

How to readfrom Spark data sources

We have already used *DataFrameReader* for CSV. So, you already know the general structure. For a typical reader, there are four methods.

1. format
2. option
3. schema
4. load

We have already seen these in action for CSV. The format method takes a string to define the data source. It could be one of the following values.

1. parquet
2. json
3. orc
4. com.databricks.spark.avro
5. com.databricks.spark.xml

The first three formats are part of the Spark Core packages. However, other two are still separate, and hence if you want to use them, you must manage the dependency

The next one is the option method. We have already used options for CSV. You can pass a key-value pair to the option method. And you can add as many option methods as you want

The next method is the schema method. We already learned that we could infer the schema from the source system or we can use this method. There are two alternatives to specify a custom schema. We have already seen the *StructType* while working with CSV. You also have another simple choice. Specify a DDL like string.

|  |  |
| --- | --- |
|  | spark.read.schema("a INT, b STRING, c DOUBLE") |

Finally, you have the load method. In case of a file-based data source, you can pass a directory location or a file name to the load method. For other sources like JDBC, there is no file. So you would call it without any argument.

## How to handle malformed records

When you are reading hundreds of thousands of rows, some of them may not confine to the schema. Those are incorrect records. What do you want to do with those?

Spark *DataFrameReader* allows you to set an option called mode. We have seen that as well for the CSV example. You can set one of the three values.

1. Permissive: the DataFrameReader will set all the column values to null and push the entire row into a string column called \_corrupt\_record. This corrupt record column allows you to dump those records in a separate file.
2. dropMalformed: The dropMalformed option will quietly drop the malformed records. You won't even notice that some rows were not accurate.
3. failFast: the failFast method raises an exception.

ow to handle file already exist condition

However, while working with a file-based destination, you may encounter a file already exists situation. To handle that scenario, we have a mode method that takes one of the following options.

1. append
2. overwrite
3. errorIfExists
4. ignore

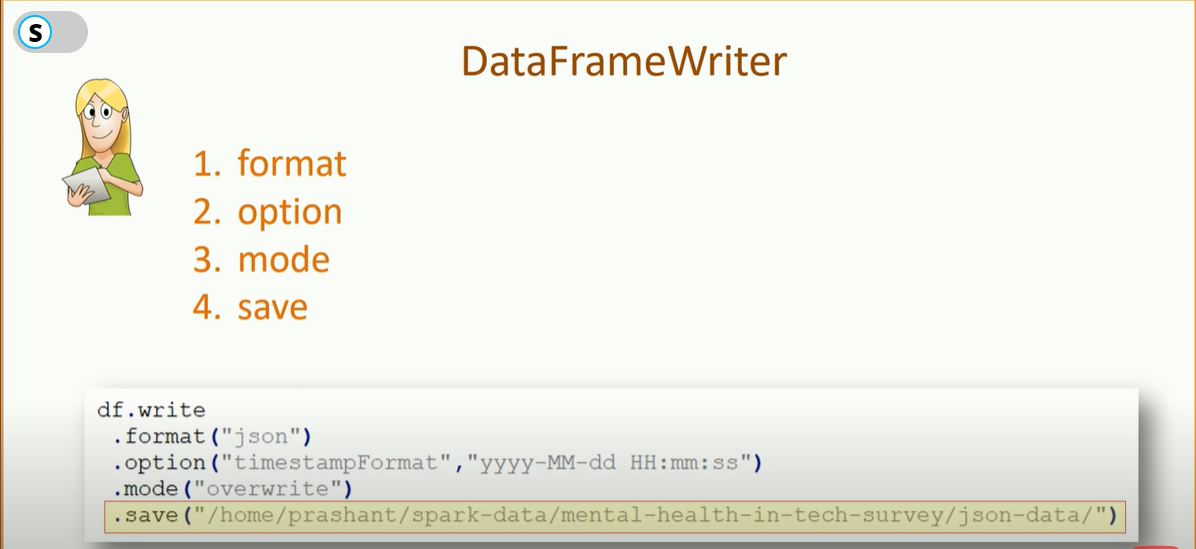
All the values are obvious. The third option will raise an exception, and the last option will keep quiet. I mean, if the file already exists, just ignore without raising any error.

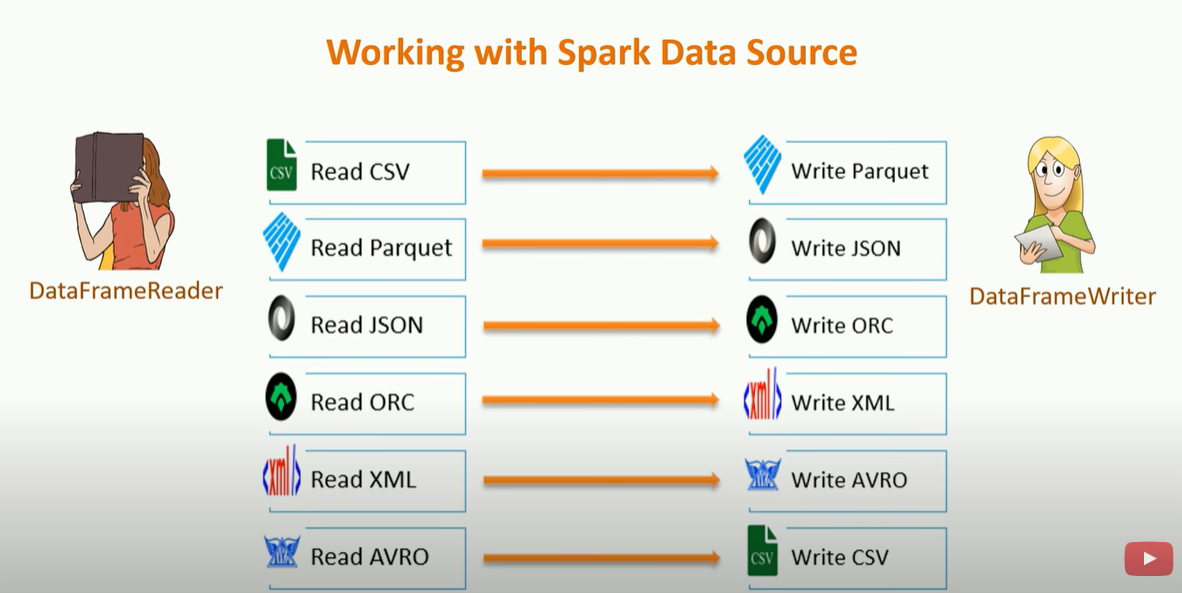
Additional Spark Data frame write options

If you are working with file-based destination system, you also have three additional methods.

1. partitionBy
2. bucketBy
3. sortBy

The *partitionBy* method allows you to partition your output file. And the *bucketBy* method allows you to create a finite number of data buckets. Both methods follow the hive partitioning and bucketing principals.





Working with CSV in Apache Spark

Here is the code to read a CSV and write into a Parquet format.

|  |  |
| --- | --- |
|  | //Read CSV into Data Frame |
|  | val df = spark.read |
|  | .format("csv") |
|  | .option("header", "true") |
|  | .option("inferSchema", "true") |
|  | .option("nullValue", "NA") |
|  | .option("timestampFormat", "yyyy-MM-dd'T'HH:mm:ss") |
|  | .option("mode", "failfast") |
|  | .load("/home/prashant/spark-data/mental-health-in-tech-survey/survey.csv") |
|  |  |
|  | //Write Data Frame to Parquet |
|  | df.write |
|  | .format("parquet") |
|  | .mode("overwrite") |
|  | .save("/home/prashant/spark-data/mental-health-in-tech-survey/parquet-data/") |

Working with JSON in Apache Spark

Let's move on to the next example. This time, we read parquet and write JSON.

|  |  |
| --- | --- |
|  | //Read Parquet into Data Frame |
|  | val df = spark.read |
|  | .format("parquet") |
|  | .option("mode", "failfast") |
|  | .load("/home/prashant/spark-data/mental-health-in-tech-survey/parquet-data/") |
|  |  |
|  | //Write Data Frame to JSON |
|  | df.write |
|  | .format("json") |
|  | .option("timestampFormat", "yyyy-MM-dd HH:mm:ss") |
|  | .mode("overwrite") |
|  | .save("/home/prashant/spark-data/mental-health-in-tech-survey/json-data/") |

Working with ORC in Apache Spark

Here is an example to read a JSON and write it back to an ORC format. There is nothing new for me to explain here.

|  |  |
| --- | --- |
|  | //Read JSON into Data Frame |
|  | val df = spark.read |
|  | .format("json") |
|  | .option("timestampFormat", "yyyy-MM-dd HH:mm:ss") |
|  | .option("mode", "failfast") |
|  | .load("/home/prashant/spark-data/mental-health-in-tech-survey/json-data/") |
|  |  |
|  | //Write Data Frame to ORC |
|  | df.write |
|  | .format("orc") |
|  | .mode("overwrite") |
|  | .save("/home/prashant/spark-data/mental-health-in-tech-survey/orc-data/") |

## Working with XML in Apache Spark

The next example is to read from ORC and write it to XML. The XML connector is not part of the Spark distribution. It is an opensource connector that Databricks has created and certified.

spark-shell --packages com.databricks:spark-xml\_2.11:0.4.1,com.databricks:spark-avro\_2.11:4.0.0

|  |
| --- |
| //Read ORC into Data Frame |
|  | val df = spark.read |
|  | .format("orc") |
|  | .option("mode", "failfast") |
|  | .load("/home/prashant/spark-data/mental-health-in-tech-survey/orc-data/") |
|  |  |
|  | //Write Data Frame to XML |
|  | df.write |
|  | .format("com.databricks.spark.xml") |
|  | .option("rootTag", "survey") |
|  | .option("rowTag", "survey-row") |
|  | .mode("overwrite") |
|  | .save("/home/prashant/spark-data/mental-health-in-tech-survey/xml-data/") |

|  |
| --- |
| //Read XML into Data Frame |
|  | val df = spark.read |
|  | .format("com.databricks.spark.xml") |
|  | .option("rowTag", "survey-row") |
|  | .option("mode", "failfast") |
|  | .load("/home/prashant/spark-data/mental-health-in-tech-survey/xml-data/") |
|  |  |
|  | //Write Data Frame to AVRO |
|  | df.write |
|  | .format("com.databricks.spark.avro") |
|  | .mode("overwrite") |
|  | .save("/home/prashant/spark-data/mental-health-in-tech-survey/avro-data/") |

s