HCatalog简介

1. 一般的hdfs读写

传统的对于hdfs的读写都是直接设置inputPath 和 outPath ，而且对于数据都是以文件的形式访问的，不涉及到结构化/半结构化的东东，及时如hive存储在hdfs的的结构化数据，外部系统访问也只能自己去了解具体的结构是如何存储的，然后自己读文件再访问，传统访问hdfs的方式如下：

使用InputFormat、Split、RecordReader用于读,具体方式如下：

**List<InputSplit> InputFormat.getSplits() ;**

**RecordReader InputFormat. createRecordReader(InputSplit split) ;**

**RecordReader. nextKeyValue();**

**RecordReader. getCurrentValue();**

这样的流程来访问 。

使用OutputFormat、RecordWriter来写入hdfs数据，具体如下：

**OutputFormat. getRecordWriter();**

**RecordWriter.write(K key, V value)**

对于一般的table结构数据，都是将Key值置成NullWritable，作为一个无用的值来存储。

1. Hcatalog与hdfs的结合：定义了文件的schema

HCatalog将每份结构化的hdfs数据定义schema和访问信息（db、table、partition），然后读和写的时候使用db、table、partition(对于无partition这个可以为空)这三部分信息来访问相应的表数据，屏蔽掉表底层InputFormat、OutFormat以及path信息，读&&写时候只许关系以下几个访问类即可：

**HCatInputFormat<WritableComparable, HCatRecord>**

**HCatRecordReader<WritableComparable, HCatRecord>**

**HCatOutputFormat<WritableComparable<?>, HCatRecord>**

**FileRecordWriterContainer<WritableComparable<?>, HCatRecord>**

几个类就行了，读写的接口Value均是HCatRecord对象，Key值是WritableComparable即可，不过一般key值都是NullWritable，并无实际用途 。

1. 对象HCatRecord

内部存储这Object[]数组，分别存储一行数据中每列的值，为hive基本类型的Java对象（int、String、list、Map等），对于复杂结构可以嵌套，主要访问接口如下 ：

**List<Object> getAll();**

**Object get(int fieldIndex);**

**Object get(String fieldName, HCatSchema recordSchema)**

通过列名称来访问具体列的值，但是需要table的schema信息HCatSchema（具体每列的colName以及每个colName对应的fieldIndex）

这点设计不是很好，感觉可以将HCatSchema内置到HCatRecord 中，这样封装性可能会更好一些 、

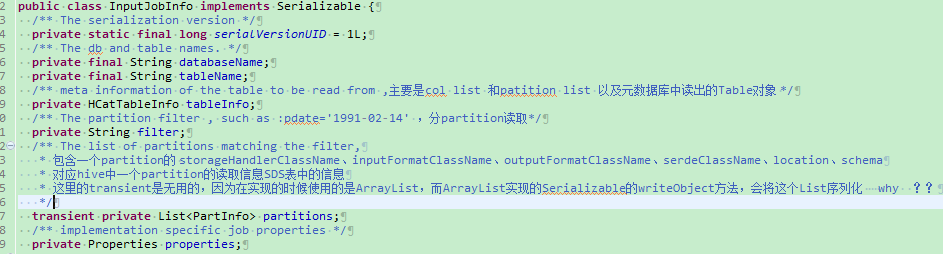
1. HCatalog的元数据存储

HCatalog存储元数据和Hive的元数据是一致的，对于元数据的存储底层也是调用hive底层HiveMetaStoreClient的接口（我们常用的Hive其实是在这个类上封装了一层），其提供对于元数据库的各种操作（creat drop alter load insert等等 ）

直接调用Hive接口/hive的客户端 写入的元数据在HCatalog中可以直接用，同样HCatalog写入的元数据以及数据在hive中也是可以直接使用的。

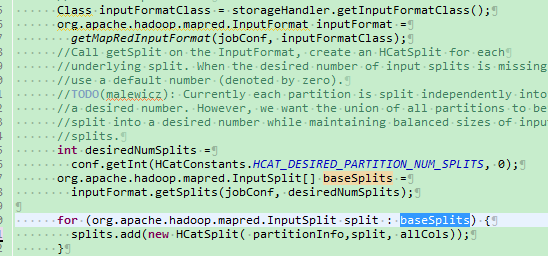
1. HCatInputFormat:读取HCatalog中数据的接口

读取的时候需要dbName、tableName、partition(这个可以没有)、tableProperties(可以没有)信息，存储于InputJobInfo中，InputJobInfo是读取时候的信息存储封装，在jobclient端初始化以后，使用Java的序列化Serializable转换成byte[],然后将byte[]编码成一个字符串存储(参见HCatUtil的encodeBytes和decodeBytes方法)到hadoop的配置Configuration来在整个mr任务中传递。

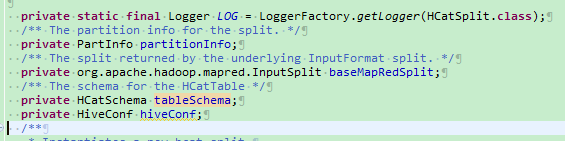


InputJobInfo初始化创建的时候只需要dbName、tableName、partition(这个可以没有)、tableProperties(可以没有)信息，在InitializeInput.getInputJobInfo方法中去连接元数据库，补齐InputJobInfo中的HCatTableInfo、partitions信息，对于不是分区的table，构造一个partition信息，将相应数据存储在里面。

**getSplits(JobContext jobContext)**方法，从conf配置中反序列化出InputJobInfo信息，然后根据partition中的信息，利用inputFormatClass、location来split成真正的底层数据的split，在封装成HCatSplit。

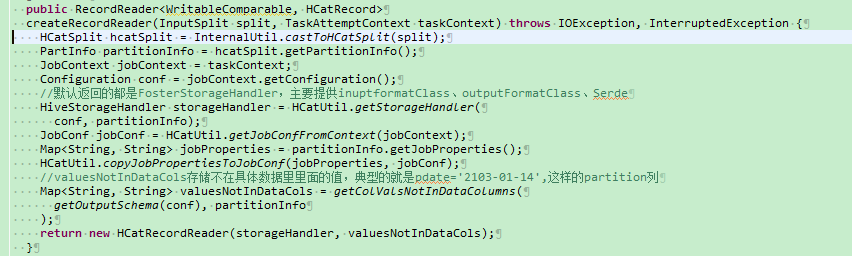


HCatSplit结构

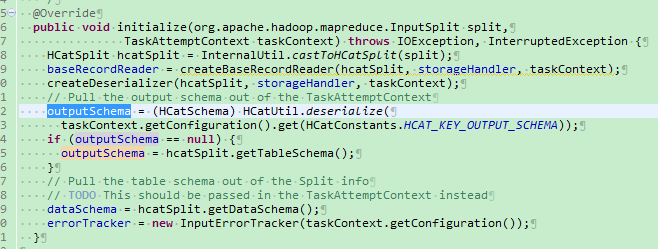


最终读取的时候还是要利用PartInfo中的SerDe来读取相应的数据，后面介绍

**createRecordReader(InputSplit split, TaskAttemptContext taskContext)方法**



返回的是封装了一层的HCatRecordReader对象，然后通过HCatRecordReader. Initialize(split,taskContext)来初始化具体的读信息baseRecordReader等 。



一切初始化完毕，就可以开始调用经典的nextKeyValue的方法了

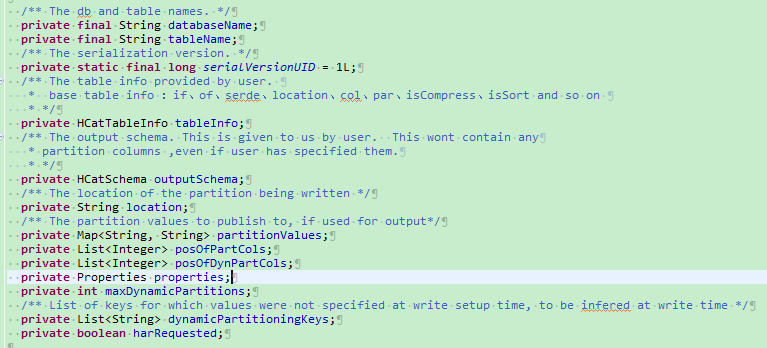


baseRecordReader是底层的RecordReader ，读出来数据后使用相应的SerDe的deserialize方法反序列化成ObjectInspector(一般是StructObjectInspector)可以访问的数据格式，然后通过相应的colnum ObjectInspector得到每列的具体Java object值，然后封装到HCatRecord对象中 。

对于hive基本类型转换成Java的基本类型，对于struct、list转换成Java的List，map转换成Java的map。

1. Hcatalog在MR程序中的输出类HCatOutputFormat

和input对应的，OutputJobInfo是存储用于输出数据时候的信息的，其结构如下：

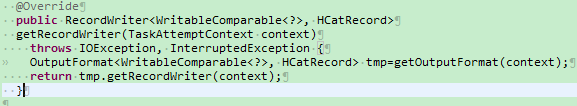


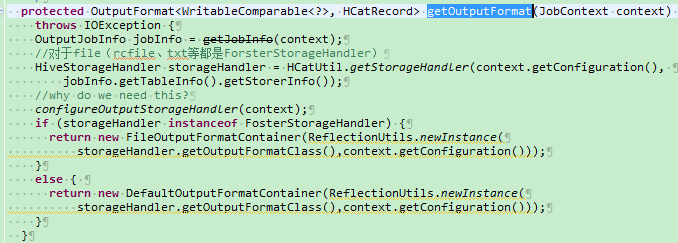
主要是table的相关信息

**OutputJobInfo outPutInfo=OutputJobInfo.create("analyse\_db","contline\_revenue\_day\_cut", partitions);**

也是需要dbName、tblName、partition就可以初始化，通过HCatOutputFormat.setOutput(conf, credentials,outPutInfo)中连接metastore，然后补充其余的schema、Serde、outputFormat等一系列信息 ，目前Hcata中对于hive的自建Index、内置压缩、分桶和sort的特性还是不支持的，然后同样在利用序列化这个对象成String，存储与MR的配置conf中 ，提供给以后的程序共享使用。

**getRecordWriter(TaskAttemptContext context)方法**

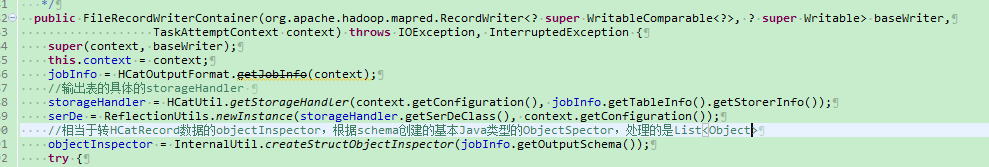
****

****

先得到一个封装后的OutputFormatContainer，这个其实就是内部包装了一个这个table数据真正的OutputFormat对象，以FileOutputFormatContainer为例，以下是FileOutputFormatContainer. getRecordWriter(TaskAttemptContext context)方法



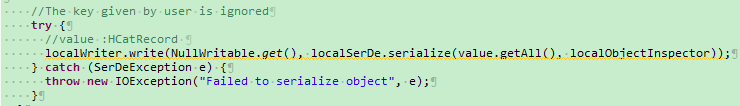
主要是做了一些MR程序中输出文件夹初始化的工作，然后就是返回了一个FileRecordWriterContainer对象，该对象和上面的OutputFormat一样，内部封装了一个table实际存储的OutputFormat返回的RecordWriter baseWriter ,最终真正用于向hdfs写数据的对象，FileRecordWriterContainer如下：



其中初始化了几个变量

serDe ：用于序列化写到hdfs的数据

objectInspector ：用于访问HCatRecord的 。



这里面还是利用了hive中的这一套Serde和ObjectInspector的封装来做的 。

1. 单机关于读/写的两个Java demo



1. 待补充