Scala for Apache Spark

Markus Dale, medale@asymmetrik.com

Jan 2019

Intro, Slides And Code

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

Goals

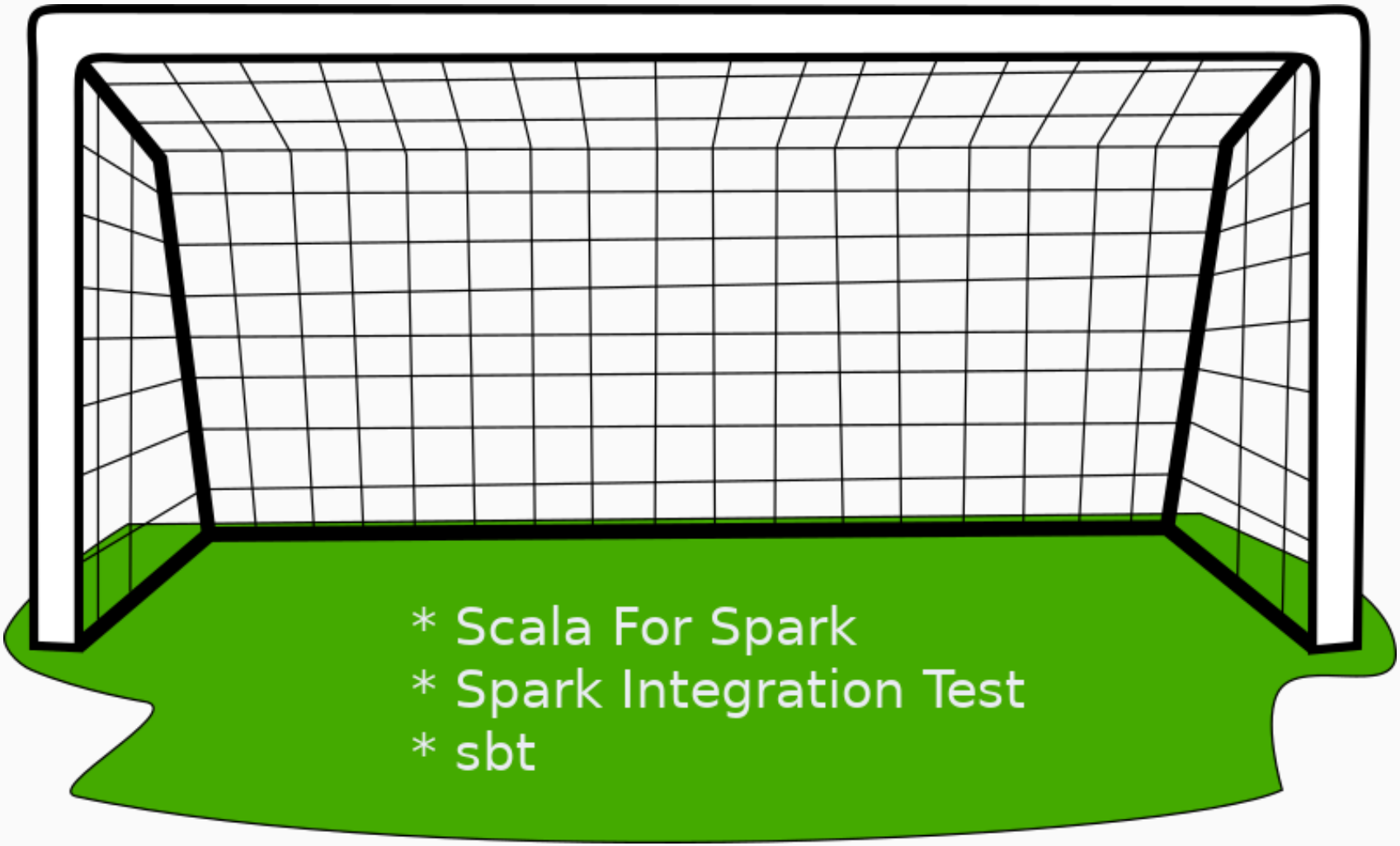
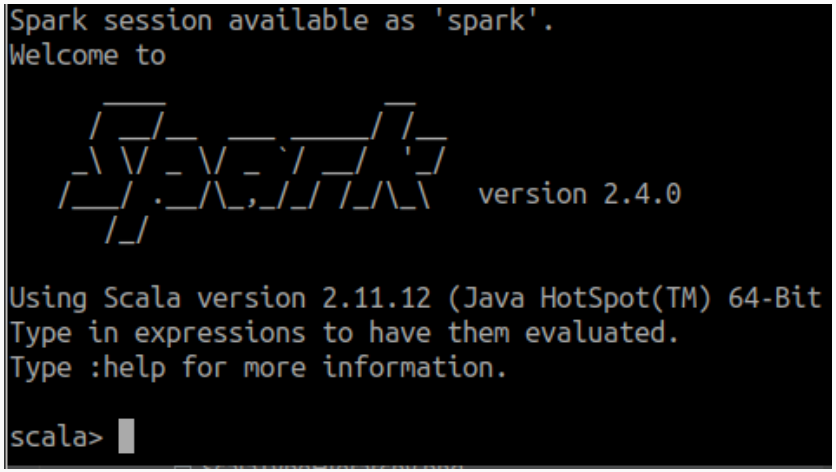


Figure 1: Intro to Scala for Spark

Why Scala for Spark?



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

Java to Scala - Java Main

```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 Main One

```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}")
  }
}
```

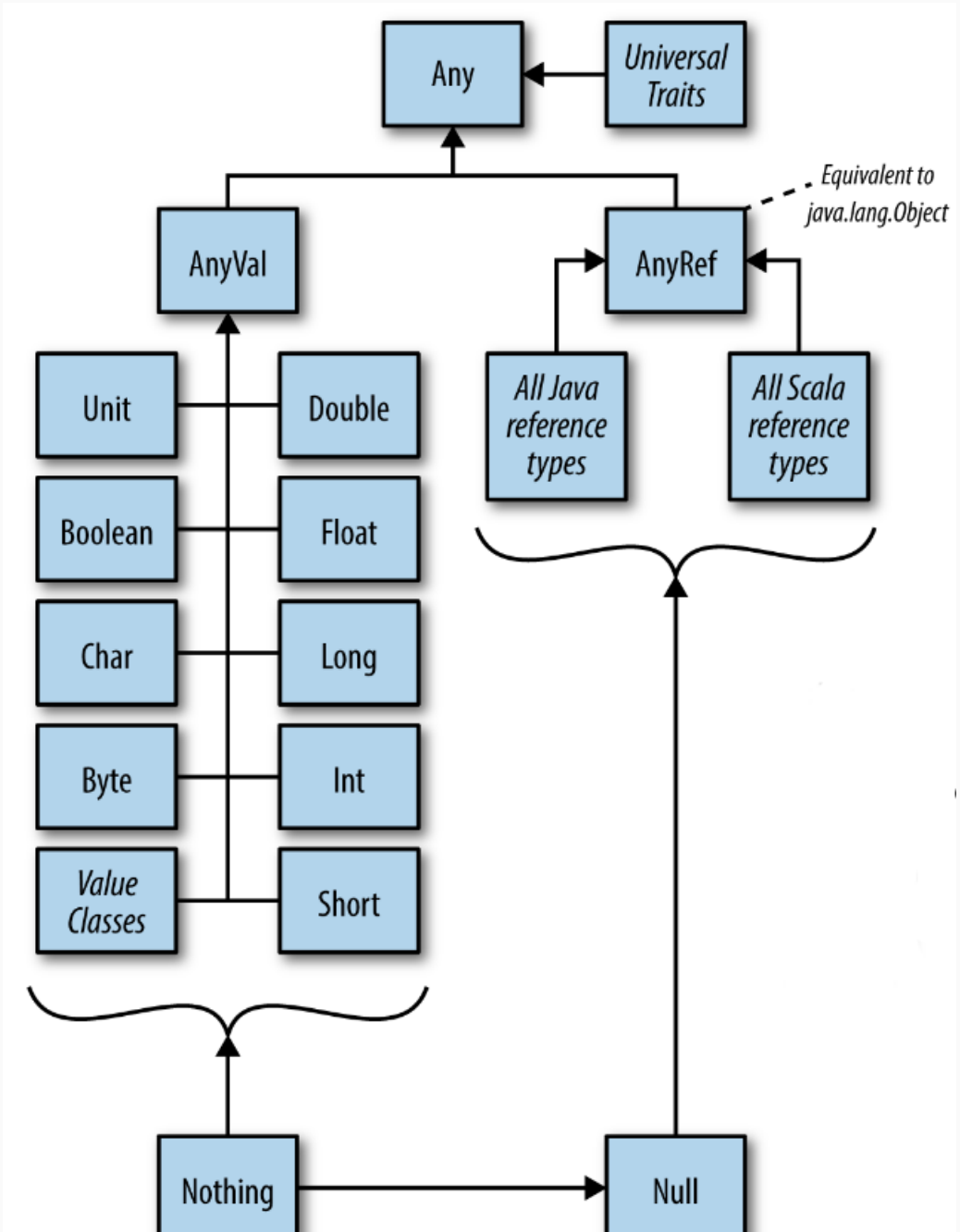
HelloSparkWorld - expression-oriented

```
object HelloSparkWorld {
...
  def main(args: Array[String]): Unit = {

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

    wordCountLocal(lines)
      ...
}
```

Scala Type Hierarchy



HelloSparkWorld - String, StringOps, implicits

```
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)
```

Java API - String

`String[]`	**`split(String regex, int limit)`** Splits this string around matches of the given **regular exp**
`boolean`	**`startsWith(String prefix)`** Tests if this string starts with the specified prefix.
`boolean`	**`startsWith(String prefix, int toffset)`** Tests if the substring of this string beginning at the specif
`CharSequence`	**`subSequence(int beginIndex, int endIndex)`** Returns a character sequence that is a subsequence of thi
`String`	**`substring(int beginIndex)`** Returns a string that is a substring of this string.
`String`	**`substring(int beginIndex, int endIndex)`** Returns a string that is a substring of this string.

Figure 3: 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 co

```
implicit def char2Character(x: Char): Character
implicit def charArrayOps(xs: Array[Char]): ArrayOps[Char]
implicit def charWrapper(c: Char): RichChar
```

Figure 4: Scala Predef API

Scala StringOps API - stripMargin

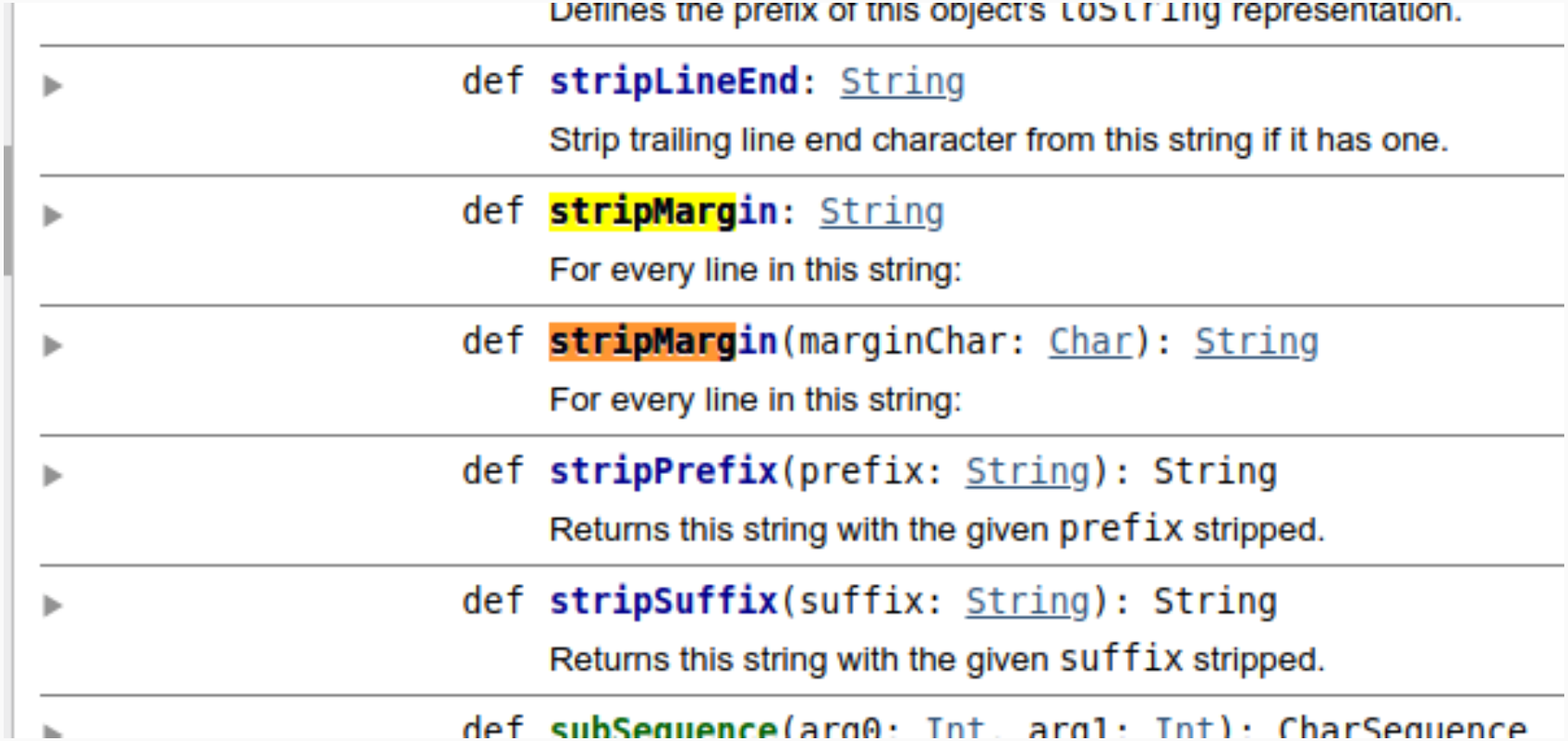


Figure 5: Scala StringOps API

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)
  ...
```

wordCountLocal: map higher-order function w/ function literal

```
//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)
```

map function

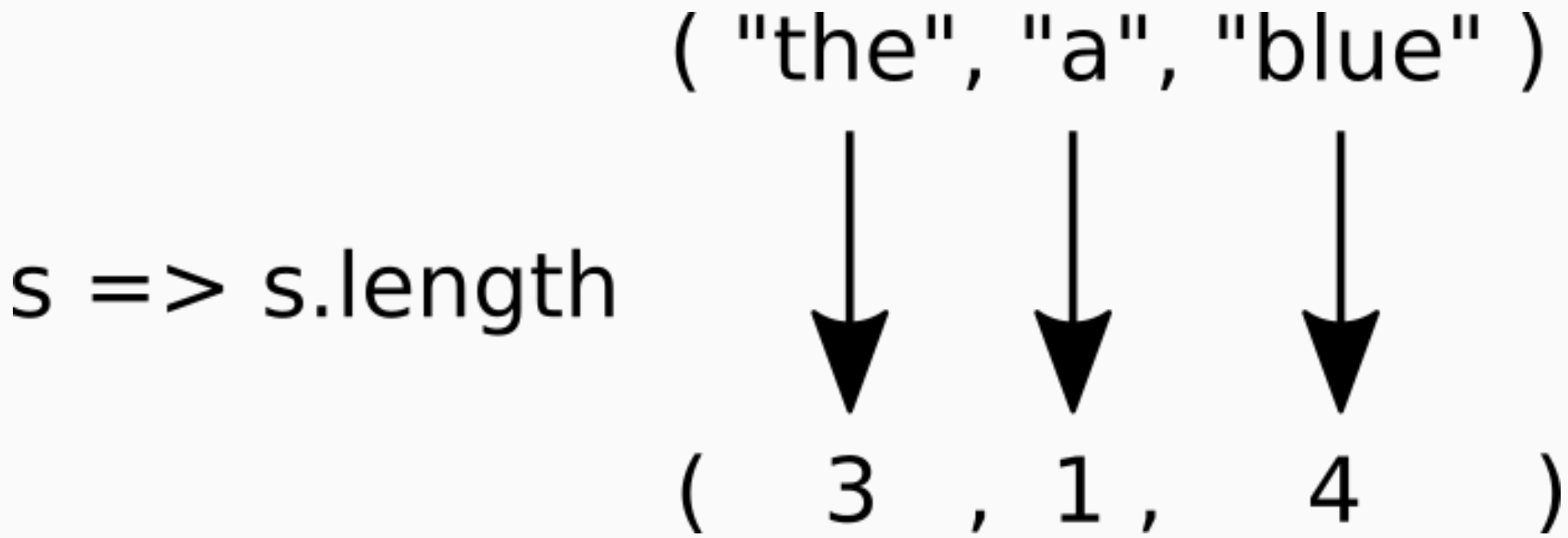


Figure 6: Map

```
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 7: flatMap

wordCountLocal: flatMap, filter

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

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

Scala Seq trait API

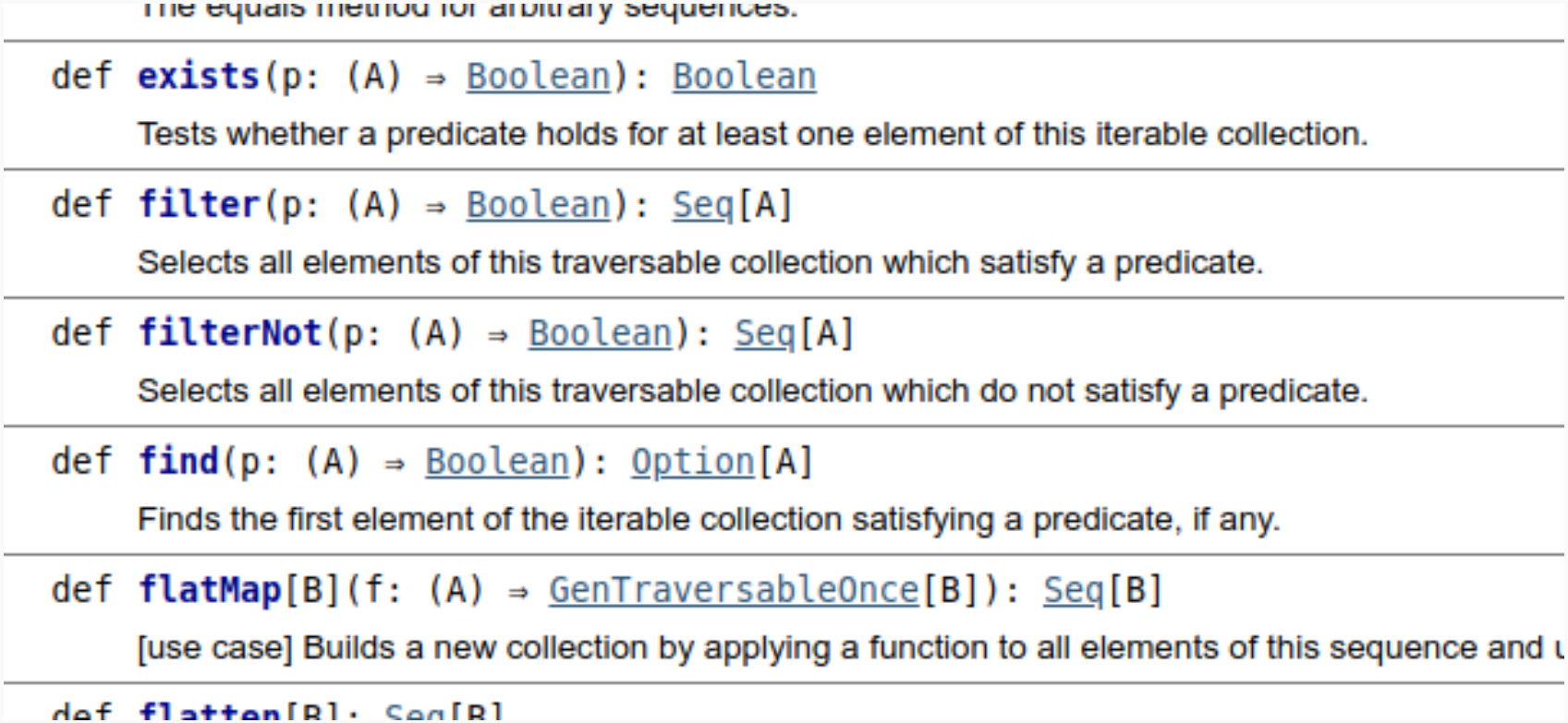


Figure 8: Scala Seq

wordCountLocal: foldLeft

```
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)
    })
```

wordCountLocal: mkString

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

```
The word counts were:
learn -> 1
if -> 2
as -> 2
you -> 2
die -> 1
...
```

HelloSparkWorld - SparkSession

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

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

   wordCountRdd(spark, lines)

   spark.close()
}
```

SparkSession Scala API

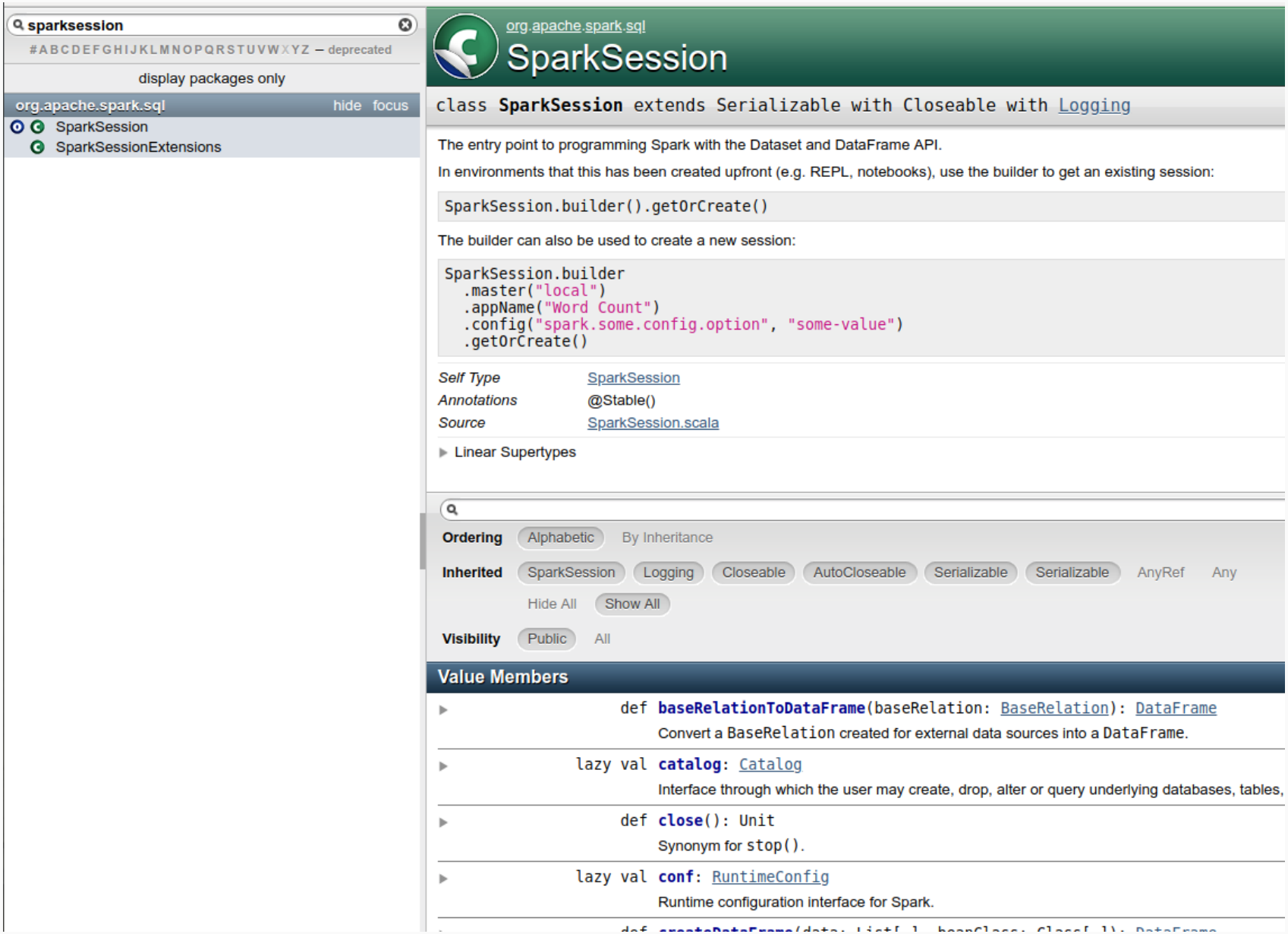


Figure 9: SparkSession class

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(_))
```

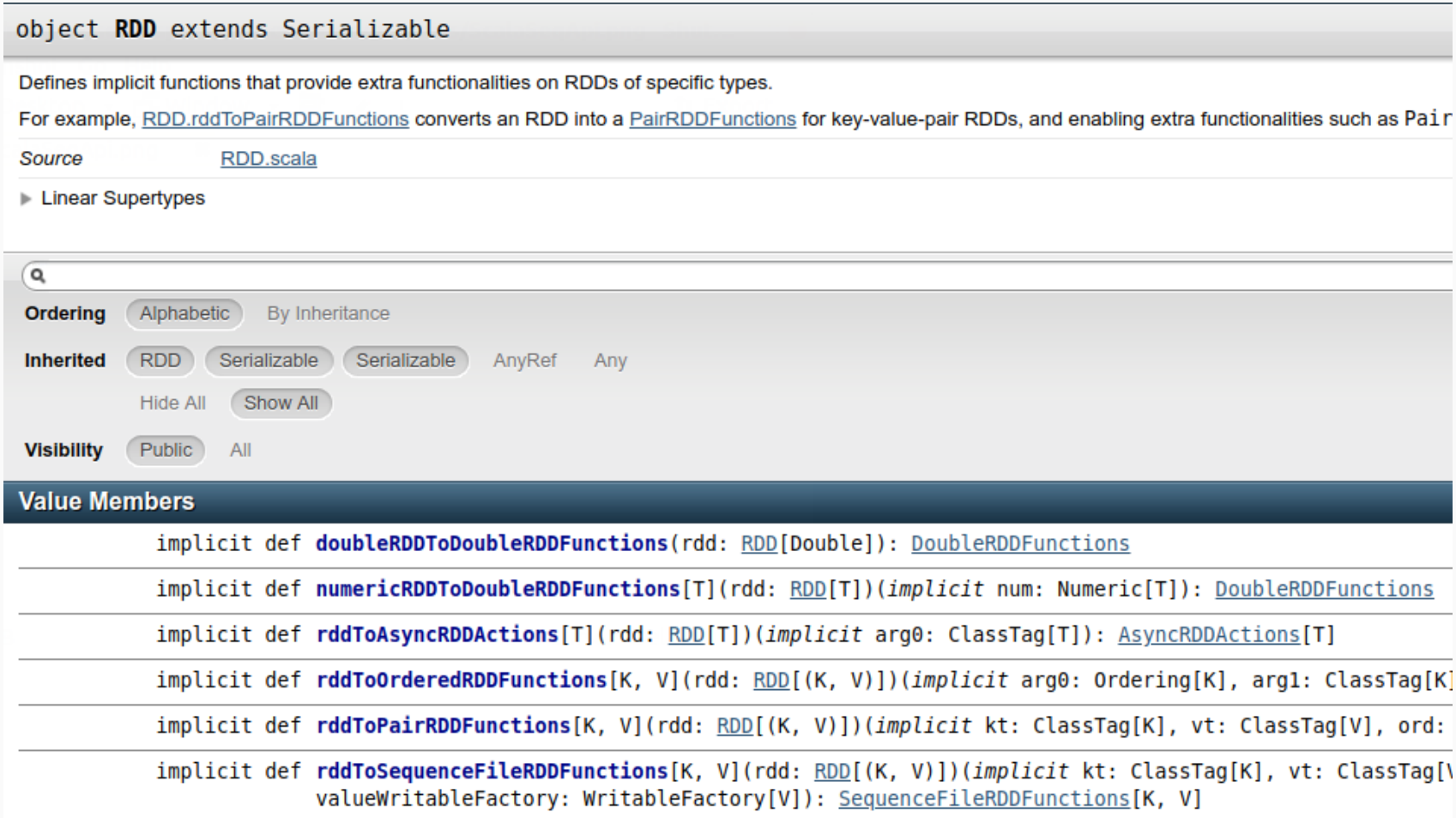
HelloSparkWorld - RDD of tuples - PairRDDFunctions

```
val wordCountTuplesRdd = noStopWordsRdd.map { (_, 1) }

//No groupBy - expensive to shuffle words!
val wordCountsRdd = wordCountTuplesRdd.reduceByKey(_ + _)

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

RDD object API



object **RDD** extends Serializable

Defines implicit functions that provide extra functionalities on RDDs of specific types.

For example, RDD.rddToPairRDDFunctions converts an RDD into a PairRDDFunctions for key-value-pair RDDs, and enabling extra functionalities such as Pair

Source RDD.scala

▶ Linear Supertypes

Ordering Alphabetic By Inheritance

Inherited RDD Serializable Serializable AnyRef Any

Hide All Show All

Visibility Public All

Value Members

```
implicit def doubleRDDToDoubleRDDFunctions(rdd: RDD[Double]): DoubleRDDFunctions
implicit def numericRDDToDoubleRDDFunctions[T](rdd: RDD[T])(implicit num: Numeric[T]): DoubleRDDFunctions
implicit def rddToAsyncRDDActions[T](rdd: RDD[T])(implicit arg0: ClassTag[T]): AsyncRDDActions[T]
implicit def rddToOrderedRDDFunctions[K, V](rdd: RDD[(K, V)])(implicit arg0: Ordering[K], arg1: ClassTag[K]
implicit def rddToPairRDDFunctions[K, V](rdd: RDD[(K, V)])(implicit kt: ClassTag[K], vt: ClassTag[V], ord:
implicit def rddToSequenceFileRDDFunctions[K, V](rdd: RDD[(K, V)])(implicit kt: ClassTag[K], vt: ClassTag[V
    valueWritableFactory: WritableFactory[V]): SequenceFileRDDFunctions[K, V]
```

Figure 10: RDD object API

HelloSparkDatasetWorld - Scala case class

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

HelloSparkDatasetWorld - javap Person.class

```
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();
```

HelloSparkDatasetWorld - Encoder,Column

```
//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
```

Column-based: org.apache.spark.sql.functions._

Figure 11: Spark SQL functions

Integration Testing - make code testable as you write

```
//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
```

Integration Testing - ScalaTest with Spark Testing Base

```
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")...
```

Integration Testing - assertExpectedCountForCutoff

```
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
```

sbt - Dependencies.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

```
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
```

And now for something completely different: Colon Cancer

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

Resources

- 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

Questions?

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