Input and Output Interfaces

Table of contents

1 Set Up....2
2 HCatInputFormat....2
3 HCatOutputFormat....3
4 Examples....4

1 Set Up

No HCatalog-specific setup is required for the HCatInputFormat and HCatOutputFormat interfaces.

2 HCatInputFormat

The HCatInputFormat is used with MapReduce jobs to read data from HCatalog managed tables.

HCatInputFormat exposes a Hadoop 0.20 MapReduce API for reading data as if it had been published to a table.

2.1 API

The API exposed by HCatInputFormat is shown below.

To use HCatInputFormat to read data, first instantiate as `InputJobInfo` with the necessary information from the table being read and then call setInput with the `InputJobInfo`.

You can use the `setOutputSchema` method to include a projection schema, to specify specific output fields. If a schema is not specified all the columns in the table will be returned.

You can use the `getTableSchema` methods to determine the table schema for a specified input table.

```
  /**
   * Set the input to use for the Job. This queries the metadata server with
   * the specified partition predicates, gets the matching partitions, puts
   * the information in the conf object. The inputInfo object is updated with
   * information needed in the client context
   * @param job the job object
   * @param inputJobInfo the input info for table to read
   * @throws IOException the exception in communicating with the metadata server
   */
  public static void setInput(Job job,
      InputJobInfo inputJobInfo) throws IOException;

  /**
   * Set the schema for the HCatRecord data returned by HCatInputFormat.
   * @param job the job object
   * @param hcatSchema the schema to use as the consolidated schema
   */
  public static void setOutputSchema(Job job,HCatSchema hcatSchema)
    throws IOException;

  /**
   * Gets the HCatTable schema for the table specified in the HCatInputFormat.setInput call
   * on the specified job context. This information is available only after
 HCatInputFormat.setInput
```

```
   * has been called for a JobContext.
   * @param context the context
   * @return the table schema
   * @throws IOException if HCatInputFormat.setInput has not been called
   *                     for the current context
   */
  public static HCatSchema getTableSchema(JobContext context)
    throws IOException;
```

3 HCatOutputFormat

HCatOutputFormat is used with MapReduce jobs to write data to HCatalog managed tables.

HCatOutputFormat exposes a Hadoop 20 MapReduce API for writing data to a table. When a MapReduce job uses HCatOutputFormat to write output, the default OutputFormat configured for the table is used and the new partition is published to the table after the job completes.

3.1 API

The API exposed by HCatOutputFormat is shown below.

The first call on the HCatOutputFormat must be `setOutput`; any other call will throw an exception saying the output format is not initialized. The schema for the data being written out is specified by the `setSchema` method. You must call this method, providing the schema of data you are writing. If your data has same schema as table schema, you can use HCatOutputFormat.getTableSchema() to get the table schema and then pass that along to setSchema().

```
   /**
    * Set the info about the output to write for the Job. This queries the metadata server
    * to find the StorageDriver to use for the table.  Throws error if partition is
already published.
    * @param job the job object
    * @param outputJobInfo the table output info
    * @throws IOException the exception in communicating with the metadata server
    */
   @SuppressWarnings("unchecked")
   public static void setOutput(Job job, OutputJobInfo outputJobInfo) throws IOException;

   /**
    * Set the schema for the data being written out to the partition. The
    * table schema is used by default for the partition if this is not called.
    * @param job the job object
    * @param schema the schema for the data
    */
   public static void setSchema(final Job job, final HCatSchema schema) throws
IOException;

  /**
   * Gets the table schema for the table specified in the HCatOutputFormat.setOutput call
```

```
   * on the specified job context.
   * @param context the context
   * @return the table schema
   * @throws IOException if HCatOutputFromat.setOutput has not been called for the passed
 context
   */
  public static HCatSchema getTableSchema(JobContext context) throws IOException;
```

4 Examples

Running MapReduce with HCatalog

Your MapReduce program will need to know where the thrift server to connect to is. The easiest way to do this is pass it as an argument to your Java program. You will need to pass the Hive and HCatalog jars MapReduce as well, via the -libjars argument.

```
export HADOOP_HOME=<path_to_hadoop_install>
export HCAT_HOME=<path_to_hcat_install>
export LIB_JARS=$HCAT_HOME/share/hcatalog/hcatalog-0.4.0.jar,
$HIVE_HOME/lib/hive-metastore-0.9.0.jar,
$HIVE_HOME/lib/libthrift-0.7.0.jar,
$HIVE_HOME/lib/hive-exec-0.9.0.jar,
$HIVE_HOME/lib/libfb303-0.7.0.jar,
$HIVE_HOME/lib/jdo2-api-2.3-ec.jar,
$HIVE_HOME/lib/slf4j-api-1.6.1.jar

export HADOOP_CLASSPATH=$HCAT_HOME/share/hcatalog/hcatalog-0.4.0.jar:
$HIVE_HOME/lib/hive-metastore-0.9.0.jar:
$HIVE_HOME/lib/libthrift-0.7.0.jar:
$HIVE_HOME/lib/hive-exec-0.9.0.jar:
$HIVE_HOME/lib/libfb303-0.7.0.jar:
$HIVE_HOME/lib/jdo2-api-2.3-ec.jar:
$HIVE_HOME/conf:$HADOOP_HOME/conf:
$HIVE_HOME/lib/slf4j-api-1.6.1.jar

$HADOOP_HOME/bin/hadoop --config $HADOOP_HOME/conf jar <path_to_jar>
<main_class> -libjars $LIB_JARS <program_arguments>
```

Authentication

If a failure results in a message like "2010-11-03 16:17:28,225 WARN hive.metastore ... - Unable to connect metastore with URI thrift://..." in /tmp/<username>/hive.log, then make sure you have run "kinit <username>@FOO.COM" to get a Kerberos ticket and to be able to authenticate to the HCatalog server.

Read Example

The following very simple MapReduce program reads data from one table which it assumes to have an integer in the second column, and counts how many different values it sees. That is, is does the equivalent of `select col1, count(*) from $table group by col1;`.

```
public class GroupByAge extends Configured implements Tool {

    public static class Map extends
            Mapper<WritableComparable, HCatRecord, IntWritable, IntWritable> {

        int age;

        @Override
        protected void map(
                WritableComparable key,
                HCatRecord value,
                org.apache.hadoop.mapreduce.Mapper<WritableComparable, HCatRecord,
                        IntWritable, IntWritable>.Context context)
                throws IOException, InterruptedException {
            age = (Integer) value.get(1);
            context.write(new IntWritable(age), new IntWritable(1));
        }
    }

    public static class Reduce extends Reducer<IntWritable, IntWritable,
    WritableComparable, HCatRecord> {


      @Override
      protected void reduce(
              IntWritable key,
              java.lang.Iterable<IntWritable> values,
              org.apache.hadoop.mapreduce.Reducer<IntWritable, IntWritable,
                      WritableComparable, HCatRecord>.Context context)
              throws IOException, InterruptedException {
          int sum = 0;
          Iterator<IntWritable> iter = values.iterator();
          while (iter.hasNext()) {
              sum++;
              iter.next();
          }
          HCatRecord record = new DefaultHCatRecord(2);
          record.set(0, key.get());
          record.set(1, sum);

          context.write(null, record);
        }
    }

    public int run(String[] args) throws Exception {
        Configuration conf = getConf();
        args = new GenericOptionsParser(conf, args).getRemainingArgs();

        String inputTableName = args[0];
        String outputTableName = args[1];
        String dbName = null;

        Job job = new Job(conf, "GroupByAge");
        HCatInputFormat.setInput(job, InputJobInfo.create(dbName,
                inputTableName, null));
        // initialize HCatOutputFormat

        job.setInputFormatClass(HCatInputFormat.class);
        job.setJarByClass(GroupByAge.class);
        job.setMapperClass(Map.class);
```

```
        job.setReducerClass(Reduce.class);
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(WritableComparable.class);
        job.setOutputValueClass(DefaultHCatRecord.class);
        HCatOutputFormat.setOutput(job, OutputJobInfo.create(dbName,
                outputTableName, null));
        HCatSchema s = HCatOutputFormat.getTableSchema(job);
        System.err.println("INFO: output schema explicitly set for writing:"
                + s);
        HCatOutputFormat.setSchema(job, s);
        job.setOutputFormatClass(HCatOutputFormat.class);
        return (job.waitForCompletion(true) ? 0 : 1);
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new GroupByAge(), args);
        System.exit(exitCode);
    }
}
```

Notice a number of important points about this program:

1. The implementation of Map takes HCatRecord as an input and the implementation of Reduce produces it as an output.
2. This example program assumes the schema of the input, but it could also retrieve the schema via HCatOutputFormat.getOutputSchema() and retrieve fields based on the results of that call.
3. The input descriptor for the table to be read is created by calling InputJobInfo.create. It requires the database name, table name, and partition filter. In this example the partition filter is null, so all partitions of the table will be read.
4. The output descriptor for the table to be written is created by calling OutputJobInfo.create. It requires the database name, the table name, and a Map of partition keys and values that describe the partition being written. In this example it is assumed the table is unpartitioned, so this Map is null.

To scan just selected partitions of a table, a filter describing the desired partitions can be passed to InputJobInfo.create. To scan a single filter, the filter string should look like: "datestamp=20120401" where datestamp is the partition column name and 20120401 is the value you want to read.

Filter Operators

A filter can contain the operators 'and', 'or', 'like', '()', '=', '<>' (not equal), '<', '>', '<=' and '>='. For example:

- `datestamp > "20110924"`
- `datestamp < "20110925`
- `datestamp <= "20110925" and datestamp >= "20110924"`

Scan Filter

Assume for example you have a web_logs table that is partitioned by the column datestamp. You could select one partition of the table by changing

```
HCatInputFormat.setInput(job, InputJobInfo.create(dbName, inputTableName, null));
```

to

```
HCatInputFormat.setInput(job,
    InputJobInfo.create(dbName, inputTableName, "datestamp=\"20110924\""));
```

This filter must reference only partition columns. Values from other columns will cause the job to fail.

Write Filter

To write to a single partition you can change the above example to have a Map of key value pairs that describe all of the partition keys and values for that partition. In our example web_logs table, there is only one partition column (datestamp), so our Map will have only one entry. Change

```
HCatOutputFormat.setOutput(job, OutputJobInfo.create(dbName, outputTableName, null));
```

to

```
Map partitions = new HashMap<String, String>(1);
partitions.put("datestamp", "20110924");
HCatOutputFormat.setOutput(job, OutputJobInfo.create(dbName, outputTableName, partitions));
```

To write multiple partitions simultaneously you can leave the Map null, but all of the partitioning columns must be present in the data you are writing.