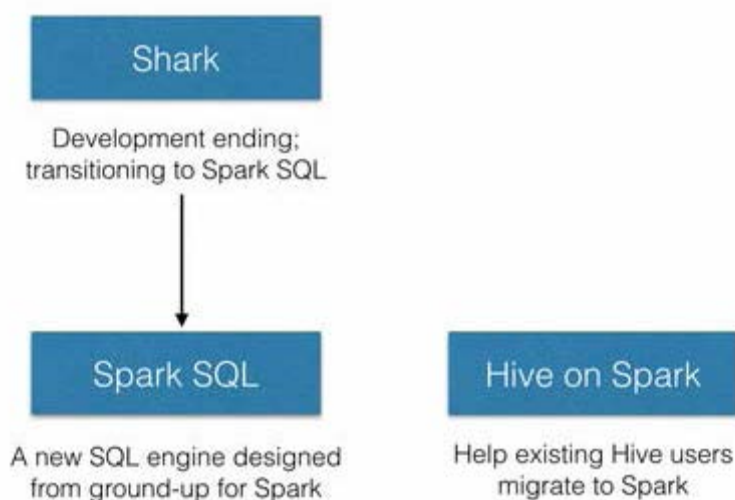
### 1.2 Shark 和 SparkSQL

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 的发展画上了句号，但也因此发展出两个直线：SparkSQL 和 Hive on Spark。

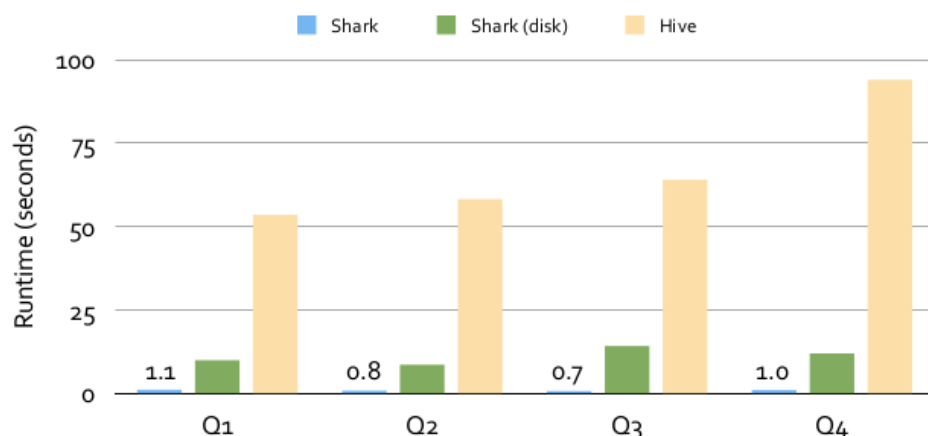


其中 SparkSQL 作为 Spark 生态的一员继续发展，而不再受限于 Hive，只是兼容 Hive；而 Hive on Spark 是一个 Hive 的发展计划，该计划将 Spark 作为 Hive 的底层引擎之一，也就是说，Hive 将不再受限于一个引擎，可以采用 Map-Reduce、Tez、Spark 等引擎。

## 1.3 SparkSQL 的性能

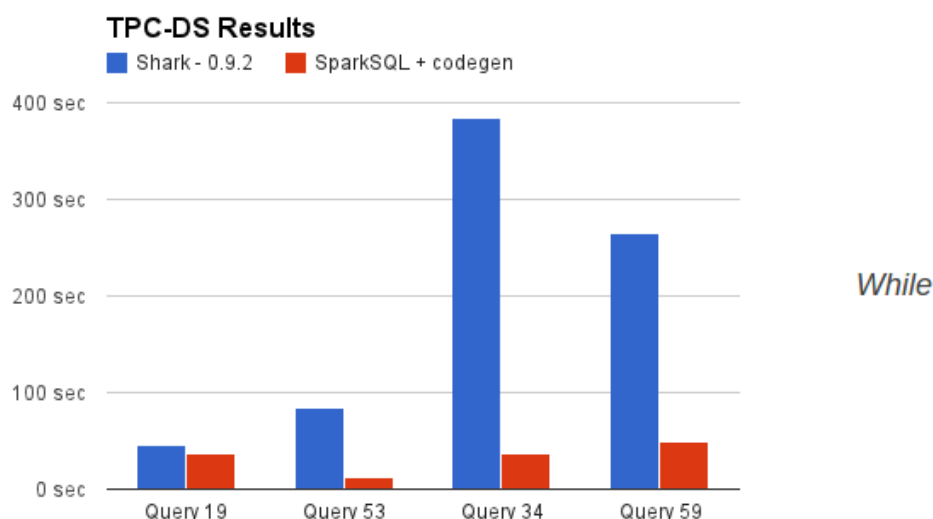
Shark 的出现，使得 SQL-on-Hadoop 的性能比 Hive 有了 10-100 倍的提高：

# Performance



1.7 TB Real Warehouse Data on 100 EC2 nodes

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：



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。



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：

```
case Add(e1, e2) => (e1, e2) evaluate { case (eval1, eval2) => q"$eval1 + $eval2" }
```

然后，通过调用，最终调用：

```
def evaluateAs(resultType: DataType)(f: (TermName, TermName) => Tree): Seq[Tree] = {
  // TODO: Right now some timestamp tests fail if we enforce this...
  if (expressions._1.dataType != expressions._2.dataType) {
    log.warn(s"${expressions._1.dataType} != ${expressions._2.dataType}")
  }

  val eval1 = expressionEvaluator(expressions._1)
  val eval2 = expressionEvaluator(expressions._2)
  val resultCode = f(eval1.primitiveTerm, eval2.primitiveTerm)

  eval1.code ++ eval2.code ++
  q"""
    val $nullTerm = ${eval1.nullTerm} || ${eval2.nullTerm}
    val $primitiveTerm: ${termForType(resultType)} =
      if($nullTerm) {
        ${defaultPrimitive(resultType)}
      } else {
        $resultCode.asInstanceOf[${termForType(resultType)}]
      }
  """.children : Seq[Tree]
}
```

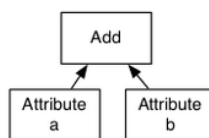
最终实现效果类似如下伪代码：

```
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：

## Interpreting “a+b”

1. Virtual call to Add.eval()
2. Virtual call to a.eval()
3. Return boxed Int
4. Virtual call to b.eval()
5. Return boxed Int
6. Integer addition
7. Return boxed result



SELECT a + b FROM table

```
def generateCode(e: Expression): Tree = e match {  
  case Attribute(ordinal) =>  
    q"inputRow.getInt($ordinal)"  
  case Add(left, right) =>  
    q""  
    {  
      val leftResult = ${generateCode(left)}  
      val rightResult = ${generateCode(right)}  
      leftResult + rightResult  
    }  
  ""  
}
```

```
val a: Int = inputRow.getInt(0)  
val b: Int = inputRow.getInt(1)  
val result: Int = a + b  
resultRow.setInt(0, result)
```

## 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：

- Scala FP features that kill performance:

- Option
- For-loop / map / filter / foreach / ...
- Numeric[T] / Ordering[T] / ...
- Immutable objects (GC stress)



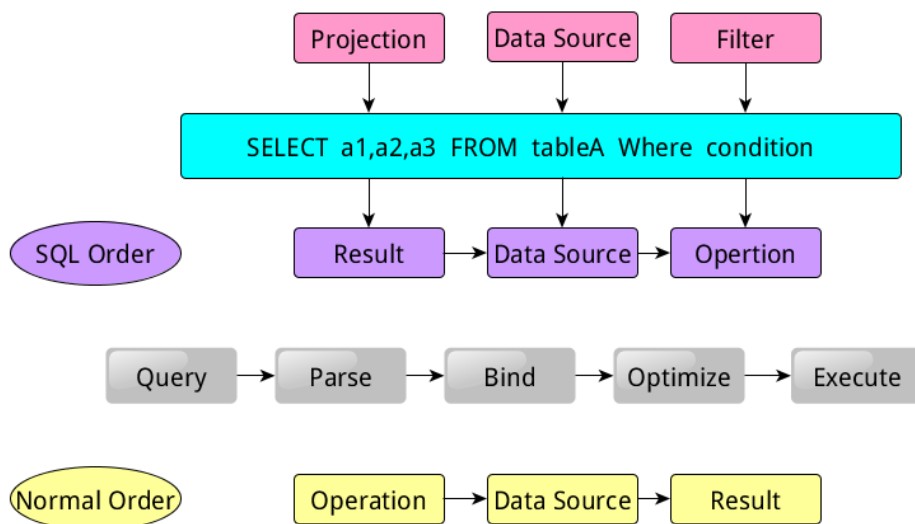
- Have To Resort To:

- null
- While-loop and vars
- Manually specialized code for primitive types
- Reusing mutable objects

## 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。





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 Join、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，

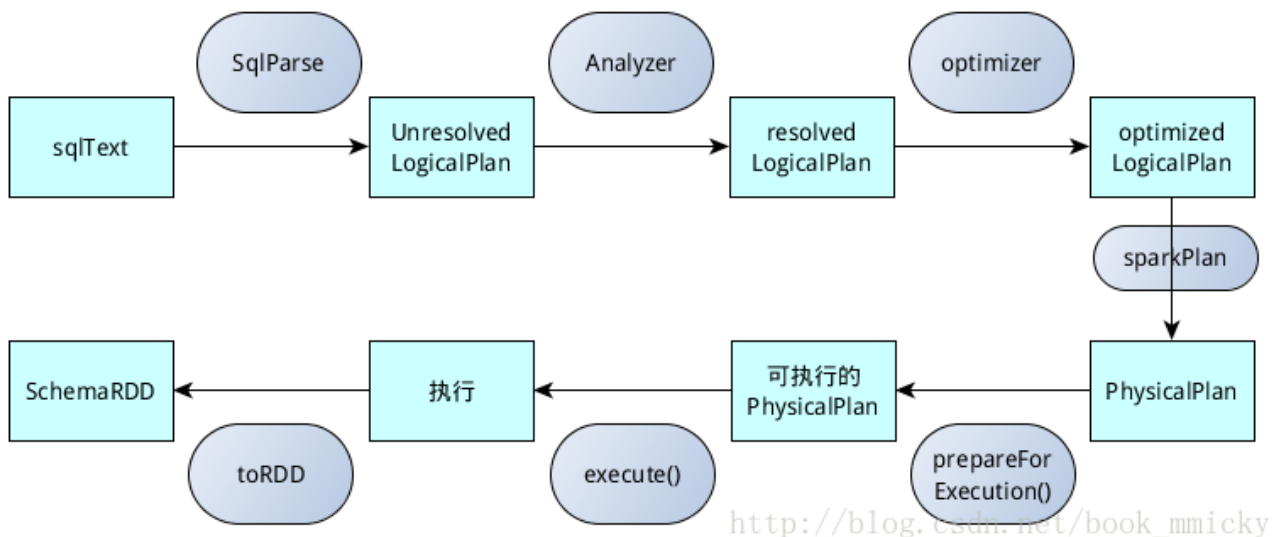
### 2.2.1 sqlContext 的运行过程

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。

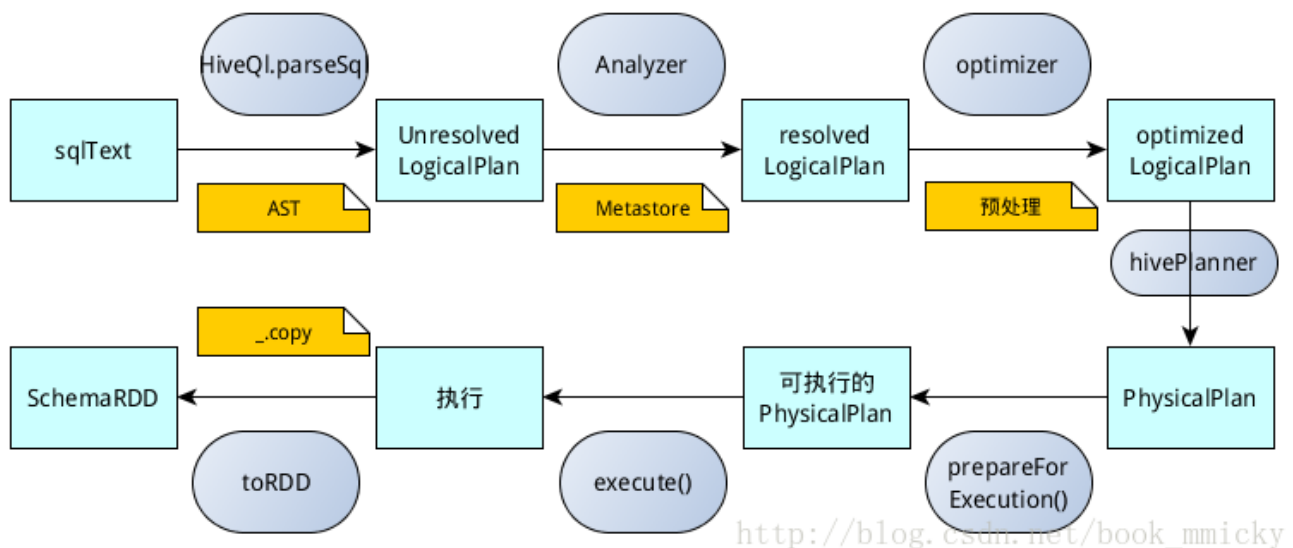
在整个运行过程中涉及到多个 SparkSQL 的组件 如 SqlParse、analyzer、optimizer、SparkPlan 等等



## 2.2.2 hiveContext 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。

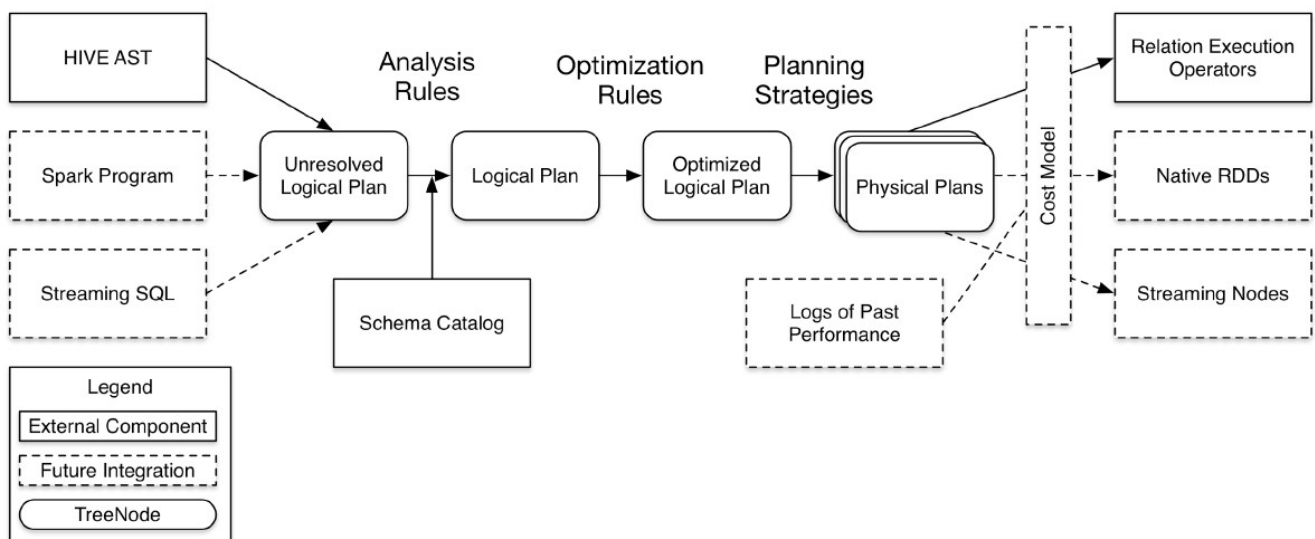


## 2.3 catalyst 优化器

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：



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>

```



```
hadoop1 | hadoop2 | hadoop3 | hadoop1 (1)
[hadoop@hadoop1 ~]$ cd /app/hadoop/spark-1.1.0/conf
[hadoop@hadoop1 conf]$ ls
fairscheduler.xml.template  log4j.properties.template  spark-defaults.conf.template
hive-site-bak.xml           metrics.properties.template  spark-env.sh
hive-site.xml               slaves                        spark-env.sh.template
[hadoop@hadoop1 conf]$ cat hive-site.xml
<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>
[hadoop@hadoop1 conf]$
```

### 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 &
```

```
hadoop1 | hadoop2 | hadoop3 | hadoop1 (1)
[hadoop@hadoop1 ~]$ $nohup hive --service metastore > metastore.log 2>&1 &
[1] 7014
[hadoop@hadoop1 ~]$ jobs
[1]+  Running                  $nohup hive --service metastore > metastore.log 2>&1 &
[hadoop@hadoop1 ~]$
```

### 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:

#### Workers

Id	Address	State	Cores	Memory
worker-20150730215205-hadoop1-50695	hadoop1:50695	ALIVE	1 (1 Used)	1024.0 MB (1024.0 MB Used)
worker-20150730215224-hadoop2-45563	hadoop2:45563	ALIVE	1 (1 Used)	1024.0 MB (1024.0 MB Used)
worker-20150730215225-hadoop3-36128	hadoop3:36128	ALIVE	1 (1 Used)	1024.0 MB (1024.0 MB Used)

#### Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20150730215311-0000	SparkSQL::hadoop1	3	1024.0 MB	2015/07/30 21:53:11	hadoop	RUNNING	9 s

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。

18	mapPartitions at Exchange.scala:48	+details	2014/09/14 22:13:53	0.7 s	20/20		1330.7 KB	139.5 KB
17	mapPartitions at Exchange.scala:48	+details	2014/09/14 22:13:52	0.7 s	20/20		1392.6 KB	1952.2 KB
16	mapPartitions at Exchange.scala:48	+details	2014/09/14 22:13:50	2 s	2/2	11.7 MB		1675.0 KB
15	mapPartitions at Exchange.scala:48	+details	2014/09/14 22:13:50	1 s	2/2	608.7 KB		317.6 KB
14	mapPartitions at Exchange.scala:48	+details	2014/09/14 22:13:50	0.9 s	2/2	171.8 KB		51.3 KB
6	collect at HiveContext.scala:415	+details	2014/09/14 22:12:52	0.1 s	8/8		299.0 B	
12	mapPartitions at Exchange.scala:71	+details	2014/09/14 22:12:50	2 s	200/200		144.3 KB	419.0 B
0	RangePartitioner at Exchange.scala:79	+details	2014/09/14 22:12:48	2 s	200/200		145.5 KB	

### 3.2.4 命令参数

通过 bin/spark-sql --help 可以查看 CLI 命令参数：

```
hadoop1 | hadoop2 | hadoop3
[hadoop@hadoop1 spark-1.1.0]$ bin/spark-sql --help
Usage: ./bin/spark-sql [options] [cli option]
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Options:
  --master MASTER_URL          spark://host:port, mesos://host:port, yarn, or local.
  --deploy-mode DEPLOY_MODE    whether to launch the driver program locally ("client") or
                                on one of the worker machines inside the cluster ("cluster")
                                (Default: client).
  --class CLASS_NAME            Your application's main class (for java / scala apps).
  --name NAME                  A name of your application.
  --jars JARS                  Comma-separated list of local jars to include on the driver
                                and executor classpaths.
  --py-files PY_FILES          Comma-separated list of .zip, .egg, or .py files to place
                                on the PYTHONPATH for Python apps.
  --files FILES                Comma-separated list of files to be placed in the working
                                directory of each executor.
  --conf PROP=VALUE            Arbitrary Spark configuration property.
  --properties-file FILE       Path to a file from which to load extra properties. If not
                                specified, this will look for conf/spark-defaults.conf.
  --driver-memory MEM          Memory for driver (e.g. 1000M, 2G) (Default: 512M).
  --driver-java-options        Extra java options to pass to the driver.
  --driver-library-path        Extra library path entries to pass to the driver.
  --driver-class-path          Extra class path entries to pass to the driver. Note that
                                jars added with --jars are automatically included in the
                                classpath.
  --executor-memory MEM        Memory per executor (e.g. 1000M, 2G) (Default: 1G).
  --help, -h                  Show this help message and exit
  --verbose, -v               Print additional debug output

Spark standalone with cluster deploy mode only:
  --driver-cores NUM          Cores for driver (Default: 1).
  --supervise                  If given, restarts the driver on failure.
```



```

CLI options:
15/07/30 22:11:29 INFO Configuration.deprecation: mapred.input.dir.recursive is deprecated. Instead, use mapreduce
15/07/30 22:11:29 INFO Configuration.deprecation: mapred.max.split.size is deprecated. Instead, use mapreduce.inp
15/07/30 22:11:29 INFO Configuration.deprecation: mapred.min.split.size is deprecated. Instead, use mapreduce.inp
15/07/30 22:11:29 INFO Configuration.deprecation: mapred.min.split.size.per.rack is deprecated. Instead, use mapr
15/07/30 22:11:29 INFO Configuration.deprecation: mapred.min.split.size.per.node is deprecated. Instead, use mapr
15/07/30 22:11:29 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.r
15/07/30 22:11:29 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instea
-d,--define <key=value>      Variable substitution to apply to hive
                             commands. e.g. -d A=B or --define A=B
--database <databasename>    Specify the database to use
-e <quoted-query-string>    SQL from command line
-f <filename>                SQL from files
-h <hostname>                connecting to Hive Server on remote host
--hiveconf <property=value> Use value for given property
--hivevar <key=value>        Variable substitution to apply to hive
                             commands. e.g. --hivevar A=B
-i <filename>                Initialization SQL file
-p <port>                    connecting to Hive Server on port number
-s,--silent                  Silent mode in interactive shell
-v,--verbose                  verbose mode (echo executed SQL to the
                             console)

```

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。

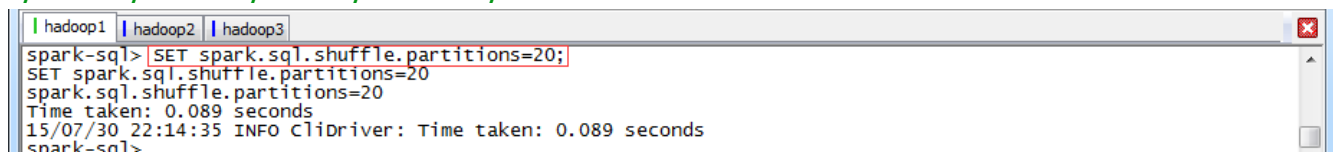
[cli option]是 CLI 的参数，通过这些参数 CLI 可以直接运行 SQL 文件、进入命令行运行 SQL 命令等等，类似以前的 Shark 的用法。需要注意的是 CLI 不是使用 JDBC 连接，所以不能连接到 ThriftServer；但可以配置 conf/hive-site.xml 连接到 Hive 的 Metastore，然后对 Hive 数据进行查询。

## 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;*



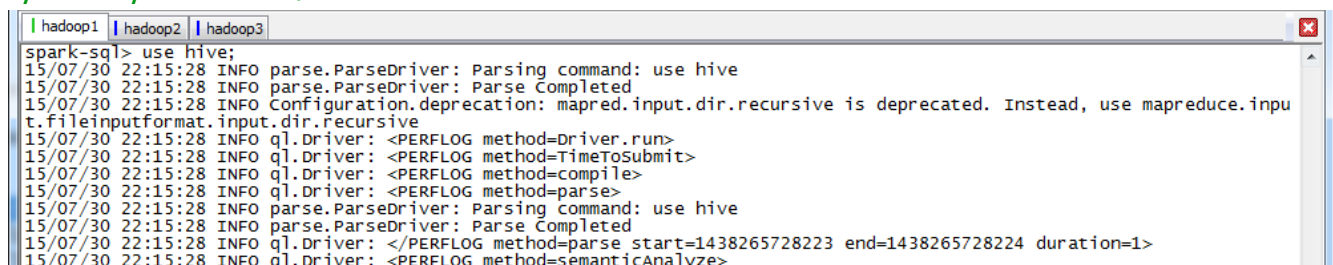
```

hadoop1 | hadoop2 | hadoop3
spark-sql> SET spark.sql.shuffle.partitions=20;
SET spark.sql.shuffle.partitions=20
spark.sql.shuffle.partitions=20
Time taken: 0.089 seconds
15/07/30 22:14:35 INFO CliDriver: Time taken: 0.089 seconds
spark-sql>

```

第二步 运行 SQL 语句

*spark-sql>use hive;*



```

hadoop1 | hadoop2 | hadoop3
spark-sql> use hive;
15/07/30 22:15:28 INFO parse.ParseDriver: Parsing command: use hive
15/07/30 22:15:28 INFO parse.ParseDriver: Parse Completed
15/07/30 22:15:28 INFO Configuration.deprecation: mapred.input.dir.recursive is deprecated. Instead, use mapreduce.inpu
t.fileinputformat.input.dir.recursive
15/07/30 22:15:28 INFO ql.Driver: <PERFLOG method=Driver.run>
15/07/30 22:15:28 INFO ql.Driver: <PERFLOG method=TimeToSubmit>
15/07/30 22:15:28 INFO ql.Driver: <PERFLOG method=compile>
15/07/30 22:15:28 INFO ql.Driver: <PERFLOG method=parse>
15/07/30 22:15:28 INFO parse.ParseDriver: Parsing command: use hive
15/07/30 22:15:28 INFO parse.ParseDriver: Parse Completed
15/07/30 22:15:28 INFO ql.Driver: </PERFLOG method=parse start=1438265728223 end=1438265728224 duration=1>
15/07/30 22:15:28 INFO ql.Driver: <PERFLOG method=semanticAnalyze>

```

*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;*

```
hadoop1 | hadoop2 | hadoop3
spark-sql> select c.theyear,count(distinct a.ordernumber),sum(b.amount) from tbstock a join tbstockDetail b on a.ordernumber=b.ordernumb
er join tbbate c on a.dateid=c.dateid group by c.theyear order by c.theyear;
15/07/30 22:17:30 INFO parse.ParserDriver: Parsing command: select c.theyear,count(distinct a.ordernumber),sum(b.amount) from tbstock a jo
in tbstockDetail b on a.ordernumber=b.ordernumber join tbbate c on a.dateid=c.dateid group by c.theyear order by c.theyear
15/07/30 22:17:30 INFO parse.ParserDriver: Parse Completed
15/07/30 22:17:30 INFO storage.MemoryStore: ensureFreeSpace(402171) called with curMem=1397516, maxMem=278302556
15/07/30 22:17:30 INFO storage.MemoryStore: Block broadcast_11 stored as values in memory (estimated size 392.7 KB, free 263.7 MB)
15/07/30 22:17:31 INFO storage.MemoryStore: ensureFreeSpace(27474) called with curMem=1799687, maxMem=278302556
15/07/30 22:17:31 INFO storage.MemoryStore: Block broadcast_11_piece0 stored as bytes in memory (estimated size 26.8 KB, free 263.7 MB)
15/07/30 22:17:31 INFO storage.BlockManagerInfo: Added broadcast_11_piece0 in memory on hadoop1:55278 (size: 26.8 KB, free: 265.3 MB)
15/07/30 22:17:31 INFO storage.BlockManagerMaster: updated info of block broadcast_11_piece0
15/07/30 22:17:31 INFO storage.MemoryStore: ensureFreeSpace(402043) called with curMem=1827161, maxMem=278302556
15/07/30 22:17:31 INFO storage.MemoryStore: Block broadcast_12 stored as values in memory (estimated size 392.6 KB, free 263.3 MB)
15/07/30 22:17:31 INFO storage.MemoryStore: ensureFreeSpace(27429) called with curMem=2229204, maxMem=278302556
15/07/30 22:17:31 INFO storage.MemoryStore: Block broadcast_12_piece0 stored as bytes in memory (estimated size 26.8 KB, free 263.3 MB)
```

### 第三步 查看运行结果

```
15/07/30 22:18:52 INFO scheduler.DAGScheduler: Stage 32 (collect at HiveContext.scala:415) finished in 0.199 s
15/07/30 22:18:52 INFO spark.SparkContext: Job finished: collect at HiveContext.scala:415, took 1.019829631 s
15/07/30 22:18:52 INFO scheduler.StatsReportListener: Finished stage: org.apache.spark.scheduler.StageInfo@4f1d6764
2004 1094 3265696
2005 3828 13247234
2006 3772 13670416
2007 4885 16711974
2008 4861 14670698
2009 2619 6322137
2010 94 210924
Time taken: 8.954 seconds
15/07/30 22:18:52 INFO CLIDriver: Time taken: 8.954 seconds
15/07/30 22:18:52 INFO scheduler.StatsReportListener: task runtime:(count: 8, mean: 74.875000, stdev: 37.747310, max: 131.000000, min: 30.000
000)
spark-sql> 15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0% 5% 10% 25% 50% 75% 90% 95% 100%
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 30.0 ms 30.0 ms 30.0 ms 45.0 ms 83.0 ms 119.0 ms 131.0 ms 131.0 ms 1
31.0 ms
15/07/30 22:18:52 INFO scheduler.StatsReportListener: fetch wait time:(count: 8, mean: 7.875000, stdev: 5.509934, max: 15.000000, min: 0.0000
00)
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0% 5% 10% 25% 50% 75% 90% 95% 100%
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0.0 ms 0.0 ms 0.0 ms 4.0 ms 11.0 ms 13.0 ms 15.0 ms 15.0 ms 15.0 ms
15/07/30 22:18:52 INFO scheduler.StatsReportListener: remote bytes read:(count: 8, mean: 44.875000, stdev: 25.910604, max: 60.000000, min: 0.
000000)
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0% 5% 10% 25% 50% 75% 90% 95% 100%
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0.0 B 0.0 B 0.0 B 59.0 B 60.0 B 60.0 B 60.0 B 60.0 B 60.0 B
15/07/30 22:18:52 INFO scheduler.StatsReportListener: task result size:(count: 8, mean: 1064.750000, stdev: 80.033977, max: 1095.000000, min:
853.000000)
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0% 5% 10% 25% 50% 75% 90% 95% 100%
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 853.0 B 853.0 B 853.0 B 1095.0 B 1095.0 B 1095.0 B 1095.0 B 1095.0 B 1
095.0 B
15/07/30 22:18:52 INFO scheduler.StatsReportListener: executor (non-fetch) time pct: (count: 8, mean: 35.884653, stdev: 11.273334, max: 57.83
1325, min: 20.370370)
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0% 5% 10% 25% 50% 75% 90% 95% 100%
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 20 % 20 % 20 % 26 % 40 % 42 % 58 % 58 % 58 %
15/07/30 22:18:52 INFO scheduler.StatsReportListener: fetch wait time pct: (count: 8, mean: 10.674235, stdev: 9.301000, max: 28.888889, min:
0.000000)
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0% 5% 10% 25% 50% 75% 90% 95% 100%
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0 % 0 % 0 % 3 % 11 % 18 % 29 % 29 % 29 %
15/07/30 22:18:52 INFO scheduler.StatsReportListener: other time pct: (count: 8, mean: 53.441112, stdev: 15.266191, max: 64.814815, min: 24.0
96386)
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 0% 5% 10% 25% 50% 75% 90% 95% 100%
15/07/30 22:18:52 INFO scheduler.StatsReportListener: 24 % 24 % 24 % 57 % 63 % 65 % 65 % 65 % 65 %
```

hadoop1:4040/stages/

Spark

StagesStorageEnvironmentExecutors

SparkSQL::hadoop1 application UI

### Spark Stages

Total Duration: 5.9 min  
Scheduling Mode: FIFO  
Active Stages: 0  
Completed Stages: 24  
Failed Stages: 0

Active Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
----------	-------------	-----------	----------	------------------------	-------	--------------	---------------

Completed Stages (24)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
32	<a href="#">collect at HiveContext.scala:415</a>	+details 2015/07/30 22:18:52	0.2 s	8/8		359.0 B	
38	<a href="#">mapPartitions at Exchange.scala:71</a>	+details 2015/07/30 22:18:51	0.7 s	20/20		93.1 KB	419.0 B
26	<a href="#">RangePartitioner at Exchange.scala:79</a>	+details 2015/07/30 22:18:50	0.8 s	20/20		92.4 KB	
31	<a href="#">mapPartitions at Exchange.scala:48</a>	+details 2015/07/30 22:18:48	2 s	20/20		1331.8 KB	139.4 KB
30	<a href="#">mapPartitions at Exchange.scala:48</a>	+details 2015/07/30 22:18:46	2 s	20/20		1298.2 KB	1955.8 KB

## 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;*

```

hadoop1 | hadoop2 | hadoop3
spark-sql> select c.theyear,count(distinct a.ordernumber),sum(b.amount) from tbStock a join tbStockDetail b on a.ordernumber=b.ordernum
ber join tbDate c on a.dateid=c.dateid group by c.theyear order by c.theyear;
15/07/30 22:27:22 INFO parse.ParseDriver: Parsing command: select c.theyear,count(distinct a.ordernumber),sum(b.amount) from tbStock a j
oin tbStockDetail b on a.ordernumber=b.ordernumber join tbDate c on a.dateid=c.dateid group by c.theyear order by c.theyear
15/07/30 22:27:22 INFO parse.ParseDriver: Parse Completed
15/07/30 22:27:23 INFO storage.MemoryStore: ensureFreeSpace(402563) called with curMem=2746678, maxMem=278302556
15/07/30 22:27:23 INFO storage.MemoryStore: Block broadcast_51 stored as values in memory (estimated size 393.1 KB, free 262.4 MB)
15/07/30 22:27:23 INFO storage.MemoryStore: ensureFreeSpace(27510) called with curMem=3149241, maxMem=278302556
15/07/30 22:27:23 INFO storage.MemoryStore: Block broadcast_51.piece0 stored as bytes in memory (estimated size 26.9 KB, free 262.4 MB)
15/07/30 22:27:23 INFO storage.BlockManagerInfo: Added broadcast_51.piece0 in memory on hadoop1:55278 (size: 26.9 KB, free: 265.2 MB)
15/07/30 22:27:23 INFO storage.BlockManagerMaster: Updated info of block broadcast_51.piece0
15/07/30 22:27:23 INFO storage.MemoryStore: ensureFreeSpace(402563) called with curMem=3176751, maxMem=278302556
15/07/30 22:27:23 INFO storage.MemoryStore: Block broadcast_52 stored as values in memory (estimated size 393.1 KB, free 262.0 MB)
15/07/30 22:27:23 INFO storage.MemoryStore: ensureFreeSpace(27491) called with curMem=3579314, maxMem=278302556
15/07/30 22:27:23 INFO storage.MemoryStore: Block broadcast_52.piece0 stored as bytes in memory (estimated size 26.8 KB, free 262.0 MB)
15/07/30 22:27:23 INFO storage.BlockManagerInfo: Added broadcast_52.piece0 in memory on hadoop1:55278 (size: 26.8 KB, free: 265.1 MB)
15/07/30 22:27:23 INFO storage.BlockManagerMaster: Updated info of block broadcast_52.piece0
15/07/30 22:27:23 INFO storage.MemoryStore: ensureFreeSpace(402659) called with curMem=3606805, maxMem=278302556
15/07/30 22:27:23 INFO storage.MemoryStore: Block broadcast_53 stored as values in memory (estimated size 393.2 KB, free 261.6 MB)
15/07/30 22:27:23 INFO storage.MemoryStore: ensureFreeSpace(27488) called with curMem=4009464, maxMem=278302556
15/07/30 22:27:23 INFO storage.MemoryStore: Block broadcast_53.piece0 stored as bytes in memory (estimated size 26.8 KB, free 261.6 MB)
15/07/30 22:27:23 INFO storage.BlockManagerInfo: Added broadcast_53.piece0 in memory on hadoop1:55278 (size: 26.8 KB, free: 265.1 MB)
15/07/30 22:27:23 INFO storage.BlockManagerMaster: Updated info of block broadcast_53.piece0
15/07/30 22:27:23 INFO spark.SparkContext: Starting job: RangePartitioner at Exchange.scala:79

```

## 第二步 执行结果

使用 CLI 执行结果如下：

```

15/07/30 22:28:45 INFO spark.SparkContext: Job finished: collect at HiveContext.scala:415, took 0.68904225 s
2004      1094      3265696
15/07/30 22:28:45 INFO scheduler.StatsReportListener: task runtime:(count: 8, mean: 52.625000, stdev: 36.993031, max: 120.000000, min: 1
6.000000)
2005      3828      13247234
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0%      5%      10%      25%      50%      75%      90%      95%      100%
2006      3772      13670416
2007      4885      1671197415/07/30 22:28:45 INFO scheduler.StatsReportListener: 16.0 ms 16.0 ms 16.0 ms 29.0 ms 32.0 ms 102.0 ms 1
20.0 ms 120.0 ms 120.0 ms
2008      4861      14670698
2009      2619      6322137
2010      94      210924
Time taken: 6.127 seconds
15/07/30 22:28:45 INFO CliDriver: Time taken: 6.127 seconds
spark-sql> 15/07/30 22:28:45 INFO scheduler.StatsReportListener: fetch wait time:(count: 8, mean: 3.625000, stdev: 6.203578, max: 19.000
000, min: 0.000000)
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0%      5%      10%      25%      50%      75%      90%      95%      100%
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0.0 ms 0.0 ms 0.0 ms 0.0 ms 0.0 ms 6.0 ms 19.0 ms 19.0 ms 19.0 ms
15/07/30 22:28:45 INFO scheduler.StatsReportListener: remote bytes read:(count: 8, mean: 22.500000, stdev: 29.047375, max: 60.000000, mi
n: 0.000000)
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0%      5%      10%      25%      50%      75%      90%      95%      100%
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0.0 B 0.0 B 0.0 B 0.0 B 0.0 B 60.0 B 60.0 B 60.0 B 60.0 B
15/07/30 22:28:45 INFO scheduler.StatsReportListener: task result size:(count: 8, mean: 1064.750000, stdev: 80.033977, max: 1095.000000, m
in: 853.000000)
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0%      5%      10%      25%      50%      75%      90%      95%      100%
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 853.0 B 853.0 B 853.0 B 1095.0 B 1095.0 B 1095.0 B 1095.0 B 1095.0 B 1095.0 B
15/07/30 22:28:45 INFO scheduler.StatsReportListener: executor (non-fetch) time pct: (count: 8, mean: 34.265778, stdev: 6.291089, max: 4
8.275862, min: 28.125000)
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0%      5%      10%      25%      50%      75%      90%      95%      100%
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 28 % 28 % 28 % 30 % 32 % 38 % 48 % 48 % 48 %
15/07/30 22:28:45 INFO scheduler.StatsReportListener: fetch wait time pct: (count: 8, mean: 5.088848, stdev: 7.925312, max: 18.750000, m
in: 0.000000)
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0%      5%      10%      25%      50%      75%      90%      95%      100%
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0 % 0 % 0 % 0 % 0 % 19 % 19 % 19 % 19 %
15/07/30 22:28:45 INFO scheduler.StatsReportListener: other time pct: (count: 8, mean: 60.645374, stdev: 7.307436, max: 71.014493, min:
50.980392)
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 0%      5%      10%      25%      50%      75%      90%      95%      100%
15/07/30 22:28:45 INFO scheduler.StatsReportListener: 51 % 51 % 51 % 53 % 63 % 69 % 71 % 71 % 71 %

```

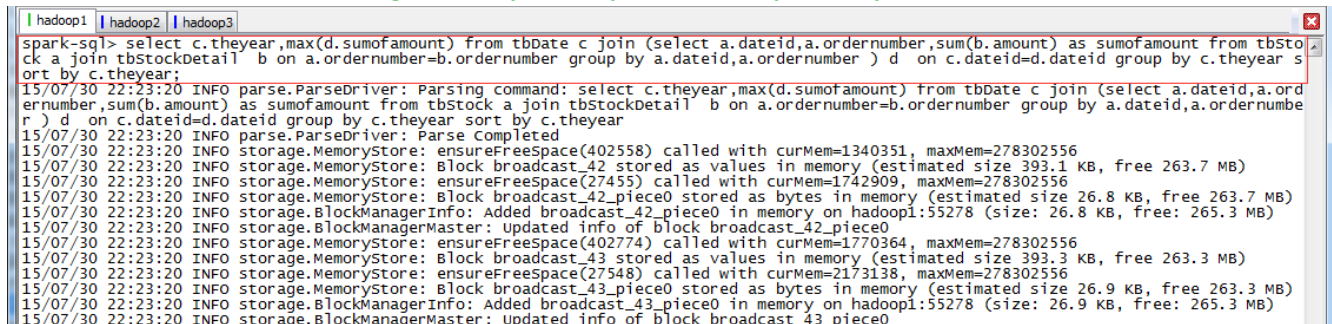
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
57	<a href="#">collect at HiveContext.scala:415</a>	<a href="#">+details</a>	2015/07/30 22:27:30	0.2 s	8/8	179.0 B	
63	<a href="#">mapPartitions at Exchange.scala:71</a>	<a href="#">+details</a>	2015/07/30 22:27:30	0.7 s	20/20	93.3 KB	419.0 B
51	<a href="#">RangePartitioner at Exchange.scala:79</a>	<a href="#">+details</a>	2015/07/30 22:27:28	0.8 s	20/20	93.8 KB	
56	<a href="#">mapPartitions at Exchange.scala:48</a>	<a href="#">+details</a>	2015/07/30 22:27:26	2 s	20/20	1299.2 KB	139.4 KB
55	<a href="#">mapPartitions at Exchange.scala:48</a>	<a href="#">+details</a>	2015/07/30 22:27:25	1 s	20/20	1359.3 KB	1895.9 KB
54	<a href="#">mapPartitions at Exchange.scala:48</a>	<a href="#">+details</a>	2015/07/30 22:27:23	2 s	2/2	11.4 MB	1675.0 KB
53	<a href="#">mapPartitions at Exchange.scala:48</a>	<a href="#">+details</a>	2015/07/30 22:27:23	0.9 s	2/2	588.0 KB	333.7 KB
52	<a href="#">mapPartitions at Exchange.scala:48</a>	<a href="#">+details</a>	2015/07/30 22:27:23	0.8 s	2/2	167.5 KB	51.3 KB



### 3.3.3 计算所有订单每年最大金额订单的销售额

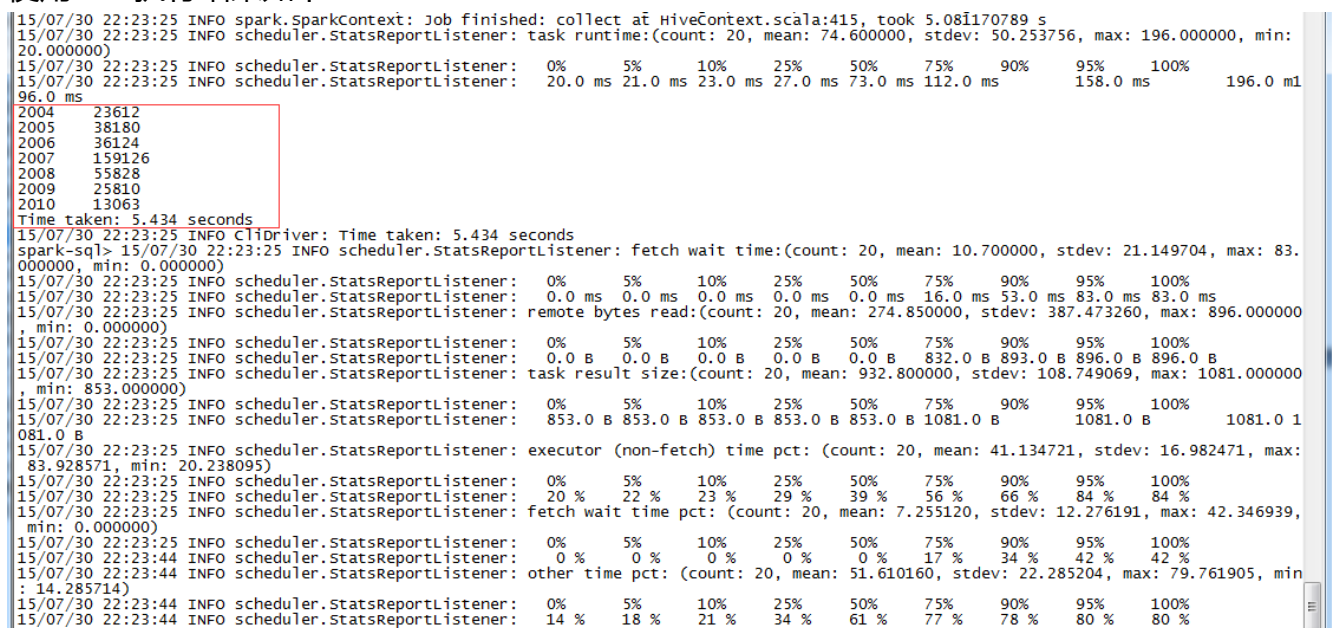
#### 第一步 执行 SQL 语句

```
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;
```



#### 第二步 执行结果

使用 CLI 执行结果如下：



Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
45	<a href="#">collect at HiveContext.scala:415</a>	2015/07/30 22:23:25	0.5 s	20/20		5.4 KB	
50	<a href="#">mapPartitions at Exchange.scala:48</a>	2015/07/30 22:23:23	1 s	20/20		266.8 KB	8.1 KB
49	<a href="#">mapPartitions at Exchange.scala:48</a>	2015/07/30 22:23:22	2 s	20/20		1296.6 KB	348.4 KB
48	<a href="#">mapPartitions at Exchange.scala:48</a>	2015/07/30 22:23:20	1 s	2/2	11.4 MB		1675.0 KB
47	<a href="#">mapPartitions at Exchange.scala:48</a>	2015/07/30 22:23:20	0.7 s	2/2	588.0 KB		333.7 KB
46	<a href="#">mapPartitions at Exchange.scala:48</a>	2015/07/30 22:23:20	0.7 s	2/2	167.5 KB		53.0 KB

## 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

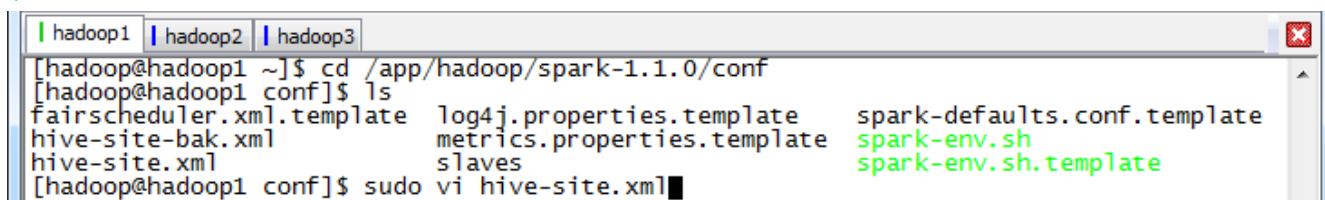
#### 4.1.1 创建并配置 hive-site.xml

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
```



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>
```

```
hadoop1 | hadoop2 | hadoop3
<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>
```

## 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 &
```

```
hadoop1 | hadoop2 | hadoop3
[hadoop@hadoop1 sbin]$ nohup hive --service metastore > metastore.log 2>&1 &
[1] 3916
[hadoop@hadoop1 sbin]$ jobs
[1]+  Running                  nohup hive --service metastore > metastore.log 2>&1 &
[hadoop@hadoop1 sbin]$ jps
3198 Main
3916 RunJar
3477 DataNode
3998 Jps
3604 SecondaryNameNode
3365 NameNode
```

## 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
```

```
hadoop1 | hadoop2 | hadoop3
[hadoop@hadoop2 sbin]$ cd /app/hadoop/spark-1.1.0/sbin
[hadoop@hadoop2 sbin]$ ls
slaves.sh      spark-daemons.sh  start-history-server.sh  start-slaves.sh      stop-history-server.sh
spark-config.sh spark-executor      start-master.sh          start-thriftserver.sh stop-master.sh
spark-daemon.sh start-all.sh        start-slave.sh           stop-all.sh          stop-slaves.sh
[hadoop@hadoop2 sbin]$ ./start-thriftserver.sh --master spark://hadoop1:7077 --executor-memory 1g
Spark assembly has been built with Hive, including Datanucleus jars on classpath
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/07/30 22:40:13 WARN NativeCodeLoader: unable to load native-hadoop library for your platform... using builtin-java classes where applicable
15/07/30 22:40:13 INFO HiveThriftServer2: Starting SparkContext
15/07/30 22:40:14 INFO securityManager: Changing view acls to: hadoop,
15/07/30 22:40:14 INFO securityManager: Changing modify acls to: hadoop,
15/07/30 22:40:14 INFO securityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(hadoop, ); users with modify permissions: Set(hadoop, )
```

**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：

#### Workers

Id	Address	State	Cores	Memory
worker-20150730215823-hadoop1-45473	hadoop1:45473	ALIVE	1 (1 Used)	1024.0 MB (1024.0 MB Used)
worker-20150730215824-hadoop2-37469	hadoop2:37469	ALIVE	1 (1 Used)	1024.0 MB (1024.0 MB Used)
worker-20150730215824-hadoop3-33258	hadoop3:33258	ALIVE	1 (1 Used)	1024.0 MB (1024.0 MB Used)

#### Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20150730224027-0005	SparkSQL::hadoop2	3	1024.0 MB	2015/07/30 22:40:27	hadoop	RUNNING	36 s

## 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
```



```
hadoop1 | hadoop2 | hadoop3 | hadoop2 (1)
[hadoop@hadoop2 spark-1.1.0]$ sbin/start-thriftserver.sh --help
Usage: ./sbin/start-thriftserver [options] [thrift server options]
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Options:
  --master MASTER_URL      spark://host:port, mesos://host:port, yarn, or local.
                           whether to launch the driver program locally ("client") or
                           on one of the worker machines inside the cluster ("cluster")
                           (Default: client).
  --deploy-mode DEPLOY_MODE
                           Your application's main class (for java / scala apps).
                           A name of your application.
  --class CLASS_NAME
                           Comma-separated list of local jars to include on the driver
                           and executor classpaths.
  --name NAME
                           Comma-separated list of .zip, .egg, or .py files to place
                           on the PYTHONPATH for Python apps.
  --jars JARS
                           Comma-separated list of files to be placed in the working
                           directory of each executor.
  --py-files PY_FILES
  --files FILES
  --conf PROP=VALUE        Arbitrary Spark configuration property.
  --properties-file FILE   Path to a file from which to load extra properties. If not
                           specified, this will look for conf/spark-defaults.conf.
  --driver-memory MEM      Memory for driver (e.g. 1000M, 2G) (Default: 512M).
  --driver-java-options    Extra Java options to pass to the driver.
  --driver-library-path    Extra library path entries to pass to the driver.
  --driver-class-path      Extra class path entries to pass to the driver. Note that
                           jars added with --jars are automatically included in the
                           classpath.
  --executor-memory MEM    Memory per executor (e.g. 1000M, 2G) (Default: 1G).
  --help, -h               Show this help message and exit
  --verbose, -v            Print additional debug output

Spark standalone with cluster deploy mode only:
  --driver-cores NUM       Cores for driver (Default: 1).
  --supervise              If given, restarts the driver on failure.

Spark standalone and Mesos only:
  --total-executor-cores NUM Total cores for all executors.

YARN-only:
  --executor-cores NUM     Number of cores per executor (Default: 1).
  --queue QUEUE_NAME       The YARN queue to submit to (Default: "default").
  --num-executors NUM      Number of executors to launch (Default: 2).
  --archives ARCHIVES      Comma separated list of archives to be extracted into the

Thrift server options:
  --hiveconf <property=value> Use value for given property
```

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
```

```
hadoop1 | hadoop2 | hadoop3
[hadoop@hadoop1 ~]$ cd /app/hadoop/spark-1.1.0/bin
[hadoop@hadoop1 bin]$ ls
beeline          load-spark-env.sh  pyspark.cmd       spark-class      spark-shell.cmd  utils.sh
compute-classpath.cmd  metastore.log      run-example       spark-class2.cmd spark-sql         spark-submit
compute-classpath.sh  pyspark            run-example2.cmd  spark-class.cmd  spark-shell      spark-submit.cmd
file:              pyspark2.cmd      run-example.cmd   spark-shell

[hadoop@hadoop1 bin]$ ./beeline
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Beeline version 1.1.0 by Apache Hive
beeline> !connect jdbc:hive2://hadoop2:10000
scan complete in 9ms
Connecting to jdbc:hive2://hadoop2:10000
Enter username for jdbc:hive2://hadoop2:10000: hadoop
Enter password for jdbc:hive2://hadoop2:10000:
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Connected to: Hive (version 0.12.0)
Driver: Spark Project Core (version 1.1.0)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://hadoop2:10000>
```

## 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;
```

```
hadoop1 | hadoop2 | hadoop3
0: jdbc:hive2://hadoop2:10000> show databases;
+-----+
| result |
+-----+
| default|
| hive   |
+-----+
2 rows selected (0.713 seconds)
0: jdbc:hive2://hadoop2:10000> use hive;
+-----+
| result |
+-----+
+-----+
No rows selected (0.272 seconds)
0: jdbc:hive2://hadoop2:10000> show tables;
+-----+
| result |
+-----+
| sogouq1|
| sogouq2|
| tbdate |
| tbstock|
| tbstockdetail|
+-----+
5 rows selected (0.316 seconds)_
```

第二步 创建表 testThrift

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

```
beeline>show tables;
```

```

hadoop1 | hadoop2 | hadoop3
0: jdbc:hive2://hadoop2:10000> create table testThrift(field1 string , field2 Int);
+-----+
| result |
+-----+
No rows selected (1.413 seconds)
0: jdbc:hive2://hadoop2:10000> show tables;
+-----+
| result |
+-----+
sogouq1
sogouq2
tbdate
tbstock
tbstockdetail
testthrift
+-----+
6 rows selected (0.298 seconds)

```

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;

```

```

hadoop1 | hadoop2 | hadoop3
0: jdbc:hive2://hadoop2:10000> insert into table testThrift select ordernumber,amount from
tbStockDetail where amount>3000;
+-----+-----+
| ordernumber | amount |
+-----+-----+
No rows selected (3.801 seconds)
0: jdbc:hive2://hadoop2:10000> select * from testThrift;
+-----+-----+
| field1 | field2 |
+-----+-----+
GHSL00000743 | 5025 |
GHSL00001503 | 10989 |
HMJSL00000706 | 4490 |
HMJSL00000706 | 10776 |
HMJSL00000826 | 5988 |
HMJSL00001769 | 6578 |
HMJSL00002224 | 3440 |
+-----+-----+

```

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;

```

```
hadoop1 | hadoop2 | hadoop3
0: jdbc:hive2://hadoop2:10000> drop table testThrift;
+-----+
| Result |
+-----+
No rows selected (1.099 seconds)
0: jdbc:hive2://hadoop2:10000> create table testThrift (field1 string , field2 Int);
+-----+
| result |
+-----+
No rows selected (0.33 seconds)
0: jdbc:hive2://hadoop2:10000> insert into table testThrift select c.theyear,max(d.sumofamount) f
rom 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;
+-----+
| theyear | c_1 |
+-----+
No rows selected (69.26 seconds)
0: jdbc:hive2://hadoop2:10000> select * from testThrift;
+-----+
| field1 | field2 |
+-----+
| 2010    | 13063  |
| 2004    | 23612  |
| 2005    | 38180  |
| 2006    | 36124  |
| 2007    | 159126 |
| 2008    | 55828  |
| 2009    | 25810  |
+-----+
7 rows selected (2.407 seconds)
```

### 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

```
hadoop1 | hadoop2 | hadoop3
0: jdbc:hive2://hadoop2:10000> select c.theyear, count(distinct a.ordernumber) from tbStock a join
n tbStockDetail b on a.ordernumber=b.ordernumber join tbDate c on a.dateid=c.dateid group by c.t
hey year order by c.theyear;
+-----+
| theyear | c_1 |
+-----+
| 2004    | 1094 |
| 2005    | 3828 |
| 2006    | 3772 |
| 2007    | 4885 |
| 2008    | 4861 |
| 2009    | 2619 |
| 2010    | 94   |
+-----+
7 rows selected (37.439 seconds)
```

Stage 监控页面：

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
28	<code>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</code> <a href="#">collect at SparkSQLOperationManager.scala:191</a> +details	2015/07/30 22:59:44	0.1 s	8/8		275.0 B	
34	<code>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</code> <a href="#">mapPartitions at Exchange.scala:71</a> +details	2015/07/30 22:59:41	3 s	200/200		143.5 KB	385.0 B
22	<code>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</code> <a href="#">RangePartitioner at Exchange.scala:79</a> +details	2015/07/30 22:59:38	3 s	200/200		143.0 KB	
27	<code>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</code> <a href="#">mapPartitions at Exchange.scala:48</a> +details	2015/07/30 22:59:23	15 s	200/200		1470.0 KB	214.7 KB
26	<code>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</code> <a href="#">mapPartitions at Exchange.scala:48</a> +details	2015/07/30 22:59:09	13 s	200/200		988.6 KB	2.1 MB
25	<code>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</code> <a href="#">mapPartitions at Exchange.scala:48</a> +details	2015/07/30 22:59:07	2 s	2/2	11.4 MB		1058.1 KB
24	<code>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</code> <a href="#">mapPartitions at Exchange.scala:48</a> +details	2015/07/30 22:59:07	1 s	2/2	588.0 KB		413.0 KB
23	<code>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</code> <a href="#">mapPartitions at Exchange.scala:48</a> +details	2015/07/30 22:59:07	0.7 s	2/2	167.5 KB		87.3 KB

## 查看 Details for Stage 28

### Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input	Shuffle Read	Shuffle Write	Shuffle Spill (Memory)	Shuffle Spill (Disk)
0	hadoop2:35048	0.1 s	3	0	3	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
1	hadoop1:60828	0.1 s	3	0	3	0.0 B	165.0 B	0.0 B	0.0 B	0.0 B
2	hadoop3:39761	0.1 s	2	0	2	0.0 B	110.0 B	0.0 B	0.0 B	0.0 B

### Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Shuffle Read	Errors
0	1631	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/30 22:59:44	31 ms			55.0 B	
1	1632	0	SUCCESS	PROCESS_LOCAL	hadoop3	2015/07/30 22:59:44	31 ms			55.0 B	
2	1633	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/30 22:59:44	49 ms			0.0 B	
3	1634	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/30 22:59:44	15 ms			55.0 B	
4	1635	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/30 22:59:44	11 ms			0.0 B	
5	1636	0	SUCCESS	PROCESS_LOCAL	hadoop3	2015/07/30 22:59:44	17 ms			55.0 B	
6	1637	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/30 22:59:44	10 ms			55.0 B	
7	1638	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/30 22:59:44	8 ms			0.0 B	

## 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

hadoop1	hadoop2	hadoop3
0: jdbc:hive2://hadoop2:10000> 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;		
theyear	themonth	sumofamount
2008	1	2444809
2006	1	1897819
2007	2	1809792
2007	10	1691198
2007	1	1606365
2008	5	1582829
2007	12	1552968
2007	5	1486832
2008	2	1484344
2005	8	1429748
10 rows selected (25.58 seconds)		

Stage 监控页面：

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
41	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 collect at SparkSQLOperationManager.scala:191	2015/07/30 23:03:42	19 ms	1/1			
35	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 takeOrdered at basicOperators.scala:171	2015/07/30 23:03:39	3 s	200/200		85.7 KB	
40	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 mapPartitions at Exchange.scala:48	2015/07/30 23:03:27	11 s	200/200		1676.6 KB	127.5 KB
39	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 mapPartitions at Exchange.scala:48	2015/07/30 23:03:18	9 s	200/200		1410.5 KB	2.4 MB
38	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 mapPartitions at Exchange.scala:48	2015/07/30 23:03:17	2 s	2/2	11.4 MB		1713.1 KB
37	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 mapPartitions at Exchange.scala:48	2015/07/30 23:03:17	0.9 s	2/2	588.0 KB		410.6 KB
36	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	2015/07/30 23:03:17	0.8 s	2/2	167.5 KB		100.1 KB

在其第一个 Task 中，从本地读入数据

#### Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input	Shuffle Read	Shuffle Write	Shuffle Spill (Memory)	Shuffle Spill (Disk)
1	hadoop1:60828	0.6 s	1	0	1	83.7 KB	0.0 B	49.6 KB	0.0 B	0.0 B
2	hadoop3:39761	0.8 s	1	0	1	83.7 KB	0.0 B	50.6 KB	0.0 B	0.0 B

#### Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Input	Write Time	Shuffle Write	Errors
0	1639	0	SUCCESS	NODE_LOCAL	hadoop1	2015/07/30 23:03:17	0.5 s			83.7 KB (hadoop)	6 ms	49.6 KB	
1	1640	0	SUCCESS	NODE_LOCAL	hadoop3	2015/07/30 23:03:17	0.7 s	30 ms		83.7 KB (hadoop)	6 ms	50.6 KB	

在后面的 Task 是从内存中获取数据



#### Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input	Shuffle Read	Shuffle Write	Shuffle Spill (Memory)	Shuffle Spill (Disk)
0	hadoop2:35048	3 s	60	0	60	0.0 B	28.8 KB	0.0 B	0.0 B	0.0 B
1	hadoop1:60828	3 s	70	0	70	0.0 B	28.2 KB	0.0 B	0.0 B	0.0 B
2	hadoop3:39761	3 s	70	0	70	0.0 B	28.7 KB	0.0 B	0.0 B	0.0 B

#### Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Shuffle Read	Errors
0	2045	0	SUCCESS	PROCESS_LOCAL	hadoop3	2015/07/30 23:03:39	0.2 s			0.0 B	
2	2047	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/30 23:03:39	0.2 s			0.0 B	
1	2046	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/30 23:03:39	0.2 s			0.0 B	
3	2048	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/30 23:03:39	3 ms			0.0 B	
4	2049	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/30 23:03:39	6 ms			0.0 B	
6	2051	0	SUCCESS	PROCESS_LOCAL	hadoop3	2015/07/30 23:03:39	75 ms			1330.0 B	
5	2050	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/30 23:03:39	73 ms			1400.0 B	
7	2052	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/30 23:03:39	67 ms			1470.0 B	
8	2053	0	SUCCESS	PROCESS_LOCAL	hadoop1	2015/07/30 23:03:39	61 ms			1469.0 B	

## 4.2.5 缓存表数据

### 第一步 缓存数据

*beeline>cache table tbStock;*

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

```

| hadoop1 | hadoop2 | hadoop3 |
0: jdbc:hive2://hadoop2:10000> cache table tbstock;
+-----+
| Result |
+-----+
No rows selected (0.125 seconds)
0: jdbc:hive2://hadoop2:10000> cache table tbstock;
+-----+
| Result |
+-----+
No rows selected (0.145 seconds)
0: jdbc:hive2://hadoop2:10000> cache table tbstock;
+-----+
| Result |
+-----+
No rows selected (0.145 seconds)

```

### 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;*

```

| hadoop1 | hadoop2 | hadoop3 |
0: jdbc:hive2://hadoop2:10000> cache table tbstock;
+-----+
| Result |
+-----+
No rows selected (0.183 seconds)
0: jdbc:hive2://hadoop2:10000> select count(*) from tbstock;
+-----+
| c_0 |
+-----+
| 21154 |
+-----+
1 row selected (11.233 seconds)

```

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

hadoop2:4040/storage/						
SparkSQL::hadoop2 application UI						
Storage						
RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size in Tachyon	Size on Disk
HiveTableScan [ordernumber#253,locationid#254,dateid#255], (MetastoreRelation hive, tbstock, None), None	Memory Deserialized 1x Replicated	2	100%	780.4 KB	0.0 B	0.0 B

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 ) :

```

hadoop1 | hadoop2 | hadoop3
0: jdbc:hive2://hadoop2:10000> select count(*) from tbstock;
+-----+
| c_0    |
+-----+
| 21154  |
+-----+
1 row selected (0.343 seconds)
0: jdbc:hive2://hadoop2:10000>

```

用时 0.343 秒 , 再查看 webUI 中的 stage :

Aggregated Metrics by Executor										
Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input	Shuffle Read	Shuffle Write	Shuffle Spill (Memory)	Shuffle Spill (Disk)
0	hadoop2:35048	68 ms	1	0	1	0.0 B	51.0 B	0.0 B	0.0 B	0.0 B

Tasks											
Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Shuffle Read	Errors
0	4082	0	SUCCESS	PROCESS_LOCAL	hadoop2	2015/07/30 23:26:26	40 ms			51.0 B	

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

```



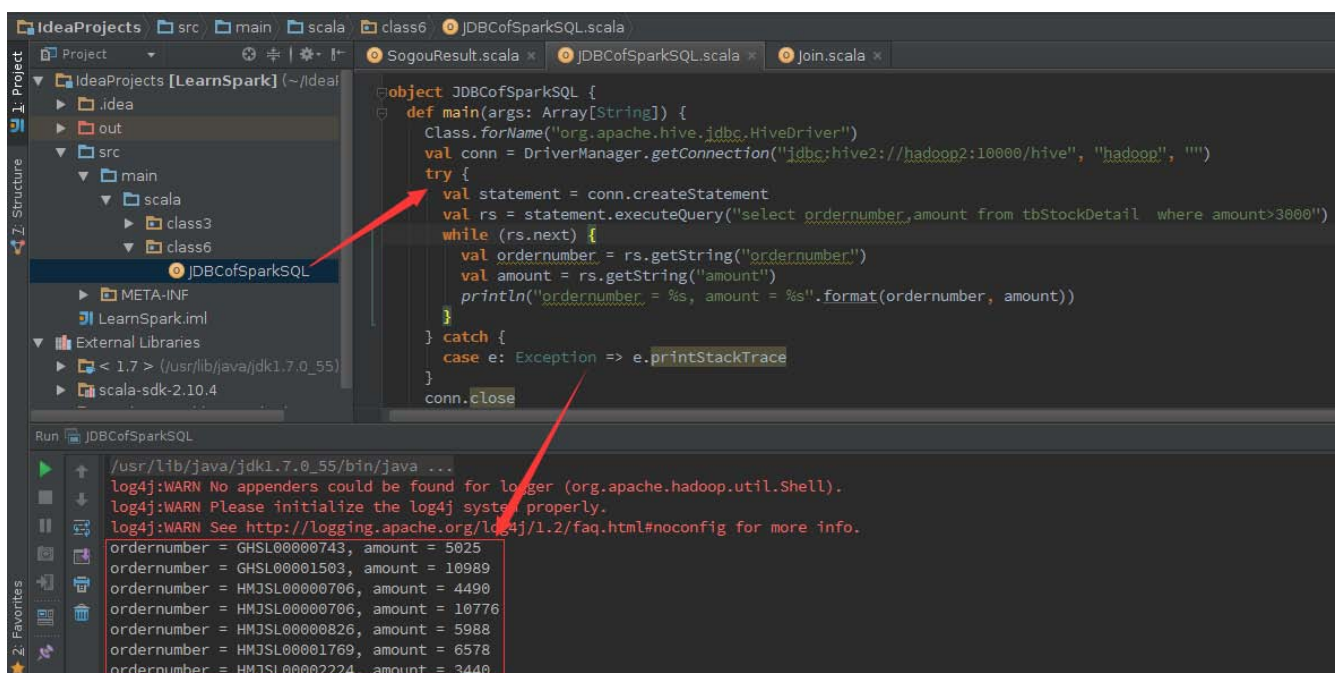
```

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



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。

Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20150731100411-0000	SparkSQL::hadoop2	3	1024.0 MB	2015/07/31 10:04:11	hadoop	RUNNING	29 min

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
6	<code>select ordernumber,amount from tbStockDetail where amount&gt;3000</code> <code>collect at SparkSQLOperationManager.scala:191</code>	2015/07/31 10:30:06	5 s	2/2	11.4 MB		

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input	Shuffle Read	Shuffle Write	Shuffle Spill (Memory)	Shuffle Spill (Disk)
0	hadoop2:43382	2 s	1	0	1	5.7 MB	0.0 B	0.0 B	0.0 B	0.0 B
2	hadoop1:46136	5 s	1	0	1	5.7 MB	0.0 B	0.0 B	0.0 B	0.0 B

Tasks

Index	ID	Attempt	Status	Locality Level	Executor	Launch Time	Duration	GC Time	Accumulators	Input	Errors
0	7	0	SUCCESS	NODE_LOCAL	hadoop1	2015/07/31 10:30:06	5 s	0.8 s		5.7 MB (hadoop)	
1	8	0	SUCCESS	NODE_LOCAL	hadoop2	2015/07/31 10:30:06	2 s	0.3 s		5.7 MB (hadoop)	

在 hadoop2 中观察 Thrift Server 运行日志如下：

