Open in app

Matthew Powers

Follow 2.1K Followers About

Different approaches to manually create Spark DataFrames

Matthew Powers May 22, 2017 · 2 min read

This blog post explains the Spark and spark-daria helper methods to manually create DataFrames for local development or testing.

We'll demonstrate why the `createDF()` method defined in spark-daria is better than the `toDF()` and `createDataFrame()` methods from the Spark source code.

See this blog post if you're working with PySpark (the rest of this post uses Scala).

toDF()

`toDF()` provides a concise syntax for creating DataFrames and can be accessed after importing Spark implicits.

```
import spark.implicits._
```

The `toDF()` method can be called on a sequence object to create a DataFrame.

```
val someDF = Seq(
  (8, "bat"),
  (64, "mouse"),
  (-27, "horse")
).toDF("number", "word")
```

`someDF` has the following schema.

```
root
 | — number: integer (nullable = false)
 | — word: string (nullable = true)
```

`toDF()` is limited because the column type and nullable flag cannot be customized. In this example, the `number` column is not nullable and the `word` column is nullable.

The `import spark.implicits._` statement can only be run inside of class definitions when the Spark Session is available. All imports should be at the top of the file before the class definition, so `toDF()` encourages bad Scala coding practices.

`toDF()` is suitable for local testing, but production grade code that's checked into master should use a better solution.

createDataFrame()

The `createDataFrame()` method addresses the limitations of the `toDF()` method and allows for full schema customization and good Scala coding practices.

Here is how to create `someDF` with `createDataFrame()`.

```
val someData = Seq(
  Row(8, "bat"),
  Row(64, "mouse"),
  Row(-27, "horse")
)

val someSchema = List(
  StructField("number", IntegerType, true),
  StructField("word", StringType, true)
)

val someDF = spark.createDataFrame(
  spark.sparkContext.parallelize(someData),
  StructType(someSchema)
)
```

`createDataFrame()` provides the functionality we need, but the syntax is verbose. Our test files will become cluttered and difficult to read if `createDataFrame()` is used frequently.

createDF()

`createDF()` is defined in spark-daria and allows for the following terse syntax.

```
val someDF = spark.createDF(
  List(
    (8, "bat"),
    (64, "mouse"),
    (-27, "horse")
  ), List(
    ("number", IntegerType, true),
    ("word", StringType, true)
  )
)
```

`createDF()` creates readable code like `toDF()` and allows for full schema customization like `createDataFrame()`. It's the best of both worlds.

Big shout out to Nithish for writing the advanced Scala code to make `createDF()` work so well.

Creating Datasets

Datasets are similar to DataFrames, but preferable at times because they offer more type safety.

See this blog post for explanations on how to create Datasets with the `toDS` and `createDataset` methods.

Including spark-daria in your projects

The spark-daria README provides the following project setup instructions.

1. Update your `build.sbt` file.

```
libraryDependencies += "com.github.mrpowers" %% "spark-daria" %
"0.38.2"
```

2. Import the spark-daria code into your project:

```
import com.github.mrpowers.spark.daria.sql.SparkSessionExt._
```

Closing Thoughts

DataFrames are a fundamental data structure that are at the core of my Spark analyses.

See this blog post for the different approaches on how to create Datasets, a related data structure.

I wrote a Beautiful Spark Code book that teaches the core aspects of Spark development with DataFrames. The book is the best way to learn how to get good at Spark quickly.

Scala