

# Scala for Apache Spark

---

Markus Dale, [medale@asymmetrik.com](mailto:medale@asymmetrik.com)

Jan 2019

- Slides: <https://github.com/medale/scala-spark/blob/master/presentation/ScalaSpark.pdf>
- Scala Spark Code Examples: <https://github.com/medale/scala-spark>

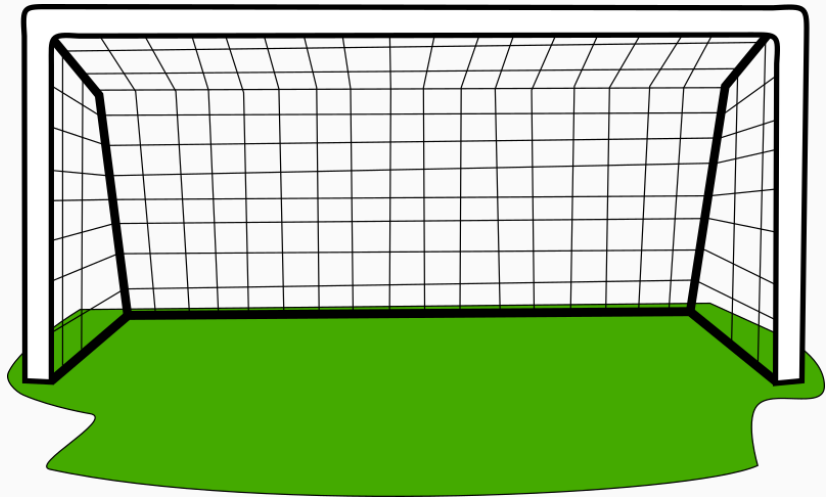


Figure 1: Intro to Scala for Spark

## Why Scala for Spark?

- full interoperability with Java
  - strong type system
  - elegant multi-paradigm (functional & OO)
  - less boilerplate/less code
- JVM

```
package com.uebercomputing.scalaspark.common;

public class JavaMain {
    private int answer = 0;
    public JavaMain(int answer) {
        this.answer = answer;
    }
    public int getAnswer() {
        return answer;
    }
    public static void main(String[] args) {
        System.out.println("Starting a Java program...");
        JavaMain jaMain = new JavaMain(42);
        int answer = jaMain.getAnswer();
        System.out.println("The answer was " + answer);
    }
}
```

# Scala Main One

```
package com.uebercomputing.scalaspark.common
```

```
class ScalaMainOne(val answer: Int)
```

```
object ScalaMainOne {
```

```
  def main(args: Array[String]): Unit = {  
    println("Starting a Scala program...")
```

```
    val scMain = new ScalaMainOne(42)
```

```
    println(scMain)
```

```
    val answer = scMain.answer
```

```
    println(s"The answer was ${answer}")
```

```
  }
```

```
}
```

Starting a Scala program...

com.uebercomputing.scalaspark.common.ScalaMainOne@256216b3

The answer was 42

## Scala Main Two - case class

```
package com.uebercomputing.scalaspark.common
```

```
case class ScalaMainTwo(answer: Int)
```

```
object ScalaMainTwo {
```

```
  def main(args: Array[String]): Unit = {  
    println("Starting a Scala program...")  
    //ScalaMainTwo.apply(42)  
    val scMain = ScalaMainTwo(42)  
    println(scMain)  
    val answer = scMain.answer  
    println(s"The answer was ${answer}")  
  }
```

```
}
```



Starting a Scala program...

ScalaMainTwo(42)

The answer was 42

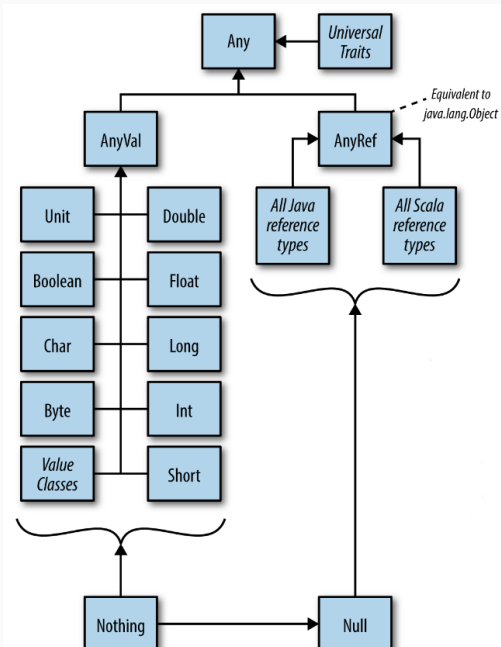
## Scala Main Two - javap ScalaMainTwo.class

```
public class ScalaMainTwo implements Product,Serializable
    public static Option<Object> unapply(ScalaMainTwo);
    public static ScalaMainTwo apply(int);
...
    public ScalaMainTwo copy(int);
...
    public int productArity();
    public Object productElement(int);
    public Iterator<Object> productIterator();
...
    public int hashCode();
    public String toString();
    public boolean equals(Object);
...
```

## HelloSparkWorld - expression-oriented

```
object HelloWorld {  
  ...  
  def main(args: Array[String]): Unit = {  
  
    val lines = if (!args.isEmpty) {  
      val inputFile = args(0)  
      readLinesFromFile(inputFile)  
    } else {  
      readLinesFromString(GhandiQuote)  
    }  
  
    ...  
  }  
}
```

# Scala Type Hierarchy



## HelloSparkWorld - SparkSession

```
import org.apache.spark.sql.SparkSession
...
def main(args: Array[String]): Unit = {
    val lines = ...

    wordCountLocal(lines)

    val spark = SparkSession.builder.
        appName("HelloSparkWorld").
        master("local[2]").
        getOrCreate()

    wordCountRdd(spark, lines)

    spark.close()
}
```

# SparkSession Scala API

sparksession

#ABCDEFGHIJKLMNOPQRSTUVWXYZ – deprecated

display packages only

org.apache.spark.sql

hide focus

SparkSession

SparkSessionExtensions

org.apache.spark.sql

SparkSession

SparkSessionExtensions

org.apache.spark.sql

SparkSession

class **SparkSession** extends Serializable with Closeable with [Logging](#)

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()
```

Self Type: [SparkSession](#)  
Annotations: [@Stable\(\)](#)  
Source: [SparkSession.scala](#)

► Linear Supertypes

Q

Ordering: [Alphabetic](#) [By Inheritance](#)

Inherited: [SparkSession](#) [Logging](#) [Closeable](#) [AutoCloseable](#) [Serializable](#) [Serializable](#) [AnyRef](#) [Any](#)

Hide All [Show All](#)

Visibility: [Public](#) [All](#)

Value Members

► def **baseRelationToDataFrame**(baseRelation: [BaseRelation](#)): [DataFrame](#)  
Convert a BaseRelation created for external data sources into a DataFrame.

► lazy val **catalog**: [Catalog](#)  
Interface through which the user may create, drop, alter or query underlying databases, tables,

► def **close**(): Unit  
Synonym for stop().

► lazy val **conf**: [RuntimeConfig](#)  
Runtime configuration interface for Spark.

► def **createDataFrame**(data: List[Row], baseClass: Class[Row], DataFormat: DataFormat): DataFrame

Figure 2: SparkSession class

```
val GhandiQuote =  
  """Live as if you were to die tomorrow  
    |Learn as if you were to live forever""".stripMargin  
  
def readLinesFromString(input: String): Seq[String] = {  
  val lines = input.split("\n")  
  lines  
}  
...  
readLinesFromString(GhandiQuote)
```

<b>String[]</b>	<b>split(String regex, int limit)</b> Splits this string around matches of the given <b>regular expression</b> .
<b>boolean</b>	<b>startsWith(String prefix)</b> Tests if this string starts with the specified prefix.
<b>boolean</b>	<b>startsWith(String prefix, int toffset)</b> Tests if the substring of this string beginning at the specified offset starts with the specified prefix.
<b>CharSequence</b>	<b>subSequence(int beginIndex, int endIndex)</b> Returns a character sequence that is a subsequence of this string.
<b>String</b>	<b>substring(int beginIndex)</b> Returns a string that is a substring of this string.
<b>String</b>	<b>substring(int beginIndex, int endIndex)</b> Returns a string that is a substring of this string.

Figure 4: Java String API



## Scala Predef API - implicit conversions

tests an expression, throwing an `ASSERTIONError` if false.

```
implicit def augmentString(x: String): StringOps
```

```
implicit def boolean2Boolean(x: Boolean): java.lang.Boolean
```

```
implicit def booleanArrayOps(xs: Array[Boolean]): ArrayOps[Boolean]
```

```
implicit def booleanWrapper(x: Boolean): RichBoolean
```

```
implicit def byte2Byte(x: Byte): java.lang.Byte
```

```
implicit def byteArrayOps(xs: Array[Byte]): ArrayOps[Byte]
```

```
implicit def byteWrapper(x: Byte): RichByte
```

We prefer the java.lang.\* boxed types to these wrappers in any potential conflict.

```
implicit def char2Character(x: Char): Character
```

```
implicit def charArrayOps(xs: Array[Char]): ArrayOps[Char]
```

```
implicit def charWrapper(c: Char): RichChar
```

Figure 5: Scala Predef API

## Scala StringOps API - stripMargin

Defines the prefix of this object's toString representation.

▶	def <b>stripLineEnd</b> : <a href="#">String</a> Strip trailing line end character from this string if it has one.
▶	def <b>stripMargin</b> : <a href="#">String</a> For every line in this string:
▶	def <b>stripMargin</b> (marginChar: <a href="#">Char</a> ): <a href="#">String</a> For every line in this string:
▶	def <b>stripPrefix</b> (prefix: <a href="#">String</a> ): <a href="#">String</a> Returns this string with the given prefix stripped.
▶	def <b>stripSuffix</b> (suffix: <a href="#">String</a> ): <a href="#">String</a> Returns this string with the given suffix stripped.
▶	def <b>subSequence</b> (arg0: <a href="#">Int</a> , arg1: <a href="#">Int</a> ): <a href="#">CharSequence</a>

Figure 6: Scala StringOps API

## HelloSparkWorld - accessing Java API/libraries

```
import java.nio.file.Files
import java.nio.file.Paths
import java.util.{List => JavaList}

import scala.collection.JavaConverters._

def readLinesFromFile(inputFile: String): Seq[String] = {
  val inputPath = Paths.get(inputFile)
  val linesJava: JavaList[String] =
    Files.readAllLines(inputPath)
  val lines = linesJava.asScala //mutable.Buffer
  lines
}
```

## wordCountLocal: map higher-order function w/named function

```
def wordCountLocal(lines: Seq[String]): Unit = {  
  
  def toLower(s: String): String = {  
    s.toLowerCase  
  }  
  
  val lowerLines = lines.map(toLower)  
  ...  
}
```

```
//function literal - anonymous function explicit type:
```

```
lines.map((l: String) => l.toLowerCase)
```

```
//function literal - anonymous with inferred type:
```

```
lines.map(l => l.toLowerCase)
```

```
//function literal with placeholder syntax
```

```
lines.map(_.toLowerCase)
```

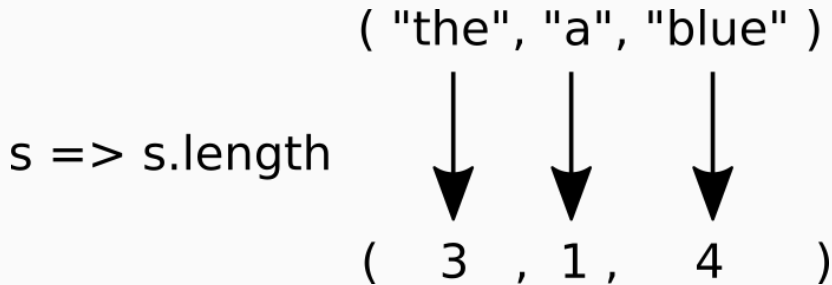


Figure 7: Map

flatMap function: `s => s.split("\\s+")`

Original: `("the quick", "brown fox")`

After map: `(Array("the", "quick"), Array("brown", "fox"))`

After flatten: `("the", "quick", "brown", "fox")`

Figure 8: flatMap

```
val words = lowerLines.flatMap { line =>  
    line.split("""\s+""")  
}
```

```
val noStopWords = words.filter(!StopWords.contains(_))
```



The equals method for arbitrary sequences.

```
def exists(p: (A) => Boolean): Boolean
```

Tests whether a predicate holds for at least one element of this iterable collection.

```
def filter(p: (A) => Boolean): Seq[A]
```

Selects all elements of this traversable collection which satisfy a predicate.

```
def filterNot(p: (A) => Boolean): Seq[A]
```

Selects all elements of this traversable collection which do not satisfy a predicate.

```
def find(p: (A) => Boolean): Option[A]
```

Finds the first element of the iterable collection satisfying a predicate, if any.

```
def flatMap[B](f: (A) => GenTraversableOnce[B]): Seq[B]
```

[use case] Builds a new collection by applying a function to all elements of this sequence and u

```
def flatten[R]: Seq[R]
```

Figure 9: Scala Seq

```
val emptyMapWithZeroDefault =  
  Map[String, Int]().withDefaultValue(0)  
  
//foldLeft(z: B)((B,A) => B): B  
val wordCountsMap: Map[String, Int] =  
  noStopWords.foldLeft(emptyMapWithZeroDefault)(  
    (wcMap, word) => {  
      val newCount = wcMap(word) + 1  
      wcMap + (word -> newCount)  
    })
```

```
val countsString = wordCountsMap.mkString("\n", "\n", "\n")  
println(s"The word counts were: ${countsString}")
```

## HelloSparkWorld - RDD map, flatMap, filter

```
//val mixedLinesRdd = spark.read.textFile(inputPath).rdd
val sc = spark.sparkContext

val mixedLinesRdd: RDD[String] =
    sc.parallelize(seq = lines, numSlices = 2)

val lowerLinesRdd = mixedLinesRdd.map(_.toLowerCase)

val wordsRdd = lowerLinesRdd.flatMap(_.split("""\s+"""))

val noStopWordsRdd = wordsRdd.filter(!StopWords.contains(_))
```

```
//Don't use groupBy - expensive to shuffle words across  
  ↪ partition!  
  
val wordCountTuplesRdd = noStopWordsRdd.map { (_, 1) }  
val wordCountsRdd = wordCountTuplesRdd.reduceByKey(_ + _)  
  
//and Action!  
val localWordCounts = wordCountsRdd.collect()
```