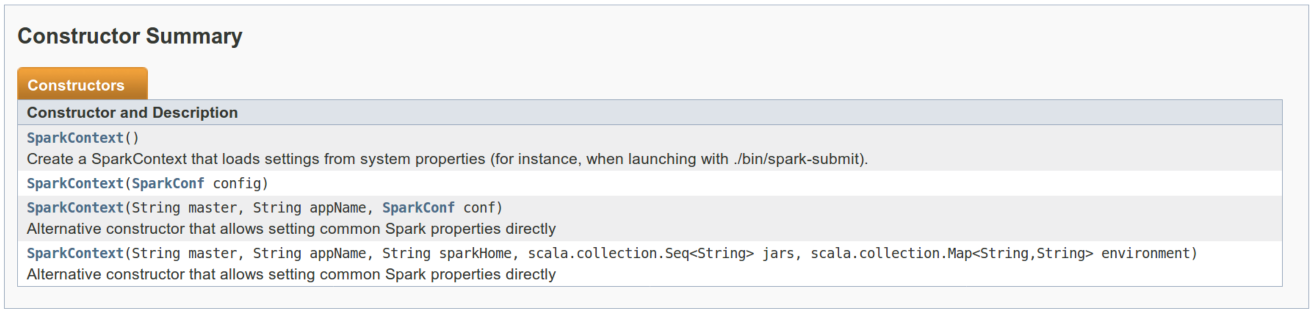
Walter Blair
603 Assignment 5

Builder Pattern

In Assignment 2 I discussed the SparkContext as a not-necessarily-enforced example of a singleton. SparkContext is the sort of managerial environment in which a Spark application or interactive shell runs and uses a not-so-interesting set of constructors (Fig. 1).



Constructor Summary

Constructors

Constructor and Description
`SparkContext()` Create a SparkContext that loads settings from system properties (for instance, when launching with ./bin/spark-submit).
`SparkContext(SparkConf config)`
`SparkContext(String master, String appName, SparkConf conf)` Alternative constructor that allows setting common Spark properties directly
`SparkContext(String master, String appName, String sparkHome, scala.collection.Seq<String> jars, scala.collection.Map<String,String> environment)` Alternative constructor that allows setting common Spark properties directly

Figure 1. Old SparkContext constructors

As we discussed in class, telescoping constructors can be replaced by a good builder pattern, and indeed that's one of the changes we see from the Spark 1.0 to 2.0. I also mentioned in Assignment 2 that Spark 2.0 created another layer of abstraction for SparkContext and the RDD data structure called SparkSession and Dataset/Dataframe data structures. As explained in the Scaladocs, and as you noted in your written comments, SparkSession provides an example of the builder pattern (Fig. 2).

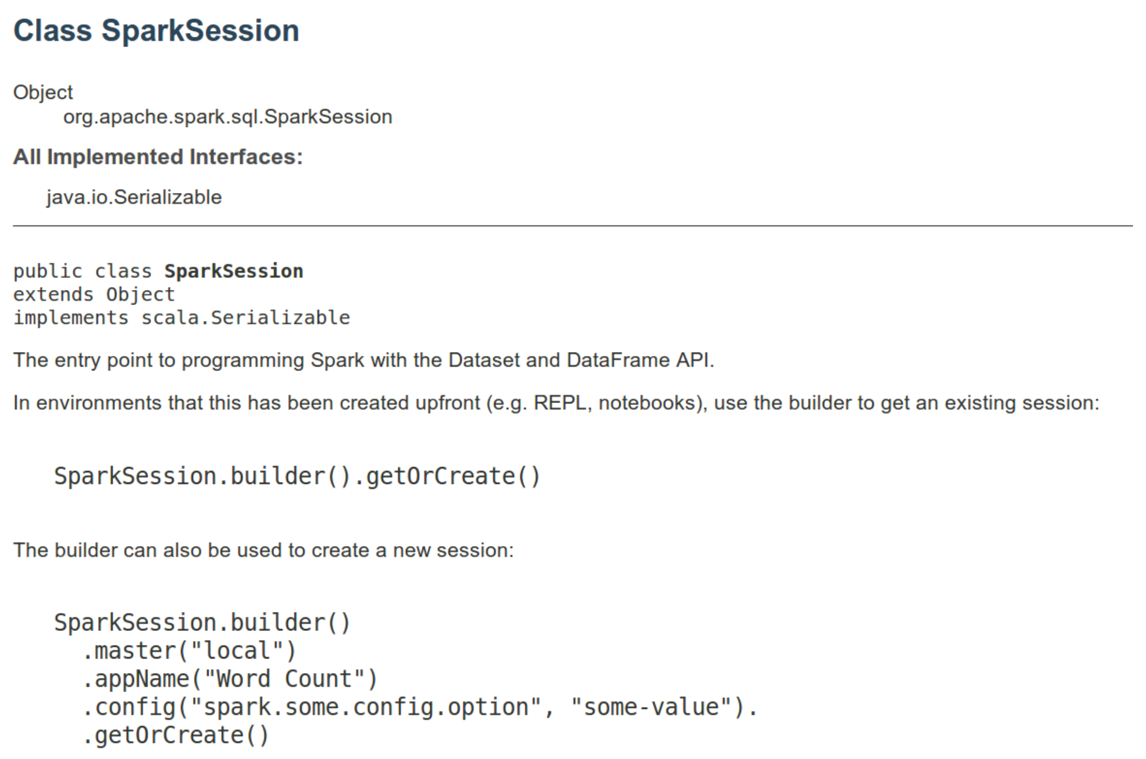
Looking at SparkSession in more detail, the SparkSession class encloses a SparkSession object (objects are easy ways to create static members in Scala) which in turn encloses the Builder class (Fig. 3). SparkSession.builder() returns the concrete Builder. SparkSession.Builder handles various configurations of the Spark environment, including where the master node is located and whether a Hive data store is supported. The example code below from one of my own applications only uses a few of these products when building the SparkSession.

```scala
class TrimS {
  // case class must be defined outside of scope
  case class Read(name1: String, seq: String, name2: String, phred: String)
  def main(args: Array[String]) {

    val spark = SparkSession
      .builder()
      .appName("Scala Trimmer")
      .master("local")    // for testing in IntelliJ
      .getOrCreate()

    val sc = spark.sparkContext
```

As demonstrated above, the Builder’s getOrCreate( ) function is used to either return an existing SparkSession (remember these things are treated as singletons!) or create the new one with all of the specified built products if none exists. Once the SparkSession is built, the old SparkContext that was built and configured inside the SparkSession can be returned.



Class SparkSession

Object
    org.apache.spark.sql.SparkSession

All Implemented Interfaces:

java.io.Serializable

```
public class SparkSession
extends Object
implements scala.Serializable
```

The entry point to programming Spark with the Dataset and DataFrame API.

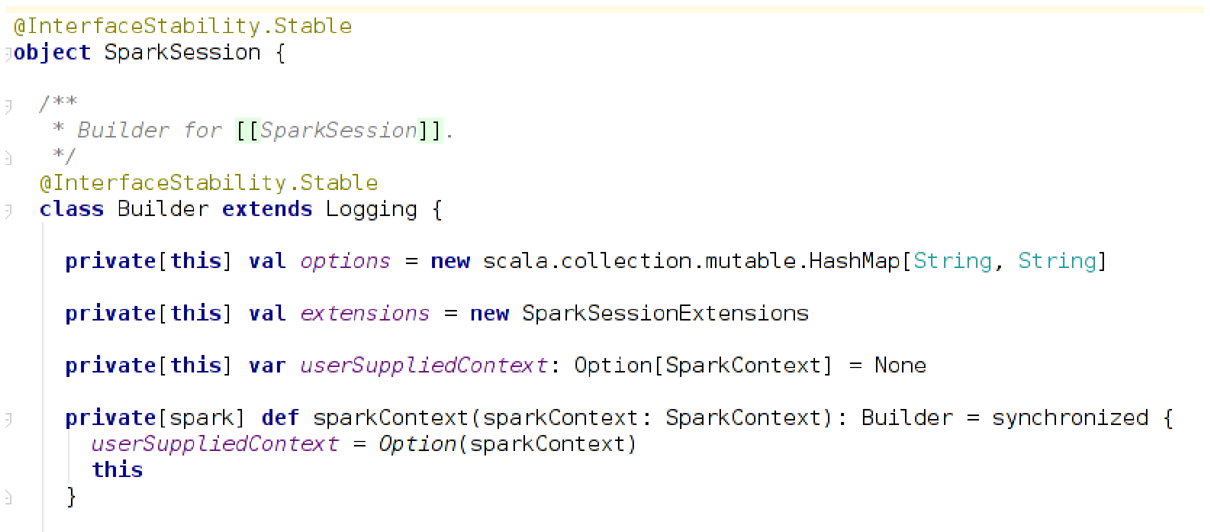
In environments that this has been created upfront (e.g. REPL, notebooks), use the builder to get an existing session:

```
SparkSession.builder().getOrCreate()
```

The builder can also be used to create a new session:

```
SparkSession.builder()
  .master("local")
  .appName("Word Count")
  .config("spark.some.config.option", "some-value").
  .getOrCreate()
```

Figure 2. New SparkSession abstraction



```
@InterfaceStability.Stable
object SparkSession {

  /**
   * Builder for [[SparkSession]].
   */
  @InterfaceStability.Stable
  class Builder extends Logging {

    private[this] val options = new scala.collection.mutable.HashMap[String, String]

    private[this] val extensions = new SparkSessionExtensions

    private[this] var userSuppliedContext: Option[SparkContext] = None

    private[spark] def sparkContext(sparkContext: SparkContext): Builder = synchronized {
      userSuppliedContext = Option(sparkContext)
      this
    }
```

Figure 3. SparkSession object nested within Class SparkSession that contains the builder. The parts built by Builder continue below the code shown here.