# Hive源码分析(五)-Compile原理分析

1. 简介

本文开始将展开介绍Compile和Execute。  
  
Hive的编译过程主要分为以下几步:

* Antlr定义SQL的语法规则，完成SQL词法，语法解析，将SQL转化为抽象语法树AST Tree
* 遍历AST Tree，抽象出查询的基本组成单元QueryBlock
* 遍历QueryBlock，翻译为执行操作树OperatorTree
* 逻辑层优化器进行OperatorTree变换，合并不必要的ReduceSinkOperator，减少shuffle数据量
* 遍历OperatorTree，翻译为MapReduce任务
* 物理层优化器进行MapReduce任务的变换，生成最终的执行计划

由于Hive Compile的过程较复杂, 所以我将用两篇文章介绍. 本文是第一篇介绍Hive利用Antlr进行词法分析与语法分析并生成抽象语法树AST. 第二篇将介绍Hive的语义解析也就是如何从抽象语法树转换成MapReduce.

2. Antlr与Hive

Hive使用Antlr实现SQL的词法和语法解析。Antlr是一种语言识别的工具，可以用来构造领域语言。下载Hive源码后, 我们可以发现在

org.apache.hadoop.hive.ql.parse

这个package下面, 存在几个.g文件:

* HiveLexer.g, 是做词法分析的，定义了所有用到的token
* HiveParser.g, 做语法解析的
* FromCaluseParser.g, from从句语法解析
* SelectCaluseParser.g, select 从句语法解析
* IdentifiersParser.g, 自定义函数的解析，hive中的自定义函数范围很广，各种内建的库函数，包括操作符之类的都被归为自定义的函数，而在语义解析的时候给以甄别.

在较早版本的Hive中是只存在一个Hive.g文件, 后来随着语法规则越来越复杂，由语法规则生成的Java解析类可能超过Java类文件的最大上限因此拆成了以下5个文件。

2.1 Hive Parse

Hive根据前面几个.g文件使用

antlr-3.5-complete.jar

来生成一堆解析java类以及tokens文件, 比如HiveLexer.g生成了HiveParser.java, HiveParser.g生成了HiveParser.java, 我们就可以直接使用或者继承这几个类来对SQL STRING进行解析成AST了。  
  
查看Hive Driver的源码不难发现跟词法分析和语法分析的都在ParseDriver类中进行。

**public** ASTNode parse(String command, Context ctx, String viewFullyQualifiedName)

**throws** ParseException {

**if** (***LOG***.isDebugEnabled()) {

***LOG***.debug("Parsing command: " + command);

}

HiveLexerX lexer = **new** HiveLexerX(**new** ANTLRNoCaseStringStream(command));

TokenRewriteStream tokens = **new** TokenRewriteStream(lexer);

**if** (ctx != **null**) {

**if** (viewFullyQualifiedName == **null**) {

// Top level query

ctx.setTokenRewriteStream(tokens);

} **else** {

// It is a view

ctx.addViewTokenRewriteStream(viewFullyQualifiedName, tokens);

}

lexer.setHiveConf(ctx.getConf());

}

HiveParser parser = **new** HiveParser(tokens);

**if** (ctx != **null**) {

parser.setHiveConf(ctx.getConf());

}

parser.setTreeAdaptor(***adaptor***);

HiveParser.statement\_return r = **null**;

**try** {

r = parser.statement();

} **catch** (RecognitionException e) {

e.printStackTrace();

**throw** **new** ParseException(parser.errors);

}

**if** (lexer.getErrors().size() == 0 && parser.errors.size() == 0) {

***LOG***.debug("Parse Completed");

} **else** **if** (lexer.getErrors().size() != 0) {

**throw** **new** ParseException(lexer.getErrors());

} **else** {

**throw** **new** ParseException(parser.errors);

}

ASTNode tree = (ASTNode) r.getTree();

tree.setUnknownTokenBoundaries();

**return** tree;

}

其中HiveLexerX继承了Antlr的HiveLexer(包装了错误信息), ANTLRNoCaseStringStream继承了Antlr的ANTLRStringStream(消除了大小写敏感), ASTNode继承了Antlr的Commontree(主要聚合了ASTNodeOrigin以求能获得对象类型，名字，定义，别名，和定义)。

parser.statement()

才是重点, 它调用了词法解析和语法解析, 当然这些代码都是通过Antlr生成的, 我就不看下去了。

2.2 AntlrWork介绍

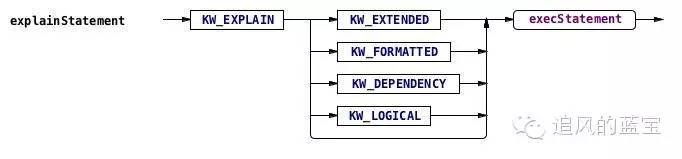
AntlrWork是专门为Antlr开发的IDE, 可以通过它来对Antlr进行调试以及可视化等功能。关于AntlrWork的使用教程请看《技术乱弹之hand in hand with antlr》。  
  
比如我导入HiveParser.g文件, 就可以点击某个rule通过它的可视化功能来查看这个。  
  
比如以下一个规则:

explainStatement

: KW\_EXPLAIN (explainOptions=KW\_EXTENDED|explainOptions=KW\_FORMATTED|explainOptions=KW\_DEPENDENCY|explainOptions=KW\_LOGICAL)? execStatement

-> ^(TOK\_EXPLAIN execStatement $explainOptions?)

;

它的可视化效果是:  
  
  
  
从图中的流程就可以看出这个语句包含三部分:

KW\_EXPLAIN

(必须有),

KW\_EXTENDED,

KW\_FORMATTED,

KW\_DEPENDENCY,

KW\_LOGICAL

(4个中可以选一个或者都省略)

execStatement

（另一个rule,必须有）

所以可见rule里面可以调用其他的rule。   
rule的基本形式如下:

rulename

: alternative\_1

| alternative\_2

...

| alternative\_n

;

这种形式叫产生式，冒号左边的是左部，作为代表这个产生式规则的符号。 冒号右边是右部， 连接符号

|

表示右部的组成部分是或者的关系。 关于Antlr的语法请看《ANTLR笔记2 - 简单语法说明》基本包含了Hive中用到的几个语法了。

3. 词法分析

所谓词法分析其实就是研究无意义的字母如何组成有意义的单词. 比如hello, my name is ryan, i come from china. 词法分析是识别出一个一个的单词 比如hello， my等。 单个的 h，e ,l,l,o 是不具备概念意义的，只有有序组成hello的时候才具备一个概念意义。  
  
HiveLexer其实做的工作就是将一串字符串流解析成一个个HiveLexer.g定义的tokens。  
  
我们首先来简单看下HiveLexer.g文件, 由于文件较大, 不全部贴了。 它有以下几部分组成:

lexer grammar HiveLexer;

*//标准格式 lexer grammar 后面跟语法名。 需要注意的是 语法名需要与文件名一致。 lexer表示该文件用于词法解析*

*//当需要向类的顶端加入代码时可以用@header定义。一般情况下（如java）用@header来加入包名和导入imports其它java类。*

@lexer::header {

package org.apache.hadoop.hive.ql.parse;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hive.conf.HiveConf;

}*//这里面给出的代码将放入到生成的语法分析器类中，作为分析器类的成员属性、方法等。*

@lexer::members {

private Configuration hiveConf;

public void setHiveConf(Configuration hiveConf) {

this .hiveConf = hiveConf;

}

protected boolean allowQuotedId() {

String supportedQIds = HiveConf.getVar(hiveConf, HiveConf.ConfVars.HIVE\_QUOTEDID\_SUPPORT);

return !"none".equals(supportedQIds);

}

}

tokens定义, 由于太多只举例部分

KW\_TRUE : &#39;TRUE&#39;;KW\_FALSE : &#39;FALSE&#39;;KW\_ALL : &#39;ALL&#39;; KW\_NONE: &#39;NONE&#39;;KW\_AND : &#39;AND&#39;;KW\_OR : &#39;OR&#39;;KW\_NOT : &#39;NOT&#39; | &#39;!&#39;;KW\_LIKE : &#39;LIKE&#39;;...fragmentLetter

: &#39;a&#39;..&#39;z&#39; | &#39;A&#39;..&#39;Z&#39;

;fragmentHexDigit

: &#39;a&#39;..&#39;f&#39; | &#39;A&#39;..&#39;F&#39;

;fragmentDigit

:

&#39;0&#39;..&#39;9&#39;

;

fragment 用于指示 Antlr这些记号只是一个记号片段，其主要作用在于构造其他记号，被其他记号调用，词法分析器 并不会把它们当成一个完整记号向外传递。也就是说fragment相当于private,只能在本.g文件中被调用。

4. 语法分析

什么是语法分析呢, 比如”lamborryan says hello”, 它的结构就是主语谓语宾语。  
  
如果我定义了一条规则即主语谓语宾语, 主语是名词, 谓语是动词, 宾语是名词或者代词组成. 那么”lamborryan says hello”无疑是符合这条规则的。  
  
在Antlr中我们定义很多规则, 就是用来做匹配各种语法的。 语法分析(Parse)的语法跟词法分析的语法(Lexer)是同一套的, 但是还是略有不同, 比如Lexer的rule是首字母大写, 而Parse的rule首字母是小写, Parse是有起始符号的(类似于入口函数),而Lexer没有。  
  
现在让我们来看HiveParser.g是怎么写的.

parser grammar HiveParser;

*//标准格式 parser grammar 后面跟语法名。 表示这是个语法解析器*

options{

tokenVocab=HiveLexer;

*//词汇表来源于*

*HiveLexer*output=AST;

*//输出 抽象语法树*

ASTLabelType=CommonTree;

*//抽象语法树类型为*

*commonTtree*backtrack= false ;

*//不回溯*

k=3;

*//前向窥看3个token的长度。*

}

*//由于生成的java解析文件太大, 所以这里将.g文件拆分成以下几个,*

*//文件的含义在前文已经讲过, 这里将这几个语法解析器的rule引入进来。*

import SelectClauseParser , FromClauseParser, IdentifiersParser;

tokens {…}，定义全局的符号表。 在AST中它们往往用作节点。

tokens {TOK\_INSERT;TOK\_QUERY;TOK\_SELECT;TOK\_SELECTDI;TOK\_SELEXPR;TOK\_FROM;TOK\_TAB;TOK\_PARTSPEC;TOK\_PARTVAL;TOK\_DIR;...}

@header 和 @memvbers 同 lexer类似, 这里就不介绍了。  
  
列举几个Parser的rule

*// starting rule*statement

: explainStatement EOF

| execStatement EOF

;explainStatement @init { msgs.push("explain statement"); } @after { msgs.pop(); }

: KW\_EXPLAIN (explainOptions=KW\_EXTENDED|explainOptions=KW\_FORMATTED|explainOptions=KW\_DEPENDENCY|explainOptions=KW\_LOGICAL)? execStatement

-> ^(TOK\_EXPLAIN execStatement $explainOptions?)

;execStatement

: queryStatementExpression

| loadStatement

| exportStatement

| importStatement

| ddlStatement

;

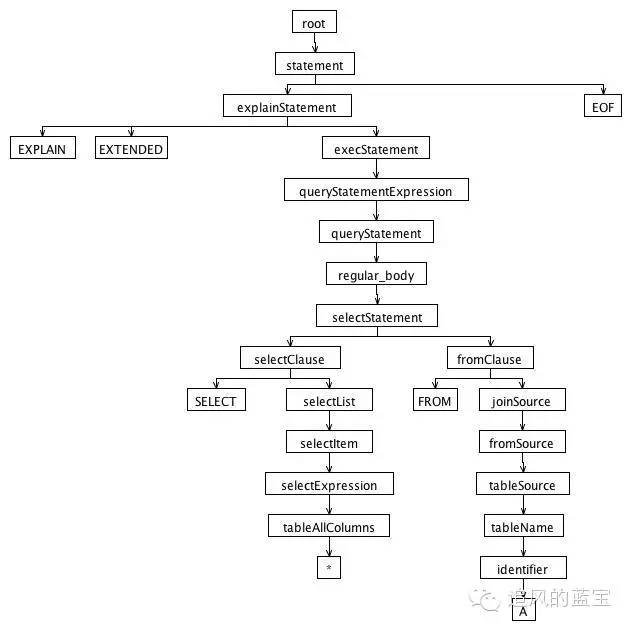
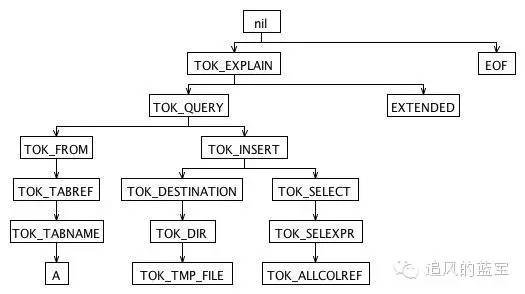
statement是Parse的入口, 也就是说语法解析都是从这里开始的，它有两个子rule explainStatement 和 execStatement, 当然这两个子rule里面还会调用其他的rule, 类似树状, 由此一层一层进行分析.  
  
简单介绍下explainStatement。  
  
explainStatement由KW\_EXPLAIN开始,中间有可选项KW\_EXTENDED, KW\_FORMATTED, KW\_DEPENDENCY, KW\_LOGICAL, 后面紧跟着执行语句。KW\_开始的token代表关键字。  
  
语法形式:

* @init表示进入规则时执行后面的{}里的动作.例中,压入trace的消息。
* @after{} 表示规则完成后执行{}里面的动作.例中,弹出trace的消息。
* ->构建语法抽象树 ^(rootnode leafnode1 leafnode2…) 如例 表示构建一个以 TOK\_EXPLAIN 为根节点 execStatement 为第一个叶结点， 可选项为第二个叶结点，如果有可选项的话。如果没有^符号的话就意味着该节点将以字符串的形式而不是后续以节点的形式展开。
* explainOptions=KW\_EXTENDED 定义了 explainOptions作为别名引用KW\_EXTENDED， 引用形式为

$explainOptions

我们通过以下一个例子来理解以上的语法: 

EXPLAIN EXTENDED SELECT \* FROM A;

它的语法解析树是  
  
  
  
它的抽象语法树如下  
  
  
  
语法解析树是按照HiveParser.g定义的rule调用顺序一层层建立的, 抽象语法树是按照rule的

->

之后的设定转换成的, 比如

^(TOK\_EXPLAIN execStatement $explainOptions?)

. 经过AST后, 树的深度大大减小了。  
  
通过AntlrWork的调试我们很容易就能理解Hive的AST, 所以建议好好利用这个工具。

在用AntlrWork调试hive sql时候, 需要将HiveLexer.g和HiveParser.g等文件中的@header,@member等跟hive其他源码有关的部分删掉。 请注意Hive的5个.g文件使用顺序, 需要编译HiveParser.g就必须先编译其他几个.g文件并生成相应的tokens文件后才能用。

5. 总结

本文主要介绍了hive是如何使用antlr来生成抽象语法树AST, 尤其强调了使用AntlrWork调试来帮助深入理解这个过程。  
  
最后用一个比较复杂的sql的解析过程来结束本章。

CREATE TABLE xml\_stat.fact\_userSTORED AS INPUTFORMAT &#39;com.hadoop.mapred.DeprecatedLzoTextInputFormat&#39;OUTPUTFORMAT &#39;org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat&#39; AS SELECT

du.user\_id,

du.user\_name,

du.register\_time,

du.gender,

CASE WHEN fu.first\_deal\_city IS NOT NULL THEN fu.first\_deal\_city WHEN fu.first\_deal\_city IS NULL THEN du.city\_code END AS city\_code FROM xml\_stat.dim\_user du LEFT OUTER JOIN xml\_stat.fact\_user\_first\_deal fu ON du.user\_id == fu.user\_id LEFT OUTER JOIN (

SELECT

t.customid AS user\_id,

t.dealid AS last\_deal\_id,

t.created AS last\_deal\_time

FROM (

SELECT customid,dealid,created, ROW\_NUMBER() OVER(

DISTRIBUTE BY customid

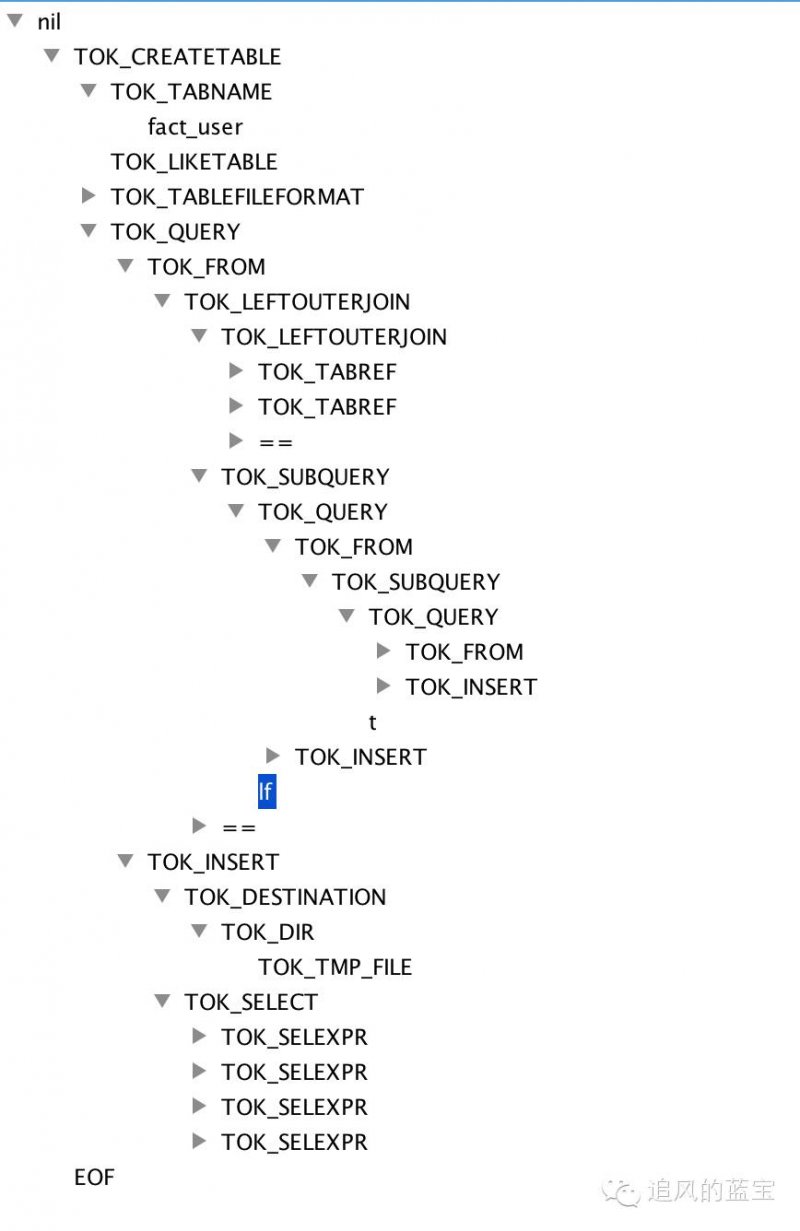
SORT BY created DESC ) rownumber

FROM xml\_stat.deal\_current

WHERE status = 4

) t

WHERE t.rownumber = 1) lf ON du.user\_id == lf.user\_id

AST(由于展开太大,所以只能看一部分了):  
  
  
对照sql查看, 很容易看懂。 本文对Hive 的AST介绍就先到这里了, 涉及到Anstr的语法请查看文档。 由于Hive SQL的语法实在太多了, 所以关于其中的细节只能等以后用到了再看了。

参考文献

* Hive SQL的编译过程
* 技术乱弹之hive源码之词法语法解析
* Compiler - lexical analysis
* 使用 Antlr 开发领域语言
* 技术乱弹之hand in hand with antlr