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

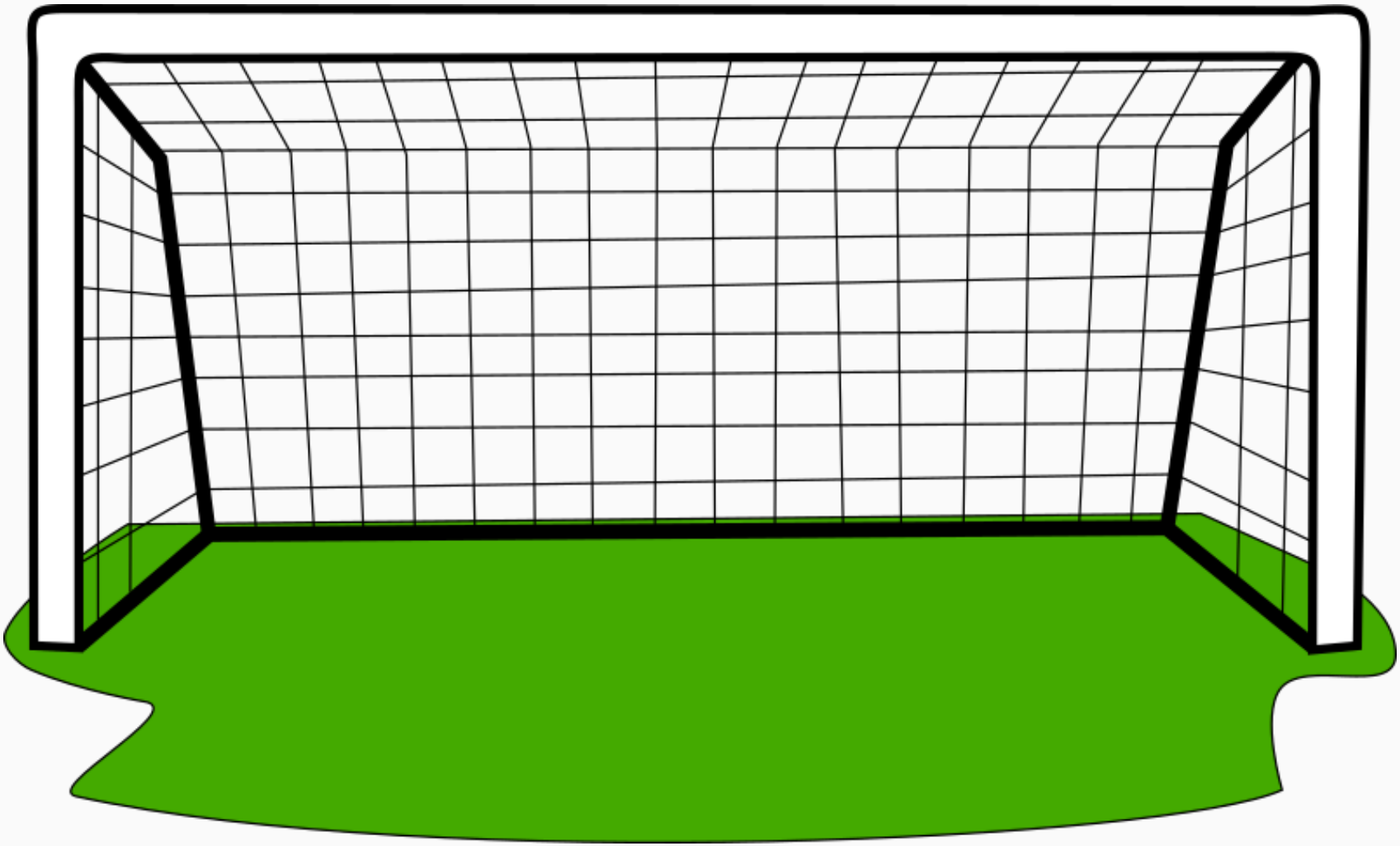


Figure 1: Intro to Scala for Spark

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)
    println(scMain)
    val answer = scMain.answer
    println(s"The answer was ${answer}")
  }
}
```

Scala Main One - Output

```
Starting a Scala program...
com.uebercomputing.scalaspark.common.ScalaMainOne@256216b3
The answer was 42
```

Scala Main Two - case class

```scala
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}")
  }

}
```

Scala Main Two - Output

```
Starting a Scala program...
ScalaMainTwo(42)
The answer was 42
```

Scala Main Two - javap ScalaMainTwo.class

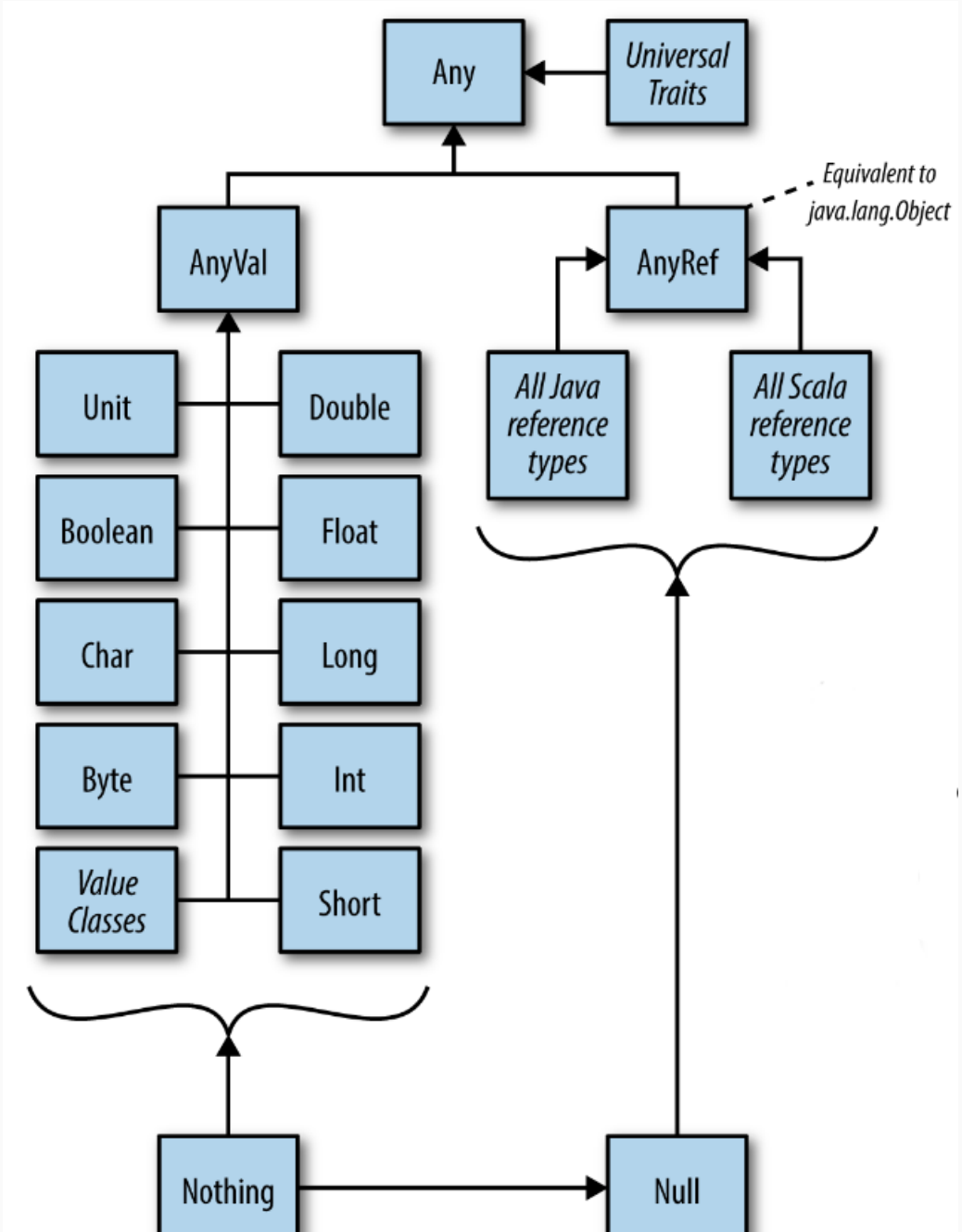
```
public class ScalaMainTwo implements Product,Serializable
  public static Option<Object> unapply(ScalaMainTwo);
  public static ScalaMainTwo apply(int);
...
  public ScalaMainTwo copy(int);
...
  public int productArity();
  public Object productElement(int);
  public Iterator<Object> productIterator();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

HelloSparkWorld - expression-oriented

```
object HelloSparkWorld {
...
  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

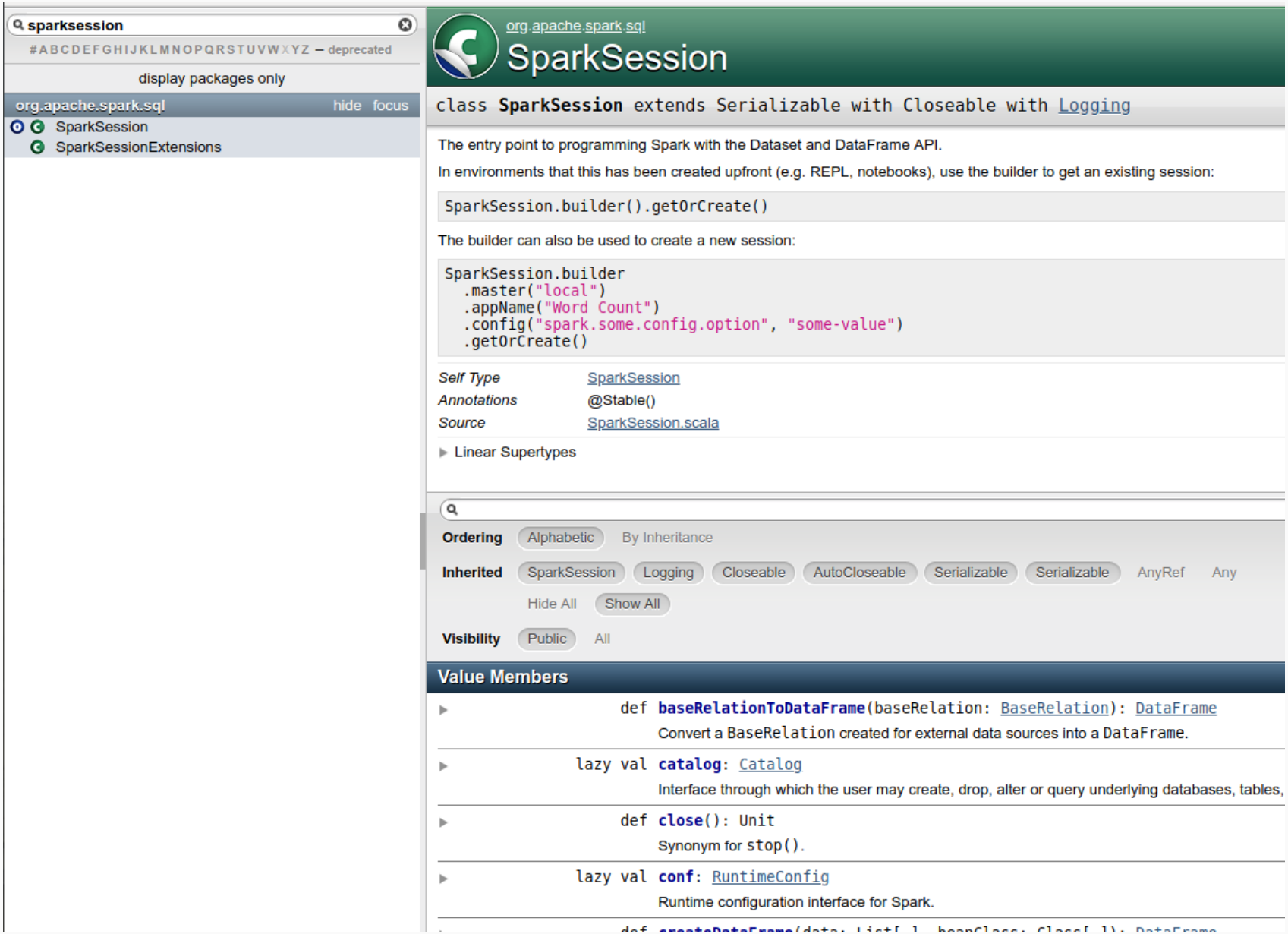


Figure 2: SparkSession class

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 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 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

Scala StringOps API - stripMargin

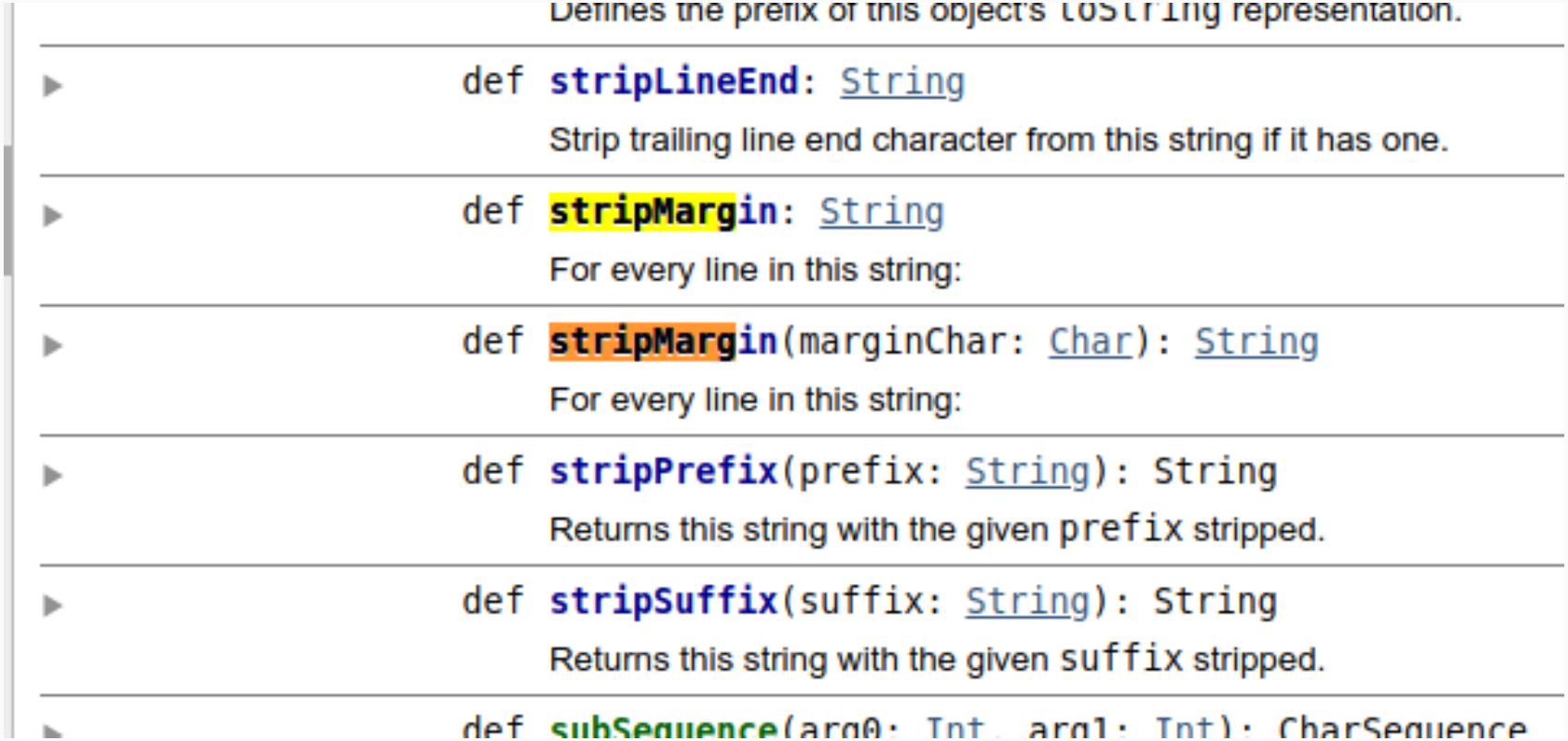


Figure 6: Scala StringOps API

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)
  ...
```

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

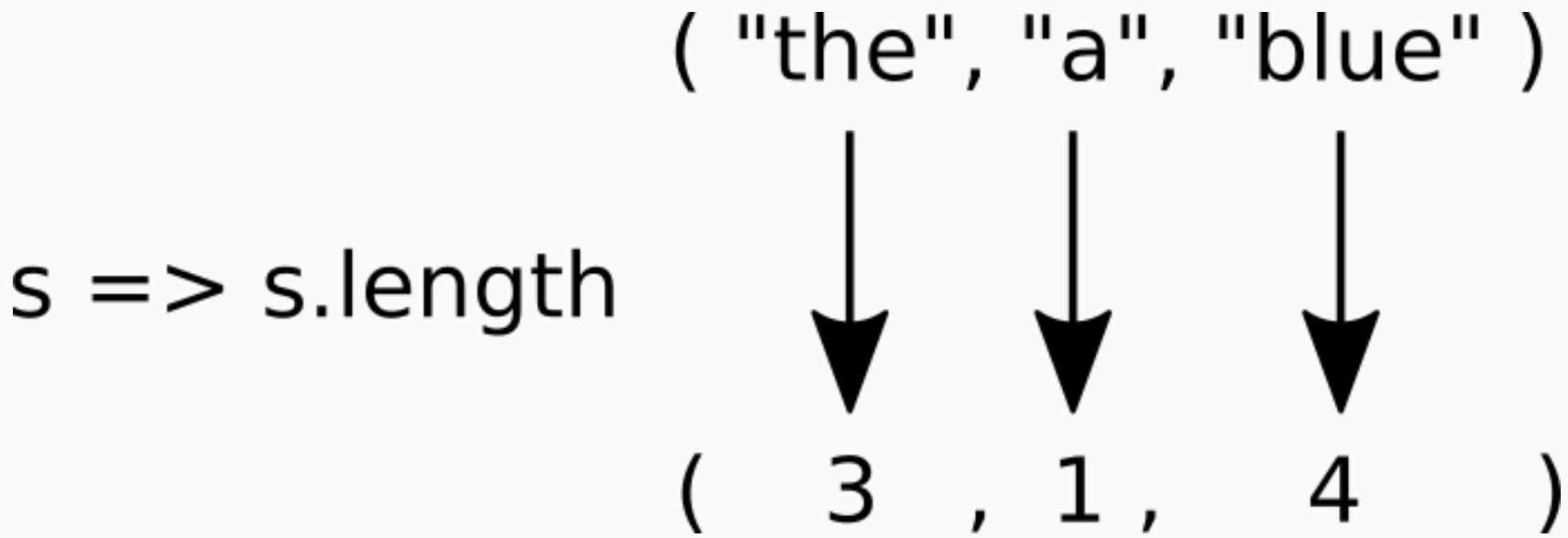


Figure 7: Map

```
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

wordCountLocal: flatMap, filter

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

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

Scala Seq trait API

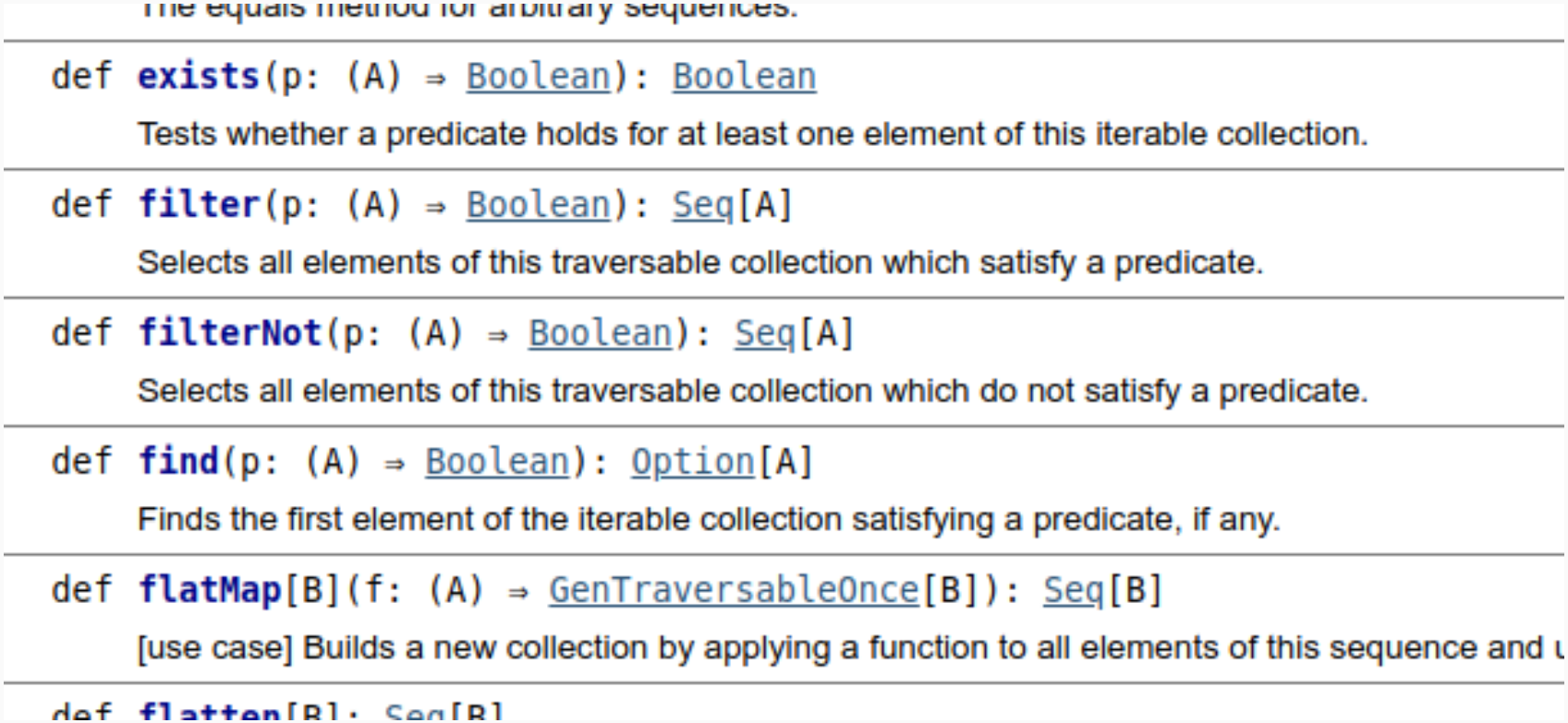


Figure 9: Scala Seq

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}")
```

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

```
 //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()
```