**Spark安装以及使用指导**

**任恒2016212063**

**2019年5月**

**Spark安装以及使用指导**

1. **关于spark基本概念**

Google 在 2003 年和 2004 年先后发表了 Google 文件系统 GFS 和 MapReduce 编程模型两篇文章,. 基于这两篇开源文档,06 年 Nutch 项目子项目之一的 Hadoop 实现了两个强有力的开源产品:HDFS 和 MapReduce. Hadoop 成为了典型的大数据批量处理架构,由 HDFS 负责静态数据的存储,并通过 MapReduce 将计算逻辑分配到各数据节点进行数据计算和价值发现.之后以 HDFS 和 MapReduce 为基础建立了很多项目,形成了 Hadoop 生态圈.

而 Spark 则是UC Berkeley AMP lab (加州大学伯克利分校AMP实验室)所开源的类Hadoop MapReduce的通用并行框架, 专门用于大数据量下的迭代式计算.是为了跟 Hadoop 配合而开发出来的,不是为了取代 Hadoop, Spark 运算比 Hadoop 的 MapReduce 框架快的原因是因为 Hadoop 在一次 MapReduce 运算之后,会将数据的运算结果从内存写入到磁盘中,第二次 Mapredue 运算时在从磁盘中读取数据,所以其瓶颈在2次运算间的多余 IO 消耗. Spark 则是将数据一直缓存在内存中,直到计算得到最后的结果,再将结果写入到磁盘,所以多次运算的情况下, Spark 是比较快的. 其优化了迭代式工作负载。

关于spark的一些概念如下：

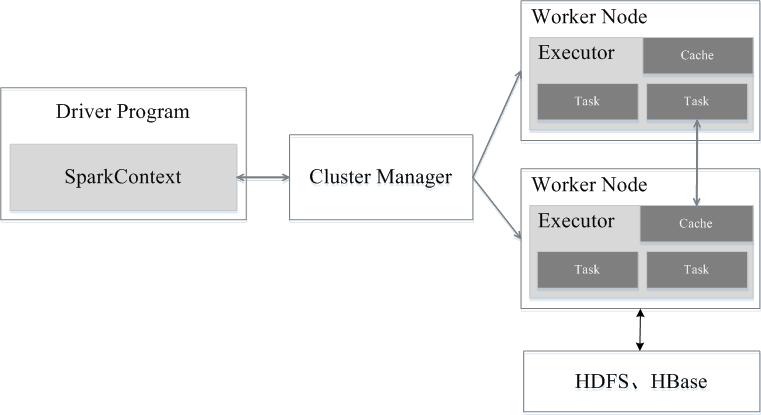
* RDD：是弹性分布式数据集（Resilient Distributed Dataset）的简称，是分布式内存的一个抽象概念，提供了一种高度受限的共享内存模型；
* DAG：是Directed Acyclic Graph（有向无环图）的简称，反映RDD之间的依赖关系；
* Executor：是运行在工作节点（Worker Node）上的一个进程，负责运行任务，并为应用程序存储数据；
* 应用：用户编写的Spark应用程序；
* 任务：运行在Executor上的工作单元；
* 作业：一个作业包含多个RDD及作用于相应RDD上的各种操作；
* 阶段：是作业的基本调度单位，一个作业会分为多组任务，每组任务被称为“阶段”，或者也被称为“任务集”。

架构设计

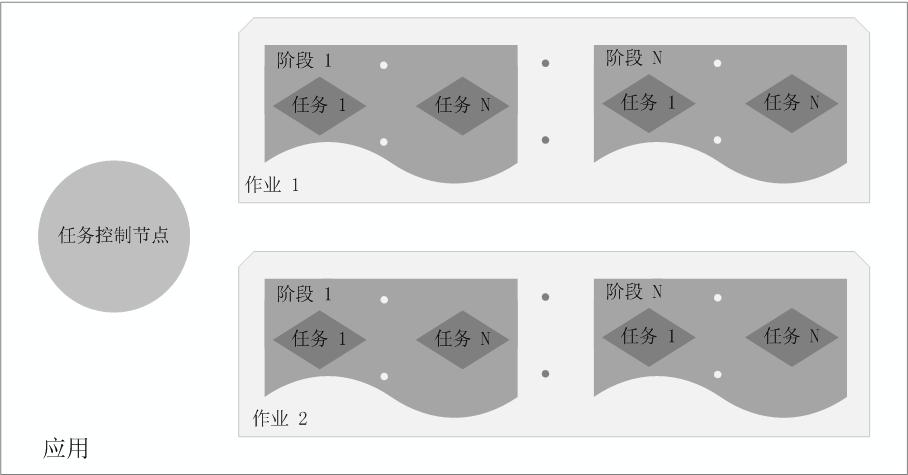
如图一所示，Spark运行架构包括集群资源管理器（Cluster Manager）、运行作业任务的工作节点（Worker Node）、每个应用的任务控制节点（Driver）和每个工作节点上负责具体任务的执行进程（Executor）。其中，集群资源管理器可以是Spark自带的资源管理器，也可以是YARN或Mesos等资源管理框架。

与Hadoop MapReduce计算框架相比，Spark所采用的Executor有两个优点：一是利用多线程来执行具体的任务（Hadoop MapReduce采用的是进程模型），减少任务的启动开销；二是Executor中有一个BlockManager存储模块，会将内存和磁盘共同作为存储设备，当需要多轮迭代计算时，可以将中间结果存储到这个存储模块里，下次需要时，就可以直接读该存储模块里的数据，而不需要读写到HDFS等文件系统里，因而有效减少了IO开销；或者在交互式查询场景下，预先将表缓存到该存储系统上，从而可以提高读写IO性能。

总体而言，如图二所示，在Spark中，一个应用（Application）由一个任务控制节点（Driver）和若干个作业（Job）构成，一个作业由多个阶段（Stage）构成，一个阶段由多个任务（Task）组成。当执行一个应用时，任务控制节点会向集群管理器（Cluster Manager）申请资源，启动Executor，并向Executor发送应用程序代码和文件，然后在Executor上执行任务，运行结束后，执行结果会返回给任务控制节点，或者写到HDFS或者其他数据库中。



图一



图二

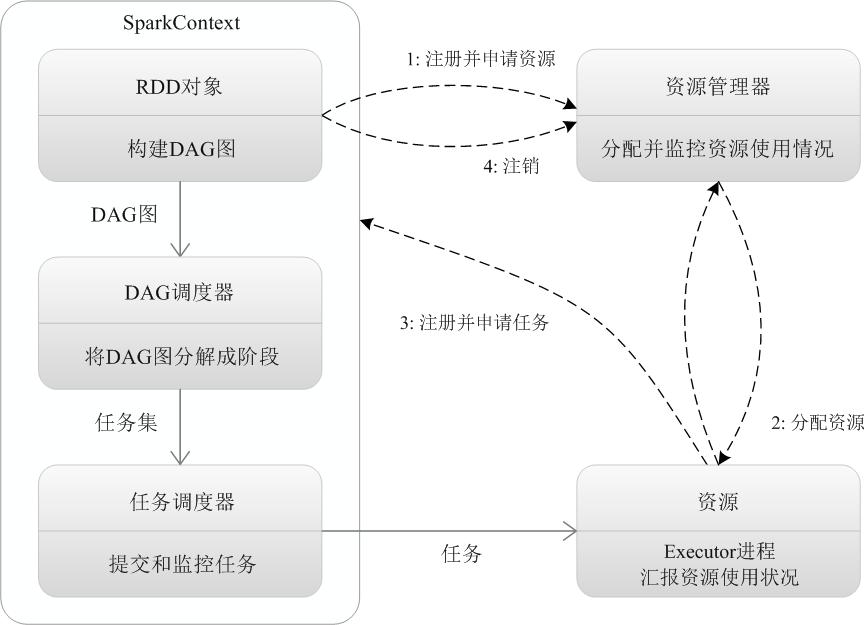
Spark基本运行过程如图三：

（1）当一个Spark应用被提交时，首先需要为这个应用构建起基本的运行环境，即由任务控制节点（Driver）创建一个SparkContext，由SparkContext负责和资源管理器（Cluster Manager）的通信以及进行资源的申请、任务的分配和监控等。SparkContext会向资源管理器注册并申请运行Executor的资源；

（2）资源管理器为Executor分配资源，并启动Executor进程，Executor运行情况将随着“心跳”发送到资源管理器上；

（3）SparkContext根据RDD的依赖关系构建DAG图，DAG图提交给DAG调度器（DAGScheduler）进行解析，将DAG图分解成多个“阶段”（每个阶段都是一个任务集），并且计算出各个阶段之间的依赖关系，然后把一个个“任务集”提交给底层的任务调度器（TaskScheduler）进行处理；Executor向SparkContext申请任务，任务调度器将任务分发给Executor运行，同时，SparkContext将应用程序代码发放给Executor；

（4）任务在Executor上运行，把执行结果反馈给任务调度器，然后反馈给DAG调度器，运行完毕后写入数据并释放所有资源。



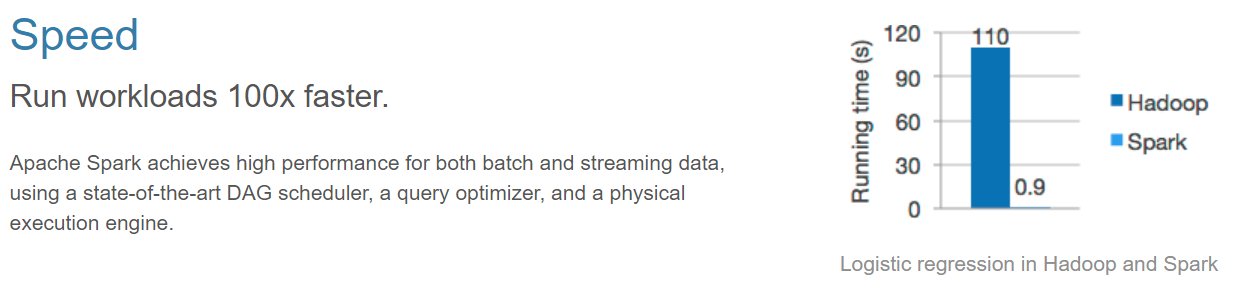
图三

1. **Spark的四大特性**

1、高效性

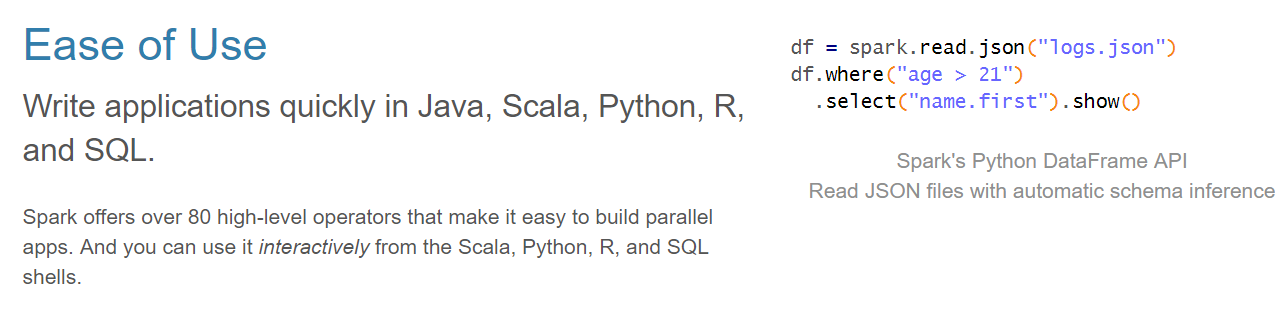
运行速度提高100倍。

Apache Spark使用最先进的DAG调度程序，查询优化程序和物理执行引擎，实现批量和流式数据的高性能。

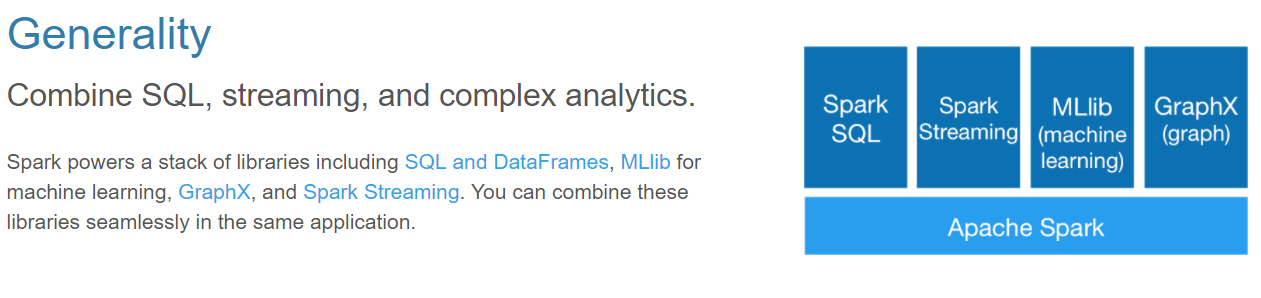


2、易用性

Spark支持Java、Python和Scala的API，还支持超过80种高级算法，使用户可以快速构建不同的应用。而且Spark支持交互式的Python和Scala的shell，可以非常方便地在这些shell中使用Spark集群来验证解决问题的方法。

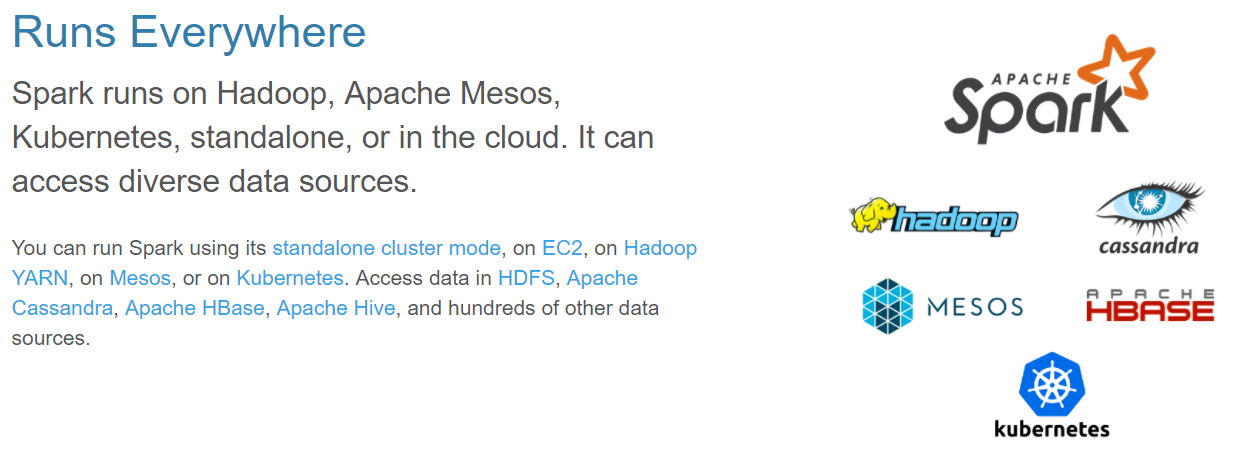
3、通用性

Spark提供了统一的解决方案。Spark可以用于批处理、交互式查询（Spark SQL）、实时流处理（Spark Streaming）、机器学习（Spark MLlib）和图计算（GraphX）。这些不同类型的处理都可以在同一个应用中无缝使用。Spark统一的解决方案非常具有吸引力，毕竟任何公司都想用统一的平台去处理遇到的问题，减少开发和维护的人力成本和部署平台的物力成本。



4、兼容性

Spark可以非常方便地与其他的开源产品进行融合。比如，Spark可以使用Hadoop的YARN和Apache Mesos作为它的资源管理和调度器，器，并且可以处理所有Hadoop支持的数据，包括HDFS、HBase和Cassandra等。这对于已经部署Hadoop集群的用户特别重要，因为不需要做任何数据迁移就可以使用Spark的强大处理能力。Spark也可以不依赖于第三方的资源管理和调度器，它实现了Standalone作为其内置的资源管理和调度框架，这样进一步降低了Spark的使用门槛，使得所有人都可以非常容易地部署和使用Spark。此外，Spark还提供了在EC2上部署Standalone的Spark集群的工具。



1. **Spark的组成**

Spark组成(BDAS)：全称伯克利数据分析栈，通过大规模集成算法、机器、人之间展现大数据应用的一个平台。也是处理大数据、云计算、通信的技术解决方案。

它的主要组件有：

SparkCore：将分布式数据抽象为弹性分布式数据集（RDD），实现了应用任务调度、RPC、序列化和压缩，并为运行在其上的上层组件提供API。

SparkSQL：Spark Sql 是Spark来操作结构化数据的程序包，可以让我使用SQL语句的方式来查询数据，Spark支持 多种数据源，包含Hive表，parquest以及JSON等内容。

SparkStreaming： 是Spark提供的实时数据进行流式计算的组件。

MLlib：提供常用机器学习算法的实现库。

GraphX：提供一个分布式图计算框架，能高效进行图计算。

BlinkDB：用于在海量数据上进行交互式SQL的近似查询引擎。

Tachyon：以内存为中心高容错的的分布式文件系统。

1. **安装scala**

4.1 下载地址（本教程下载为2.10.7）

|  |
| --- |
| >>wget <http://www.scala-lang.org/download/> |

4.2 解压安装包：

|  |
| --- |
| >>mv scala-2.10.7.tgz ~/app  >>tar -zxvf scala-2.10.7.tgz |

4.3 编辑.bash\_profile文件：

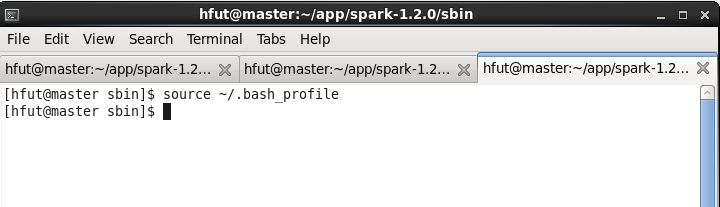
|  |
| --- |
| >>gedit ~/.bash\_profile |

4.4 在.bash\_profile文件中添加：

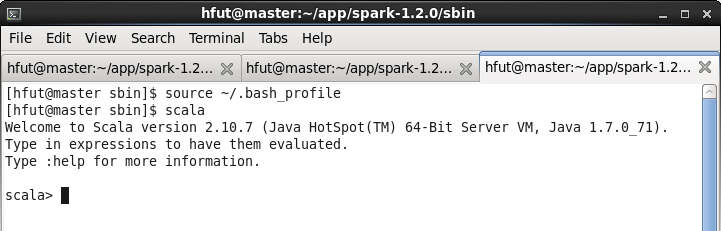
|  |
| --- |
| #Scala  export SCALA\_HOME=/home/hfut/app/scala-2.10.7  export PATH=$JAVA\_HOME/bin$HADOOP\_HOME/bin:$HIVE\_HOME/bin:$SCALA\_HOME/bin:$PATH |

4.5 使环境变量生效

|  |
| --- |
| source ~./bash\_profile |



4.6 验证scala安装：



1. **安装spark（本文档使用的是spark-1.2.0版本）**

5.1 下载安装包

|  |
| --- |
| >>wget <http://d3kbcqa49mib13.cloudfront.net/spark-1.2.0-bin->hadoop1.tgz |

5.2 解压安装包：

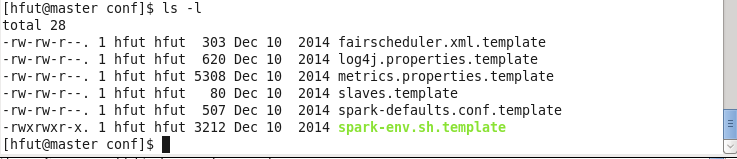
|  |
| --- |
| >>mv spark-1.2.0-bin-hadoop1.tgz ~/app  >>cd ~/app  >>tar -zxvf spark-1.2.0-bin-hadoop1.tgz |



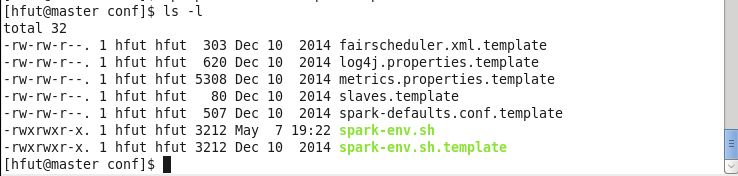
5.3 配置spark环境：

|  |
| --- |
| >>cd /spark-1.2.0-bin-hadoop1/conf/  >>cp spark-env.sh.template spark-env.sh |

5.4 查看原始文件列表：



5.5 查看复制之后的文件列表：



5.6 **将文件夹spark-1.2.0-bin-hadoop1改名为spark-1.2.0**（开始配置了spark-1.2.0包的环境但是失败，后来重新下载了-bin-hadoop1的包，环境就按照原先的配置了）。

5.7 编辑spark-env.sh文件：

|  |
| --- |
| >>gedit spark-env.sh |

5.8 在spark-env.sh中添加下内容：

（其中将JAVA\_HOME、SCALA\_HOME、HADOOP\_CONF\_DI以此改为java安装目录，scala安装目录以及$HADOOP\_HOME/ect/hadoop）

|  |
| --- |
| export JAVA\_HOME=/usr/java/jdk1.7.0\_71  export SCALA\_HOME=/home/hfut/app/scala-2.10.7  export HADOOP\_CONF\_DIR=/home/hfut/hadoop-2.5.2/etc/hadoop |

5.9 配置slaves：

|  |
| --- |
| >>cp slaves.templsate slaves |

5.10 编辑slave文件：

|  |
| --- |
| >>gedit slaves |

5.11 在slaves中将localhost改为：（其中master和slave均为主机名）

|  |
| --- |
| master  slave |

5.12 将.bash\_profile文件传到slave上：

|  |
| --- |
| >>scp ~/.bash\_profile hfut@192.168.81.139:~/.bash\_profile |

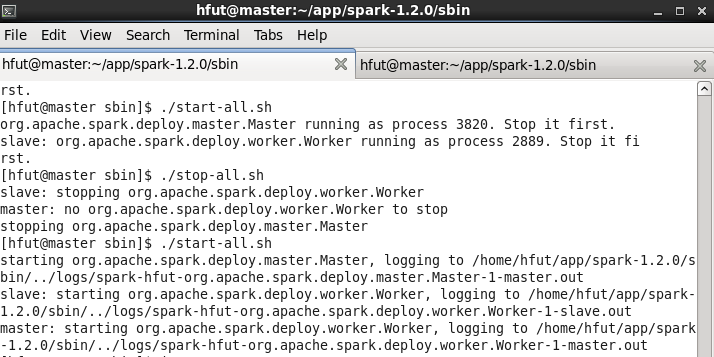
5.13 将spark拷贝到slave：（由于已经设置了免密登录，因此此处并不需要登陆密码，不然请检查免密登录）

|  |
| --- |
| >>scp ~/app/spark-1.2.0-bin-hadoop1  [hfut@192.168.81.139:~/.bash\_profile](mailto:hfut@192.168.81.139:~/.bash_profile) hfut@192.168.81.139:~/app/ |

1. **验证spark安装**

6.1 启动spark：

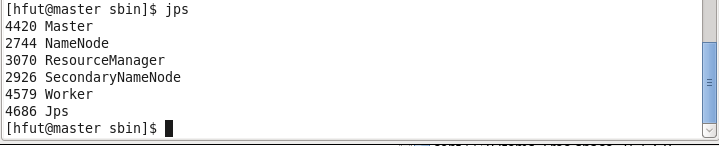
|  |
| --- |
| >>cd ~/app/spark-1.2.0/conf/sbin  >>start-all.sh |



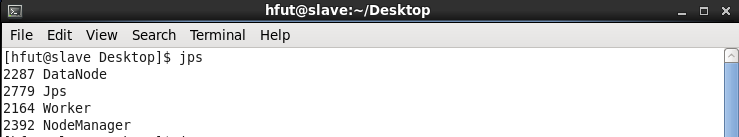
6.2 查看启动结果：

在master端看到Master和Worker说明启动成功，此处是已经启动了hadoop因此还有其他的出现。

|  |
| --- |
| Master>>jps |



|  |
| --- |
| Slave>>jps |



打开火狐浏览器，输入网址spark://master:7070查看启动情况:

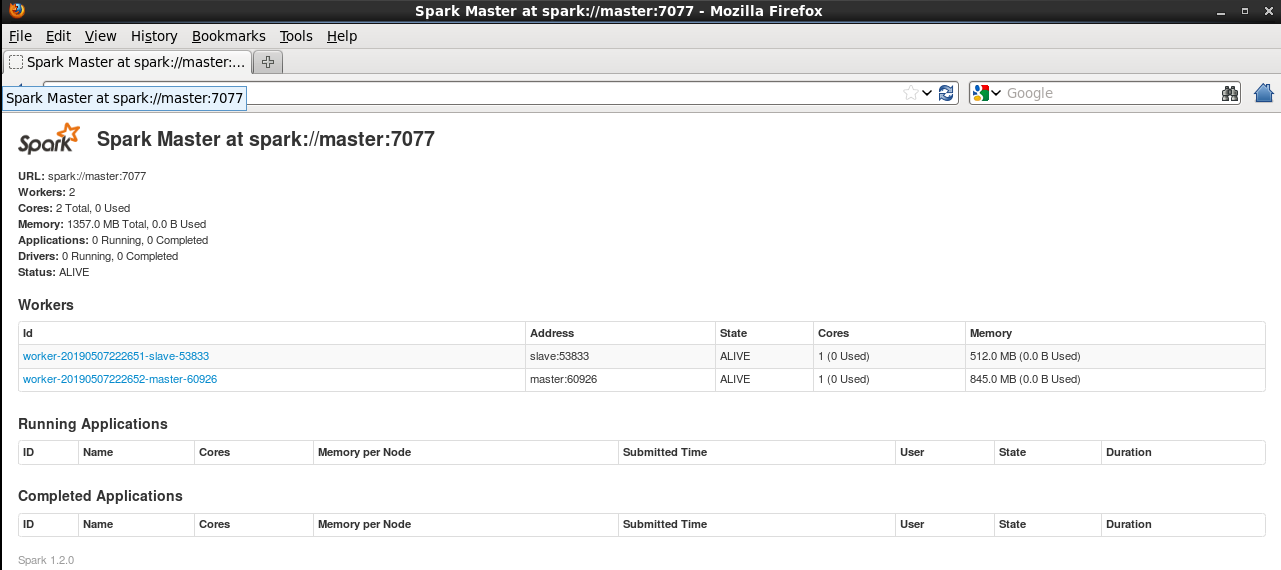
其他端口：

namenode的webUI端口：50070

yarn的web端口：8088

spark集群的web端口：8080

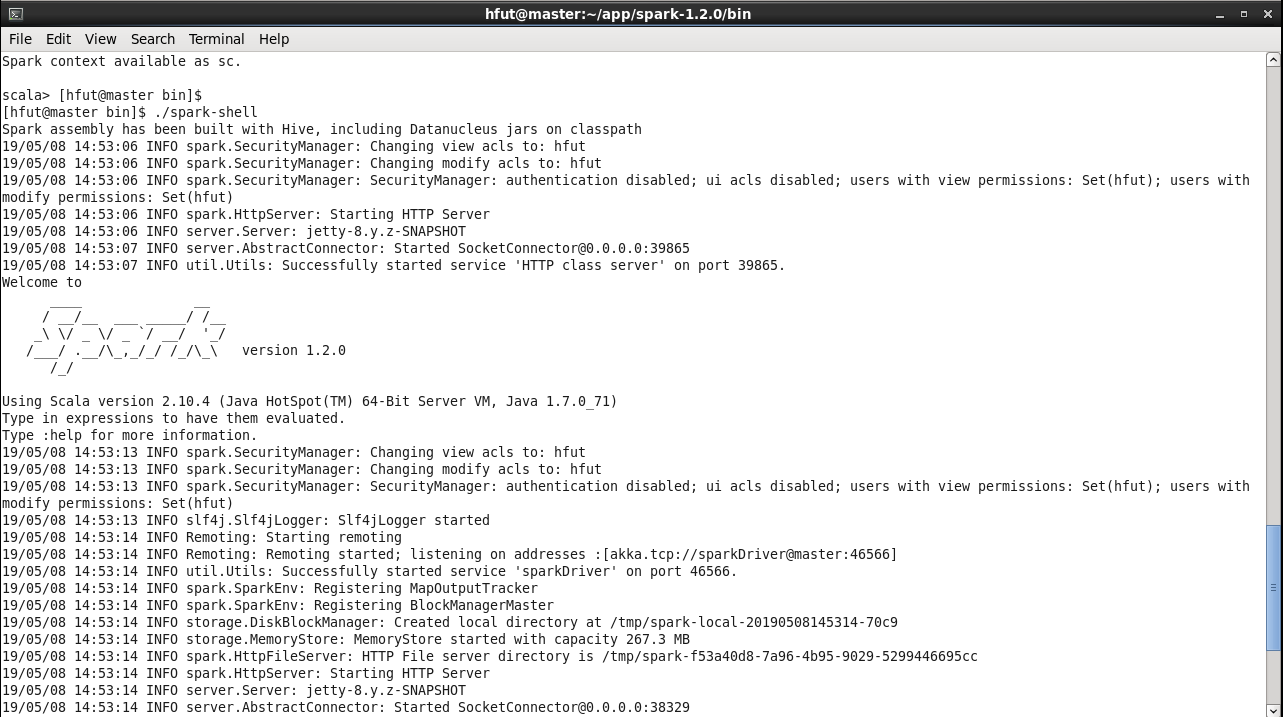
spark-job监控端口：4040

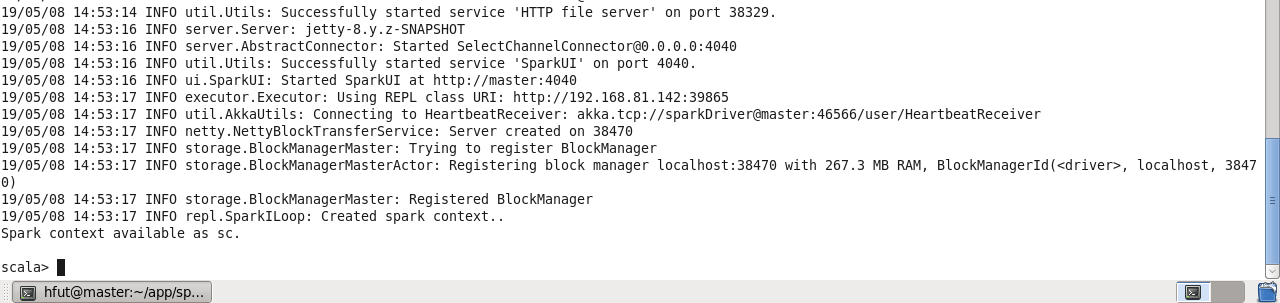


6.3 打开spark-shell：

|  |
| --- |
| >>cd ~/app/spark-1.2.0/bin  >>./sprak-shell |

出现如下结果说明安装成功：（此处建议第二行的命令改为：../bin/spark-shell,因为上一步的命令可能会在后面执行scala语句的时候出错，原因未知）



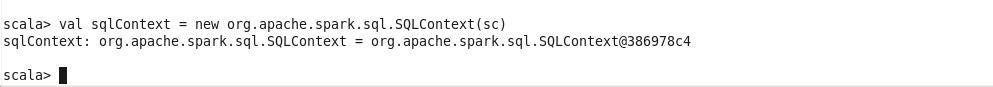


1. **Spark-shell的简单使用**

7.1 入口：SQLContext（Starting Point: SQLContext）

Spark SQL程序的主入口是SQLContext类或它的子类。创建一个基本的SQLContext，你只需要SparkContext，创建代码示例如下：

|  |
| --- |
| val sqlContext = new org.apache.spark.sql.SQLContext(sc) |



除了基本的SQLContext，也可以创建HiveContext。SQLContext和HiveContext区别与联系为：

SQLContext现在只支持SQL语法解析器（SQL-92语法）

HiveContext现在支持SQL语法解析器和HiveSQL语法解析器，默认为HiveSQL语法解析器，用户可以通过配置切换成SQL语法解析器，来运行HiveSQL不支持的语法。

使用HiveContext可以使用Hive的UDF，读写Hive表数据等Hive操作。SQLContext不可以对Hive进行操作。

Spark SQL未来的版本会不断丰富SQLContext的功能，做到SQLContext和HiveContext的功能容和，最终可能两者会统一成一个Context

HiveContext包装了Hive的依赖包，把HiveContext单独拿出来，可以在部署基本的Spark的时候就不需要Hive的依赖包，需要使用HiveContext时再把Hive的各种依赖包加进来。

SQL的解析器可以通过配置spark.sql.dialect参数进行配置。在SQLContext中只能使用Spark SQL提供的”sql“解析器。在HiveContext中默认解析器为”hiveql“，也支持”sql“解析器。

7.2 RDD创建：

|  |
| --- |
| val intRDD = sc.sparkContext.parallelize(List(3,1,2,5,5))  print("打印intRDD的结果："+intRDD.collect().mkString(","))  println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*单个RDD转换运算\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  def addOne(x:Int):Int =  {  return (x+1)  }  //map运算：对RDD中每一个元素做一个转换操作，生成一个新的RDD  println("使用具体的函数完成map运算："+intRDD.map(addOne).collect().mkString(","))  println("使用匿名函数完成map运算："+intRDD.map(x=>x+1).collect().mkString(","))  println("使用匿名函数和匿名参数完成map运算："+intRDD.map(\_+1).collect().mkString(","))  println("使用匿名函数完成map运算："+stringRDD.map(x=>"fruit"+x).collect().mkString(","))  //filter运算：对RDD中每一个元素进行筛选，生成一个新的RDD  println("使用匿名函数筛选intRDD中小于5的元素："+intRDD.filter(x=>x < 5).collect().mkString(","))  println("使用匿名函数和匿名参数筛选intRDD中小于5的元素："+intRDD.filter(\_ < 5).collect().mkString(","))  println("使用匿名函数筛选stringRDD中包含ra的元素："+stringRDD.filter(x=>x.contains('ra)).collect().mkString(","))  //distinct运算：对RDD中元素进行去重  println("对intRDD元素进行去重"+intRDD.distinct().collect().mkString(","))  println("对StringRDD元素进行去重"+stringRDD.distinct().collect().mkString(","))  //randomSplit运算，按照指定的比例将RDD进行划分  val sRDD = intRDD.randomSplit(Array(0.4,0.6))  println("分割后第一个RDD为："+sRDD(0).collect().mkString(","))  println("分割后第二个RDD为："+sRDD(1).collect().mkString(","))  //groupBy运算：可以按照传入的匿名函数规则，将数据分为多个Array,返回Array[(String,Iterable[Int])]  val gRDD = intRDD.groupBy(x => {if(x%2==0) "even" else "odd"}).collect()  println("偶数数组为："+gRDD(0))  println("奇数数组为："+gRDD(1)) |

结果：

|  |
| --- |
| 打印intRDD的结果：3,1,2,5,5  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*单个RDD转换运算\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  使用具体的函数完成map运算：4,2,3,6,6  使用匿名函数完成map运算：4,2,3,6,6  使用匿名函数和匿名参数完成map运算：4,2,3,6,6  使用匿名函数完成map运算：fruitApple,fruitOrange,fruitBanana,fruitGrape,fruitApple  使用匿名函数筛选intRDD中小于5的元素：3,1,2  使用匿名函数和匿名参数筛选intRDD中小于5的元素：3,1,2  使用匿名函数筛选stringRDD中包含ra的元素：  对intRDD元素进行去重2,1,3,5  对StringRDD元素进行去重Orange,Apple,Grape,Banana  分割后第一个RDD为：1,2,5  分割后第二个RDD为：3,5  偶数数组为：(even,CompactBuffer(2))  奇数数组为：(odd,CompactBuffer(3, 1, 5, 5)) |

7.3 案例分析

该案例中，我们将假设我们需要统计一个 1 万人口的所有人的平均年龄，当然如果您想测试 Spark 对于大数据的处理能力，您可以把人口数放的更大，比如 1 亿人口，当然这个取决于测试所用集群的存储容量。假设这些年龄信息都存储在一个文件里，并且该文件的格式如下，第一列是 ID，第二列是年龄。

年龄生成代码：

|  |
| --- |
| FILE \* fp;  fp = fopen("ageNum.txt", "w+");  for (int i = 0; i < 10000; i++) {  srand((UINT)GetCurrentTime());  fprintf(fp, "%d %d\n",i,rand()%100);  printf("%d %d\n", i, rand() % 100);  }  fclose(fp); |

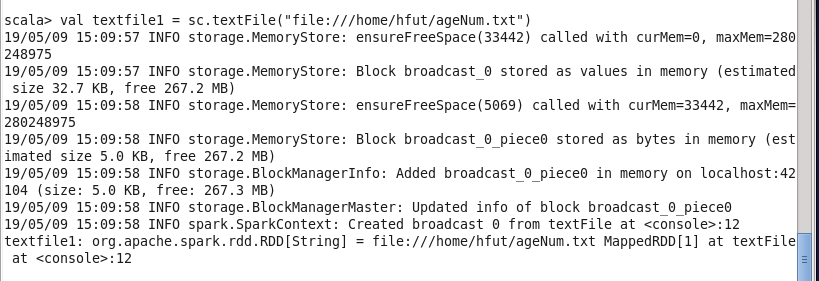


要计算平均年龄，那么首先需要对源文件对应的 RDD 进行处理，也就是将它转化成一个只包含年龄信息的 RDD，其次是计算元素个数即为总人数，然后是把所有年龄数加起来，最后平均年龄=总年龄/人数。

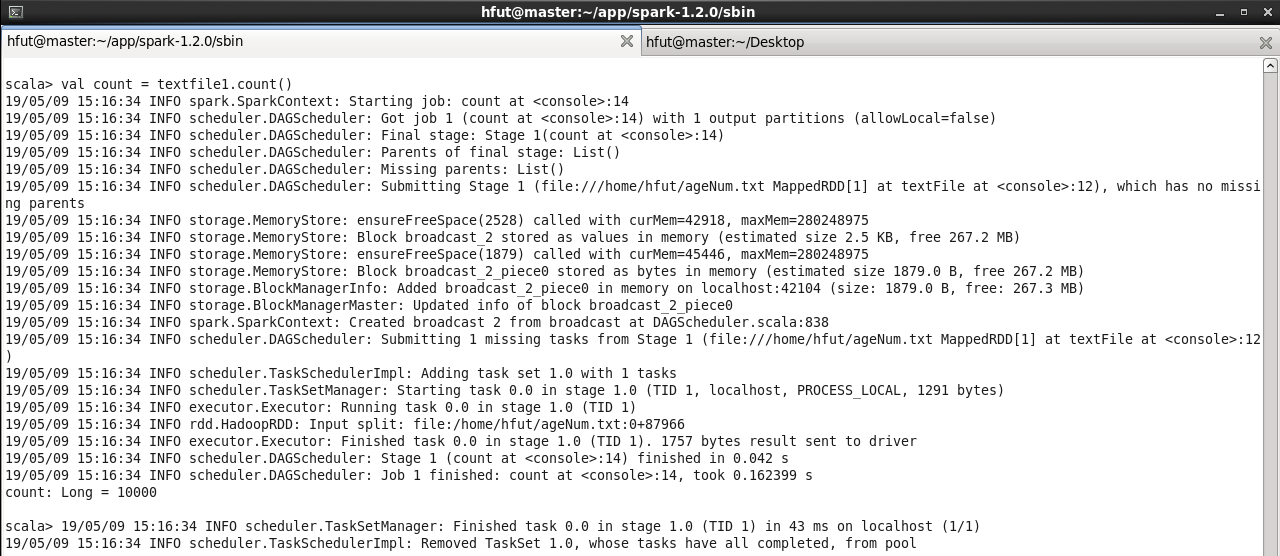
对于第一步我们需要使用 map 算子把源文件对应的 RDD 映射成一个新的只包含年龄数据的 RDD，很显然需要对在 map 算子的传入函数中使用 split 方法，得到数组后只取第二个元素即为年龄信息；第二步计算数据元素总数需要对于第一步映射的结果 RDD 使用 count 算子；第三步则是使用 reduce 算子对只包含年龄信息的 RDD 的所有元素用加法求和；最后使用除法计算平均年龄即可。

由于本例输出结果很简单，所以只打印在控制台即可。

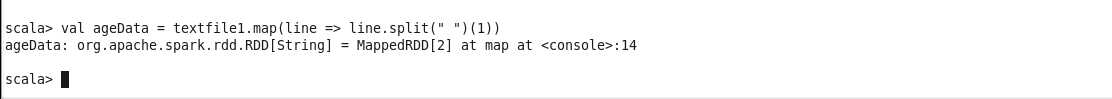
|  |
| --- |
| >>val textfile1 = sc.textFile("file:///home/hfut/ageNum.txt") |



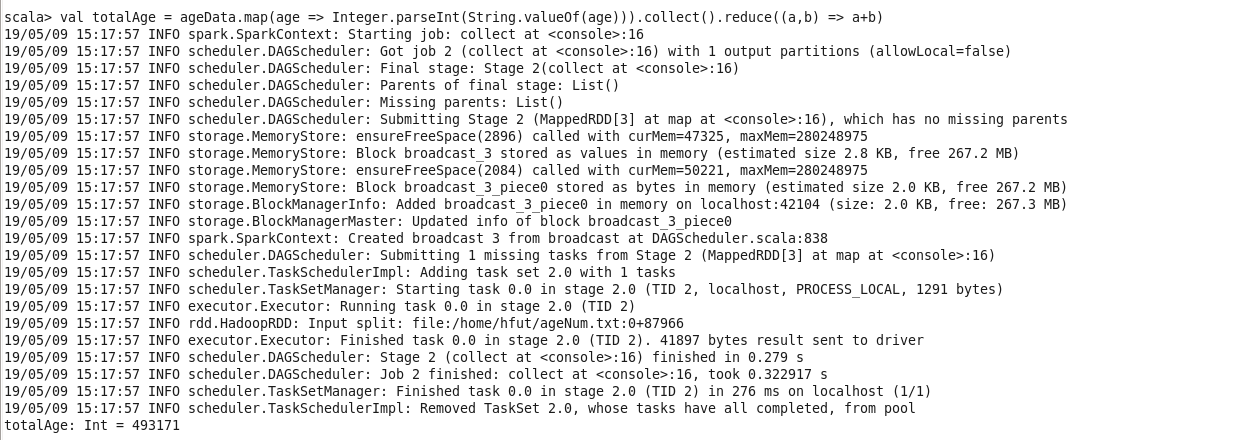
|  |
| --- |
| >>val count = textfile1.count() |



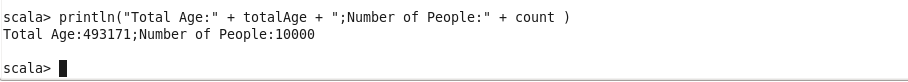
|  |
| --- |
| >>val ageData = textfile1.map(line => line.split(" ")(1)) |



|  |
| --- |
| >>val totalAge = ageData.map(age => Integer.parseInt  (String.valueOf(age))).collect().reduce((a,b) => a+b) |



|  |
| --- |
| >>println("Total Age:" + totalAge + ";Number of People:" + count ) |



|  |
| --- |
| >>val avgAge : Double = totalAge.toDouble / count.toDouble |



|  |
| --- |
| >>println("Average Age is " + avgAge) |



参考文献：

<http://dblab.xmu.edu.cn/blog/1709-2/>

<https://www.w3cschool.cn/spark/>

<https://blog.csdn.net/u011630575/article/details/56556949>