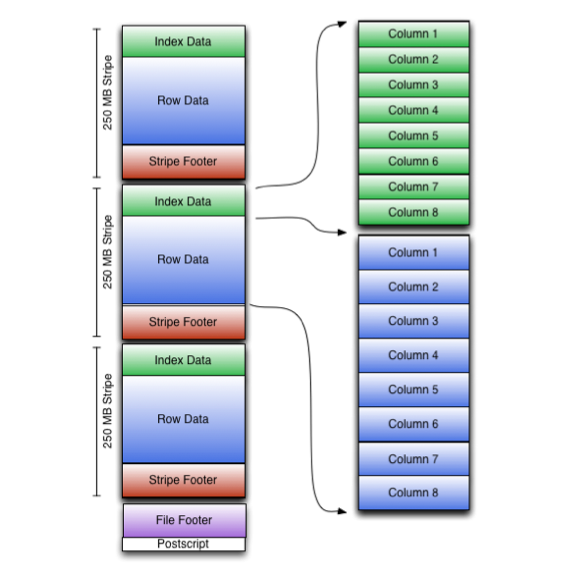
Orcfile

Orcfile && 一些优点

<http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.0.2/ds_Hive/orcfile.html#ORCFiles-ORCFileFormat>

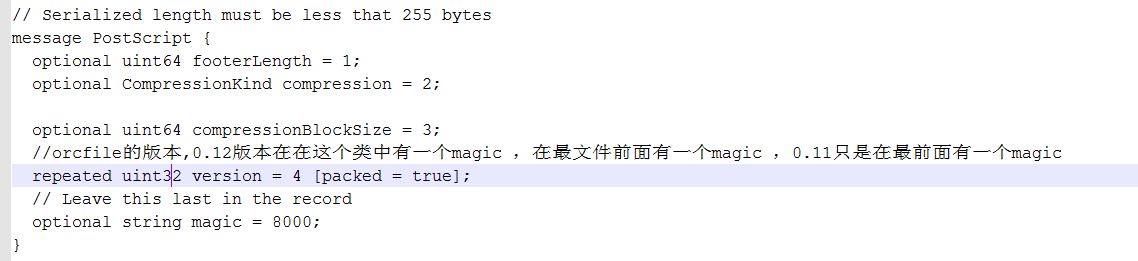
Orcfile存储格式



各部分结构

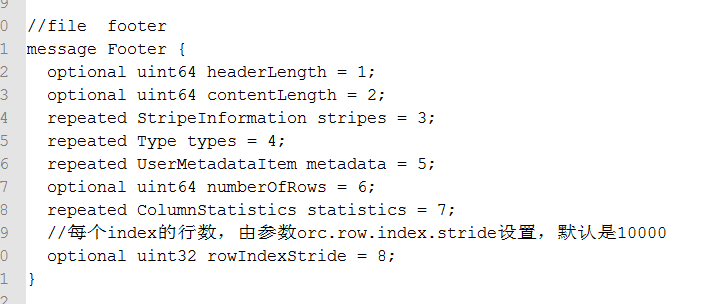
整个文件部分：

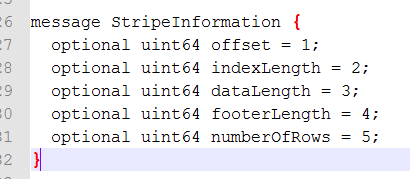
* 最后一个字节是postscript的长度，也决定了post的长度不超过255（unsigned byte）
* Postscript数据类型



**主要信息 ：file footer的长度、文件压缩类型、orcfile版本以及magic等信息 。**

* File footer ，整个文件的概述



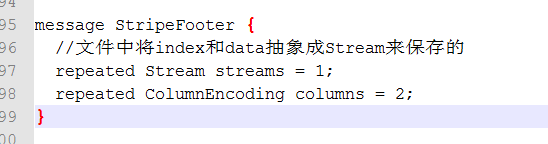


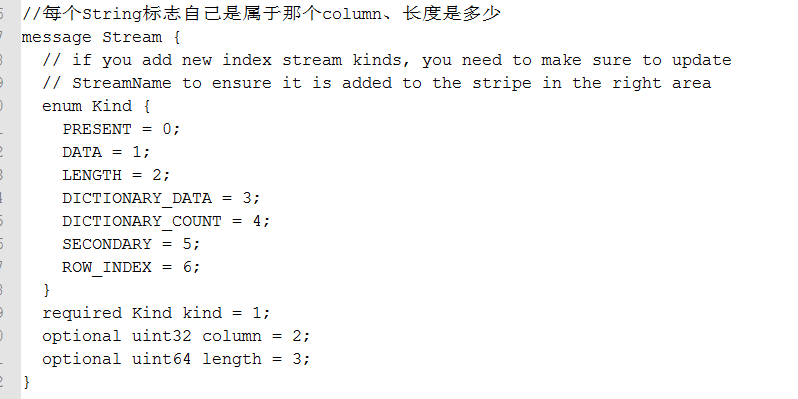
主要信息：一些orcfile的参数、orcfile文件的column类型、key-value个人信息、**Stripe信息以及每个stripe的各个数据部分数据长度信息..**

* 待补充

每个stripe部分结构

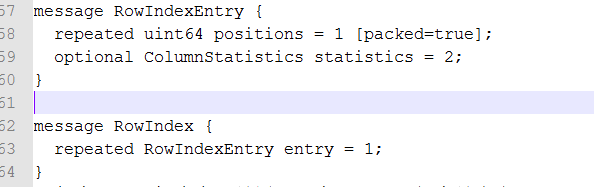
* Stripe脚（StripeFooter）

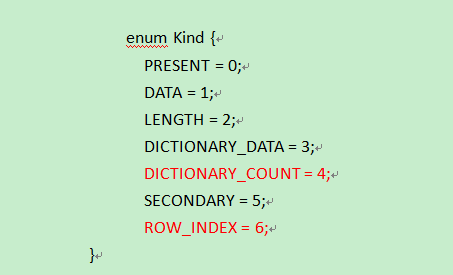




**这个stripe中的stream的概要信息**

**RowIndex，同样采用protobuf的格式来存储**

**  
data信息，采用自己编码的方式来存储 ，data的分类**

****

**红色的为index的信息，黑色的都为data中的不同类型stream**

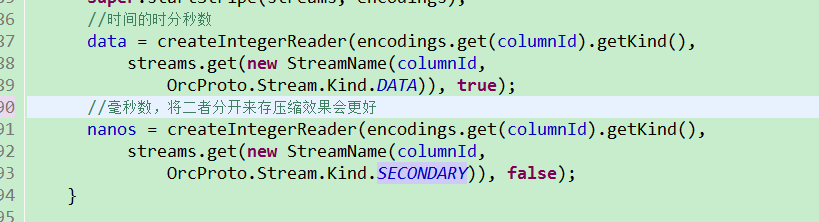
**Present：bit数组，针对null值的优化**

**Data：具体的data信息**

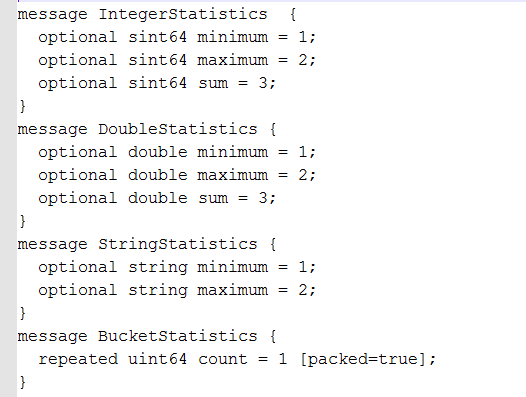
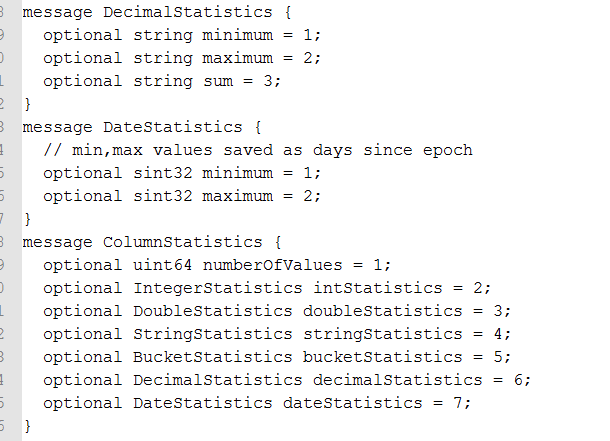
**Length：字符串的长度**

**dictionary\_data：字典编码的字典**

**secondary：针对于timestamp的优化**

****

* Index的统计信息

* 待补充

orcfile的读写

orcfile的写

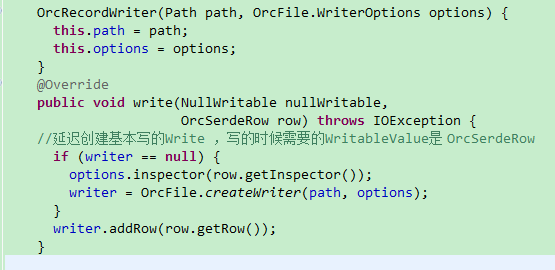
* 创建write方法



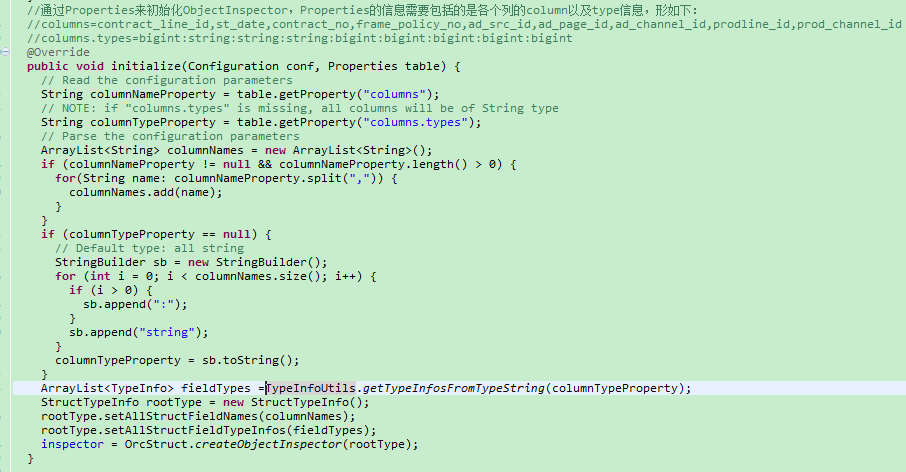
**Orcfile的配置项：**

[**http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.0.2/ds\_Hive/orcfile.html#ORCFiles-HiveQLSyntax**](http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.0.2/ds_Hive/orcfile.html#ORCFiles-HiveQLSyntax)

**最终写key-value数据的wrieter，和rcfile、squence等file一样key为null**



* 关于OrcSerdeRow，通过OrcSerde的initialize方法，或者自己创建：

****

**private** Object realRow;

**private** ObjectInspector inspector;

两个变量，realRow真实的一行，是一个writable[]类型，里面存储的是各个列的值 。

Inspector是一个OrcStructInspector类型，里面存储这每个filed的ObjectInspector（hive中的一个概念，我现在的理解是将数据交换用的类型转换为Java的基本类型的一个东东，将存储和计算中的数值解耦开，个人可以自定义以适应不同的存储类型，例如writable、avro等类型）

realRow 是 OrcStruct类型，其 **private** Object[] fields 存储的是writable类型的数据。

* 关于writer：实现的类为WriterImpl，主要的是创建TreeWriter，负责写各个列的是数据，与其对应的是在读的时候有个TreeReader，负责具体的各个列数据的读取

****

* 关于orcfile的TreeWriter：具体负责写各个列的writer，是一棵树的结构（以下以table中都是基本类型数据为类），基本的列类型为：int:string:timestampe:short:date:double



**IntegerTreeWriter:**

byte、short、int、long均转换为该类型的write，最底层的为一个RunLengthIntegerWriter(V2) stream 。

**StringTreeWriter ：**

一般分为两种情况：直接写，写每个字符串的长度，char数组；使用字典编码：字典字符串(长度+char数组)、字符在字典中的位置 。

写的时候两种流同时写，最后计算所有已写的字符串中non-null字符串 和 字典个数的比例，若是超过一点的阀值（比如说1，就是没有重复的字符串，字典大小和字符串一直）就只写写，废弃字典 。

**TimestampTreeWriter：**

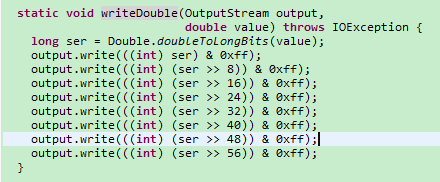
精确的时间读写，将时间分成秒数(linux时间)+好描述，两个RunLengthInteger Stream数组流，利于压缩 。

**DateTreeWriter：**

转换成linux时间的秒数，最终使用RunLengthInteger Stream来存储 。

**DoubleTreeWriter：**

没有特殊处理，直接转换成byte数组 。



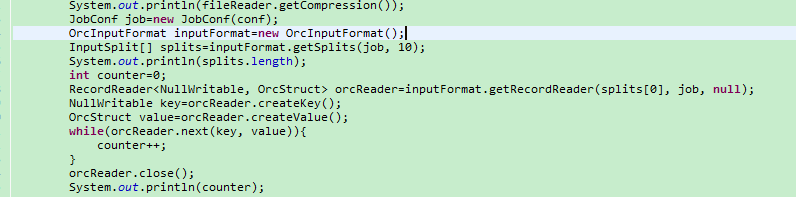
从低位到高位依次写入stream 。

Orcfile的写对于一个stripe数据是全部写入内存stream buffer 后，在fulsh硬盘上的，对于读去也是，整个stripe读取到内存buffer中的 。

其它方式待补充，感兴趣可以参考相应的类

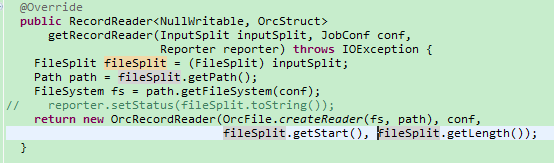
orcfile的读

* 创建基本的RecordReader<NullWritable, OrcStruct> ,然后按照hadoop的基本模型，一条条的读取具体的数据



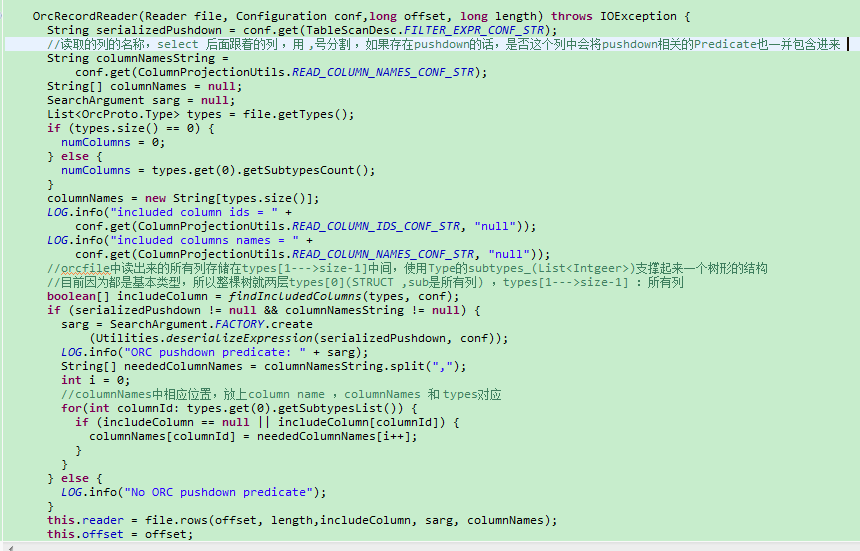
Split方法直接继承自fileInputFormat，按照文件和大小两个因素来拆分 。

关于inputFormat.getRecordReader

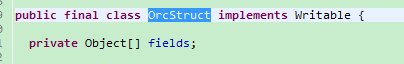


**然后OrcFile.*createReader*** ，这个方法其实就是读取一个orcfile文件的PostScript以及file footer信息：一些orcfile的参数、orcfile文件的column类型、key-value个人信息、Stripe信息以及每个stripe的各个数据部分数据长度信息，详述见上面 。无论文件一个orcfile被拆成基本split(一个map处理，有文件的offset length)，但是这部分信息都是读取的同一个orcfile的文件尾部 。

**接着OrcRecordReader的构造函数，利用conf中的hive.io.filter.expr.serialized(谓词下推的条件数) 和hive.io.file.readcolumn.ids、hive.io.file.readcolumn.names（需要的列），计算出需要读取的存储列以及需要过滤掉的行， 构造本次读取的Stream 。**

****

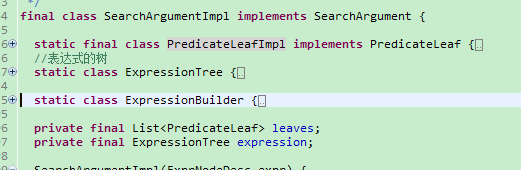
最终通过**OrcRecordReader.nex()读出来的是**OrcStruct ，

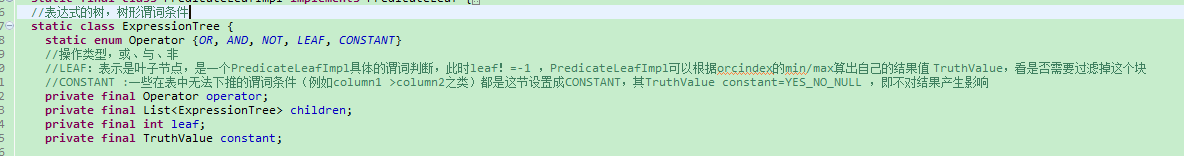


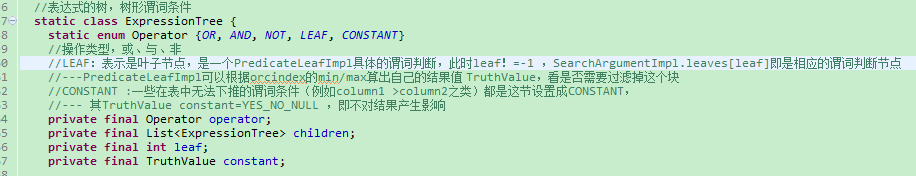
fields是所有列存储数组，实际是writable[]类型 。

**关于谓词下推pushdown ，将谓词条件数转换成orcfile的SearchArgument对象 。**

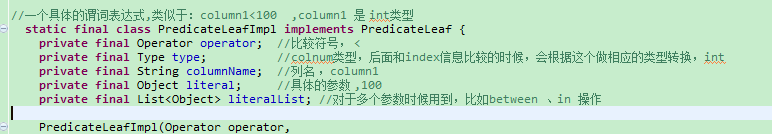
SearchArgumentImp的实现中有两个属性：







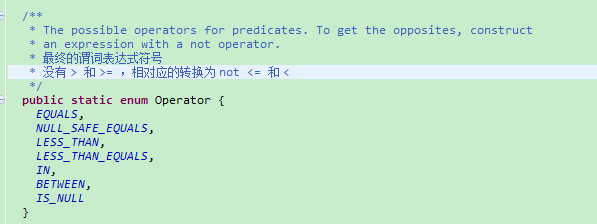
ExpressionTree类 : 一棵谓词的条件树 。



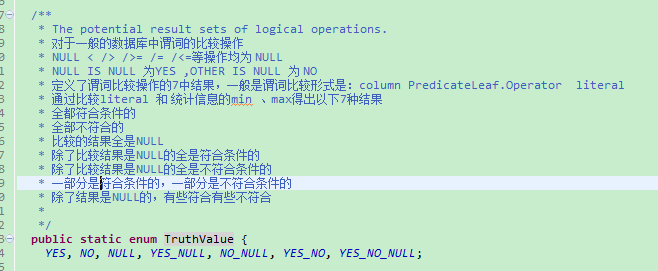
PredicateLeaf 类：一个具体的谓词判断条件 ，在后续做谓词下推的判断条件时候可以将index中的信息带入这里面，然后得出一个判断值 TruthValue 。

**两个枚举类介绍**

谓词表达式符号 ：



谓词判断逻辑结果类TruthValue ：



对于一个完整的SearchArgumentImpl ，demo如下

谓词逻辑树是：((prodline\_id>100 and st\_date>'20130101' ) or ad\_src\_id in(15 ,20 ,30) or alb\_cust\_id between 0 and 200000) and contract\_line\_id > prodline\_id ，其数据结结构如下：



RecordReaderImpl. pickRowGroups(),判断一个stripe中多少个index对应的数据段应该被过滤，得到一个boolean[]result ,若是result[i]==false ,则表示第i个小块的数据被过滤 。

RecordReaderImpl. readAllDataStreams(),不需要过滤的时候将整个stripe读取到内存buffer中，使用InStream(orc自定义的一个内存inputStream，里面有)来存储 。

RecordReaderImpl. readPartialDataStreams() ,读取过滤后的一些小块 ，通过index得到的过滤结果，判断哪些位置的数据需要读，读取出来到一个bytebuffer[]中，然后再计算每个column的一系列不连续的块的offset和endoffset ，然后封装成InStream ，主要的几个方法 ：

planReadPartialDataStreams():计算每个列的哪些位置数据应该被读出来；

mergeDiskRanges() :将连续的数据合并起来，读到一个bytebuffer中；

createStreams(streamList, chunks, bytes, included, codec, bufferSize,streams):根据上面计算的未知信息，将所有数据读出来并构造需要的InStream ；

**demo ，以下数据在一个orcfile中的数据结构(data部分和footer的streams)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id(long) | Name(string) | Sex(string) | Socre(double) | Address(string) |
| 14 | lili | male | 98.75 | Street.5 |
| 15 | xiaoming | male | 80.25 | Null |
| 16 | lucy | fmale | 85.7 | Street.7 |
| 25 | leke | male | 79 | Null |
| 27 | null | male | 83.25 | Street.9.shangdi |

Stripte-data

StripeFooter List<Stream> :index和data中的stream

|  |  |  |
| --- | --- | --- |
| Col号 | Stream类型 | Stream长度（byte[]的长度） |
| 1 | ROW\_INDEX |  |
| …… |  |  |
| 5 | ROW\_INDEX |  |
| 1 | DATA(long，以run-longth存储) |  |
| 2 | PRESENT |  |
| 2 | LENGTH（long,每列字符串的char个数） |  |
| 2 | DATA（一个个的char） |  |
| 3 | LENGTH(long，字段的每个单词包含的char) |  |
| 3 | DICTIONARY\_DATA（字典的char集，和上面一个构成字典） |  |
| 3 | DATA(long,一个field在字典中的未知) |  |
| 4 | DATA(double，以64位定长编码double) |  |
| 5 | PRESENT |  |
| 5 | LENGTH(long,每列字符串的char个数) |  |
| 5 | DATA（一个个的char） |  |

|  |
| --- |
| Data |
|  |
|  |
|  |
| 14 15 16 25 27 |
| 11110 |
| 4 8 4 4 |
| Lilixiaominglucyleke |
| 3 4 |
| malefmale |
| 0 0 1 0 0 |
| 98.75 80.25 85.7 79 83.25 |
| 10101 |
| 7 7 15 |
| Street.5Street.7Street.9.shangdi |

测试数据结果，在总共4200万行的数据中过滤选择，数据大小在3G左右

测试分为两种：

一：按照某一列排序，然后按照这列的条件来做过滤，这样就会出现的是从某一个index开始后的所有数据被连续读出来，避免不断的随机寻址

二 ：修改部分orc谓词下推的代码，在所有的index中随机间隔的选出一个作为命中块，这样会出现数据过滤了一下，但是会出现多次磁盘寻址在读取（有的可能是读取很小的数据段）的情况，具体测试如下：

|  |  |  |
| --- | --- | --- |
| Row num | Cost(s) | Read buffer count |
| 顺序读数据 | | |
| 42064498 | 117 | 47 |
| 40634498 | 112 | 46 |
| 38559498 | 106 | 43 |
| 36024498 | 103 | 41 |
| 31924498 | 88 | 36 |
| 27399498 | 75 | 31 |
| 22034498 | 63 | 25 |
| 14359498 | 41 | 16 |
| 12209498 | 35 | 14 |
| 随机读数据 | | |
| 42064498 | 118 | 47 |
| 37929498 | 122 | 171 |
| 33454498 | 121 | 401 |
| 29215000 | 122 | 637 |
| 25189498 | 116 | 877 |
| 21449498 | 110 | 1090 |
| 16725000 | 96 | 1291 |
| 13030000 | 84 | 1382 |
| 8354498 | 66 | 1411 |
| 4130000 | 45 | 1303 |

后续使用OrcFile的一些改进想法

1. 在保证计算性能的前提下，扩展orcfile的统计信息，用作**table数据统计画像**和**改进谓词下推过滤**。
2. 在扩展统计信息的基础上改进谓词下推的算法.
3. 可以参考infobright的<<Infobright\_–\_Analytic\_Database\_Engine>>和<Brighthouse\_An\_Analytic\_Data\_Warehouse\_for\_Ad-hoc\_Queries>论文以及infobright的一些查询优化 .
4. 针对hdfs的block大小，来控制stripe文件大小，以尽量保证local .
5. 待补充.