


Java-Success.com

Prepare to fast-track, choose & go places with 800+ Java & Big Data Q&As with lots of code & diagrams.

[Home](#) [Why? ▾](#) [300+ Java FAQs ▾](#) [300+ Big Data FAQs ▾](#) [Courses ▾](#)[👤 Membership ▾](#) [Your Career ▾](#)[Home](#) > [bigdata-success.com](#) > [Tutorials - Big Data](#) > [TUT - File Formats](#) > 02: Convert

XML file To Sequence File with Apache Spark – writing & reading

02: Convert XML file To Sequence File with Apache Spark – writing & reading

 Posted on [May 2, 2016](#)

This extends the [Convert XML file To Sequence File With Hadoop libraries](#), by using Apache Spark.

Step 1: The pom.xml file should include the Apache Spark libraries as shown below.

```
1
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.mytutorial</groupId>
```

300+ Java Interview FAQs

300+ Java FAQs



16+ Java Key Areas Q&As



150+ Java Architect FAQs



80+ Java Code Quality Q&As



150+ Java Coding Q&As



300+ Big Data Interview FAQs

300+ Big Data FAQs



Tutorials - Big Data

TUT -  Starting Big Data

TUT - Starting Spark & Scala

```

6      <artifactId>sequence-file</artifactId>
7      <packaging>jar</packaging>
8      <version>1.0-SNAPSHOT</version>
9      <name>sequence-file</name>
10     <url>http://maven.apache.org</url>
11
12     <properties>
13         <maven.compiler.source>1.8</maven.compiler.source>
14         <maven.compiler.target>1.8</maven.compiler.target>
15         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16         <junit.version>4.8.1</junit.version>
17         <hadoop.version>2.7.2</hadoop.version>
18         <spark-version>1.3.0</spark-version>
19     </properties>
20
21     <dependencies>
22         <!-- JUnit -->
23
24         <dependency>
25             <groupId>junit</groupId>
26             <artifactId>junit</artifactId>
27             <version>${junit.version}</version>
28             <scope>test</scope>
29         </dependency>
30
31         <!-- Hadoop -->
32         <dependency>
33             <groupId>org.apache.hadoop</groupId>
34             <artifactId>hadoop-hdfs</artifactId>
35             <version>${hadoop.version}</version>
36             <exclusions>
37                 <exclusion>
38                     <groupId>javax.servlet</groupId>
39                     <artifactId>*</artifactId>
40                 </exclusion>
41             </exclusions>
42         </dependency>
43         <dependency>
44             <groupId>org.apache.hadoop</groupId>
45             <artifactId>hadoop-client</artifactId>
46             <version>${hadoop.version}</version>
47             <exclusions>
48                 <exclusion>
49                     <groupId>javax.servlet</groupId>
50                     <artifactId>*</artifactId>
51                 </exclusion>
52             </exclusions>
53         </dependency>
54
55         <!-- Apache Spark -->
56         <dependency>
57             <groupId>org.apache.spark</groupId>
58             <artifactId>spark-core_2.11</artifactId>
59             <version>${spark-version}</version>
60             <exclusions>

```

TUT - Starting with Python

TUT - Kafka

TUT - Pig

TUT - Apache Storm

TUT - Spark Scala on Zeppelin

TUT - Cloudera

TUT - Cloudera on Docker

TUT - File Formats

TUT - Spark on Docker

TUT - Flume

TUT - Hadoop (HDFS)

TUT - HBase (NoSQL)

TUT - Hive (SQL)

TUT - Hadoop & Spark

TUT - MapReduce

TUT - Spark and Scala

TUT - Spark & Java

TUT - PySpark on Databricks

TUT - Zookeeper

800+ Java Interview Q&As

300+ Core Java Q&As



300+ Enterprise Java Q&As



150+ Java Frameworks Q&As



120+ Companion Tech Q&As



Tutorials - Enterprise Java



```
61         <exclusion>
62             <groupId>javax.servlet</groupId>
63             <artifactId>*</artifactId>
64         </exclusion>
65     </exclusions>
66 </dependency>
67
68 </dependencies>
69
70 </project>
71
72
```

Step 2: The XML file report.xml.

```
1
2 <?xml version="1.0" encoding="UTF-8"?>
3 <transactionReports xmlns="http://mytutorial.com/">
4     <transactionReport>
5         <report>
6             <reportNumber>9999</reportNumber>
7             <createdDatetime>2015-06-15T11:29:52+1
8             <processedDatetime>2015-06-15T11:29:52
9             <reportStatusCode>Active</reportStatus
10        </report>
11    </transactionReport>
12 </transactionReports>
13
```

Step 3: The Java class

"ConvertXmlToSequenceWithSpark".

```
1
2 package com.mytutorial;
3
4 import java.io.File;
5 import java.io.IOException;
6 import java.net.URL;
7 import java.util.ArrayList;
8 import java.util.List;
9 import java.util.stream.Collectors;
10
11 import org.apache.commons.io.FileUtils;
12 import org.apache.hadoop.io.ByteWritable;
13 import org.apache.hadoop.io.BytesWritable;
14 import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
15 import org.apache.spark.SparkConf;
16 import org.apache.spark.api.java.JavaPairRDD;
```

```
17 import org.apache.spark.api.java.JavaSparkContext
18
19 import scala.Tuple2;
20
21 public class ConvertXmlToSequenceWithSpark {
22
23     private static final String FILE_IN_PATH = "d
24     private static final String FILE_OUT_PATH = "c
25
26     final static JavaSparkContext sc;
27
28     static {
29         SparkConf conf = new SparkConf().setAppNar
30             .set("spark.executor.memory", "1g"
31             .set("spark.serializer", "org.apac
32         sc = new JavaSparkContext(conf);
33     }
34
35     public static void main(String[] args) throws
36         URL resource = ConvertXmlToSequenceWithSp
37
38         File inputFile = new File(resource.getPat
39         File outputFile = new File(resource.getPat
40
41         String readXmlFileContents = FileUtils.rea
42
43         write(readXmlFileContents, outputFile); /
44
45         read(outputFile);
46
47         sc.stop();
48     }
49
50     /**
51     * Write a text file to sequence file
52     *
53     * @param conf
54     * @param inputFile
55     * @param outputFile
56     * @throws IOException
57     */
58     public static void write(String readXmlFileCon
59
60         List<Tuple2<BytesWritable, BytesWritable>>
61         arrayList.add(new Tuple2<>(new BytesWrital
62             .getBytes())));
63
64         if (outputFile.exists()) {
65             FileUtils.deleteQuietly(outputFile);
66         }
67
68         try {
69             JavaPairRDD<BytesWritable, BytesWrital
70             seqRDD.saveAsNewAPIHadoopFile(outputF
71                 SequenceFileOutputFormat.class;
```

```

72
73     } catch (Exception e) {
74         System.out.println("Error writing: " + e.getMessage());
75     }
76 }
77
78 /**
79  * Read a sequence file
80  *
81  * @param conf
82  * @param sequenceFileToRead
83  */
84 public static void read(File sequenceFileToRead) {
85
86     sc.hadoopConfiguration().set("mapreduce.input.sequence.file.name", sequenceFileToRead.getAbsolutePath());
87
88     JavaPairRDD<ByteWritable, BytesWritable> rdd = sc.newAPIHadoopFile(sequenceFileToRead.getAbsolutePath(),
89         ByteWritable.class, BytesWritable.class, SequenceFileAsBinaryInputFormat.class);
90
91     List<String> valuesXml = rdd.map(pair -> pair._2().toString()).collect();
92     String xml = valuesXml.stream().collect(Collectors.joining("\n"));
93     System.out.println("xml=" + xml);
94 }
95 }
96
97

```

The sequence file output is written as something like **part-r-00000** in **data/report-seq** folder, and then read back again.

Using SequenceFileAsBinaryOutputFormat & SequenceFileAsBinaryInputFormat

As you can see the reading method uses "sc.newAPIHadoopFile(.....)".

```

1
2 package com.mytutorial;
3
4 import java.io.File;
5 import java.io.IOException;
6 import java.net.URL;
7 import java.util.ArrayList;
8 import java.util.List;
9 import java.util.stream.Collectors;
10
11 import org.apache.commons.io.FileUtils;

```

```
12 import org.apache.hadoop.conf.Configuration;
13 import org.apache.hadoop.io.ByteWritable;
14 import org.apache.hadoop.io.BytesWritable;
15 import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
16 import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
17 import org.apache.hadoop.mapreduce.lib.output.SequenceFileWriter;
18 import org.apache.spark.SparkConf;
19 import org.apache.spark.api.java.JavaPairRDD;
20 import org.apache.spark.api.java.JavaSparkContext;
21
22 import scala.Tuple2;
23
24 public class ConvertXmlToSequenceWithSpark {
25
26     private static final String FILE_IN_PATH = "data.xml";
27     private static final String FILE_OUT_PATH = "sequence.txt";
28
29     final static JavaSparkContext sc;
30
31     static {
32         SparkConf conf = new SparkConf().setAppName("ConvertXmlToSequenceWithSpark")
33             .set("spark.executor.memory", "1g")
34             .set("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
35         sc = new JavaSparkContext(conf);
36     }
37
38     public static void main(String[] args) throws IOException {
39         URL resource = ConvertXmlToSequenceWithSpark.class.getResource(FILE_IN_PATH);
40
41         File inputFile = new File(resource.getPath());
42         File outputFile = new File(resource.getPath().replace(FILE_IN_PATH, FILE_OUT_PATH));
43
44         String readXmlFileContents = FileUtils.readFileToString(inputFile, "UTF-8");
45
46         write(readXmlFileContents, outputFile); // Write a text file to sequence file
47
48         read(outputFile);
49
50         sc.stop();
51     }
52
53     /**
54      * Write a text file to sequence file
55      *
56      * @param conf
57      * @param inputFile
58      * @param outputFile
59      * @throws IOException
60      */
61     public static void write(String readXmlFileContents, File outputFile) throws IOException {
62
63         List
```

```

67         if (outputFile.exists()) {
68             FileUtils.deleteQuietly(outputFile);
69         }
70
71         try {
72             JavaPairRDD<BytesWritable, BytesWritable>
73             seqRDD.saveAsNewAPIHadoopFile(outputFile,
74                 SequenceFileAsBinaryOutputFormat.class,
75                 null, null);
76         } catch (Exception e) {
77             System.out.println("Error writing: " + e.getMessage());
78         }
79     }
80
81     /**
82     * Read a sequence file
83     *
84     * @param conf
85     * @param sequenceFileToRead
86     * @throws IOException
87     */
88     public static void read(File sequenceFileToRead) {
89
90         JavaPairRDD<BytesWritable, BytesWritable>
91         seqRDD = new SequenceFileAsBinaryInputFormat<>().read(sequenceFileToRead);
92
93         List<String> valuesXml = seqRDD.mapValues(pair -> pair._2().toString()).collect();
94         String xml = valuesXml.stream().collect(Collectors.joining("\n"));
95         System.out.println("xml=" + xml);
96     }
97 }
98
99

```

◀ 01a: Convert XML file To Sequence File – writing & reading – Local File

System

XML Parsing with JAXB implementation called MOXy ▶

Disclaimer

The contents in this Java-Success are copyrighted and from EmpoweringTech pty ltd. The EmpoweringTech pty ltd has the right to correct or enhance the current content without any prior notice. These are general advice only, and one needs to take his/her own circumstances into consideration. The EmpoweringTech pty ltd will not be held liable for any damages caused or alleged to be caused either directly or

indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. Links to external sites do not imply endorsement of the linked-to sites. [Privacy Policy](#).

© 2022 [java-success.com](https://www.java-success.com)