Scala for Apache Spark

Markus Dale, medale@asymmetrik.com

Jan 2019

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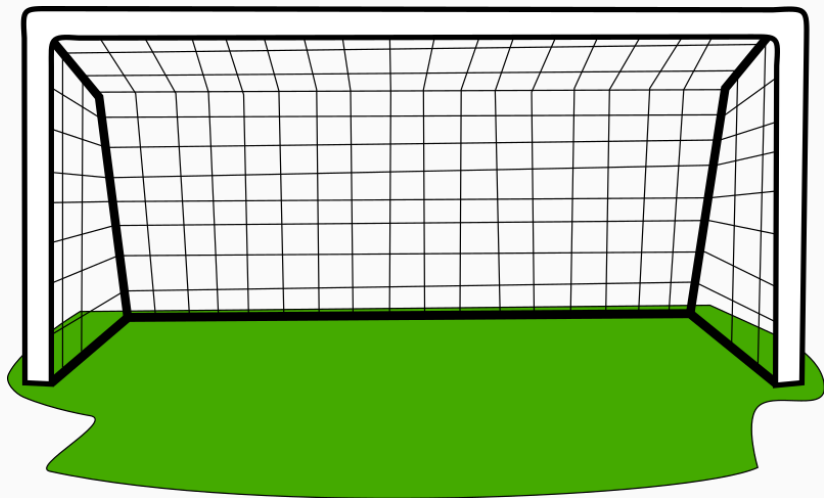- Scala Spark Code Examples: https://github.com/medale/scala-spark

Figure 1: Intro to Scala for Spark

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

```java
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
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)
    val answer = scMain.answer
    println(s"The answer was ${answer}")
  }
}
```
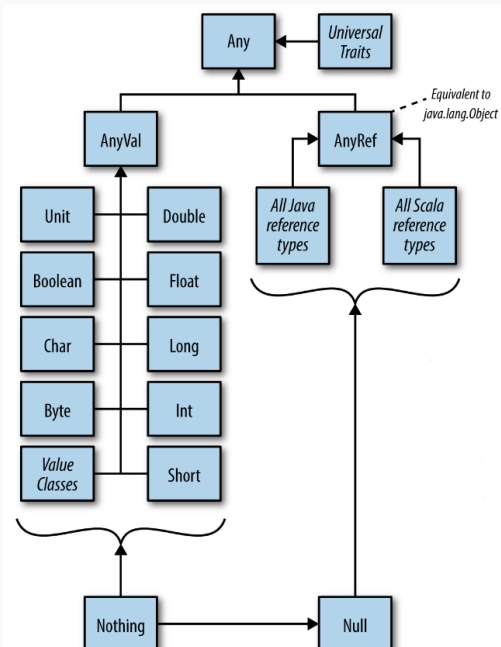
```scala
object HelloSparkWorld {
...
  def main(args: Array[String]): Unit = {

    val lines = if (!args.isEmpty) {
      val inputFile = args(0)
      readLinesFromFile(inputFile)
    } else {
      readLinesFromString(GhandiQuote)
    }
    ...
}
```

Scala Type Hierarchy

```scala
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()
}
```

Figure 2: SparkSession class

```scala
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)
```

Figure 4: Java String API

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 co

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

Figure 6: Scala StringOps API

```scala
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

```scala
def wordCountLocal(lines: Seq[String]): Unit = {

  def toLower(s: String): String = {
    s.toLowerCase
  }

  val lowerLines = lines.map(toLower)
  ...
```

wordCountLocal: map higher-order function w/ function literal

```scala
//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)
```

("the", "a", "blue")

s => s.length

(3 , 1, 4)
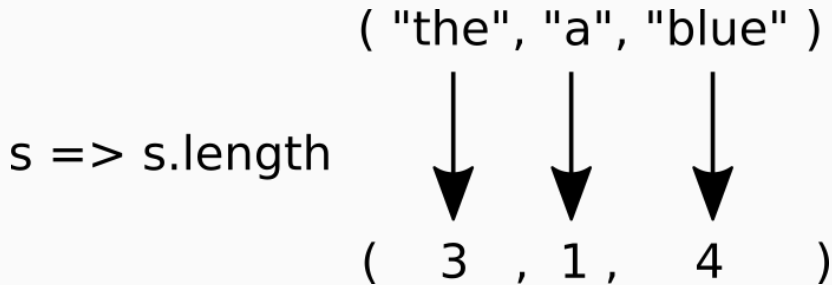
Figure 7: Map

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) ⇒ <u>Boolean</u>): <u>Boolean</u>
  Tests whether a predicate holds for at least one element of this iterable collection.

def **filter**(p: (A) ⇒ <u>Boolean</u>): <u>Seq</u>[A]
  Selects all elements of this traversable collection which satisfy a predicate.

def **filterNot**(p: (A) ⇒ <u>Boolean</u>): <u>Seq</u>[A]
  Selects all elements of this traversable collection which do not satisfy a predicate.

def **find**(p: (A) ⇒ <u>Boolean</u>): <u>Option</u>[A]
  Finds the first element of the iterable collection satisfying a predicate, if any.

def **flatMap**[B](f: (A) ⇒ <u>GenTraversableOnce</u>[B]): <u>Seq</u>[B]
  [use case] Builds a new collection by applying a function to all elements of this sequence and u

def **flatten**[B]: Seq[B]

Figure 9: Scala Seq

```scala
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)
    })
```

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

```scala
//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(_))
```

```scala
 //groupBy - expensive to shuffle words across partition!
val wordCountTuplesRdd = noStopWordsRdd.map { (_, 1) }

val wordCountsRdd = wordCountTuplesRdd.reduceByKey(_ + _)

//and Action!
val localWordCounts = wordCountsRdd.collect()
```

Figure 10: RDD object API

```scala
case class Person(firstName: String,
                  lastName: String,
                  age: Int)
```

```
public class Person implements Product,Serializable {
  public static Option<Tuple3<String,String,Object>>
  ↪   unapply(Person);
  public static Person apply(String, String, int);
...
  public String firstName();
  public String lastName();
  public int age();
...
  public Person copy(String, String, int);
...
  public String productPrefix();
  public int productArity();
  public Object productElement(int);
  public Iterator<Object> productIterator();
...
  public int hashCode();
```

28

```scala
//Person.apply("John...
val persons = List(Person("John","Doe",42),
      Person("Jane","Doe",43))

//createDataset[T : Encoder](data: Seq[T]): Dataset[T]
import spark.implicits._
val people: Dataset[Person] =
↪  spark.createDataset(persons)

//people.where(people("age").>(olderCutoff))
val youngers: Dataset[Row] = people.
  where($"age" < ageCutoff).
  select("firstName")

youngers.count
```

▶ def from_utc_timestamp(ts: Column, tz: String): Column
Given a timestamp like '2017-07-14 02:40:00.0', interprets it as a time i

▶ def hour(e: Column): Column
Extracts the hours as an integer from a given date/timestamp/string.

▶ def last_day(e: Column): Column
Returns the last day of the month which the given date belongs to.

▶ def minute(e: Column): Column
Extracts the minutes as an integer from a given date/timestamp/string.

▶ def month(e: Column): Column
Extracts the month as an integer from a given date/timestamp/string.

▶ def months_between(end: Column, start: Column, roundOff:
Returns number of months between dates end and start.

▶ def months_between(end: Column, start: Column): Column
Returns number of months between dates start and end.

▶ def next_day(date: Column, dayOfWeek: String): Column
Returns the first date which is later than the value of the date column

▶ def quarter(e: Column): Column

Figure 11: Spark SQL functions

```scala
//load initial dataset (file, directory...)
def createPersonDataset(spark: SparkSession,
                        persons: Seq[Person]):
↪ Dataset[Person]

def countAgeLessThanCutoff(spark: SparkSession,
                          people: Dataset[Person],
                          ageCutoff: Int = 42): Long
```

```scala
class HelloSparkDatasetWorldIntegrationTest extends
  FunSpec with Matchers with DatasetSuiteBase {
...
  describe("countAgeLessThanCutoff") {
    def assertExpectedCountForCutoff(ageCutoff: Int,
                        expectedCount: Int): Assertion...
    it("should return count = all for high cutoff") {
      val ageCutoff = 99
      val expectedCount = MyPersons.size
      assertExpectedCountForCutoff(ageCutoff, expectedCount)
    }
    it("should return count = 0 for low cutoff")...
    it("should return count = 2 for cutoff of 43")...
```

```scala
def assertExpectedCountForCutoff(ageCutoff: Int,
    expectedCount: Int): Assertion = {
  val people = HelloSparkDatasetWorld.
    createPersonDataset(spark, MyPersons)

  val actualCount = HelloSparkDatasetWorld.
    countAgeLessThanCutoff(spark, people, ageCutoff)

  actualCount should equal (expectedCount)
}
```

sbt - "build tool for Scala, Java and more"

```
- build.sbt
- version.sbt
- project/
    - build.properties
    - plugins.sbt
    - Dependencies.scala
- common
    - src
        - it/resources
        - it/scala
        - main
        - test
    - target
        - scala-2.11/analytics-0.9.0-SNAPSHOT-fat.jar
        - scala-2.11/classes
```

```scala
import sbt._

object Dependencies {
  //match Spark's pom for dependencies!
  val sparkVersion = "2.4.0"

  lazy val commonDependencies = Seq(
    ("commons-io" % "commons-io" % "2.4")
  )

  lazy val sparkDependencies = Seq(
    ("org.apache.spark" %% "spark-core" % sparkVersion)
    ...
```

sbt - build.sbt

```scala
ThisBuild / scalaVersion := "2.11.8"
...
lazy val analytics = project
  .dependsOn(common)
  .configs(IntegrationTest)
  .settings(Defaults.itSettings: _*)
  .settings(
    name := s"${namePrefix}-analytics",
    libraryDependencies :=
      commonDependencies ++
        sparkDependencies ++
        testDependencies ++
        sparkTestDependencies
  )
lazy val root = (project in file("."))
  .aggregate(common, analytics)
...
```

sbt tasks

```
sbt
sbt:root> compile
sbt:root> test
sbt:root> analytics/it:test
sbt:root> common/console
sbt:root> common/run
sbt:root> assembly
sbt:root> publish
```

- Screening saves lives!
  - Colonoscopy - talk to your doc
  - Dave Barry: A journey into my colon — and yours
- Colorectal Cancer Alliance

- Dean Wampler, Alex Payne, "Programming Scala, 2nd Edition", O'Reilly, 2014
- Jacek Laskowski, Mastering Spark SQL, Gitbook
- ScalaTest
- Holden Karau Spark Testing Base
- sbt reference manual

- medale@asymmetrik.com
- Infrequent blog/past presentations http://uebercomputing.com/
- Scala Spark repo: https://github.com/medale/scala-spark