使用MapReduce读取XML文件

XML（可扩展标记语言，英语：eXtensible Markup Language，简称: XML）是一种标记语言，也是行业标准数据交换交换格式，它很适合在系统之间进行数据存储和交换（话说Hadoop、Hive等的配置文件就是XML格式的）。本文将介绍如何使用MapReduce来读取XML文件。但是Hadoop内部是无法直接解析XML文件；而且XML格式中没有同步标记，所以并行地处理单个XML文件比较棘手。本文将一一介绍如何使用MapReduce处理XML文件，并且能够对其进行分割，然后并行处理这些分片。

MapReduce里面读取文件一般都是使用InputFormat类进行的，比如读取文本文件可以使用TextInputFormat类进行（关于InputFormat类的源码分析可以参见《MapReduce数据输入中InputFormat类源码解析》），所以我们在MapReduce中解析XML文件也可以自定义类似的XMLInputFormat。熟悉Mahout的同学应该知道，它里面实现了一个XmlInputFormat（https://github.com/apache/mahout/blob/ad84344e4055b1e6adff5779339a33fa29e1265d/examples/src/main/java/org/apache/mahout/classifier/bayes/XmlInputFormat.java）类，我们可以指定指定的开始和结束标记来分隔XML文件，然后我们就可以像解析Text文件一样来解析Xml文件，如下：

```
Configuration conf = new Configuration();
conf.set("xmlinput.start", "<property>");
conf.set("xmlinput.end", "</property>");

Job job = new Job(conf);

 job.setInputFormatClass(XmlInputFormat.class);
```

指定了xmlinput.start和xmlinput.end就可以搜索XML文件中的开始和结束标记，从而获取到XML里面的数据。完整的map程序如下：

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.slf4j.*;

import javax.xml.stream.*;
import java.io.*;
```

```
public static class Map extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    protected void map(LongWritable key, Text value,
                       Mapper.Context context)
      throws   IOException, InterruptedException {
     String document = value.toString();
     System.out.println("'" + document + "'");
     try {
       XMLStreamReader reader =
          XMLInputFactory.newInstance().createXMLStreamReader(new
             ByteArrayInputStream(document.getBytes()));
       String propertyName = "";
       String propertyValue = "";
       String currentElement = "";
       while (reader.hasNext()) {
         int code = reader.next();
         switch (code) {
           case START_ELEMENT:
             currentElement = reader.getLocalName();
             break;
           case CHARACTERS:
             if (currentElement.equalsIgnoreCase("name")) {
               propertyName += reader.getText();
             } else if (currentElement.equalsIgnoreCase("value")) {
               propertyValue += reader.getText();
             }
             break;
         }
       }
       reader.close();
       context.write(propertyName.trim(), propertyValue.trim());
     } catch (Exception e) {
       log.error("Error processing '" + document + "'", e);
     }
}
```

map接收一个Text的值，里面的值包含了开始和结束的标记之间的数据，然后我们就可以使用java内置的XML Streaming
API解析器提取每个属性的key和value，最后输出key和value。完整的代码如下：

```
package com.iteblog.hadoop;
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.slf4j.*;

import javax.xml.stream.*;
import java.io.*;

import static javax.xml.stream.XMLStreamConstants.*;

public final class XML{
  private static final Logger log = LoggerFactory.getLogger
     (HadoopPropertyXMLMapReduce.class);

  public static class Map extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    protected void map(LongWritable key, Text value,
                       Mapper.Context context)
      throws IOException, InterruptedException {
     String document = value.toString();
     System.out.println("""" + document + """");
     try {
       XMLStreamReader reader =
          XMLInputFactory.newInstance().createXMLStreamReader(new
             ByteArrayInputStream(document.getBytes()));
       String propertyName = "";
       String propertyValue = "";
       String currentElement = "";
       while (reader.hasNext()) {
         int code = reader.next();
         switch (code) {
           case START_ELEMENT:
             currentElement = reader.getLocalName();
             break;
           case CHARACTERS:
             if (currentElement.equalsIgnoreCase("name")) {
               propertyName += reader.getText();
             } else if (currentElement.equalsIgnoreCase("value")) {
               propertyValue += reader.getText();
             }
             break;
         }
```

```
      }
      reader.close();
      context.write(propertyName.trim(), propertyValue.trim());
    } catch (Exception e) {
      log.error("Error processing '" + document + "'", e);
    }
  }
 }

 public static void main(String... args) throws Exception {
  runJob(args[0], args[1]);
 }

 public static void runJob(String input,
                           String output)
    throws Exception {
  Configuration conf = new Configuration();
  conf.set("key.value.separator.in.input.line", " ");
  conf.set("xmlinput.start", "<property>");
  conf.set("xmlinput.end", "</property>");

  Job job = new Job(conf);
  job.setJarByClass(XML.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(Text.class);
  job.setMapperClass(Map.class);
  job.setInputFormatClass(XmlInputFormat.class);
  job.setNumReduceTasks(0);
  job.setOutputFormatClass(TextOutputFormat.class);

  FileInputFormat.setInputPaths(job, new Path(input));
  Path outPath = new Path(output);
  FileOutputFormat.setOutputPath(job, outPath);
  outPath.getFileSystem(conf).delete(outPath, true);

  job.waitForCompletion(true);
 }
}
```

我们可以使用hadoop内部的core-site.xml文件做测试，如下：

```
$ bin/hadoop jar xml.jar  com.iteblog.hadoop.XML  input/core-site.xml  output/
```

运行完上面的程序，可以在output目录下产生几个part*的文件，这说明解析XML文件的程序起作用了。