关于hive中的~~SerDe~~ AbstractSerDe 和 ObjectInspector

1. 继承关系

AbstractSerDe是继承了接口SerDe 的abstarct类，SerDe是继承了Deserializer, Serializer接口的，新的hive使用AbstractSerDe来代替接口，将序列化和反序列化合到一起 。

1. AbstractSerDe的方法

initialize(Configuration conf, Properties tbl)

一般是根据conf和tbl中的key-value配置来初始化一个序列化/反序列化对象，一般的操作是初始化其中的ObjectInspector inspector ,因为SerDe是处理行的，所以一般都初始化成StructObjectInspector对象 。

Class<? extends Writable> getSerializedClass()

这是继承自Serializer接口的方法，用于获取要序列化成的数据的Writable子类型的Class，应为hive是支持hdfs的，所以序列化也是序列化成hadoop可写的writable格式。

这个类是在某些RecordeWriter写的时候会用到。

但是有些实现可能会加一个writable的壳，只是为了兼容以前的接口，其实是不能直接使用调用hadoop的序列化方法的，其将数据存储在内部，然后再重写其相应的RecoderWrite来接收这些数据格式，例如OrcSerde 。

Writable serialize(Object obj, ObjectInspector objInspector)

这个方法是和上面的那个方法对应的，是将一个对象通过objInspector序列化成Writable的对象，Object obj, ObjectInspector objInspector是要序列化的对象以及对这个序列化的对象进行访问的objInspector ,对于行数据的序列化 objInspector都是StructObjectInspector对象，可以通过以下的方法来访问相应的具体列数值：

StructObjectInspector soi = (StructObjectInspector) objInspector;

List<? **extends** StructField> fields = soi.getAllStructFieldRefs();

List<Object> list = soi.getStructFieldsDataAsList(obj);

**for** (**int** i = 0; i < fields.size(); i++) {

// Append the separator if needed.

**if** (i > 0) {

serializeStream.write(serdeParams.separators[0]);

}

// Get the field objectInspector and the field object.

ObjectInspector foi = fields.get(i).getFieldObjectInspector();

Object f = (list == **null** ? **null** : list.get(i));

serializeField(serializeStream, f, foi, serdeParams);

}

然后具体的列值再根据ObjectInspector类型来取到具体的值，具体的可以参见LazySimpleSerDe. serializeField方法 。

Object deserialize(Writable blob)

ObjectInspector getObjectInspector()

将从hdfs读出来的数据(通过RecodeReader读出来的Writable blob,一般是支持hdfs可写数据的)，然后通过获得的ObjectInspector来解析内部的结构 。

1. 关于重新定义一个自己的序列化反序列化工具

如果你Ser和Dser的都是正常的writable对象，且也打算把他们写到hdfs上并读出来的话，那样直接定义自己的SerDe接口就行了，需要做的是：

**initialize**：主要是更具tbl中的列名称以及类型，来构造一StructObjectInspector以及其它的一些初始化工作，比如构造一个自定义的内部Writable对象等。

**getSerializedClass** ：返回你自己的writable类型的Class，当然也可以自己定义一个Writeable对象 。

**Writable serialize (Object obj, ObjectInspector objInspector) :**通过参数中的ObjectInspector来访问object的具体数据（具体方法如上），然后将这些数据再转化为本SerDe产生的那种Writable类型对象 。

这个方法中的obj 和 objInspector和你要实现的SerDe以及ObjInspector没有关系，其实是上一个operator传过来的数据和它自己的Objinspector

**Object deserialize :**将你自定义的Writable对象反序列化成Object的过程，需要自定义 。

**ObjectInspector getObjectInspector() :** 返回上面所创建的StructObjectInspector对象 。

一般来说，定义这些，然后其产出的Writable对象可以被通用的InputFormat以及Outputformat识别并操作就可以了，如果产生的Writable对象的读写需要自己自定义的话，还需要自定义自己的InputFormat/OutputFormat,这样凑齐hadoop的一套读写工具 。

1. 一个demo

InputFormat inputA ;

AbstractSerDe serdeA ;

AbstractSerDe serdeB ;

OutputFormat outputB;

RecordReader readerA=inputA.createRecordReader(**null**, **null**);

RecordWriter writeB=outputB.getRecordWriter(**null**);

StructObjectInspector soiA = (StructObjectInspector) serdeA.getObjectInspector();

List<? **extends** StructField> fieldsA = soiA.getAllStructFieldRefs();

**while**(readerA.nextKeyValue()){

Object obj=readerA.getCurrentValue();

List<Object> colList = soiA.getStructFieldsDataAsList(obj);

**for** (**int** i = 0; i < fieldsA.size(); i++) {

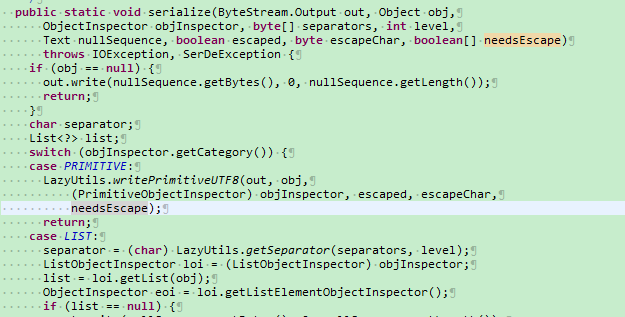
ObjectInspector foi = fieldsA.get(i).getFieldObjectInspector();

//\*\*\*\*\*\*\*\*\*\*\*\* 此处可以判断具体的ObjectInspector的具体类型，然后再轻质转换成相应类型，调用相应方法

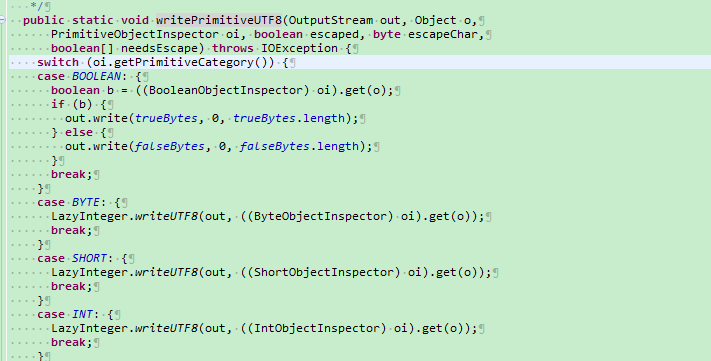
}

writeB.write(**null**, serdeA.serialize(obj, serdeA.getObjectInspector()));

}



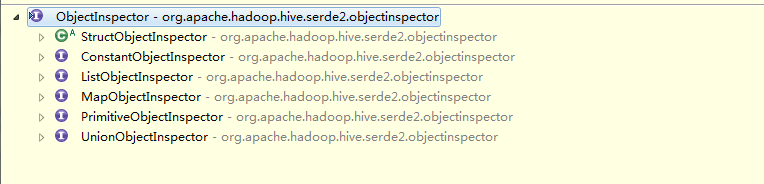
根据不同的ObjectSpector类型来选择相应的解析方式，是switch – case 的方式。



这是对于基本类型的处理，因为基本类型的Object都是继承自PrimitiveObjectInspector ，有

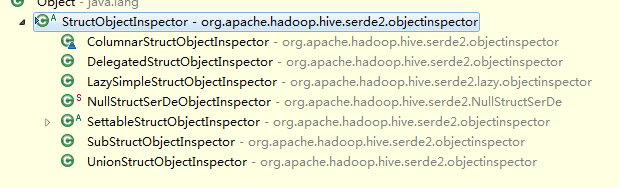
Object getPrimitiveWritableObject(Object o)、Object getPrimitiveJavaObject(Object o)两个方法，可以获得数据的Java类型对象或者相应的writable对象，然后使用者根据自己的需求来处理。参见org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

1. 关于ObjectInspector

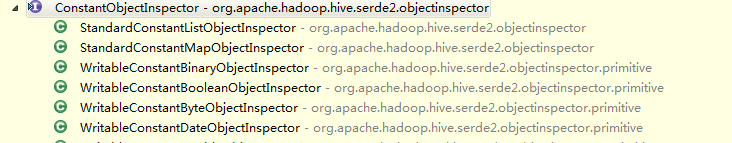


这个是ObjectInspector的继承类图，主要是

StructObjectInspector :这个主要是处理针对hive中的行row来做转换的，对于row-object的处理都是使用这个的，其下面可以获得row的具体列的方法。



ConstantObjectInspector:看代码主要是用做一个固定值的获取，估计是hive中为了将所有的列（包括去固定值得列）都封装成使用ObjectInspector来访问，它的那些基本类型的子类都是同时又继承了另外一个PrimitiveObjectInspector接口的



对于Map List union\*\*\*Spector由于使用不多，暂时未看代码 。

**PrimitiveObjectInspector**：是对于基本类型（就是hive支持的double、int、date、boolen等类型）的objectInspector的封装，使用的最多 。

其基本方法如下：

PrimitiveCategory getPrimitiveCategory() :获得类型的枚举，就是那几种基本类型

Class<?> getPrimitiveWritableClass() ：或者writable的class

Class<?> getJavaPrimitiveClass() ：同上一个，获取基本的Java对象的类型

对于这三个方法，都是在AbstractPrimitiveObjectInspector中通过构造方法注入PrimitiveTypeEntry对象提供的，每个具体的基本类型对象在构造方法中会注入相应的实例 。

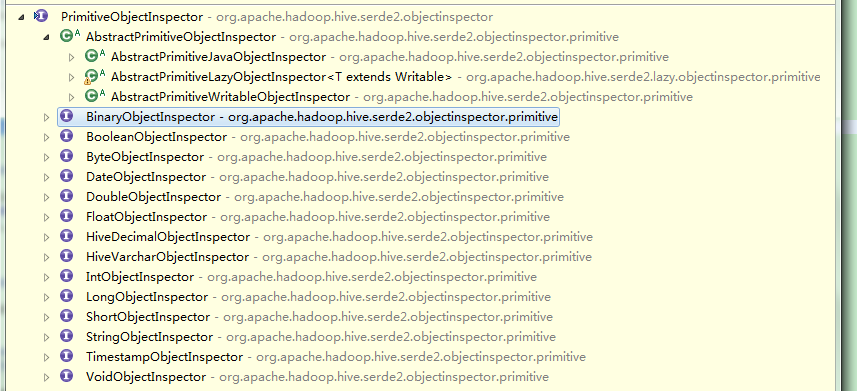
可以参见：JavaDoubleObjectInspector的构造方法 。

Object getPrimitiveWritableObject(Object o) ：一般是从struct中获得访问的对象后，调用这个方法获得基本的writable（例如DoubleWritable，TextWritable等）类型对象 。

Object getPrimitiveJavaObject(Object o) ：获得基本的Java对象

Object copyObject(Object o)：将从sturct中获得的对象copy，一般是用在writable中，其读取的对象是复用的 。

其继承关系如下：



上面那三个系类:

AbstractPrimitiveJavaObjectInspector :针对实际的row中出来的object为基本Java类型时候的转换器，对于基本类型，每个基本类型有一个实现。其对于PrimitiveObjectInspector的getPrimitiveJavaObject method返回的是本身的值 ，对于getPrimitiveWritableObject method是每个具体的实现返回通过基本类型构造的writable类型 。

AbstractPrimitiveWritableObjectInspector :针对实际row中出来的Object对象是Writable对象的，对于每个基本类型有一个具体实现 。其对于PrimitiveObjectInspector的getPrimitiveJavaObject method返回的是writable类型获得基本类型，对于getPrimitiveWritableObject method返回的是其本身值。

AbstractPrimitiveLazyObjectInspector:针对row中转换出来的是LazyPrimitive类型的值，这个类型其实是在writable对象上封装了一层（感觉主要是为了减少writable对象转换，只有当用到的时候才init，延迟加载的，可以参考下具体的LazyPrimitive系类的实现），这个类主要是LazySimpleSerDe以及相应的LazySimpleStructObjectInspector这一系列产生出来的 。其对于PrimitiveObjectInspector的getPrimitiveJavaObject method返回的是相应的writable转换成java基本类型，对于getPrimitiveWritableObject method返回的是其中保存的那个data(writable)。

1. 一些相关的网址

<http://blog.cloudera.com/blog/2012/12/how-to-use-a-serde-in-apache-hive/>

<http://db3.iteye.com/blog/1072778>

<https://cwiki.apache.org/confluence/display/Hive/SerDe>

1. 待补充