快速开始

- 使用Spark Shell进行交互式分析
  - 基本
  - 有关数据集操作的更多信息
  - 高速缓存
- 自包含的应用程序
- 从这往哪儿走

本教程简要介绍了如何使用Spark。 我们将首先通过Spark的交互式shell（在Python或Scala中）介绍API，然后展示如何使用Java，Scala和Python编写应用程序。

要继续本指南，首先，从Spark网站下载Spark的打包版本。 由于我们不会使用HDFS，您可以下载任何版本的Hadoop的软件包。

请注意，在Spark 2.0之前，Spark的主要编程接口是弹性分布式数据集（RDD）。 在Spark 2.0之后，RDD被数据集取代，数据集类似于RDD一样强类型，但在底层有更丰富的优化。 仍然支持RDD接口，您可以在RDD编程指南中获得更详细的参考。 但是，我们强烈建议您切换到使用Dataset，它具有比RDD更好的性能。 请参阅SQL编程指南以获取有关数据集的更多信息。

使用Spark Shell进行交互式分析

基本

Spark的shell提供了一种学习API的简单方法，以及一种以交互方式分析数据的强大工具。 它可以在Scala（在Java VM上运行，因此是使用现有Java库的好方法）或Python中使用。 通过在Spark目录中运行以下命令来启动它：

Scala Python

```
./bin/spark-shell
```

Spark的主要抽象是一个名为Dataset的分布式项目集合。 可以从Hadoop InputFormats（例如HDFS文件）或通过转换其他数据集来创建数据集。 让我们从Spark源目录中的README文件的文本中创建一个新的数据集：

```
scala> val textFile = spark.read.textFile("README.md") textFile: org.apache.spark.sql.Dataset[String] = [value: string]
```

您可以通过调用某些操作直接从Dataset获取值，或者转换数据集以获取新值。 有关更多详细信息，请阅读*API文档*。

```
scala> textFile.count() // Number of items in this Dataset res0: Long = 126 // May be different from yours as README.md will change over time, simila
```

```
r to other outputs scala > textFile . first () // First item in this Dataset r
es1 : String = # Apache Spark
```

现在让我们将这个数据集转换为新数据集。 我们调用filter来返回一个新的数据集，其中包含文件中项目的子集。

```
  scala > val linesWithSpark = textFile . filter ( line => line . contains (
"Spark" )) linesWithSpark : org.apache.spark.sql.Dataset [ String ] = [ value:
 string ]
```

我们可以将转换和行动联系在一起：

```
  scala > textFile . filter ( line => line . contains ( "Spark" )). count ()
// How many lines contain "Spark"? res3 : Long = 15
```

有关数据集操作的更多信息

数据集操作和转换可用于更复杂的计算。 假设我们想要找到含有最多单词的行：

Scala Python

```
  scala > textFile . map ( line => line . split ( " " ). size ). reduce (( a ,
 b ) => if ( a > b ) a else b ) res4 : Long = 15
```

这首先将一行映射为整数值，从而创建一个新的数据集。 在该数据集上调用reduce以查找最大字数。 map和reduce的参数是Scala函数文字（闭包），可以使用任何语言特性或Scala / Java库。 例如，我们可以轻松调用其他地方声明的函数。 我们将使用Math.max()函数使这段代码更容易理解：

```
  scala > import java.lang.Math import java.lang.Math scala > textFile . map (
line => line . split ( " " ). size ). reduce (( a , b ) => Math . max ( a , b
)) res5 : Int = 15
```

一种常见的数据流模式是MapReduce，由Hadoop推广。 Spark可以轻松实现MapReduce流程：

```
  scala > val wordCounts = textFile . flatMap ( line => line . split ( " " )).
 groupByKey ( identity ). count () wordCounts : org.apache.spark.sql.Dataset
[( String , Long )] = [ value: string , count ( 1 ) : bigint ]
```

在这里，我们调用flatMap将行数据集转换为单词数据集，然后将groupByKey和count结合起来计算文件中的单词计数作为（String，Long）对的数据集。 要收集我们的shell中的单词计数，我们可以调用collect：

```
  scala > wordCounts . collect () res6 : Array [( String , Int )] = Array (( m
eans , 1 ), ( under , 2 ), ( this , 3 ), ( Because , 1 ), ( Python , 2 ), ( ag
ree , 1 ), ( cluster ., 1 ), ...)
```

高速缓存

Spark还支持将数据集提取到群集范围的内存缓存中。 这在重复访问数据时非常有用，例如查询小的“热”数据集或运行像PageRank这样的迭代算法时。 举个简单的例子，让我们标记要缓存的`linesWithSpark`数据集：

Scala Python

```
scala > linesWithSpark . cache () res7 : linesWithSpark. type = [ value: str
ing ] scala > linesWithSpark . count () res8 : Long = 15 scala > linesWithSpar
k . count () res9 : Long = 15
```

使用Spark来探索和缓存100行文本文件似乎很愚蠢。 有趣的是，这些相同的功能可用于非常大的数据集，即使它们跨越数十个或数百个节点进行条带化。 您也可以通过将`bin/spark-shell`连接到群集来交互式地执行此操作，如RDD编程指南中所述 。

自包含的应用程序

假设我们希望使用Spark API编写一个自包含的应用程序。 我们将在Scala（使用sbt），Java（使用Maven）和Python（pip）中使用简单的应用程序。

Scala Java Python

我们将在Scala中创建一个非常简单的Spark应用程序 - 事实上，它简单地命名为`SimpleApp.scala`：

```
/* SimpleApp.scala */ import org.apache.spark.sql.SparkSession object Simple
App { def main ( args : Array [ String ]) { val logFile = "YOUR_SPARK_HOME/REA
DME.md" // Should be some file on your system val spark = SparkSession . build
er . appName ( "Simple Application" ). getOrCreate () val logData = spark . re
ad . textFile ( logFile ). cache () val numAs = logData . filter ( line => lin
e . contains ( "a" )). count () val numBs = logData . filter ( line => line .
contains ( "b" )). count () println ( s"Lines with a: $numAs , Lines with b:
$numBs " ) spark . stop () } }
```

请注意，应用程序应定义`main()`方法，而不是扩展`scala.App` 。 `scala.App`子类可能无法正常工作。

该程序只计算包含'a'的行数和Spark README中包含'b'的数字。 请注意，您需要将YOUR_SPARK_HOME替换为安装Spark的位置。 与之前使用Spark shell（初始化自己的SparkSession）的示例不同，我们将SparkSession初始化为程序的一部分。

我们调用`SparkSession.builder`来构造[[SparkSession]]，然后设置应用程序名称，最后调用`getOrCreate`来获取[[SparkSession]]实例。

我们的应用程序依赖于Spark API，因此我们还将包含一个sbt配置文件`build.sbt` ，它解释了Spark是一个依赖项。 此文件还添加了Spark依赖的存储库：

```
name := "Simple Project" version := "1.0" scalaVersion := "2.11.12" libraryD
ependencies += "org.apache.spark" %% "spark-sql" % "2.4.0"
```

为了使sbt正常工作，我们需要根据典型的目录结构布局`build.sbt`和`build.sbt` 。 一旦到位，我们可以创建一个包含应用程序代码的JAR包，然后使用`spark-submit`脚本来运行我们的程序。

```
 # Your directory layout should look like this $ find . . ./build.sbt ./src
./src/main ./src/main/scala ./src/main/scala/SimpleApp.scala # Package a jar
containing your application $ sbt package ... [ info ] Packaging { .. } / {
.. } /target/scala-2.11/simple-project_2.11-1.0.jar # Use spark-submit to run
your application $ YOUR_SPARK_HOME/bin/spark-submit \ --class "SimpleApp" \ -
-master local [ 4 ] \ target/scala-2.11/simple-project_2.11-1.0.jar ... Lines
with a: 46 , Lines with b: 23
```

从这往哪儿走

恭喜您运行第一个Spark应用程序！

- 有关API的深入概述，请从RDD编程指南和SQL编程指南开始 ，或参阅其他组件的“编程指南”菜单。
- 要在群集上运行应用程序，请转至部署概述 。
- 最后，Spark在`examples`目录（ Scala ， Java ， Python ， R ）中包含了几个示例。 您可以按如下方式运行它们：

```
 # For Scala and Java, use run-example: ./bin/run-example SparkPi # For Pytho
n examples, use spark-submit directly: ./bin/spark-submit examples/src/main/py
thon/pi.py # For R examples, use spark-submit directly: ./bin/spark-submit exa
mples/src/main/r/dataframe.R
```