

尚硅谷大数据技术之 Spark

版本：V3.0

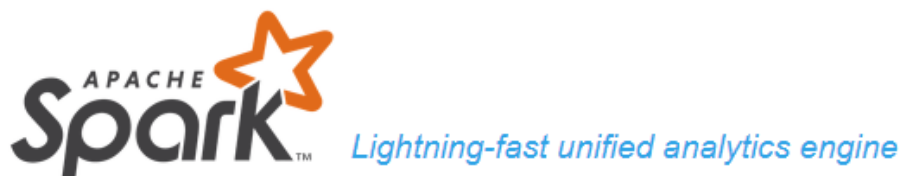


作者：尚硅谷大数据研发部

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网

第1章 Spark 概述

1.1 Spark 是什么



Apache Spark™ is a unified analytics engine for large-scale data processing.

Spark 是一种基于内存的快速、通用、可扩展的大数据分析计算引擎。

1.2 Spark and Hadoop

在之前的学习中，Hadoop 的 MapReduce 是大家广为熟知的计算框架，那为什么咱们还要学习新的计算框架 Spark 呢，这里就不得不提到 Spark 和 Hadoop 的关系。

首先从时间节点上来看：

➤ Hadoop

- 2006 年 1 月，Doug Cutting 加入 Yahoo，领导 Hadoop 的开发
- 2008 年 1 月，Hadoop 成为 Apache 顶级项目
- 2011 年 1.0 正式发布
- 2012 年 3 月稳定版发布
- 2013 年 10 月发布 2.X (Yarn)版本

➤ Spark

- 2009 年，Spark 诞生于伯克利大学的 AMPLab 实验室
- 2010 年，伯克利大学正式开源了 Spark 项目
- 2013 年 6 月，Spark 成为了 Apache 基金会下的项目
- 2014 年 2 月，Spark 以飞快的速度成为了 Apache 的顶级项目
- 2015 年至今，Spark 变得愈发火爆，大量的国内公司开始重点部署或者使用 Spark

然后我们再从功能上来看：

➤ Hadoop

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网

- Hadoop 是由 java 语言编写的，在分布式服务器集群上存储海量数据并运行分布式分析应用的开源框架
- 作为 Hadoop 分布式文件系统，HDFS 处于 Hadoop 生态圈的最下层，存储着所有的数据，支持着 Hadoop 的所有服务。它的理论基础源于 Google 的 TheGoogleFileSystem 这篇论文，它是 GFS 的开源实现。
- **MapReduce** 是一种编程模型，Hadoop 根据 Google 的 MapReduce 论文将其实现，作为 Hadoop 的分布式计算模型，是 Hadoop 的核心。基于这个框架，分布式并行程序的编写变得异常简单。综合了 HDFS 的分布式存储和 MapReduce 的分布式计算，Hadoop 在处理海量数据时，性能横向扩展变得非常容易。
- HBase 是对 Google 的 Bigtable 的开源实现，但又和 Bigtable 存在许多不同之处。HBase 是一个基于 HDFS 的分布式数据库，擅长实时地随机读/写超大规模数据集。它也是 Hadoop 非常重要的组件。

➤ **Spark**

- Spark 是一种由 Scala 语言开发的快速、通用、可扩展的**大数据分析引擎**
- Spark Core 中提供了 Spark 最基础与最核心的功能
- Spark SQL 是 Spark 用来操作结构化数据的组件。通过 Spark SQL，用户可以使用 SQL 或者 Apache Hive 版本的 SQL 方言（HQL）来查询数据。
- Spark Streaming 是 Spark 平台上针对实时数据进行流式计算的组件，提供了丰富的处理数据流的 API。

由上面的信息可以获知，Spark 出现的时间相对较晚，并且主要功能主要是用于数据计算，所以其实 Spark 一直被认为是 Hadoop 框架的升级版。

1.3 Spark or Hadoop

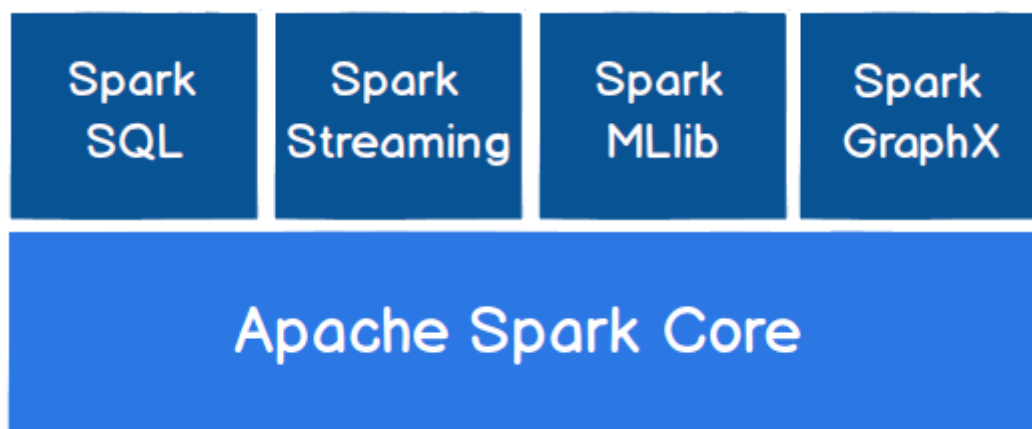
Hadoop 的 MR 框架和 Spark 框架都是数据处理框架，那么我们在使用时如何选择呢？

- Hadoop MapReduce 由于其设计初衷并不是为了满足循环迭代式数据流处理，因此在多并行运行的数据可复用场景（如：机器学习、图挖掘算法、交互式数据挖掘算法）中存在诸多计算效率等问题。所以 Spark 应运而生，Spark 就是在传统的 MapReduce 计算框架的基础上，利用其计算过程的优化，从而大大加快了数据分析、挖掘的运行和读写速度，并将计算单元缩小到更适合并行计算和重复使用的 RDD 计算模型。

- 机器学习中 ALS、凸优化梯度下降等。这些都需要基于数据集或者数据集的衍生数据反复查询反复操作。MR 这种模式不太合适，即使多 MR 串行处理，性能和时间也是一个问题。数据的共享依赖于磁盘。另外一种交互式数据挖掘，MR 显然不擅长。而 Spark 所基于的 scala 语言恰恰擅长函数的处理。
- Spark 是一个分布式数据快速分析项目。它的核心技术是弹性分布式数据集（Resilient Distributed Datasets），提供了比 MapReduce 丰富的模型，可以快速在内存中对数据集进行多次迭代，来支持复杂的数据挖掘算法和图形计算算法。
- Spark 和 Hadoop 的根本差异是多个作业之间的数据通信问题：Spark 多个作业之间数据通信是基于内存，而 Hadoop 是基于磁盘。
- Spark Task 的启动时间快。Spark 采用 fork 线程的方式，而 Hadoop 采用创建新的进程的方式。
- Spark 只有在 shuffle 的时候将数据写入磁盘，而 Hadoop 中多个 MR 作业之间的数据交互都要依赖于磁盘交互
- Spark 的缓存机制比 HDFS 的缓存机制高效。

经过上面的比较，我们可以看出在绝大多数的数据计算场景中，Spark 确实会比 MapReduce 更有优势。但是 Spark 是基于内存的，所以在实际的生产环境中，由于内存的限制，可能会由于内存资源不够导致 Job 执行失败，此时，MapReduce 其实是一个更好的选择，所以 Spark 并不能完全替代 MR。

1.4 Spark 核心模块



➤ **Spark Core**

Spark Core 中提供了 Spark 最基础与最核心的功能，Spark 其他的功能如：Spark SQL，Spark Streaming，GraphX，MLlib 都是在 Spark Core 的基础上进行扩展的

➤ **Spark SQL**

Spark SQL 是 Spark 用来操作结构化数据的组件。通过 Spark SQL，用户可以使用 SQL 或者 Apache Hive 版本的 SQL 方言（HQL）来查询数据。

➤ **Spark Streaming**

Spark Streaming 是 Spark 平台上针对实时数据进行流式计算的组件，提供了丰富的处理数据流的 API。

➤ **Spark MLlib**

MLlib 是 Spark 提供的一个机器学习算法库。MLlib 不仅提供了模型评估、数据导入等额外的功能，还提供了一些更底层的机器学习原语。

➤ **Spark GraphX**

GraphX 是 Spark 面向图计算提供的框架与算法库。

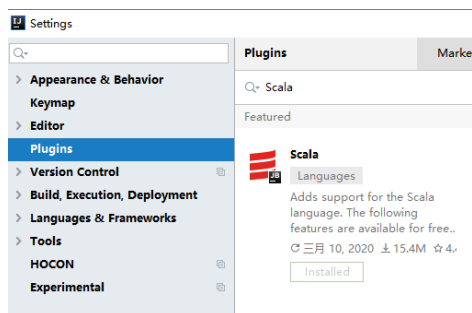
第2章 Spark 快速上手

在大数据早期的课程中我们已经学习了 MapReduce 框架的原理及基本使用，并了解了其底层数据处理的实现方式。接下来，就让咱们走进 Spark 的世界，了解一下它是如何带领我们完成数据处理的。

2.1 创建 Maven 项目

2.1.1 增加 Scala 插件

Spark 由 Scala 语言开发的，所以本课件接下来的开发所使用的语言也为 Scala，咱们当前使用的 Spark 版本为 3.0.0，默认采用的 Scala 编译版本为 2.12，所以后续开发时。我们依然采用这个版本。开发前请保证 IDEA 开发工具中含有 Scala 开发插件



2.1.2 增加依赖关系

修改 Maven 项目中的 POM 文件，增加 Spark 框架的依赖关系。本课件基于 Spark3.0 版本，使用时请注意对应版本。

```
<dependencies>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.12</artifactId>
    <version>3.0.0</version>
  </dependency>
</dependencies>
<build>
  <plugins>
    <!-- 该插件用于将 Scala 代码编译成 class 文件 -->
    <plugin>
      <groupId>net.alchim31.maven</groupId>
      <artifactId>scala-maven-plugin</artifactId>
      <version>3.2.2</version>
      <executions>
        <execution>
          <!-- 声明绑定到 maven 的 compile 阶段 -->
          <goals>
            <goal>testCompile</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网

```
</executions>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>3.1.0</version>
  <configuration>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>
  <executions>
    <execution>
      <id>make-assembly</id>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
    </execution>
  </executions>
</plugin>
</plugins>
</build>
```

2.1.3 WordCount

为了能直观地感受 Spark 框架的效果，接下来我们实现一个大数据学科中最常见的教学

案例 WordCount

```
// 创建 Spark 运行配置对象
val sparkConf = new SparkConf().setMaster("local[*]").setAppName("WordCount")

// 创建 Spark 上下文环境对象（连接对象）
val sc : SparkContext = new SparkContext(sparkConf)

// 读取文件数据
val fileRDD: RDD[String] = sc.textFile("input/word.txt")

// 将文件中的数据进行分词
val wordRDD: RDD[String] = fileRDD.flatMap( _.split(" ") )

// 转换数据结构 word => (word, 1)
val word2OneRDD: RDD[(String, Int)] = wordRDD.map((_,1))

// 将转换结构后的数据按照相同的单词进行分组聚合
val word2CountRDD: RDD[(String, Int)] = word2OneRDD.reduceByKey(_+_ )

// 将数据聚合结果采集到内存中
val word2Count: Array[(String, Int)] = word2CountRDD.collect()

// 打印结果
word2Count.foreach(println)

//关闭 Spark 连接
sc.stop()
```

执行过程中，会产生大量的执行日志，如果为了能够更好的查看程序的执行结果，可以在项

目的 resources 目录中创建 log4j.properties 文件，并添加日志配置信息：

```
log4j.rootCategory=ERROR, console
```

更多 Java -大数据 -前端 -python 人工智能资料下载，可百度访问：尚硅谷官网

```
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd
HH:mm:ss} %p %c{1}: %m%n

# Set the default spark-shell log level to ERROR. When running the spark-shell,
the
# log level for this class is used to overwrite the root logger's log level, so
that
# the user can have different defaults for the shell and regular Spark apps.
log4j.logger.org.apache.spark.repl.Main=ERROR

# Settings to quiet third party logs that are too verbose
log4j.logger.org.spark_project.jetty=ERROR
log4j.logger.org.spark_project.jetty.util.component.AbstractLifeCycle=ERROR
log4j.logger.org.apache.spark.repl.SparkIMain$exprTyper=ERROR
log4j.logger.org.apache.spark.repl.SparkILoop$SparkILoopInterpreter=ERROR
log4j.logger.org.apache.parquet=ERROR
log4j.logger.parquet=ERROR

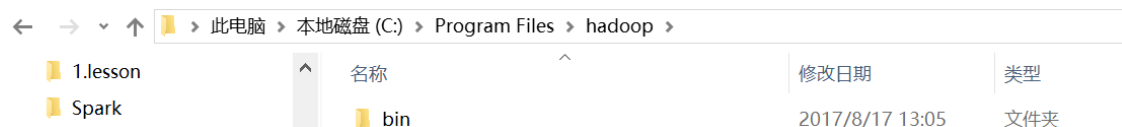
# SPARK-9183: Settings to avoid annoying messages when looking up nonexistent
UDFs in SparkSQL with Hive support
log4j.logger.org.apache.hadoop.hive.metastore.RetryingHMSHandler=FATAL
log4j.logger.org.apache.hadoop.hive.ql.exec.FunctionRegistry=ERROR
```

2.1.4 异常处理

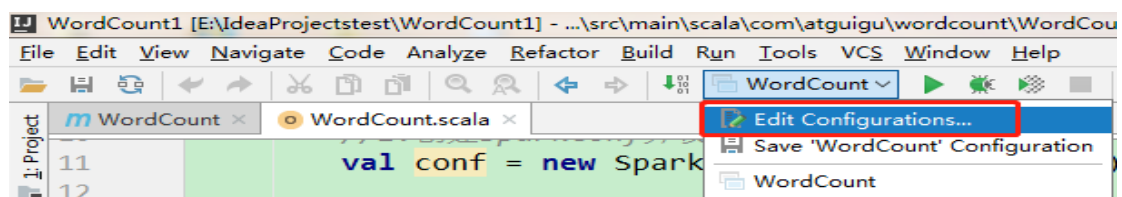
如果本机操作系统是 Windows，在程序中使用到了 Hadoop 相关的东西，比如写入文件到 HDFS，则会遇到如下异常：

```
2017-09-14 16:08:34,907 ERROR --- [main] org.apache.hadoop.util.Shell(line:303) : Failed to locate the winutils binary in the hadoop binary path
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
    at org.apache.hadoop.util.Shell.getQualifiedBinPath(Shell.java:278)
    at org.apache.hadoop.util.Shell.getWinUtilsPath(Shell.java:300)
    at org.apache.hadoop.util.Shell.<clinit>(Shell.java:293)
    at org.apache.hadoop.util.StringUtils.<clinit>(StringUtils.java:76)
    at org.apache.hadoop.conf.Configuration.getTrimmedStrings(Configuration.java:1546)
    at org.apache.hadoop.hdfs.DFSClient.<init>(DFSClient.java:519)
    at org.apache.hadoop.hdfs.DFSClient.<init>(DFSClient.java:453)
    at org.apache.hadoop.hdfs.DistributedFileSystem.initialize(DistributedFileSystem.java:136)
    at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:2433)
```

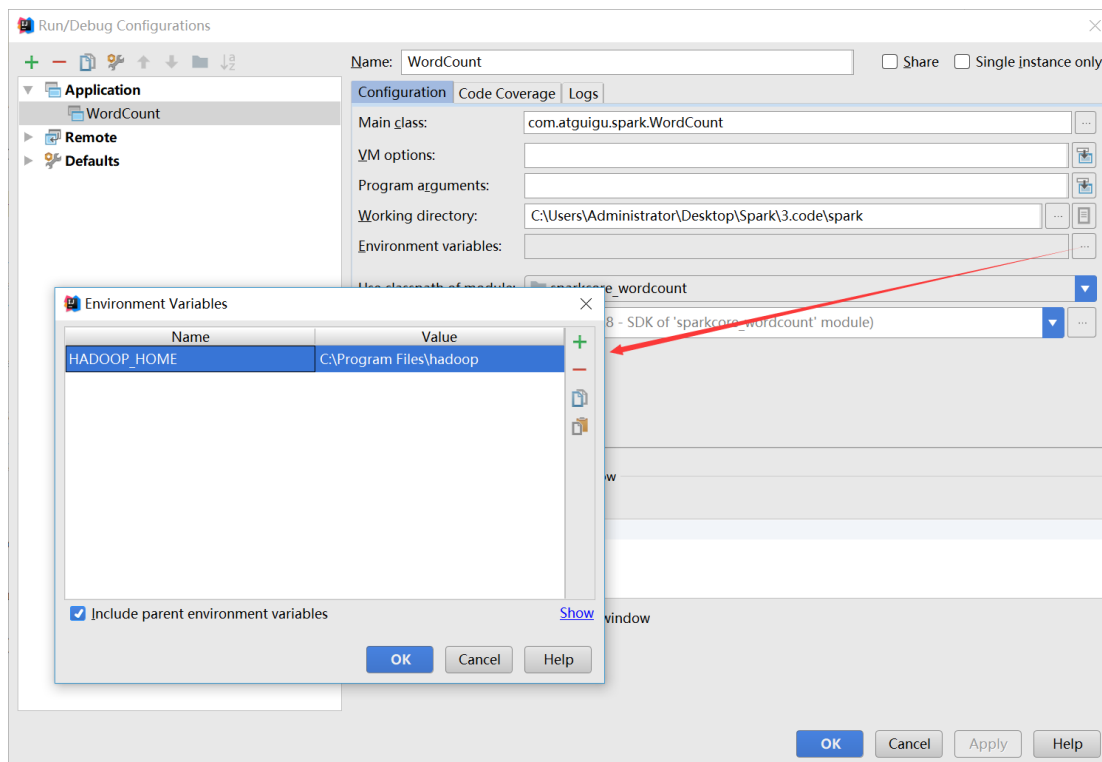
出现这个问题的原因，并不是程序的错误，而是 windows 系统用到了 hadoop 相关的服务，解决办法是通过配置关联到 windows 的系统依赖就可以了



在 IDEA 中配置 Run Configuration，添加 HADOOP_HOME 变量



更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网



第3章 Spark 运行环境

Spark 作为一个数据处理框架和计算引擎，被设计在所有常见的集群环境中运行，在国内工作中主流的环境为 Yarn，不过逐渐容器式环境也慢慢流行起来。接下来，我们就分别看看不同环境下 Spark 的运行



3.1 Local 模式

想啥呢，你之前一直在使用的模式可不是 Local 模式哟。所谓的 Local 模式，就是不需要其他任何节点资源就可以在本地执行 Spark 代码的环境，一般用于教学，调试，演示等，之前在 IDEA 中运行代码的环境我们称之为开发环境，不太一样。

3.1.1 解压缩文件

将 spark-3.0.0-bin-hadoop3.2.tgz 文件上传到 Linux 并解压缩，放置在指定位置，路径中不要包含中文或空格，课件后续如果涉及到解压缩操作，不再强调。

```
tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz -C /opt/module
cd /opt/module
mv spark-3.0.0-bin-hadoop3.2 spark-local
```

3.1.2 启动 Local 环境

1) 进入解压缩后的路径，执行如下指令

```
bin/spark-shell

[root@linux1 spark-local]# bin/spark-shell
20/06/19 16:10:23 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://linux1:4040
Spark context available as 'sc' (master = local[*], app id = local-1592554242091).
Spark session available as 'spark'.
Welcome to

  ____  _
 / ___|| | | |
| |___| |_| |
 \___|_||_|_|_|

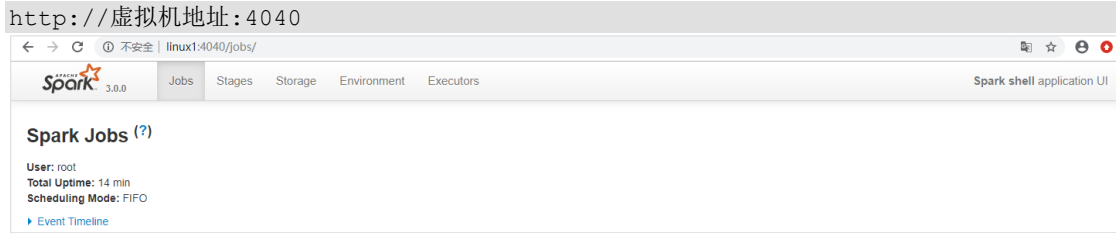
version 3.0.0

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_212)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网

2) 启动成功后，可以输入网址进行 Web UI 监控页面访问



3.1.3 命令行工具

在解压缩文件夹下的 data 目录中，添加 word.txt 文件。在命令行工具中执行如下代码指令（和 IDEA 中代码简化版一致）

```
sc.textFile("data/word.txt").flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).collect

scala> sc.textFile("data/word.txt").flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).collect
res0: Array[(String, Int)] = Array((Hello,2), (Scala,1), (Spark,1))
```

3.1.4 退出本地模式

按键 Ctrl+C 或输入 Scala 指令

```
:quit
```

3.1.5 提交应用

```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master local[2] \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```

- 1) --class 表示要执行程序的主类，此处可以更换为咱们自己写的应用程序
- 2) --master local[2] 部署模式，默认为本地模式，数字表示分配的虚拟 CPU 核数量
- 3) spark-examples_2.12-3.0.0.jar 运行的应用类所在的 jar 包，实际使用时，可以设定为咱们自己打的 jar 包
- 4) 数字 10 表示程序的入口参数，用于设定当前应用的任务数量

```
20/06/19 16:28:20 INFO Executor: Running task 9.0 in stage 0.0 (TID 9)
20/06/19 16:28:20 INFO TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 60 ms on linux1 (executor driver) (8/10)
20/06/19 16:28:20 INFO TaskSetManager: Finished task 7.0 in stage 0.0 (TID 7) in 77 ms on linux1 (executor driver) (9/10)
20/06/19 16:28:20 INFO Executor: Finished task 9.0 in stage 0.0 (TID 9). 957 bytes result sent to driver
20/06/19 16:28:20 INFO TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 28 ms on linux1 (executor driver) (10/10)
20/06/19 16:28:20 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
20/06/19 16:28:20 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 1.436 s
20/06/19 16:28:20 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
20/06/19 16:28:20 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
20/06/19 16:28:20 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 1.553778 s
Pi is roughly 3.1403991403991403
20/06/19 16:28:20 INFO SparkUI: Stopped Spark web UI at http://linux1:4040
20/06/19 16:28:20 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
20/06/19 16:28:20 INFO MemoryStore: MemoryStore cleared
20/06/19 16:28:20 INFO BlockManager: BlockManager stopped
20/06/19 16:28:20 INFO BlockManagerMaster: BlockManagerMaster stopped
20/06/19 16:28:20 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
20/06/19 16:28:20 INFO SparkContext: Successfully stopped SparkContext
20/06/19 16:28:20 INFO ShutdownHookManager: Shutdown hook called
20/06/19 16:28:20 INFO ShutdownHookManager: Deleting directory /tmp/spark-c447deaf-e3f1-4207-8d7e-d084860315dd
20/06/19 16:28:20 INFO ShutdownHookManager: Deleting directory /tmp/spark-61053b82-261e-4d9e-b7a7-031c18849b09
```

3.2 Standalone 模式

local 本地模式毕竟只是用来进行练习演示的，真实工作中还是要将应用提交到对应的集群中去执行，这里我们来看看只使用 Spark 自身节点运行的集群模式，也就是我们所谓的独立部署（Standalone）模式。Spark 的 Standalone 模式体现了经典的 master-slave 模式。

集群规划:

	Linux1	Linux2	Linux3
Spark	Worker Master	Worker	Worker

3.2.1 解压缩文件

将 spark-3.0.0-bin-hadoop3.2.tgz 文件上传到 Linux 并解压缩在指定位置

```
tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz -C /opt/module
cd /opt/module
mv spark-3.0.0-bin-hadoop3.2 spark-standalone
```

3.2.2 修改配置文件

1) 进入解压缩后路径的 conf 目录，修改 slaves.template 文件名为 slaves

```
mv slaves.template slaves
```

2) 修改 slaves 文件，添加 work 节点

```
linux1
linux2
linux3
```

3) 修改 spark-env.sh.template 文件名为 spark-env.sh

```
mv spark-env.sh.template spark-env.sh
```

4) 修改 spark-env.sh 文件，添加 JAVA_HOME 环境变量和集群对应的 master 节点

```
export JAVA_HOME=/opt/module/jdk1.8.0_144
SPARK_MASTER_HOST=linux1
SPARK_MASTER_PORT=7077
```

注意: 7077 端口，相当于 hadoop3 内部通信的 8020 端口，此处的端口需要确认自己的 Hadoop 配置

5) 分发 spark-standalone 目录

```
xsync spark-standalone
```

3.2.3 启动集群

1) 执行脚本命令:

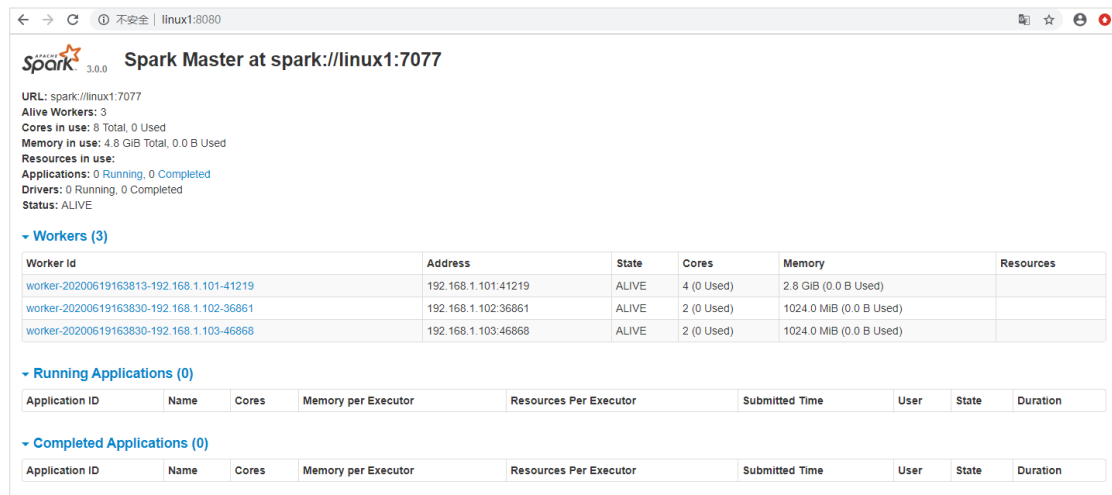
```
sbin/start-all.sh
```

```
[root@linux1 spark-standalone]# sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/module/spark-standalone/logs/spark-root-org.apache.spark.deploy.master.Master-1-linux1.out
linux1: starting org.apache.spark.deploy.worker.Worker, logging to /opt/module/spark-standalone/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-linux1.out
linux3: starting org.apache.spark.deploy.worker.Worker, logging to /opt/module/spark-standalone/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-linux3.out
linux2: starting org.apache.spark.deploy.worker.Worker, logging to /opt/module/spark-standalone/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-linux2.out
[root@linux1 spark-standalone]#
```

2) 查看三台服务器运行进程

```
=====linux1=====
3330 Jps
3238 Worker
3163 Master
=====linux2=====
2966 Jps
2908 Worker
=====linux3=====
2978 Worker
3036 Jps
```

3) 查看 Master 资源监控 Web UI 界面: <http://linux1:8080>



The screenshot shows the Spark Master Web UI at <http://linux1:8080>. The page title is "Spark Master at spark://linux1:7077". It displays various metrics: URL, Alive Workers (3), Cores in use (8 Total, 0 Used), Memory in use (4.8 GiB Total, 0.0 B Used), Resources in use, Applications (0 Running, 0 Completed), Drivers (0 Running, 0 Completed), and Status (ALIVE).

Under the "Workers (3)" section, there is a table with the following data:

Worker Id	Address	State	Cores	Memory	Resources
worker-20200619163813-192.168.1.101-41219	192.168.1.101:41219	ALIVE	4 (0 Used)	2.8 GiB (0.0 B Used)	
worker-20200619163830-192.168.1.102-36861	192.168.1.102:36861	ALIVE	2 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20200619163830-192.168.1.103-46868	192.168.1.103:46868	ALIVE	2 (0 Used)	1024.0 MiB (0.0 B Used)	

Below the workers table, there are sections for "Running Applications (0)" and "Completed Applications (0)", each with a table header showing columns for Application ID, Name, Cores, Memory per Executor, Resources Per Executor, Submitted Time, User, State, and Duration.

3.2.4 提交应用

```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://linux1:7077 \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```

- 1) --class 表示要执行程序的主类
- 2) --master spark://linux1:7077 独立部署模式，连接到 Spark 集群
- 3) spark-examples_2.12-3.0.0.jar 运行类所在的 jar 包
- 4) 数字 10 表示程序的入口参数，用于设定当前应用的任务数量

```

20/06/19 16:42:59 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Registered executor NettyRpcEndpointRef(spark-client://Executor)
(192.168.1.101:54494) with ID 0
20/06/19 16:42:59 INFO TaskSetManager: Finished task 3.0 in stage 0.0 (TID 3) in 13420 ms on 192.168.1.102 (executor 1) (9/10)
20/06/19 16:42:59 INFO TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 13431 ms on 192.168.1.102 (executor 1) (10/10)
20/06/19 16:42:59 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
20/06/19 16:42:59 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 17.999 s
20/06/19 16:42:59 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
20/06/19 16:42:59 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
20/06/19 16:42:59 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 18.633508 s
Pi is roughly 3.141943141943142
20/06/19 16:42:59 INFO SparkUI: Stopped Spark web UI at http://linux1:4040
20/06/19 16:42:59 INFO StandaloneSchedulerBackend: Shutting down all executors
20/06/19 16:42:59 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
20/06/19 16:42:59 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
20/06/19 16:43:00 INFO MemoryStore: MemoryStore cleared
20/06/19 16:43:00 INFO BlockManager: BlockManager stopped
20/06/19 16:43:00 INFO BlockManagerMaster: BlockManagerMaster stopped
20/06/19 16:43:00 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
20/06/19 16:43:00 INFO SparkContext: Successfully stopped SparkContext
20/06/19 16:43:00 INFO ShutdownHookManager: Shutdown hook called
20/06/19 16:43:00 INFO ShutdownHookManager: Deleting directory /tmp/spark-62035251-f7d8-46d7-95c7-a03c9a08edce
20/06/19 16:43:00 INFO ShutdownHookManager: Deleting directory /tmp/spark-2d166a60-b447-43fb-8115-811f379fea8e

```

执行任务时，会产生多个 Java 进程

```

-----linux1-----
4227 Master
4475 CoarseGrainedExecutorBackend
4332 Worker
4524 Jps
4399 SparkSubmit
-----linux2-----
3463 Worker
3561 Jps
3515 CoarseGrainedExecutorBackend
-----linux3-----
3573 Jps
3462 Worker
3516 CoarseGrainedExecutorBackend

```

执行节点进程

提交节点进程

执行任务时，默认采用服务器集群节点的总核数，每个节点内存 1024M。

Completed Applications (2)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20200619164410-0001	Spark Pi	8	1024.0 MIB		2020/06/19 16:44:10	root	FINISHED	14 s
app-20200619164212-0000	Spark Pi	8	1024.0 MIB		2020/06/19 16:42:12	root	FINISHED	47 s

3.2.5 提交参数说明

在提交应用中，一般会同时一些提交参数

```

bin/spark-submit \
--class <main-class>
--master <master-url> \
... # other options
<application-jar> \
[application-arguments]

```

参数	解释	可选值举例
--class	Spark 程序中包含主函数的类	
--master	Spark 程序运行的模式(环境)	模式: local[*]、spark://linux1:7077、Yarn
--executor-memory 1G	指定每个 executor 可用内存为 1G	符合集群内存配置即可，具体情况具体分析。
--total-executor-cores 2	指定所有 executor 使用的 cpu 核数为 2 个	
--executor-cores	指定每个 executor 使用的 cpu 核数	
application-jar	打包好的应用 jar，包含依赖。这个 URL 在集群中全局可见。比如 hdfs:// 共享存储系统，如果是	

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网

	file:// path，那么所有的节点的 path 都包含同样的 jar	
application-arguments	传给 main()方法的参数	

3.2.6 配置历史服务

由于 spark-shell 停止掉后，集群监控 linux1:4040 页面就看不到历史任务的运行情况，所以开发时都配置历史服务器记录任务运行情况。

1) 修改 spark-defaults.conf.template 文件名为 spark-defaults.conf

```
mv spark-defaults.conf.template spark-defaults.conf
```

2) 修改 spark-default.conf 文件，配置日志存储路径

```
spark.eventLog.enabled      true
spark.eventLog.dir          hdfs://linux1:8020/directory
```

注意：需要启动 **hadoop** 集群，**HDFS** 上的 **directory** 目录需要提前存在。

```
sbin/start-dfs.sh
hadoop fs -mkdir /directory
```

3) 修改 spark-env.sh 文件，添加日志配置

```
export SPARK_HISTORY_OPTS="
-Dspark.history.ui.port=18080
-Dspark.history.fs.logDirectory=hdfs://linux1:8020/directory
-Dspark.history.retainedApplications=30"
```

- 参数 1 含义：WEB UI 访问的端口号为 18080
- 参数 2 含义：指定历史服务器日志存储路径
- 参数 3 含义：指定保存 Application 历史记录的个数，如果超过这个值，旧的应用程序信息将被删除，这个是内存中的应用数，而不是页面上显示的应用数。

4) 分发配置文件

```
xsync conf
```

5) 重新启动集群和历史服务

```
sbin/start-all.sh
sbin/start-history-server.sh
```

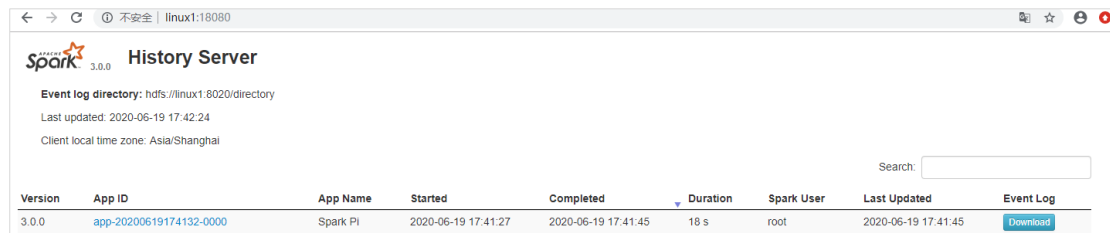
6) 重新执行任务

```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://linux1:7077 \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```



```
20/06/19 17:41:32 INFO StandaloneSchedulerBackend: Connected to Spark cluster with app ID app-20200619174132-0000
20/06/19 17:41:32 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 34345.
20/06/19 17:41:32 INFO NettyBlockTransferService: Server created on linux1:34345
20/06/19 17:41:32 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
20/06/19 17:41:32 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20200619174132-0000/0 on worker-20200619174053-192.168.1.101-33517 (192.168.1.101:33517) with 4 core(s)
20/06/19 17:41:32 INFO StandaloneSchedulerBackend: Granted executor ID app-20200619174132-0000/0 on hostPort 192.168.1.101:33517 with 4 core(s), 1024.0 MiB RAM
20/06/19 17:41:32 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20200619174132-0000/1 on worker-20200619174040-192.168.1.102-44849 (192.168.1.102:44849) with 2 core(s)
20/06/19 17:41:32 INFO StandaloneSchedulerBackend: Granted executor ID app-20200619174132-0000/1 on hostPort 192.168.1.102:44849 with 2 core(s), 1024.0 MiB RAM
20/06/19 17:41:32 INFO StandaloneAppClient$ClientEndpoint: Executor added: app-20200619174132-0000/2 on worker-20200619174047-192.168.1.103-33749 (192.168.1.103:33749) with 2 core(s)
20/06/19 17:41:32 INFO StandaloneSchedulerBackend: Granted executor ID app-20200619174132-0000/2 on hostPort 192.168.1.103:33749 with 2 core(s), 1024.0 MiB RAM
20/06/19 17:41:32 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, linux1, 34345, None)
20/06/19 17:41:32 INFO BlockManagerMasterEndpoint: Registering block manager linux1:34345 with 366.3 MiB RAM, BlockManagerId(driver, linux1, 34345, None)
20/06/19 17:41:32 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, linux1, 34345, None)
20/06/19 17:41:32 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, linux1, 34345, None)
```

7) 查看历史服务: <http://linux1:18080>



Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.0.0	app-20200619174132-0000	Spark Pi	2020-06-19 17:41:27	2020-06-19 17:41:45	18 s	root	2020-06-19 17:41:45	Download

3.2.7 配置高可用 (HA)

所谓的高可用是因为当前集群中的 Master 节点只有一个, 所以会存在单点故障问题。所以为了解决单点故障问题, 需要在集群中配置多个 Master 节点, 一旦处于活动状态的 Master 发生故障时, 由备用 Master 提供服务, 保证作业可以继续执行。这里的高可用一般采用 Zookeeper 设置

集群规划:

	Linux1	Linux2	Linux3
Spark	Master Zookeeper Worker	Master Zookeeper Worker	Zookeeper Worker

1) 停止集群

```
sbin/stop-all.sh
```

2) 启动 Zookeeper

```
xstart zk
```

3) 修改 spark-env.sh 文件添加如下配置

注释如下内容:

```
#SPARK_MASTER_HOST=linux1
#SPARK_MASTER_PORT=7077
```

添加如下内容:

```
#Master 监控页面默认访问端口为 8080, 但是可能会和 Zookeeper 冲突, 所以改成 8989, 也可以自定义, 访问 UI 监控页面时请注意
SPARK_MASTER_WEBUI_PORT=8989

export SPARK_DAEMON_JAVA_OPTS=""
```

更多 Java - 大数据 - 前端 - python 人工智能资料下载, 可百度访问: 尚硅谷官网

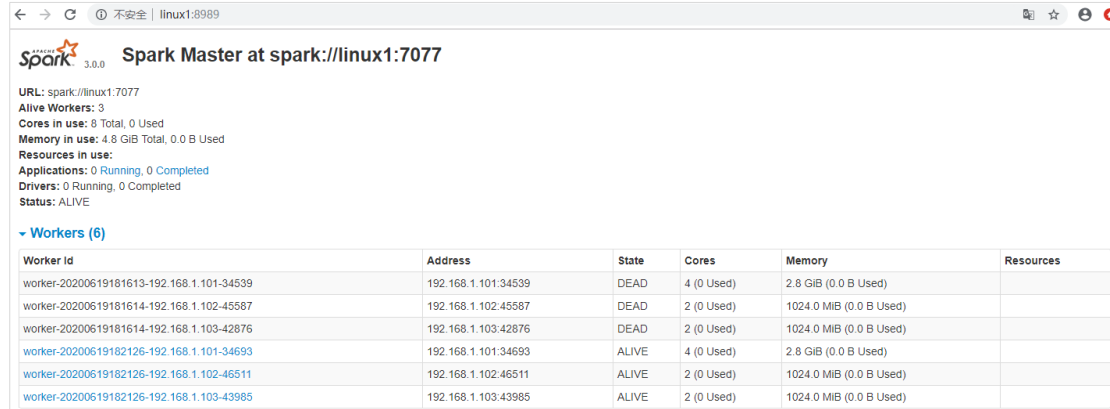

```
-Dspark.deploy.recoveryMode=ZOOKEEPER
-Dspark.deploy.zookeeper.url=linux1,linux2,linux3
-Dspark.deploy.zookeeper.dir=/spark"
```

4) 分发配置文件

```
xsync conf/
```

5) 启动集群

```
sbin/start-all.sh
```



Spark Master at spark://linux1:7077

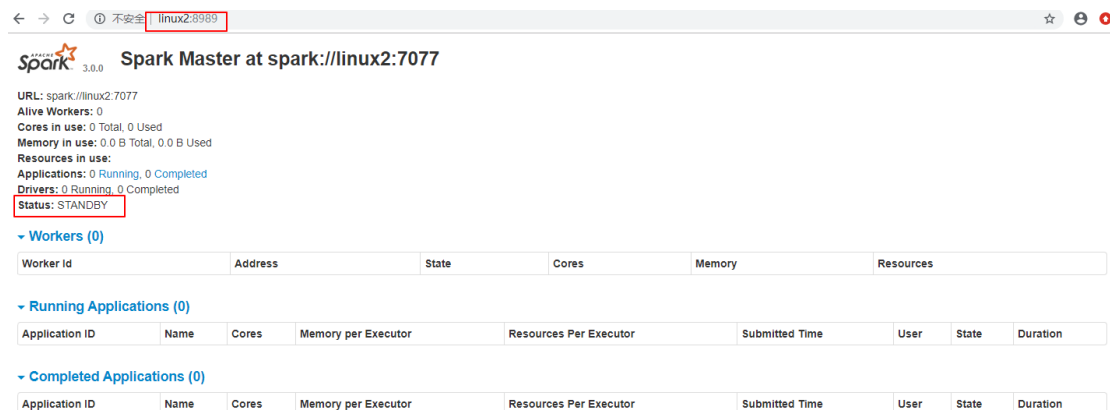
URL: spark://linux1:7077
 Alive Workers: 3
 Cores in use: 8 Total, 0 Used
 Memory in use: 4.8 GiB Total, 0.0 B Used
 Resources in use:
 Applications: 0 Running, 0 Completed
 Drivers: 0 Running, 0 Completed
 Status: ALIVE

Workers (6)

Worker Id	Address	State	Cores	Memory	Resources
worker-20200619181613-192.168.1.101-34539	192.168.1.101:34539	DEAD	4 (0 Used)	2.8 GiB (0.0 B Used)	
worker-20200619181614-192.168.1.102-45587	192.168.1.102:45587	DEAD	2 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20200619181614-192.168.1.103-42876	192.168.1.103:42876	DEAD	2 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20200619182126-192.168.1.101-34693	192.168.1.101:34693	ALIVE	4 (0 Used)	2.8 GiB (0.0 B Used)	
worker-20200619182126-192.168.1.102-46511	192.168.1.102:46511	ALIVE	2 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20200619182126-192.168.1.103-43985	192.168.1.103:43985	ALIVE	2 (0 Used)	1024.0 MiB (0.0 B Used)	

6) 启动 linux2 的单独 Master 节点，此时 linux2 节点 Master 状态处于备用状态

```
[root@linux2 spark-standalone]# sbin/start-master.sh
```



Spark Master at spark://linux2:7077

URL: spark://linux2:7077
 Alive Workers: 0
 Cores in use: 0 Total, 0 Used
 Memory in use: 0.0 B Total, 0.0 B Used
 Resources in use:
 Applications: 0 Running, 0 Completed
 Drivers: 0 Running, 0 Completed
 Status: STANDBY

Workers (0)

Worker Id	Address	State	Cores	Memory	Resources
-----------	---------	-------	-------	--------	-----------

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

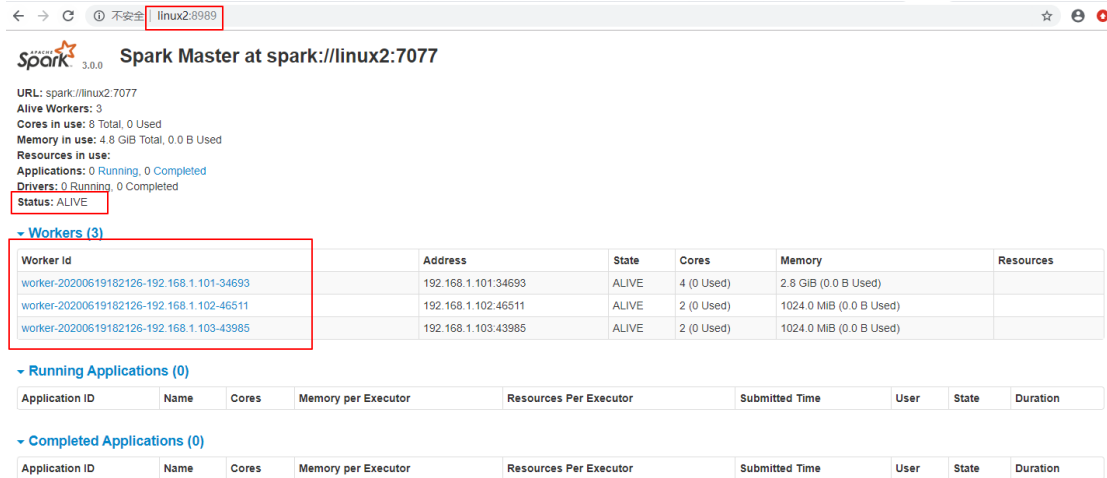
7) 提交应用到高可用集群

```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://linux1:7077,linux2:7077 \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```

8) 停止 linux1 的 Master 资源监控进程

```
[root@linux1 spark-standalone]# jps
4673 JobHistoryServer
6802 Worker
6900 Jps
4342 DataNode
4966 QuorumPeerMain
4151 NameNode
4794 NodeManager
6703 Master
[root@linux1 spark-standalone]# kill -9 6703
[root@linux1 spark-standalone]#
```

- 9) 查看 linux2 的 Master 资源监控 Web UI，稍等一段时间后，linux2 节点的 Master 状态提升为活动状态



The screenshot shows the Spark Master Web UI for spark://linux2:7077. The status is ALIVE. The Workers section shows 3 workers, all in an ALIVE state. The Running Applications section shows 0 applications.

Worker Id	Address	State	Cores	Memory	Resources
worker-20200619182126-192.168.1.101-34693	192.168.1.101:34693	ALIVE	4 (0 Used)	2.8 GiB (0.0 B Used)	
worker-20200619182126-192.168.1.102-46511	192.168.1.102:46511	ALIVE	2 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20200619182126-192.168.1.103-43985	192.168.1.103:43985	ALIVE	2 (0 Used)	1024.0 MiB (0.0 B Used)	

3.3 Yarn 模式

独立部署（Standalone）模式由 Spark 自身提供计算资源，无需其他框架提供资源。这种方式降低了和其他第三方资源框架的耦合性，独立性非常强。但是你也要记住，Spark 主要是计算框架，而不是资源调度框架，所以本身提供的资源调度并不是它的强项，所以还是和其他专业的资源调度框架集成会更靠谱一些。所以接下来我们来学习在强大的 Yarn 环境下 Spark 是如何工作的（其实是因为在国内工作中，Yarn 使用的非常多）。

3.3.1 解压缩文件

将 spark-3.0.0-bin-hadoop3.2.tgz 文件上传到 linux 并解压缩，放置在指定位置。

```
tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz -C /opt/module
cd /opt/module
mv spark-3.0.0-bin-hadoop3.2 spark-yarn
```

3.3.2 修改配置文件

- 1) 修改 hadoop 配置文件/opt/module/hadoop/etc/hadoop/yarn-site.xml，并分发

```
<!--是否启动一个线程检查每个任务正使用的物理内存量，如果任务超出分配值，则直接将其杀掉，默认是 true -->
<property>
  <name>yarn.nodemanager.pmem-check-enabled</name>
  <value>>false</value>
</property>

<!--是否启动一个线程检查每个任务正使用的虚拟内存量，如果任务超出分配值，则直接将其杀掉，默认是 true -->
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>>false</value>
</property>
```

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网

2) 修改 conf/spark-env.sh, 添加 JAVA_HOME 和 YARN_CONF_DIR 配置

```
mv spark-env.sh.template spark-env.sh
...
export JAVA_HOME=/opt/module/jdk1.8.0_144
YARN_CONF_DIR=/opt/module/hadoop/etc/hadoop
```

3.3.3 启动 HDFS 以及 YARN 集群

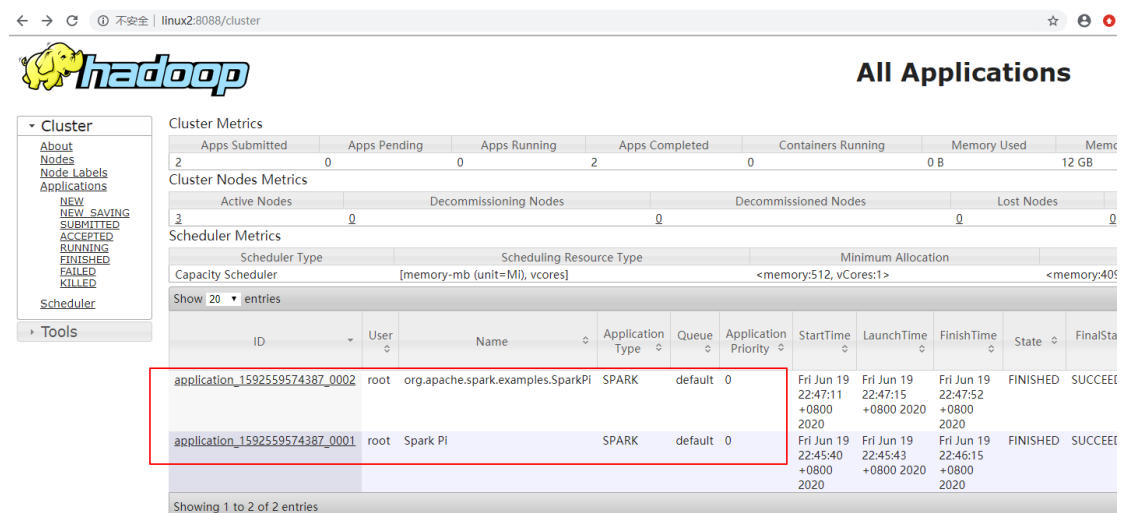
瞅啥呢, 自己启动去!

3.3.4 提交应用

```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master yarn \
--deploy-mode cluster \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```

```
2020-06-19 22:47:42,632 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:43,875 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:44,879 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:46,358 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:47,601 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:48,621 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:49,629 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:50,643 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:51,673 INFO yarn.Client: Application report for application_1592559574387_0002 (state: RUNNING)
2020-06-19 22:47:52,675 INFO yarn.Client: Application report for application_1592559574387_0002 (state: FINISHED)
2020-06-19 22:47:52,676 INFO yarn.Client:
client token: N/A
diagnostics: N/A
ApplicationMaster host: linux1
ApplicationMaster RPC port: 33892
queue: default
start time: 1592578031667
final status: SUCCEEDED
tracking URL: http://linux2:8088/proxy/application_1592559574387_0002/
user: root
2020-06-19 22:47:52,726 INFO util.ShutdownHookManager: Shutdown hook called
2020-06-19 22:47:52,730 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-aedaddf7-b276-49ee-a1b6-d98e31ab52da
2020-06-19 22:47:52,736 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-4850a8e3-e949-490d-95cb-649f423d678d
```

查看 <http://linux2:8088> 页面, 点击 History, 查看历史页面



The screenshot shows the Hadoop UI at <http://linux2:8088>. The left sidebar contains navigation links: Cluster, About, Nodes, Node Labels, Applications, NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, and Scheduler. The main content area displays 'All Applications' with a table of application metrics. The table has columns for ID, User, Name, Application Type, Queue, Application Priority, Start Time, Launch Time, Finish Time, State, and Final State. Two applications are listed, both in a 'FINISHED' state. The first application is 'application_1592559574387_0002' and the second is 'application_1592559574387_0001'. Both were submitted by 'root' and are of type 'SPARK'.

ID	User	Name	Application Type	Queue	Application Priority	Start Time	Launch Time	Finish Time	State	Final State
application_1592559574387_0002	root	org.apache.spark.examples.SparkPi	SPARK	default	0	Fri Jun 19 22:47:11 +0800 2020	Fri Jun 19 22:47:15 +0800 2020	Fri Jun 19 22:47:52 +0800 2020	FINISHED	SUCCEEDED
application_1592559574387_0001	root	Spark Pi	SPARK	default	0	Fri Jun 19 22:45:40 +0800 2020	Fri Jun 19 22:45:43 +0800 2020	Fri Jun 19 22:46:15 +0800 2020	FINISHED	SUCCEEDED

```
command:
{{JAVA_HOME}}/bin/java \
  -server \
  -Xmx1024m \
  -Djava.io.tmpdir={{PWD}}/tmp \
  '-Dspark.driver.port=48441' \
  -Dspark.yarn.app.container.log.dir=LOG_DIR \
  -XX:OnOutOfMemoryError=kill %p \
  org.apache.spark.executor.CoarseGrainedExecutorBackend \
  --driver-url \
  spark://CoarseGrainedScheduler@linux1:48441 \
  --executor-id \
  <executorId> \
  --hostname \
  <hostname> \
  --cores \
  1 \
  --app-id \
  application_1587439373824_0001 \
  --user-class-path \
  file:{{PWD}}/__app__.jar \
  1><LOG_DIR>/stdout \
  2><LOG_DIR>/stderr

resources:
__spark_libs__ -> resource { scheme: "hdfs" host: "linux1" port: 9000 file: "/user/root/.sparkStaging/application_1587439373824_0001/__spark_libs__2267073519899007339.zip" }
__spark_conf__ -> resource { scheme: "hdfs" host: "linux1" port: 9000 file: "/user/root/.sparkStaging/application_1587439373824_0001/__spark_conf__.zip" } size: 193430 timestamp: 1587439373824000000 }
```

3.3.5 配置历史服务器

1) 修改 spark-defaults.conf.template 文件名为 spark-defaults.conf

```
mv spark-defaults.conf.template spark-defaults.conf
```

2) 修改 spark-default.conf 文件，配置日志存储路径

```
spark.eventLog.enabled      true
spark.eventLog.dir          hdfs://linux1:8020/directory
```

注意：需要启动 hadoop 集群，HDFS 上的目录需要提前存在。

```
[root@linux1 hadoop]# sbin/start-dfs.sh
[root@linux1 hadoop]# hadoop fs -mkdir /directory
```

3) 修改 spark-env.sh 文件，添加日志配置

```
export SPARK_HISTORY_OPTS="
-Dspark.history.ui.port=18080
-Dspark.history.fs.logDirectory=hdfs://linux1:8020/directory
-Dspark.history.retainedApplications=30"
```

- 参数 1 含义：WEB UI 访问的端口号为 18080
- 参数 2 含义：指定历史服务器日志存储路径
- 参数 3 含义：指定保存 Application 历史记录个数，如果超过这个值，旧的应用程序信息将被删除，这个是内存中的应用数，而不是页面上显示的应用数。

4) 修改 spark-defaults.conf

```
spark.yarn.historyServer.address=linux1:18080
spark.history.ui.port=18080
```

5) 启动历史服务

```
sbin/start-history-server.sh
```

6) 重新提交应用

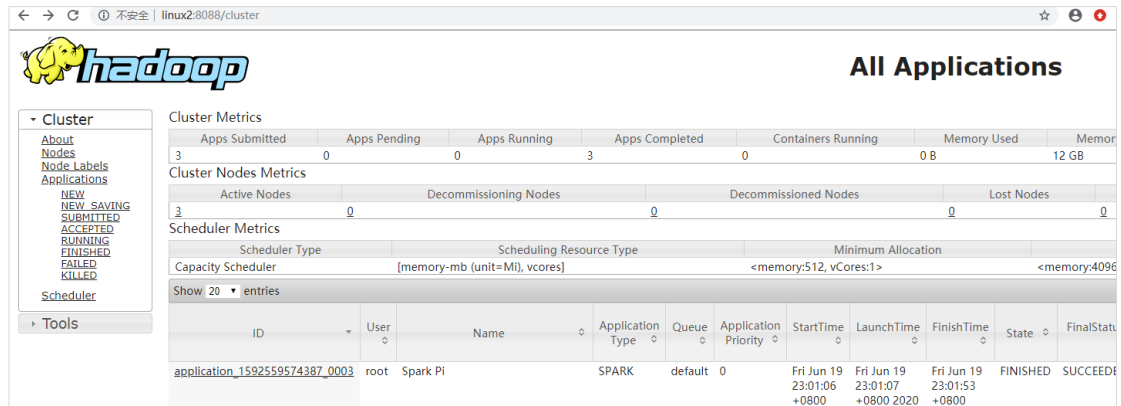
```
bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master yarn \
--deploy-mode client \
./examples/jars/spark-examples_2.12-3.0.0.jar \
10
```

```

2020-06-19 23:01:53,036 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool
2020-06-19 23:01:53,038 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 14.375 s
2020-06-19 23:01:53,048 INFO cluster.YarnScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
2020-06-19 23:01:53,051 INFO cluster.YarnScheduler: Killing all running tasks in stage 0: Stage finished
2020-06-19 23:01:53,056 INFO scheduler.DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 14.488243 s
Pi is roughly 3.138131138131138
2020-06-19 23:01:53,088 INFO server.AbstractConnector: Stopped Spark@3e587920(HTTP/1.1,[http/1.1]){0.0.0.0:4040}
2020-06-19 23:01:53,092 INFO ui.SparkUI: Stopped Spark web UI at http://linux1:4040
2020-06-19 23:01:53,103 INFO cluster.YarnClientSchedulerBackend: Interrupting monitor thread
2020-06-19 23:01:53,159 INFO cluster.YarnClientSchedulerBackend: Shutting down all executors
2020-06-19 23:01:53,160 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
2020-06-19 23:01:53,173 INFO cluster.YarnClientSchedulerBackend: YARN client scheduler backend Stopped
2020-06-19 23:01:53,308 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2020-06-19 23:01:53,338 INFO memory.MemoryStore: MemoryStore cleared
2020-06-19 23:01:53,339 INFO storage.BlockManager: BlockManager stopped
2020-06-19 23:01:53,357 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2020-06-19 23:01:53,362 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!

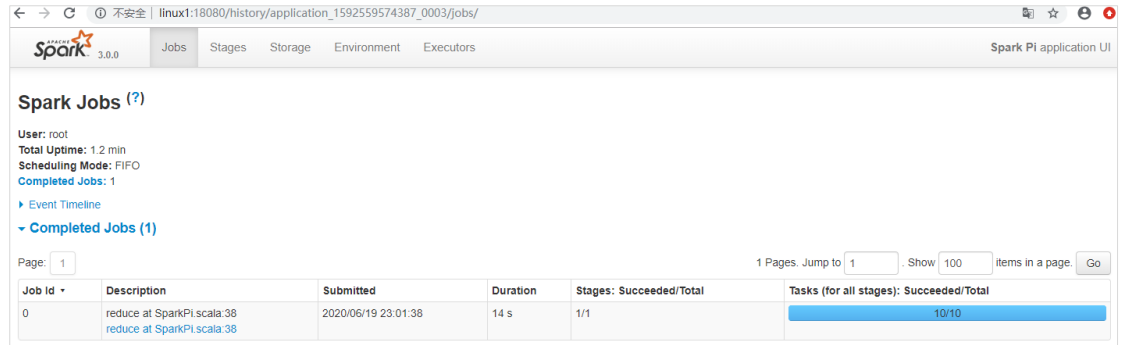
```

7) Web 页面查看日志: <http://linux2:8088>



The screenshot shows the Hadoop All Applications web interface. It displays cluster metrics, scheduler metrics, and a table of applications. The application 'application_1592559574387_0003' is highlighted, showing it is a SPARK job in a FINISHED state.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalState
application_1592559574387_0003	root	Spark Pi	SPARK	default	0	Fri Jun 19 23:01:06 +0800	Fri Jun 19 23:01:07 +0800	Fri Jun 19 23:01:53 +0800	FINISHED	SUCCEEDED

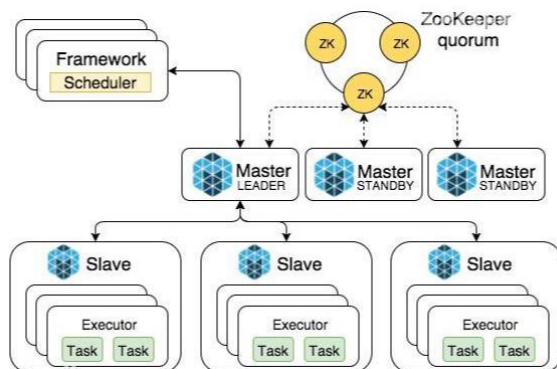


The screenshot shows the Spark Jobs web interface. It displays job details for 'application_1592559574387_0003'. The job is in a completed state, and the table shows the job details.

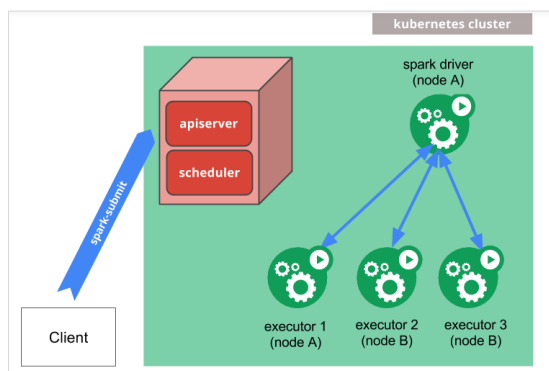
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	reduce at SparkPi.scala:38 reduce at SparkPi.scala:38	2020/06/19 23:01:38	14 s	1/1	10/10

3.4 K8S & Mesos 模式

Mesos 是 Apache 下的开源分布式资源管理框架，它被称为是分布式系统的内核,在 Twitter 得到广泛使用,管理着 Twitter 超过 30,000 台服务器上的应用部署，但是在国内，依然使用着传统的 Hadoop 大数据框架，所以国内使用 Mesos 框架的并不多，但是原理其实都差不多，这里我们就不做过多讲解了。



容器化部署是目前业界很流行的一项技术，基于 Docker 镜像运行能够让用户更加方便地对应用进行管理和运维。容器管理工具中最为流行的就是 Kubernetes (k8s)，而 Spark 也在最近的版本中支持了 k8s 部署模式。这里我们也不做过多的讲解。给个链接大家自己感受一下：<https://spark.apache.org/docs/latest/running-on-kubernetes.html>



3.5 Windows 模式

在同学们自己学习时，每次都需要启动虚拟机，启动集群，这是一个比较繁琐的过程，并且会占大量的系统资源，导致系统执行变慢，不仅仅影响学习效果，也影响学习进度，Spark 非常暖心地提供了可以在 windows 系统下启动本地集群的方式，这样，在不使用虚拟机的情况下，也能学习 Spark 的基本使用，摸摸哒！



在后续的教学，为了能够给同学们更加流畅的教学效果和教学体验，我们一般情况下都会采用 windows 系统的集群来学习 Spark。

3.5.1 解压缩文件

将文件 spark-3.0.0-bin-hadoop3.2.tgz 解压缩到无中文无空格的路径中

更多 Java -大数据 -前端 -python 人工智能资料下载，可百度访问：尚硅谷官网

3.5.2 启动本地环境

- 1) 执行解压缩文件路径下 bin 目录中的 spark-shell.cmd 文件，启动 Spark 本地环境

```
20/06/19 23:38:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://windows10.microdone.cn:4040
Spark context available as 'sc' (master = local[*], app id = local-1592581095879).
Spark session available as 'spark'.
Welcome to

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  __/
    _||_|

Spark version 3.0.0

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_111)
Type in expressions to have them evaluated.
Type :help for more information.

scala> _
```

- 2) 在 bin 目录中创建 input 目录，并添加 word.txt 文件，在命令行中输入脚本代码

```
20/06/19 23:43:33 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://windows10.microdone.cn:4040
Spark context available as 'sc' (master = local[*], app id = local-1592581423102).
Spark session available as 'spark'.
Welcome to

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  __/
    _||_|

Spark version 3.0.0

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_111)
Type in expressions to have them evaluated.
Type :help for more information.

scala> sc.textFile("input/word.txt").flatMap(_._split(" ")).map(_._1).reduceByKey(_+_).collect()
20/06/19 23:43:55 WARN ProcessMetrics: Exception when trying to compute page size, as a result reporting of ProcessMetrics metrics is stopped
res0: Array[(String, Int)] = Array((world,1), (hello,1))
```

3.5.3 命令行提交应用

在 DOS 命令行窗口中执行提交指令

```
spark-submit --class org.apache.spark.examples.SparkPi --master local[2] ../examples/jars/spark-examples_2.12-3.0.0.jar 10
```

```
20/06/19 23:45:44 INFO Executor: Running task 9.0 in stage 0.0 (TID 9)
20/06/19 23:45:44 INFO TaskSetManager: Finished task 7.0 in stage 0.0 (TID 7) in 58 ms on windows10.microdone.cn (executor driver) (8/10)
20/06/19 23:45:44 INFO Executor: Finished task 8.0 in stage 0.0 (TID 8). 914 bytes result sent to driver
20/06/19 23:45:44 INFO TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 38 ms on windows10.microdone.cn (executor driver) (9/10)
20/06/19 23:45:44 INFO Executor: Finished task 9.0 in stage 0.0 (TID 9). 914 bytes result sent to driver
20/06/19 23:45:44 INFO TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 37 ms on windows10.microdone.cn (executor driver) (10/10)
20/06/19 23:45:44 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
20/06/19 23:45:44 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 1.780 s
20/06/19 23:45:44 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
20/06/19 23:45:44 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
20/06/19 23:45:44 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 1.853504 s
Pi is roughly 3.1391991391991394
20/06/19 23:45:44 INFO SparkUI: Stopped Spark web UI at http://windows10.microdone.cn:4040
20/06/19 23:45:44 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
20/06/19 23:45:44 INFO MemoryStore: MemoryStore cleared
20/06/19 23:45:44 INFO BlockManager: BlockManager stopped
20/06/19 23:45:44 INFO BlockManagerMaster: BlockManagerMaster stopped
20/06/19 23:45:44 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
20/06/19 23:45:44 INFO SparkContext: Successfully stopped SparkContext
20/06/19 23:45:44 INFO ShutdownHookManager: Shutdown hook called
20/06/19 23:45:44 INFO ShutdownHookManager: Deleting directory C:\Users\18801\AppData\Local\Temp\spark-8bd95ad0-30ae-45d7-9257-94304c799329
20/06/19 23:45:44 INFO ShutdownHookManager: Deleting directory C:\Users\18801\AppData\Local\Temp\spark-c471b8d1-6da1-482f-b2bd-d2f5be278e48
```

更多 Java - 大数据 - 前端 - python 人工智能资料下载，可百度访问：尚硅谷官网