Hive

TsFile's Hive connector

TsFile's Hive connector implements support for reading external TsFile type file formats through Hive, enabling users to manipulate TsFile through Hive.

The main functions of the connector:

- Load a single TsFile file into Hive, whether the file is stored on the local file system or in HDFS
- Load all files in a specific directory into Hive, whether the files are stored in the local file system or HDFS
- Querying TsFile with HQL
- Until now, write operations are not supported in hive-connector. Therefore, insert operations in HQL are not allowed

Design principle

The Hive connector needs to be able to parse the TsFile file format and convert it into a line-by-line format that Hive can recognize. You also need to be able to format the output according to the form of a user-defined Table. Therefore, the function implementation of the Hive connector is mainly divided into four parts

- Slicing the entire TsFile file
- Read data from shards and convert it into a data type that Hive can recognize
- Parse user-defined Table
- Deserialize data into Hive's output format

Concrete implementation class

The above four main functional modules have their corresponding implementation classes. The four implementation classes are introduced below.

org.apache.iotdb.hive.TSFHiveInputFormat

This class is mainly responsible for formatting the input TsFile file. It inherits the `FileInputFormat <NullWritable, MapWritable>` class. Some general formatting operations have been implemented in `FileInputFormat`. This class overrides its The getSplits (JobConf, int) `method customizes the sharding method for TsFile files; and the` getRecordReader (InputSpli, JobConf, Reporter) method is used to generate a `TSFHiveRecordReader` that specifically reads data from a slice.

org.apache.iotdb.hive.TSFHiveRecordReader

This class is mainly responsible for reading TsFile data from a shard.

It implements the `IReaderSet` interface. This interface is a set of methods for setting internal properties of the class, mainly to extract the duplicated code sections in`TSRecordReader` and `TSHiveRecordReader`.

```
public interface IReaderSet {

  void setReader(TsFileSequenceReader reader);

  void setMeasurementIds(List<String> measurementIds);

  void setReadDeviceId(boolean isReadDeviceId);

  void setReadTime(boolean isReadTime);
}
```

Let's first introduce some important fields of this class

- private List<QueryDataSet> dataSetList = new ArrayList<>();

  All QueryDataSets generated by this shard
- private List<String> deviceIdList = new ArrayList<>();

  Device name list, this order is consistent with the order of dataSetList, that is, deviceIdList [i] is the device name of dataSetList [i].
- private int currentIndex = 0;

  The index of the QueryDataSet currently being processed

This class calls the `initialize (TSFInputSplit, Configuration, IReaderSet, List <QueryDataSet>, List <String>)` method of `TSFRecordReader` in the constructor to initialize some of the class fields mentioned above. It overrides the `next ()` method of `RecordReader` to return the data read from TsFile.

next(NullWritable, MapWritable)

We noticed that after reading the data from TsFile, it was returned in the form of `MapWritable`. Here`MapWritable` is actually a `Map ', except that its key and value are serialized and deserialized. Special adaptation, its reading process is as follows

1. First determine if there is a value for `QueryDataSet` at the current position of`dataSetList`. If there is no value, then increase `currentIndex` by 1 until the first`QueryDataSet` with a value is found
2. Then call `next ()` method of `QueryDataSet` to get`RowRecord`
3. Finally, the getCurrentValue () method of TSFRecordReader is called, and the value in RowRecord is placed in MapWritable.

org.apache.iotdb.hive.TsFileSerDe

This class inherits `AbstractSerDe` and is also necessary for us to implement Hive to read data from custom input formats.

It overrides the Initialize () method of AbstractSerDe. In this method, the corresponding device name, sensor name, and corresponding type of the sensor are parsed from the user-created table sql. An ObjectInspector object is also constructed. This object is mainly responsible for the conversion of data types. Since TsFile only supports primitive data types, when other data types occur, an exception needs to be thrown. The specific construction process can be seen in the createObjectInspectorWorker () method. .

The main responsibility of this class is to serialize and deserialize data in different file formats. As our Hive connector only supports read operations for the time being, it does not support insert operations, so only the deserialization process, so only overwrite The `deserialize (Writable)` method is called, which calls the `deserialize ()` method of `TsFileDeserializer`.

org.apache.iotdb.hive.TsFileDeserializer

This class deserializes the data into Hive's output format. There is only one `deserialize ()` method.

public Object deserialize(List<String>, List<TypeInfo>, Writable, String)

The `Writable` parameter of this method is the`MapWritable` generated by `next ()` of `TSFHiveRecordReader`.

First determine if the `Writable` parameter is of type`MapWritable`, if not, throw an exception.

Then take out the value of the sensor of the device from `MapWritable` in turn, throw an exception if a type mismatch is encountered, and finally return the generated result set.