# 第1章 Spark概述

## 1.1 什么是Spark

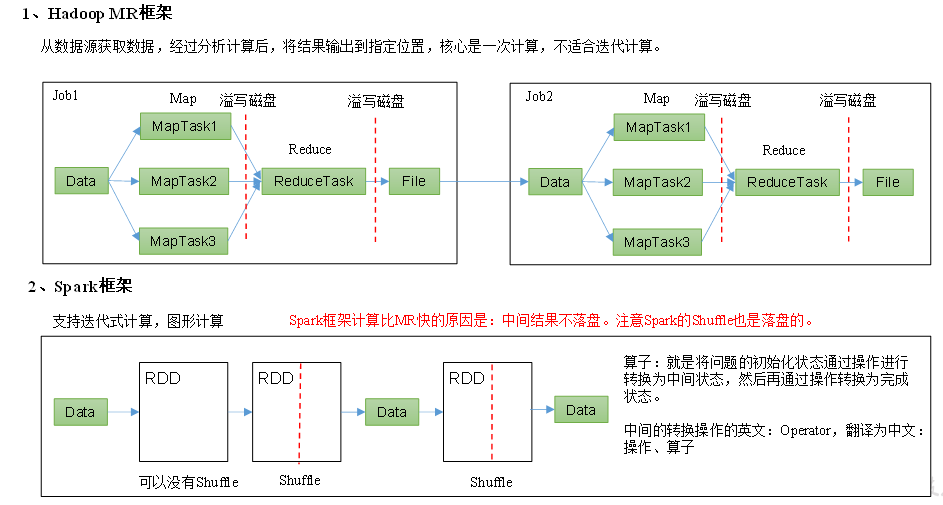
对比：Hadoop主要解决，海量数据的存储和海量数据的分析计算。

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。

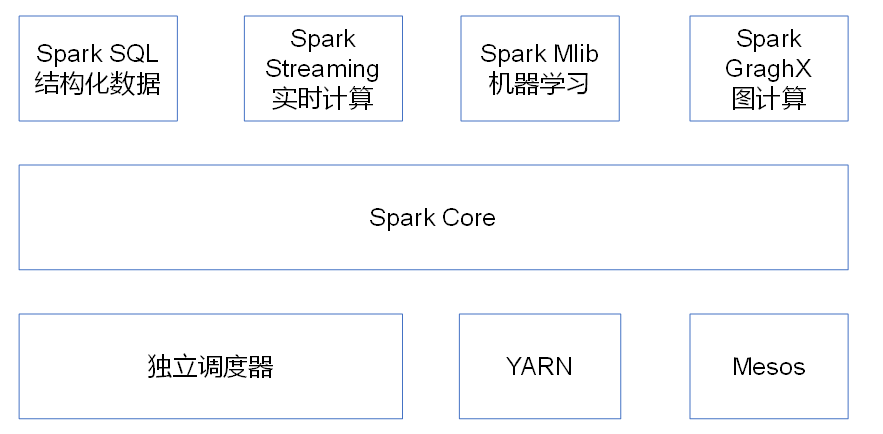
## 1.2 Hadoop与Spark历史

Hadoop的Yarn框架比Spark框架诞生的晚，所以Spark自己也设计了一套资源调度框架。

## 1.3 Hadoop与Spark框架对比



## 1.4 Spark内置模块



Spark Core：实现了Spark的基本功能，包含任务调度、内存管理、错误恢复、与存储系统交互等模块。Spark Core中还包含了对弹性分布式数据集(Resilient Distributed DataSet，简称RDD)的API定义。

Spark SQL：是Spark用来操作结构化数据的程序包。通过Spark SQL，我们可以使用 SQL或者Apache Hive版本的HQL来查询数据。Spark SQL支持多种数据源，比如Hive表、Parquet以及JSON等。

Spark Streaming：是Spark提供的对实时数据进行流式计算的组件。提供了用来操作数据流的API，并且与Spark Core中的 RDD API高度对应。

Spark MLlib：提供常见的机器学习功能的程序库。包括分类、回归、聚类、协同过滤等，还提供了模型评估、数据 导入等额外的支持功能。

Spark GraphX：主要用于图形并行计算和图挖掘系统的组件。

集群管理器：Spark设计为可以高效地在一个计算节点到数千个计算节点之间伸缩计算。为了实现这样的要求，同时获得最大灵活性，Spark支持在各种集群管理器（Cluster Manager）上运行，包括Hadoop YARN、Apache Mesos，以及Spark自带的一个简易调度器，叫作独立调度器。

# 第2章 Spark运行模式

部署Spark集群大体上分为两种模式：**单机模式与集群模式**

大多数分布式框架都支持单机模式，方便开发者调试框架的运行环境。但是在生产环境中，并不会使用单机模式。因此，后续直接按照集群模式部署Spark集群。

下面详细列举了Spark目前支持的部署模式。

（1）**Local模式：**在本地部署单个Spark服务

（2）**Standalone模式**：Spark自带的任务调度模式。（国内常用）

（3）**YARN模式**：Spark使用Hadoop的YARN组件进行资源与任务调度。（国内常用）

（4）**Mesos模式**：Spark使用Mesos平台进行资源与任务的调度。

## 2.1 Spark安装地址

1）官网地址：<http://spark.apache.org/>

2）文档查看地址：<https://spark.apache.org/docs/3.0.0/>

3）下载地址：<https://spark.apache.org/downloads.html>

https://archive.apache.org/dist/spark/

## 2.2 Local模式

Local模式就是运行在一台计算机上的模式，通常就是用于在本机上练手和测试。

### 2.2.1 安装使用

1）上传并解压Spark安装包

[user1@hadoop102 sorfware]$ tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz -C /opt/module/

[user1@hadoop102 module]$ mv spark-3.0.0-bin-hadoop3.2 spark-local

2）官方求PI案例

[user1@hadoop102 spark-local]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

--master local[2] \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

10

可以查看spark-submit所有参数：

[user1@hadoop102 spark-local]$ bin/spark-submit

* --class：表示要执行程序的主类；
* --master local[2]

（1）local: 没有指定线程数，则所有计算都运行在一个线程当中，没有任何并行计算

（2）local[K]:指定使用K个Core来运行计算，比如local[2]就是运行2个Core来执行

20/09/20 09:30:53 INFO TaskSetManager:

20/09/15 10:15:00 INFO Executor: Running task 1.0 in stage 0.0 (TID 1)

20/09/15 10:15:00 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)

（3）local[\*]：默认模式。自动帮你按照CPU最多核来设置线程数。比如CPU有8核，Spark帮你自动设置8个线程计算。

20/09/20 09:30:53 INFO TaskSetManager:

20/09/15 10:15:58 INFO Executor: Running task 1.0 in stage 0.0 (TID 1)

20/09/15 10:15:58 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)

20/09/15 10:15:58 INFO Executor: Running task 2.0 in stage 0.0 (TID 2)

20/09/15 10:15:58 INFO Executor: Running task 4.0 in stage 0.0 (TID 4)

20/09/15 10:15:58 INFO Executor: Running task 3.0 in stage 0.0 (TID 3)

20/09/15 10:15:58 INFO Executor: Running task 5.0 in stage 0.0 (TID 5)

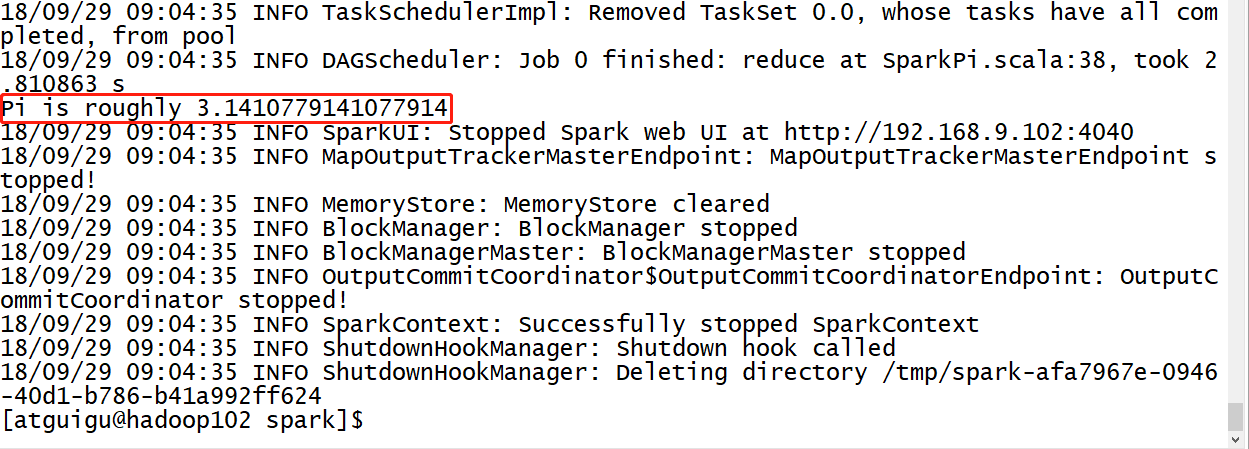
20/09/15 10:15:59 INFO Executor: Running task 7.0 in stage 0.0 (TID 7)

20/09/15 10:15:59 INFO Executor: Running task 6.0 in stage 0.0 (TID 6)

* spark-examples\_2.12-3.0.0.jar：要运行的程序；
* 10：要运行程序的输入参数（计算圆周率π的次数，计算次数越多，准确率越高）；

3）结果展示

该算法是利用蒙特·卡罗算法求PI



### 2.2.2 官方WordCount案例

1）需求：读取多个输入文件，统计每个单词出现的总次数。

2）需求分析：



3）代码实现：

（1）准备文件

[user1@hadoop102 spark-local]$ mkdir input

在input下创建2个文件1.txt和2.txt，并输入以下内容

hello user1

hello spark

（2）启动spark-shell

[user1@hadoop102 spark-local]$ bin/spark-shell

20/07/02 10:17:11 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://hadoop102:4040

Spark context available as 'sc' (master = local[\*], app id = local-1593656236294).

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>

注意：sc是SparkCore程序的入口；spark是SparkSQL程序入口；master = local[\*]表示本地模式运行。

（3）再开启一个hadoop102远程连接窗口，发现了一个SparkSubmit进程

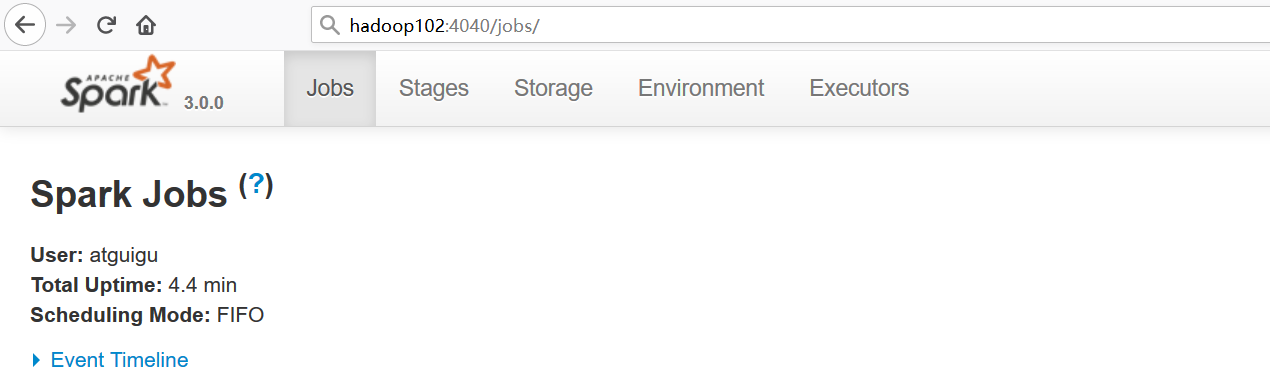
[user1@hadoop102 spark-local]$ jps

3627 SparkSubmit

4047 Jps

运行任务方式说明：spark-submit，是将jar上传到集群，执行Spark任务；spark-shell，相当于命令行工具，本身也是一个Application。

（4）登录hadoop102:4040，查看程序运行情况（注意：spark-shell窗口关闭掉，则hadoop102:4040页面关闭）



说明：本地模式下，默认的调度器为FIFO。

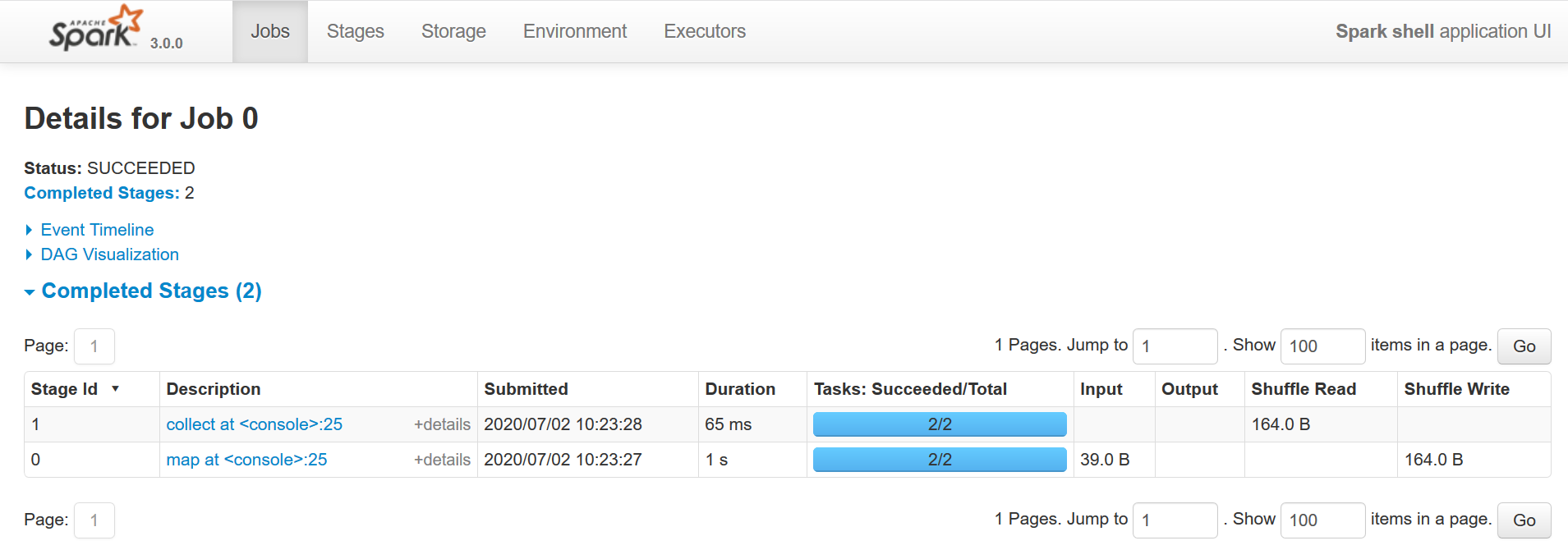
（5）运行WordCount程序

scala>sc.textFile("/opt/module/spark-local/input").flatMap(\_.split(" ")).map((\_,1)).reduceByKey(\_+\_).collect

res0: Array[(String, Int)] = Array((hadoop,6), (oozie,3), (spark,3), (hive,3), (user1,3), (hbase,6))

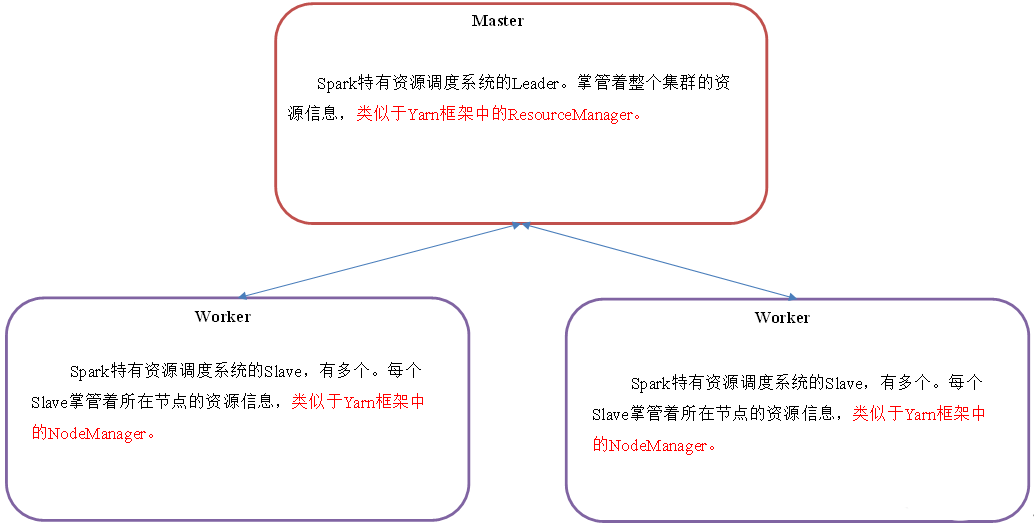
注意：只有collect开始执行时，才会加载数据。

**可登录hadoop102:4040查看程序运行结果**



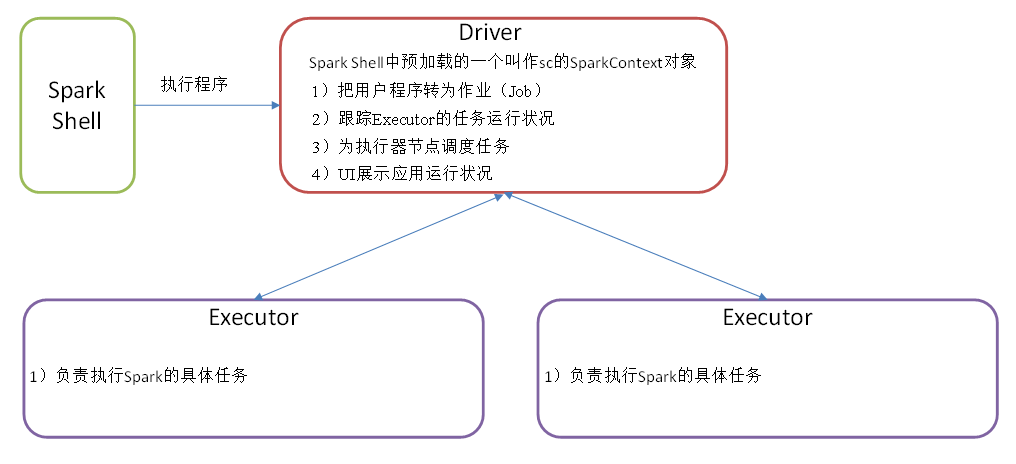
## 2.3 集群角色

### 2.3.1 Master和Worker集群资源管理



Master和Worker是Spark的守护进程、集群资源管理者，即Spark在特定模式下正常运行所必须的进程。

### 2.3.2 Driver和Executor任务的管理者



Driver和Executor是临时程序，当有具体任务提交到Spark集群才会开启的程序。

## 2.4 Standalone模式

Standalone模式是Spark自带的资源调动引擎，构建一个由Master + Slave构成的Spark集群，Spark运行在集群中。

这个要和Hadoop中的Standalone区别开来。这里的Standalone是指只用Spark来搭建一个集群，不需要借助Hadoop的Yarn和Mesos等其他框架。

### 2.4.1 安装使用

1）集群规划

|  |  |  |  |
| --- | --- | --- | --- |
|  | hadoop102 | hadoop103 | hadoop104 |
| Spark | Master  Worker | Worker | Worker |

2）再解压一份Spark安装包，并修改解压后的文件夹名称为spark-standalone

[user1@hadoop102 sorfware]$ tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz -C /opt/module/

[user1@hadoop102 module]$ mv spark-3.0.0-bin-hadoop3.2 spark-standalone

3）进入Spark的配置目录/opt/module/spark-standalone/conf

[user1@hadoop102 spark-standalone]$ cd conf

4）修改slave文件，添加work节点：

[user1@hadoop102 conf]$ mv slaves.template slaves

[user1@hadoop102 conf]$ vim slaves

hadoop102

hadoop103

hadoop104

5）修改spark-env.sh文件，添加master节点

[user1@hadoop102 conf]$ mv spark-env.sh.template spark-env.sh

[user1@hadoop102 conf]$ vim spark-env.sh

SPARK\_MASTER\_HOST=hadoop102

SPARK\_MASTER\_PORT=7077

6）分发spark-standalone包

[user1@hadoop102 module]$ xsync spark-standalone/

7）启动spark集群

[user1@hadoop102 spark-standalone]$ sbin/start-all.sh

查看三台服务器运行进程（xcall.sh是以前数仓项目里面讲的脚本）

[user1@hadoop102 spark-standalone]$ xcall.sh jps

================user1@hadoop102================

3330 Jps

3238 Worker

3163 Master

================user1@hadoop103================

2966 Jps

2908 Worker

================user1@hadoop104================

2978 Worker

3036 Jps

注意：如果遇到 “JAVA\_HOME not set” 异常，可以在sbin目录下的spark-config.sh 文件中加入如下配置：

export JAVA\_HOME=XXXX

8）网页查看：hadoop102:8080（master web的端口，相当于hadoop的8088端口）

目前还看不到任何任务的执行信息。

9）官方求PI案例

[user1@hadoop102 spark-standalone]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

--master spark://hadoop102:7077 \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

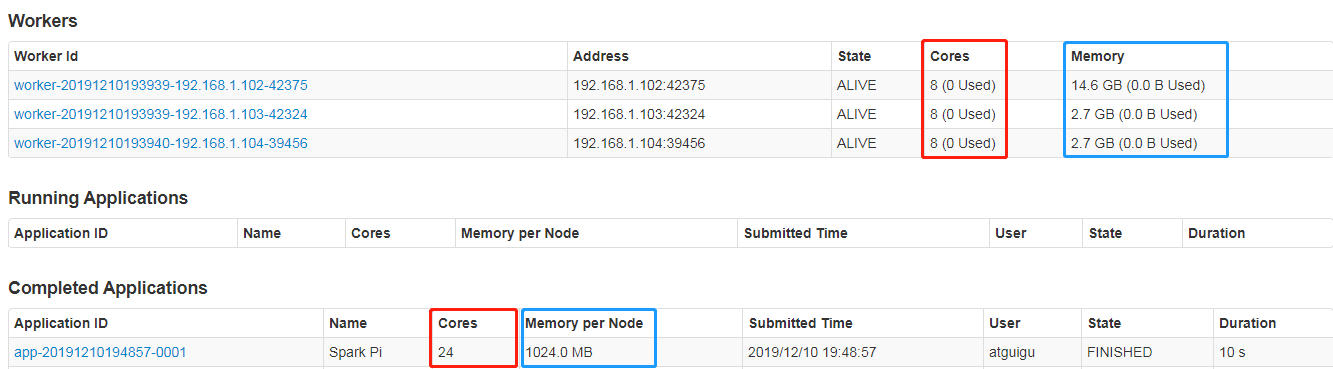
10

参数：--master spark://hadoop102:7077指定要连接的集群的master

10）页面查看[http://hadoop102:8080/，发现执行本次任务，默认采用三台服务器节点的总核数24](http://hadoop102:8080/，发现运行本次任务采用了24)核，每个节点内存1024M。

8080：master的webUI

4040：application的webUI的端口号



### 2.4.2 参数说明

1）配置Executor可用内存为2G，使用CPU核数为2个

[user1@hadoop102 spark-standalone]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

--master spark://hadoop102:7077 \

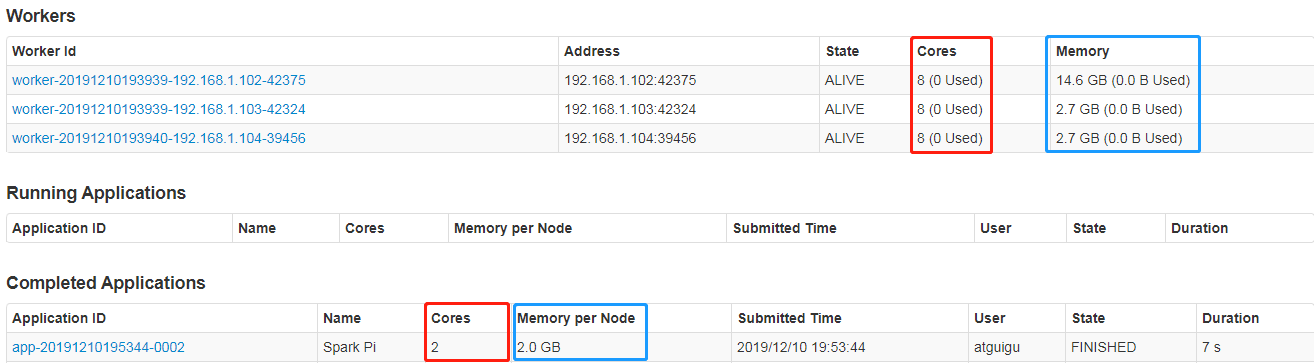
--executor-memory 2G \

--total-executor-cores 2 \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

10

2）页面查看<http://hadoop102:8080/>



3）基本语法

bin/spark-submit \

--class <main-class>

--master <master-url> \

... # other options

<application-jar> \

[application-arguments]

4）参数说明

|  |  |  |
| --- | --- | --- |
| 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 | 本地模式：local[\*]、spark://hadoop102:7077、  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 |
| 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 |

### 2.4.3 配置历史服务

由于spark-shell停止掉后，hadoop102:4040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-default.conf.template名称

[user1@hadoop102 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（写），并分发

[user1@hadoop102 conf]$ vi spark-defaults.conf

spark.eventLog.enabled true

spark.eventLog.dir hdfs://hadoop102:**8020**/directory

[user1@hadoop102 conf]$ xsync spark-defaults.conf

注意：需要启动Hadoop集群，HDFS上的目录需要提前存在。

[user1@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh

[user1@hadoop102 hadoop-3.1.3]$ hadoop fs -mkdir /directory

3）修改spark-env.sh文件，添加如下配置：

[user1@hadoop102 conf]$ vi spark-env.sh

export SPARK\_HISTORY\_OPTS="

-Dspark.history.ui.port=18080

-Dspark.history.fs.logDirectory=hdfs://hadoop102:**8020**/directory

-Dspark.history.retainedApplications=30"

# 参数1含义：WEBUI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（读）

# 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

[user1@hadoop102 conf]$ xsync spark-env.sh

5）启动历史服务

[user1@hadoop102 spark-standalone]$

sbin/start-history-server.sh

6）再次执行任务

[user1@hadoop102 spark-standalone]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

--master spark://hadoop102:7077 \

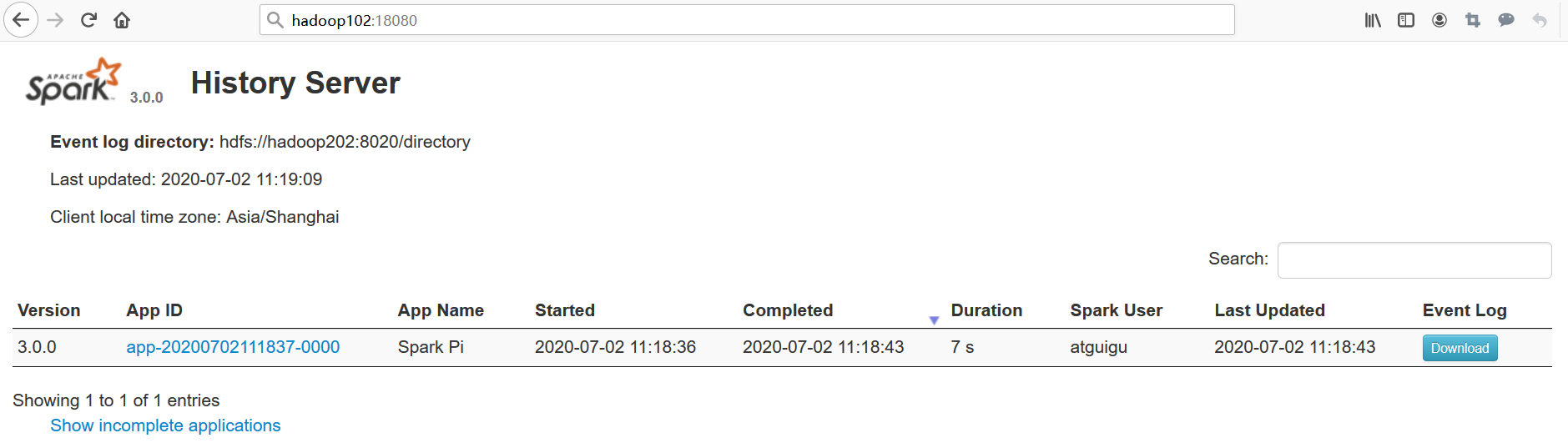
--executor-memory 1G \

--total-executor-cores 2 \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

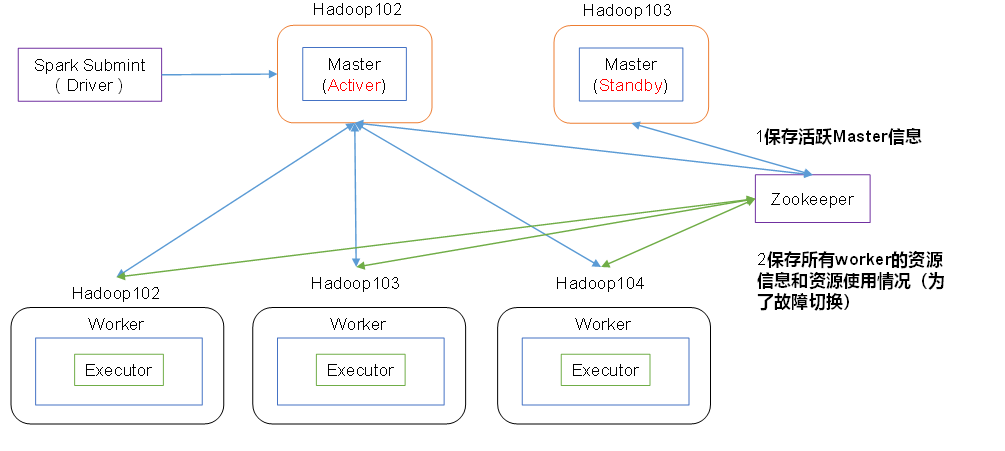
10

7）查看Spark历史服务地址：hadoop102:18080



### 2.4.4 配置高可用（HA）

1）高可用原理



2）配置高可用

（0）停止集群

[user1@hadoop102 spark-standalone]$ sbin/stop-all.sh

（1）Zookeeper正常安装并启动（基于以前讲的数仓项目脚本）

[user1@hadoop102 zookeeper-3.4.10]$ zk.sh start

（2）修改spark-env.sh文件添加如下配置：

[user1@hadoop102 conf]$ vi spark-env.sh

#注释掉如下内容：

#SPARK\_MASTER\_HOST=hadoop102

#SPARK\_MASTER\_PORT=7077

#添加上如下内容。配置由Zookeeper管理Master，在Zookeeper节点中自动创建/spark目录，用于管理：

export SPARK\_DAEMON\_JAVA\_OPTS="

-Dspark.deploy.recoveryMode=ZOOKEEPER

-Dspark.deploy.zookeeper.url=hadoop102,hadoop103,hadoop104

-Dspark.deploy.zookeeper.dir=/spark"

**#添加如下代码**

**#Zookeeper3.5的AdminServer默认端口是8080，和Spark的WebUI冲突**

**export SPARK\_MASTER\_WEBUI\_PORT=8989**

（3）分发配置文件

[user1@hadoop102 conf]$ xsync spark-env.sh

（4）在hadoop102上启动全部节点

[user1@hadoop102 spark-standalone]$ sbin/start-all.sh

（5）在hadoop103上单独启动master节点

[user1@hadoop103 spark-standalone]$ sbin/start-master.sh

（6）在启动一个hadoop102窗口，将/opt/module/spark-local/input数据上传到hadoop集群的/input目录

[user1@hadoop102 spark-standalone]$ hadoop fs -put /opt/module/spark-local/input/ /input

（7）Spark HA集群访问

[user1@hadoop102 spark-standalone]$

bin/spark-shell \

--master spark://hadoop102:7077,hadoop103:7077 \

--executor-memory 2g \

--total-executor-cores 2

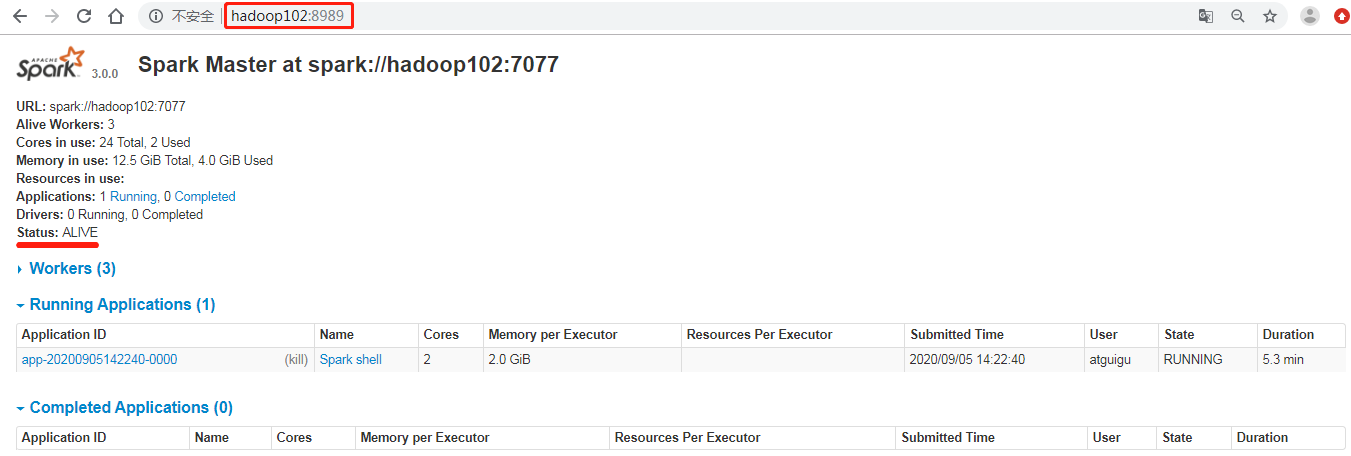
参数：--master spark://hadoop102:7077指定要连接的集群的master

（8）执行WordCount程序

scala>sc.textFile("hdfs://hadoop102:**8020**/input").flatMap(\_.split(" ")).map((\_,1)).reduceByKey(\_+\_).collect

res0: Array[(String, Int)] = Array((hadoop,6), (oozie,3), (spark,3), (hive,3), (user1,3), (hbase,6))

3）高可用测试





（1）查看hadoop102的master进程

[user1@hadoop102 ~]$ jps

5506 Worker

5394 Master

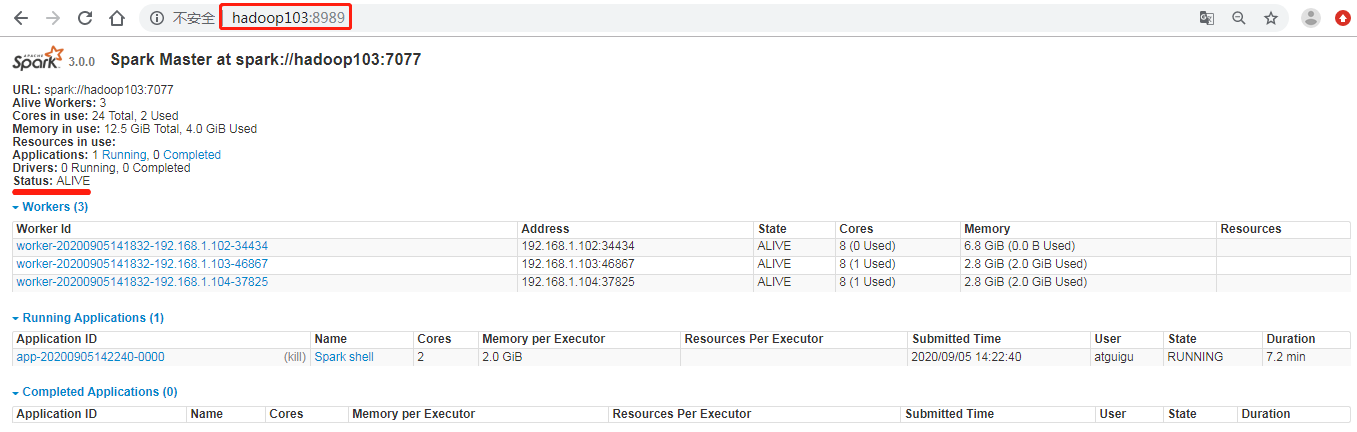
5731 SparkSubmit

4869 QuorumPeerMain

5991 Jps

5831 CoarseGrainedExecutorBackend

（2）Kill掉hadoop102的master进程，页面中观察http://hadoop103:8080/的状态是否切换为active。



[user1@hadoop102 ~]$ kill -9 5394

（3）再启动hadoop102的master进程

[user1@hadoop102 spark-standalone]$ sbin/start-master.sh

### 2.4.5 运行流程

Spark有standalone-client和standalone-cluster两种模式，主要区别在于：Driver程序的运行节点。

1）客户端模式

[user1@hadoop102 spark-standalone]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

--master spark://hadoop102:7077,hadoop103:7077 \

--executor-memory 2G \

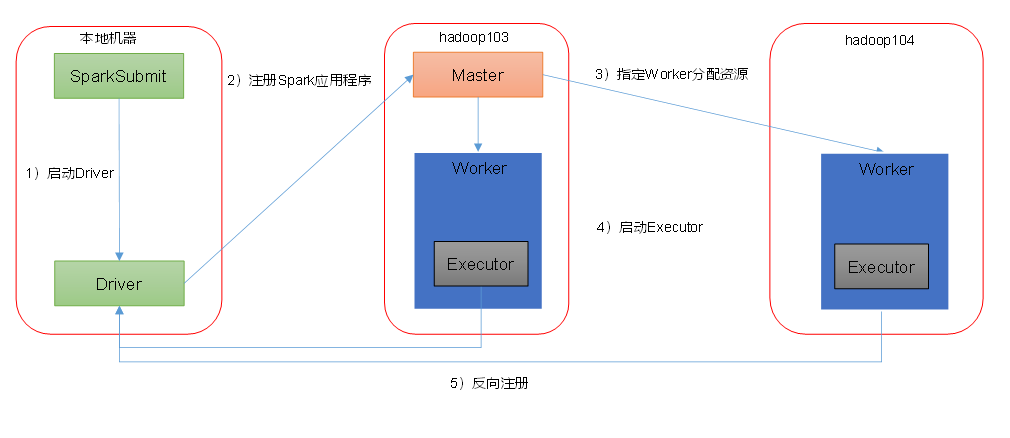
--total-executor-cores 2 \

--deploy-mode client \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

10

--deploy-mode client，表示Driver程序运行在本地客户端



2）集群模式模式

[user1@hadoop102 spark-standalone]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

--master spark://hadoop102:7077,hadoop103:7077 \

--executor-memory 2G \

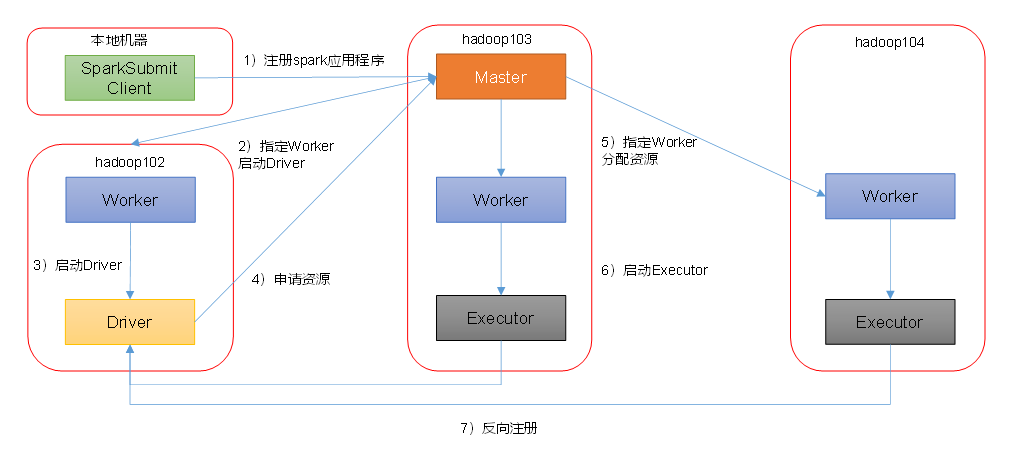
--total-executor-cores 2 \

--deploy-mode cluster \

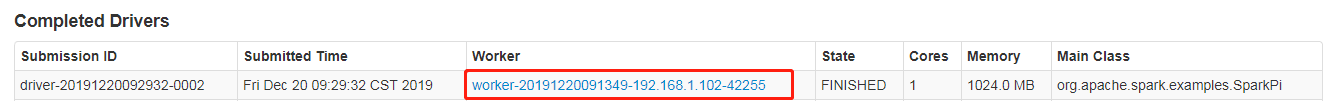
./examples/jars/spark-examples\_2.12-3.0.0.jar \

10

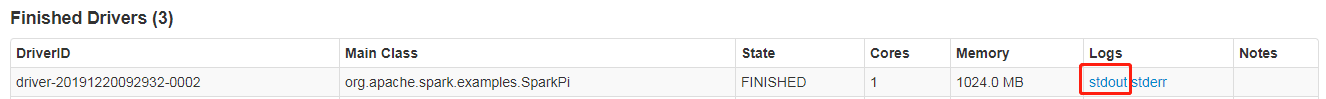
--deploy-mode cluster，表示Driver程序运行在集群



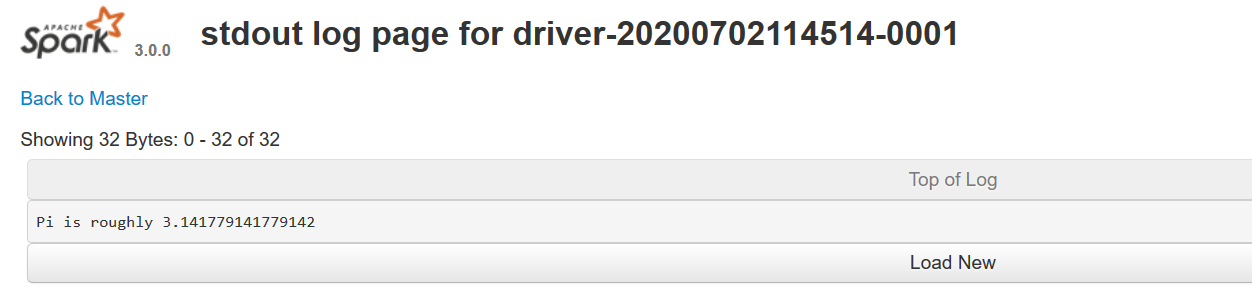
（1）查看<http://hadoop102:8989/>页面，点击Completed Drivers里面的Worker



（2）跳转到Spark Worker页面，点击Finished Drivers中Logs下面的stdout



（3）最终打印结果如下



注意：在测试Standalone模式，cluster运行流程的时候，阿里云用户访问不到Worker，因为Worker是从Master内部跳转的，这是正常的，实际工作中我们不可能通过客户端访问的，这些端口对外都会禁用，需要的时候会通过授权到Master访问Worker

## 2.5 Yarn模式

Spark客户端直接连接Yarn，不需要额外构建Spark集群。

### 2.5.1 安装使用

0）停止Standalone模式下的spark集群

[user1@hadoop102 spark-standalone]$ sbin/stop-all.sh

[user1@hadoop102 spark-standalone]$ zk.sh stop

[user1@hadoop103 spark-standalone]$ sbin/stop-master.sh

1）为了防止和Standalone模式冲突，再单独解压一份spark

[user1@hadoop102 software]$ tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz -C /opt/module/

2）进入到/opt/module目录，修改spark-3.0.0-bin-hadoop3.2名称为spark-yarn

[user1@hadoop102 module]$ mv spark-3.0.0-bin-hadoop3.2/ spark-yarn

3）修改hadoop配置文件/opt/module/hadoop-3.1.3/etc/hadoop/yarn-site.xml，添加如下内容

**因为测试环境虚拟机内存较少，防止执行过程进行被意外杀死，做如下配置**

[user1@hadoop102 hadoop]$ vi yarn-site.xml

<!--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>

4）分发配置文件

[user1@hadoop102 conf]$ xsync /opt/module/hadoop-3.1.3/etc/hadoop/yarn-site.xml

5）修改/opt/module/spark-yarn/conf/spark-env.sh，添加YARN\_CONF\_DIR配置，保证后续运行任务的路径都变成集群路径

[user1@hadoop102 conf]$ mv spark-env.sh.template spark-env.sh

[user1@hadoop102 conf]$ vi spark-env.sh

YARN\_CONF\_DIR=/opt/module/hadoop-3.1.3/etc/hadoop

6）启动HDFS以及YARN集群

[user1@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh

[user1@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh

7）执行一个程序

[user1@hadoop102 spark-yarn]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

--master yarn \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

10

参数：--master yarn，表示Yarn方式运行；--deploy-mod表示客户端方式运行程序

8）如果运行的时候，抛出如下异常ClassNotFoundException:com.sun.jersey.api.client.config.ClientConfig

-原因分析：Spark2中jersey版本是2.22，但是yarn中还需要依赖1.9，版本不兼容

-解决方式：在yarn-site.xml中，添加

<property>

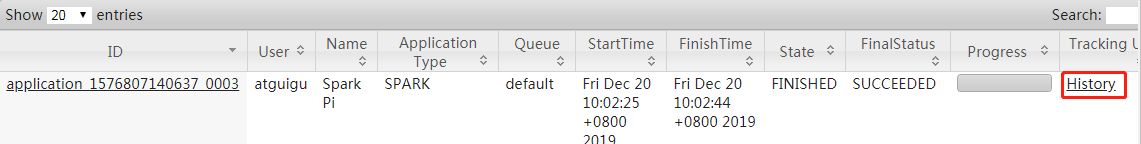
<name>yarn.timeline-service.enabled</name>

<value>false</value>

</property>

9）查看hadoop103:8088页面，点击History，查看历史页面

思考：目前是Hadoop的作业运行日志展示，如果想获取Spark的作业运行日志，怎么办？



### 2.5.2 配置历史服务

由于是重新解压的Spark压缩文件，所以需要针对Yarn模式，再次配置一下历史服务器。

1）修改spark-default.conf.template名称

[user1@hadoop102 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（写）

[user1@hadoop102 conf]$ vi spark-defaults.conf

spark.eventLog.enabled true

spark.eventLog.dir hdfs://hadoop102:**8020**/directory

3）修改spark-env.sh文件，添加如下配置：

[user1@hadoop102 conf]$ vi spark-env.sh

export SPARK\_HISTORY\_OPTS="

-Dspark.history.ui.port=18080

-Dspark.history.fs.logDirectory=hdfs://hadoop102:**8020**/directory

-Dspark.history.retainedApplications=30"

# 参数1含义：WEBUI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。

### 2.5.3 配置查看历史日志

为了从Yarn上关联到Spark历史服务器，需要配置关联路径。

1）修改配置文件/opt/module/spark-yarn/conf/spark-defaults.conf

添加如下内容：

spark.yarn.historyServer.address=hadoop102:18080

spark.history.ui.port=18080

2）重启Spark历史服务

[user1@hadoop102 spark-yarn]$ sbin/stop-history-server.sh

[user1@hadoop102 spark-yarn]$ sbin/start-history-server.sh

3）提交任务到Yarn执行

[user1@hadoop102 spark-yarn]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

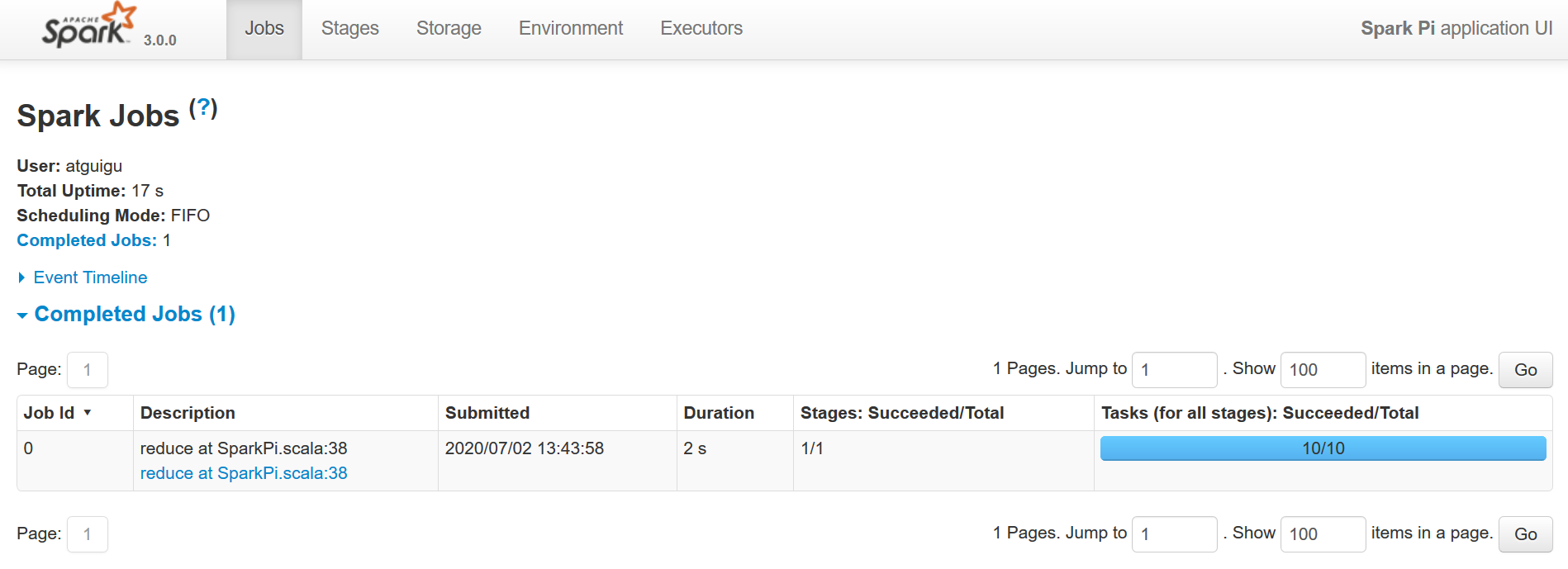
--master yarn \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

10

4）Web页面查看日志：http://hadoop103:8088/cluster

点击“history”跳转到http://hadoop102:18080/



### 2.5.4 运行流程

Spark有yarn-client和yarn-cluster两种模式，主要区别在于：Driver程序的运行节点。

yarn-client：Driver程序运行在客户端，适用于交互、调试，希望立即看到app的输出。

yarn-cluster：Driver程序运行在由ResourceManager启动的APPMaster适用于生产环境。

1）客户端模式（默认）

[user1@hadoop102 spark-yarn]$ bin/spark-submit \

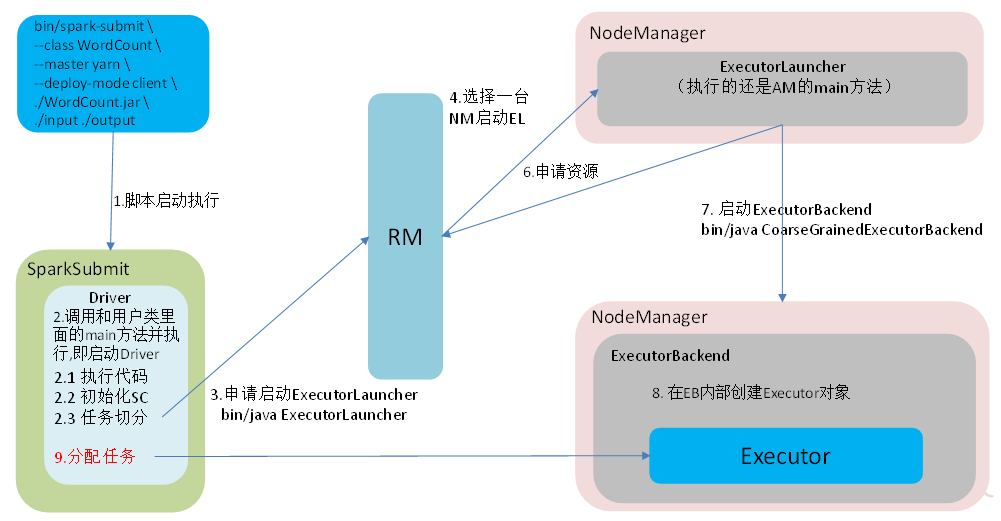
--class org.apache.spark.examples.SparkPi \

--master yarn \

--deploy-mode client \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

10



2）集群模式

[user1@hadoop102 spark-yarn]$ bin/spark-submit \

--class org.apache.spark.examples.SparkPi \

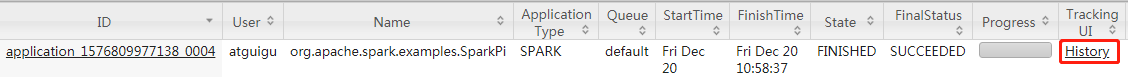
--master yarn \

--deploy-mode cluster \

./examples/jars/spark-examples\_2.12-3.0.0.jar \

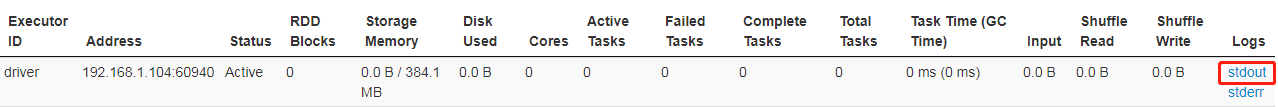
10

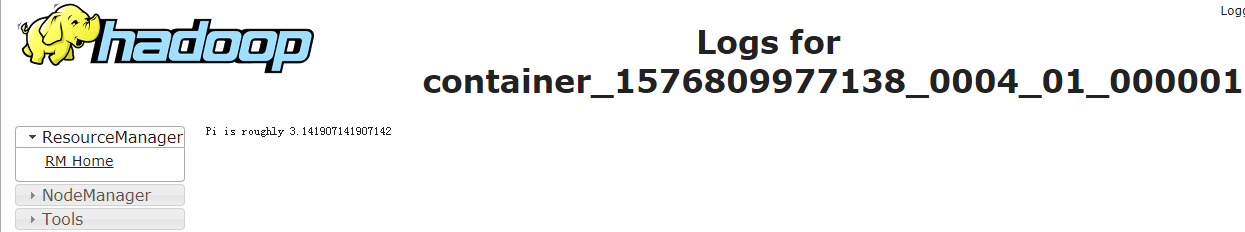
（1）查看<http://hadoop103:8088/cluster>页面，点击History按钮，跳转到历史详情页面



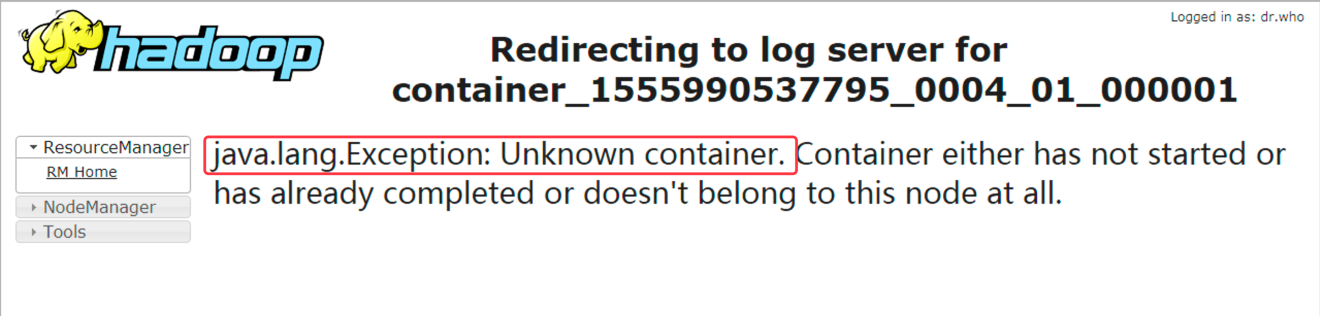
（2）http://hadoop102:18080点击Executors->点击driver中的stdout





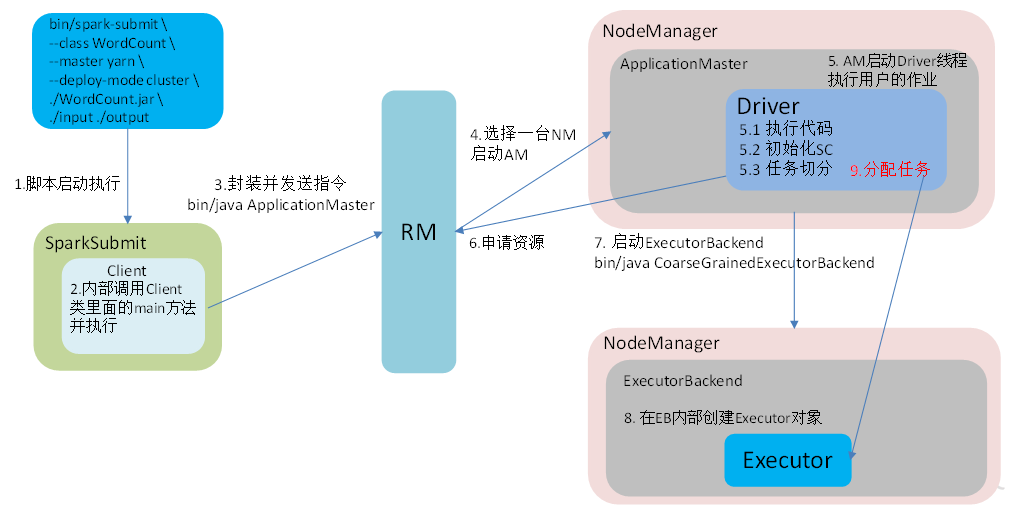


**注意：**如果在yarn日志端无法查看到具体的日志，则在*yarn-site.xml*中添加如下配置并启动Yarn历史服务器



**<property>**  
 **<name>**yarn.log.server.url**</name>**  
 **<value>**http://hadoop102:19888/jobhistory/logs**</value>**  
**</property>**

**注意：hadoop历史服务器也要启动 mr-jobhistory-daemon.sh start historyserver**



## 2.6 Mesos模式

Spark客户端直接连接Mesos；不需要额外构建Spark集群。国内应用比较少，更多的是运用Yarn调度。

## 2.7 几种模式对比

|  |  |  |  |
| --- | --- | --- | --- |
| 模式 | Spark安装机器数 | 需启动的进程 | 所属者 |
| Local | 1 | 无 | Spark |
| Standalone | 3 | Master及Worker | Spark |
| Yarn | 1 | Yarn及HDFS | Hadoop |

## 2.8 端口号总结

1）Spark查看当前Spark-shell运行任务情况端口号：4040

2）Spark Master内部通信服务端口号：7077 （类比于Hadoop的**8020（**9000）端口）

3）Spark Standalone模式Master Web端口号：8080（类比于Hadoop YARN任务运行情况查看端口号：8088）

4）Spark历史服务器端口号：18080 （类比于Hadoop历史服务器端口号：19888）

# 第3章 WordCount案例实操

Spark Shell仅在测试和验证我们的程序时使用的较多，在生产环境中，通常会在IDEA中编制程序，然后打成Jar包，然后提交到集群，最常用的是创建一个Maven项目，利用Maven来管理Jar包的依赖。

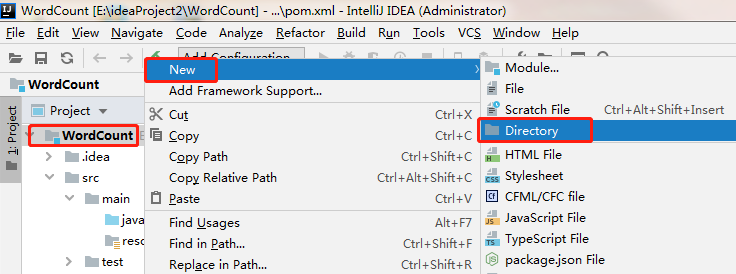
## 3.1 编写程序

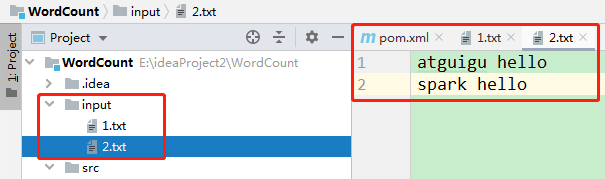
1）创建一个Maven项目WordCount

2）在项目WordCount上点击右键，Add Framework Support=》勾选scala

3）在main下创建scala文件夹，并右键Mark Directory as Sources Root=>在scala下创建包名为com.user1.spark

4）输入文件夹准备：在新建的WordCount项目名称上右键=》新建input文件夹=》在input文件夹上右键=》分别新建1.txt和2.txt。每个文件里面准备一些word单词。





5）导入项目依赖

<dependencies>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.12</artifactId>

<version>3.0.0</version>

</dependency>

</dependencies>

<build>

<finalName>WordCount</finalName>

<plugins>

<plugin>

<groupId>net.alchim31.maven</groupId>

<artifactId>scala-maven-plugin</artifactId>

<version>3.4.6</version>

<executions>

<execution>

<goals>

<goal>compile</goal>

<goal>testCompile</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

注意：如果maven版本为3.2.x，插件下载报错，那么修改插件版本为3.3.2

6）创建伴生对象WordCount，编写代码

package com.user1.spark

import org.apache.spark.rdd.RDD

import org.apache.spark.{SparkConf, SparkContext}

object WordCount {

def main(args: Array[String]): Unit = {

//1.创建SparkConf并设置App名称

val conf = new SparkConf().setAppName("WC").setMaster("local[\*]")

//2.创建SparkContext，该对象是提交Spark App的入口

val sc = new SparkContext(conf)

//3.读取指定位置文件:hello user1 user1

val lineRdd: RDD[String] = sc.textFile("input")

//4.读取的一行一行的数据分解成一个一个的单词（扁平化）(hello)(user1)(user1)

val wordRdd: RDD[String] = lineRdd.flatMap(line => line.split(" "))

//5. 将数据转换结构：(hello,1)(user1,1)(user1,1)

val wordToOneRdd: RDD[(String, Int)] = wordRdd.map(word => (word, 1))

//6.将转换结构后的数据进行聚合处理 user1:1、1 =》1+1 (user1,2)

val wordToSumRdd: RDD[(String, Int)] = wordToOneRdd.reduceByKey((v1, v2) => v1 + v2)

//7.将统计结果采集到控制台打印

val wordToCountArray: Array[(String, Int)] = wordToSumRdd.collect()

wordToCountArray.foreach(println)

//一行搞定

//sc.textFile(args(0)).flatMap(\_.split(" ")).map((\_, 1)).reduceByKey(\_ + \_).saveAsTextFile(args(1))

//8.关闭连接

sc.stop()

}

}

7）打包插件

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-assembly-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<archive>

<manifest>

<mainClass>com.user1.spark.WordCount</mainClass>

</manifest>

</archive>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

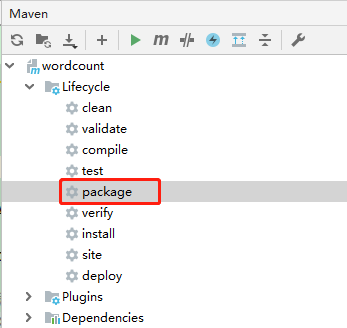
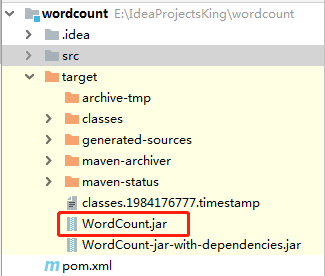
</execution>

</executions>

</plugin>

8）打包到集群测试

（1）点击package打包，然后，查看打完后的jar包

（2）将WordCount.jar上传到/opt/module/spark-yarn目录

（3）在HDFS上创建，存储输入文件的路径/input

[user1@hadoop102 spark-yarn]$ hadoop fs -mkdir /input

（4）上传输入文件到/input路径

[user1@hadoop102 spark-yarn]$ hadoop fs -put /opt/module/spark-local/input/1.txt /input

（5）执行任务

[user1@hadoop102 spark-yarn]$ bin/spark-submit \

--class com.user1.spark.WordCount \

--master yarn \

WordCount.jar \

/input \

/output

注意：input和ouput都是HDFS上的集群路径。

（6）查询运行结果

[user1@hadoop102 spark-yarn]$ hadoop fs -cat /output/\*

9）注意：如果运行发生压缩类没找到

（1）原因

Spark on Yarn会默认使用Hadoop集群配置文件设置编码方式，但是Spark在自己的spark-yarn/jars 包里面没有找到支持lzo压缩的jar包，所以报错。

（2）解决方案一：拷贝lzo的包到/opt/module/spark-yarn/jars目录

[user1@hadoop102 common]$

cp /opt/module/hadoop-3.1.3/share/hadoop/common/hadoop-lzo-0.4.20.jar /opt/module/spark-yarn/jars

（3）解决方案二：在执行命令的时候指定lzo的包位置

[user1@hadoop102 spark-yarn]$

bin/spark-submit \

--class com.user1.spark.WordCount \

--master yarn \

--driver-class-path /opt/module/hadoop-3.1.3/share/hadoop/common/hadoop-lzo-0.4.20.jar \

WordCount.jar \

/input \

/output

## 3.2 本地调试

本地Spark程序调试需要使用Local提交模式，即将本机当做运行环境，Master和Worker都为本机。运行时直接加断点调试即可。如下：

1）代码实现

package com.user1.spark

import org.apache.spark.rdd.RDD

import org.apache.spark.{SparkConf, SparkContext}

object WordCount {

def main(args: Array[String]): Unit = {

//1.创建SparkConf并设置App名称

val conf = new SparkConf().setAppName("WC").setMaster("local[\*]")

//2.创建SparkContext，该对象是提交Spark App的入口

val sc = new SparkContext(conf)

//3.读取指定位置文件:hello user1 user1

val lineRdd: RDD[String] = sc.textFile("input")

//4.读取的一行一行的数据分解成一个一个的单词（扁平化）(hello)(user1)(user1)

val wordRdd: RDD[String] = lineRdd.flatMap(\_.split(" "))

//5. 将数据转换结构：(hello,1)(user1,1)(user1,1)

val wordToOneRdd: RDD[(String, Int)] = wordRdd.map((\_, 1))

//6.将转换结构后的数据进行聚合处理 user1:1、1 =》1+1 (user1,2)

val wordToSumRdd: RDD[(String, Int)] = wordToOneRdd.reduceByKey(\_+\_)

//7.将统计结果采集到控制台打印

wordToSumRdd.collect().foreach(println)

//8.关闭连接

sc.stop()

}

}