使用Apache Beam读写HDFS上的文件

Apache Beam(原名Google DataFlow)是Google在2016年2月份贡献给Apache基金会的Apache孵化项目，被认为是继MapReduce，GFS和BigQuery等之后，Google在大数据处理领域对开源社区的又一个非常大的贡献。Apache Beam的主要目标是统一批处理和流处理的编程范式，为无限，乱序，web-scale的数据集处理提供简单灵活，功能丰富以及表达能力十分强大的SDK。此项目于2017年01月10正式成为Apache顶级项目。

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

本文并不打算介绍更多关于Apache Beam的概念等信息，而是介绍如何在程序里面使用Apache Beam读写HDFS上的文件。为了能够与HDFS通信，我们需要在项目的pom.xml文件里面引入以下依赖：

```
<dependency>
  <groupId>org.apache.beam</groupId>
  <artifactId>beam-sdks-java-io-hdfs</artifactId>
  <version>${beam.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-common</artifactId>
  <version>2.2.0</version>
</dependency>

<dependency>
  <groupId>org.apache.hadoop</groupId>
```

```
    <artifactId>hadoop-hdfs</artifactId>
    <version>2.2.0</version>
</dependency>

<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-core</artifactId>
    <version>2.2.0</version>
</dependency>
```

beam.version 根据你自己的需求选择，我这里选择的是最新版：0.6.0，0.5.0版本及其以下版本和这里介绍的略有不同。

读HDFS上的文件

读HDFS上的文件我们可以使用HDFSFileSource类来实现，其内置为我们提供了读取Avro（fromAvro）、文本文件（fromText）和任意文件（from），具体读文件如下：

```
Read.Bounded<KV<LongWritable, Text>> from = Read.from(HDFSFileSource.from("/iteblog/data", TextInputFormat.class, LongWritable.class, Text.class);
PCollection<KV<LongWritable, Text>> data = p.apply(from);
```

from函数是用来读取HDFS上多种文件的，我们只需要传递相应的InputFormat，Key和Vaule的类型即可读取里面的文件；在本类中，我们读取的是普通文本文件，所以传进来的是TextInputFormat，普通文件的Key是LongWritable类型的，Value类型是Text。我们可以进一步简化这个：

```
Read.Bounded<String> from = Read.from(HDFSFileSource.fromText("/iteblog/data"));
PCollection<KV<LongWritable, Text>> data = p.apply(from);
```

这个效果和上面的类似，因为读取文本文件的时候，Key存储的是文件偏移量，一般都是丢弃。

在读取HDFS文件的时候，我们还可以传递相关的Hadoop配置，如下：

```
Configuration conf = new Configuration();
conf.set(key,value);
Read.Bounded<String> from = Read.from(HDFSFileSource.fromText("/iteblog/data").withConfi
```

```
guration(conf));
PCollection<KV<LongWritable, Text>> data = p.apply(from);
```

将数据写入HDFS

和HDFSFileSource类相对应，我们可以使用HDFSFileSink将数据写入到HDFS上。同样，HDFSFileSink类也直接提供了写Avro文件（toAvro）、文本文件（toText）和其他文件的方式（to），具体使用如下：

```
PCollection<String> data = ....
data.apply("x", Write.to(HDFSFileSink.<string>toText("/iteblog/output/")));
```

这样我们就可以将数据写入到HDFS上。同样，我们也可以在写入数据的时间设置相关的配置：

```
Configuration conf = new Configuration();
conf.set(key,value);
PCollection<String> data = ....
data.apply("x", Write.to(HDFSFileSink.<string>toText("/iteblog/output/").withConfiguration(conf)));
```

注意事项

如果你想把数据写入HDFS上，你需要手动指定集群的地址，比如：hdfs://iteblogcluster/iteblog/output，否则Beam将默认使用的是本地文件系统。当然你也可以通过配置文件Configuration，设置HDFS的地址，如下：

```
<property>
   <name>fs.defaultFS</name>
   <value>hdfs://iteblogcluster</value>
</property>
```

这样你就可以不在输入输出路径上指定hdfs模式。

当前版本的HDFSFileSink有个bug（参见BEAM-1856

），如果输入的文件路径不带文件模式，比如：/iteblog/output，也没有设置fs.defaultFS，那么HDFSFileSink类会首先检测程序Master运行节点上的本地文件系统是否存在/iteblog/output/路径：

```
public void validate(PipelineOptions options) {
  if (validate()) {
    try {
      UGIHelper.getBestUGI(username()).doAs(new PrivilegedExceptionAction<Void>() {
        @Override
        public Void run() throws Exception {
          FileSystem fs = FileSystem.get(new URI(path()),
              SerializableConfiguration.newConfiguration(serializableConfiguration()));
          checkState(!fs.exists(new Path(path())), "Output path %s already exists", path());
          return null;
        }
      });
    } catch (IOException | InterruptedException e) {
      throw new RuntimeException(e);
    }
  }
}
```

path()这个就是我们传进来的/iteblog/output，使用这个路径初始化出来的fs其实是LocalFileSystem实例；而后面HDFSFileSink写文件的时候却会将文件写入到HDFS上：

```
public void initialize(PipelineOptions options) throws Exception {
      Job job = sink.newJob();
      FileOutputFormat.setOutputPath(job, new Path(path));
}

private Job newJob() throws IOException {
  Job job = SerializableConfiguration.newJob(serializableConfiguration());
  job.setJobID(jobId);
  job.setOutputKeyClass(keyClass());
  job.setOutputValueClass(valueClass());
  return job;
}

public static Job newJob(@Nullable SerializableConfiguration conf) throws IOException {
  if (conf == null) {
    return Job.getInstance();
  } else {
    Job job = Job.getInstance();
```

```
    for (Map.Entry<String, String> entry : conf.get()) {
      job.getConfiguration().set(entry.getKey(), entry.getValue());
    }
    return job;
  }
}
```

注意Job job = Job.getInstance();，其实job实例的初始化会加载Runner（SparkRunner、FlinkRunner等）的配置，如果这时候Runner的默认文件系统是HDFS的话，这也就导致了作业的Master和Slave节点的配置不一致；最后导致FileOutputFormat.setOutputPath(job, new Path(path));使用的是HDFS上的目录！这个Bug我已经修复完整，应该会在Apache Beam第一个稳定版本（First stable release）发布。