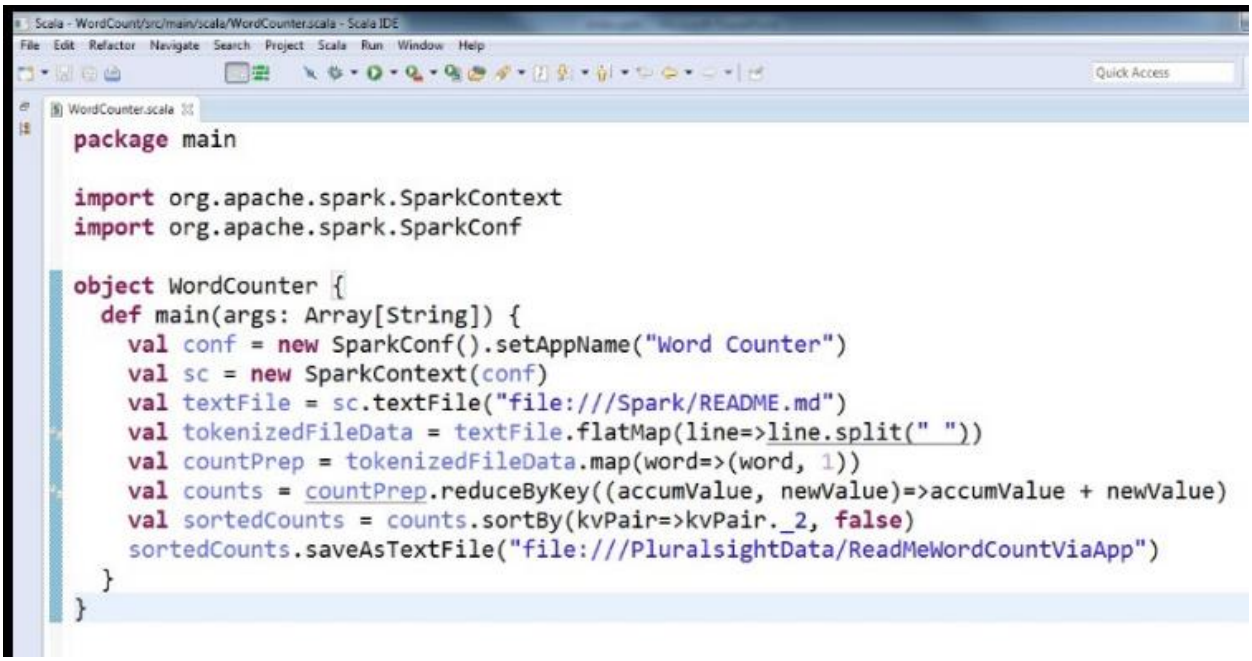
SPARK WC

```
scala> val textFile = sc.textFile("file:///Spark/README.md")
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile at <console>:21

la> val countPrep = tokenizedFileData.map(word=>(word, 1))
ntPrep: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:25

la> val counts = countPrep.reduceByKey((accumValue, newValue)=>accumValue + newValue)
nts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:27

la> val sortedCounts = counts.sortBy(kvPair=>kvPair._2, false)
tedCounts: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[9] at sortBy at <console
9

la> sortedCounts.saveAsTextFile("file:///PluralsightData/ReadMeWordCount")
```

```
ala> tokenizedFileData.countByValue
s2: scala.collection.Map[String,Long] = Map(site, -> 1, Please -> 3, GraphX -> 1, application -
1, "" -> 66, for -> 11, find -> 1, Apache -> 1, works -> 1, package -> 1, Hadoop, -> 2, Once ->
, For -> 2, name -> 1, this -> 1, protocols -> 1, Hive -> 2, in -> 5, "local[N]" -> 1, MASTER=s
rk://host:7077 -> 1, have -> 1, your -> 1, are -> 1, is -> 6, HDFS -> 1, Data. -> 1, built -> 1
thread, -> 1, examples -> 2, using -> 2, system -> 1, Shell -> 2, mesos:// -> 1, easiest -> 1,
is -> 2, [Apache -> 1, N -> 1, <class> -> 1, different -> 1, "local" -> 1, README -> 1, online
 1, spark:// -> 1, return -> 2, Note -> 1, if -> 4, project -> 1, Scala -> 2, You -> 3, running
> 1, usage -> 1, versions -> 1, uses -> 1, must -> 1, do -> 2, all -> 1, programming -> 1, runs
-> 1, distribution ...
```

Resources

- https://spark.apache.org
  - /documentation.html
  - /docs/latest/
  - /community.html
  - /examples.html
- Learning Spark: Lightning-Fast Big Data Analysis by Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia
- https://www.typesafe.com/activator/templates#filter:spark
- https://github.com/apache/spark



RDD

EACH PIECE OF PARALLEL DATA IS RDDD THAT WILL BE USED IN LINEAGE IN DAG

DATA WILL BE LOADED USING PARALLELISE OR RANGE OR MAKERDD

```
cala> sc.parallelize(1 to 100)
es0: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:22

cala> res0.collect
es1: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45
 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93,
4, 95, 96, 97, 98, 99, 100)

cala> sc.parallelize


ef parallelize[T](seq: Seq[T], numSlices: Int)(implicit scala.reflect.ClassTag[T]): rdd.RDD[T]


cala> sc.parallelize
```

```
scala> sc.makeRDD

def makeRDD[T](seq: Seq[(T, Seq[String])])(implicit scala.reflect.ClassTag[T]): rdd.RDD[T]
def makeRDD[T](seq: Seq[T], numSlices: Int)(implicit scala.reflect.ClassTag[T]): rdd.RDD[T]

scala> sc.range(1, 100).collect
res2: Array[Long] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 4
5, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69
, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93,
 94, 95, 96, 97, 98, 99)
```

Sc.wholetextfile is used for loading all the files from a given path

```
scala> sc.wholeTextFiles

def wholeTextFiles(path: String, minPartitions: Int): rdd.RDD[(String, String)]

scala> sc.wholeTextFiles
```

We can also load sequence file and other files by importing hadoop io after we load using hadoop datatype we convert to conventional datatypes using .map

```
def wholeTextFiles(path: String, minPartitions: Int): rdd.RDD[(String, String)]

scala> import org.apache.hadoop.io._
import org.apache.hadoop.io._

scala> sc.sequenceFile("file:///Data/SampleSequenceFile", classOf[Text], classOf[IntWritable])
res3: org.apache.spark.rdd.RDD[(org.apache.hadoop.io.Text, org.apache.hadoop.io.IntWritable)] =
ile:///Data/SampleSequenceFile HadoopRDD[3] at sequenceFile at <console>:25

scala> .map(kv=>(kv._1.toString, kv._2.get))
res4: org.apache.spark.rdd RDD[(String, Int)] = MapPartitionsRDD[4] at map at <console>:27
```

We can also load sequence file by using as shown

```
def sequenceFile[K, V](path: String, keyClass: Class[K], valueClass: Class[V], minPartitions: Int
): rdd.RDD[(K, V)]

def sequenceFile[K, V](path: String, minPartitions: Int)(implicit km: scala.reflect.ClassTag[K],
vm: scala.reflect.ClassTag[V], kcf: () => org.apache.spark.WritableConverter[K], vcf: () => org.a
pache.spark.WritableConverter[V]): rdd.RDD[(K, V)]

scala> sc.sequenceFile[String, Int]("file:///Data/SampleSequenceFile")
res6: org.apache.spark.rdd RDD[(String, Int)] = MapPartitionsRDD[6] at sequenceFile at <console>:
25

scala>
```

LAMBDA EXPRESSION

Named Method

```
def addOne(item: Int) = {
  item + 1
}
```

```
val intList = List(1,2)

for(item <- intList) yield {
  addOne(item)
}
```

THIS METHOD CAN BE REWRITTEN BY USING LAMDA FUNCTION

Lambda Functions

```
def addOne(item: Int) = {
  item + 1
}
```

```
val intList = List(1,2)

intList.map(x => {
  addOne(x)
})
```

THE ITEMS ON THE LEFT SIDE ARE SIMILAR TO PARAMETERS AND THE RIGHT SIDE ARE THE CODE TO BLOCK AS SHOWN BELOW

Lambda Functions

```
val intList = List(1,2)

intList.map(item => item + 1)//List(2,3)
```

```
def addOne(item: Int) = {
  item + 1
}
val intList = List(1,2)
for(item <- intList) yield {
  addOne(item)
}//List(2,3)
```

RDD HAS 2 METHOD

1.TRANSFORMATION

2.ACTIONS

TRANSFORMATION:

IF AN RDD RETURNS ANOTHER RDD IT IS A TRANSFORMATION

```scala
val wikiDocuments = sc.hadoopRDD(jobConf,
  classOf[org.apache.hadoop.streaming.StreamInputFormat],
  classOf[Text], classOf[Text])

val deHadoopedWikis = wikiDocuments.map(hadoopXML=>hadoopXML._1.toString)

import scala.xml.XML
val rawWikiPages = deHadoopedWikis.map(wikiString=>{
  val wikiXML = XML.loadString(wikiString)
  val wikiPageText = (wikiXML \ "revision" \ "text").text
  WikiCleaner.parse(wikiPageText)
})

val tokenizedWikiData = rawWikiPages.flatMap(wikiText=>wikiText.split(" "))
val pertinentWikiData = tokenizedWikiData
                        .map(wikiToken => wikiToken.replaceAll("[.|,|'|\"|?|)|(]", "").trim)
                        .filter(wikiToken=>wikiToken.length > 2)

val wikiDataSortedByLength = pertinentWikiData.distinct
         .sortBy(wikiToken=>wikiToken.length, ascending = false)
         .sample withReplacement = false  fraction = .01)
```

We can use methods like map items and intersect union ,subtract cartestian method for combining data from various

Sources by using rdd.union

ACTION

IT SENDS THE RESULT TO AN DRIVER PROGRAM AND DOESNOT RESULT IN AN RDD

Evaluator.scala | HadoopWikiRDDGenerator.scala

```scala
val conf = new SparkConf().setAppName("Language Evaluator")
val sc = new SparkContext(conf)

val wikiDocuments = HadoopWikiRDDGenerator
                    .createUsing(sc, withPath = "file:///Data/WikiPages_BigData.xml")

val rawWikiPages = WikiCleaner.clean(wikiDocuments)

val tokenizedWikiData = rawWikiPages.flatMap(wikiText=>wikiText.split("\\W+"))

val pertinentWikiData = tokenizedWikiData
                        .map(wikiToken => wikiToken.replaceAll("[.|,|'|\"|?|)|(]", "").trim)
                        .filter(wikiToken=>wikiToken.length > 2)

val wikiDataSortedByLength = pertinentWikiData.distinct
         .sortBy(wikiToken=>wikiToken.length, ascending = false)
         .sample(withReplacement = false, fraction = .01)

wikiDataSortedByLength
  .collect
  .foreach(println)
}
```

→ These are Action

COLLECT THE MAP RESULT AND SEND IT TO DRIVER AS AN ARRAY

SINCE IN COLLECT METHOD EVERYTHING IS SENT TO DRIVER MEMORY OUT OF EXCEPTION MAY TAKES PLACE

WE CAN USE TAKE METHOD WHICK TAKES FIRST n ELEMENT FROM THE RESULT

WE CAN ALSO USE TAKESAMPLE,TAKEORDER AND TAKETOP METHODS INSTEAD OF COLLECT METHOD

See reduce method fold method aggregate method

Storing the data

Instead of sending the result to driver they can directly written to mongo db or hadoop

- saveAsObjectFile(path)
- saveAsTextFile(path)
- External connector
- foreach(T => Unit)
  - foreachPartition(Iterator[T] => Unit)

Resources

- RDD Research Paper
  - http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf
- Lambdas
  - What's New in Java 8: Jose Paumard
  - Functional Programming With Java: Jessica Kerr
- Add ALL the Things: Avi Bryant
  - http://www.infoq.com/presentations/abstract-algebra-analytics

Implicit Conversion

Instead of defining a class we can shorthand it by using implicit

Conversion which makes use of the parameters and does the operation

```
scala> case class IntExtensions(value: Int) {
     |   def plus(operand: Int) = value + operand
     | }
defined class IntExtensions

scala> IntExtensions(1).plus(1)
res1: Int = 2

scala> import scala.language.implicitConversions
import scala.language.implicitConversions

scala> implicit def intToIntExtensions(value: Int) = {
     |   IntExtensions(value)
     | }
intToIntExtensions: (value: Int)IntExtensions

scala> 1.plus(1)
res2: Int = 2
```

RDD Implicits

doubleRDDToDoubleRDDFunctions(rdd: RDD[Double]): DoubleRDDFunctions

numericRDDToDoubleRDDFunctions[T](rdd: RDD[T]): DoubleRDDFunctions

rddToAsyncRDDActions[T](rdd: RDD[T]): AsyncRDDActions[T]

rddToOrderedRDDFunctions[K,V](rdd: RDD[(K,V)]): OrderedRDDFunctions

rddToPairRDDFunctions[K,V](rdd: RDD[(K,V)]): PairRDDFunctions[K,V]

rddToSequenceFileRDDFunctions[K,V](rdd: RDD[(K,V)]): SequenceFileRDDFunctions[K,V]