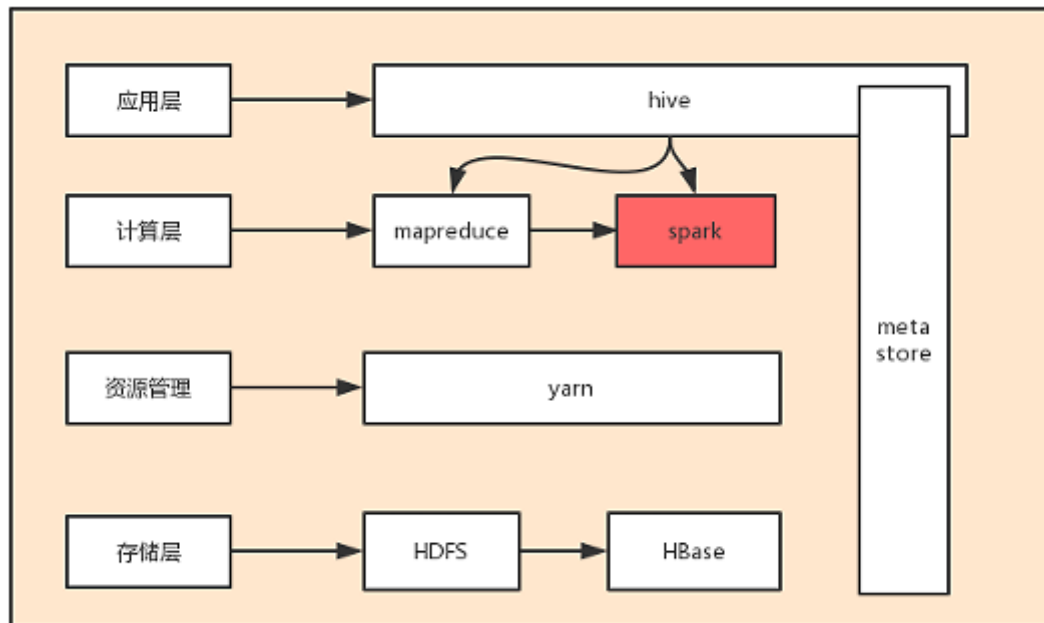


# spark简介

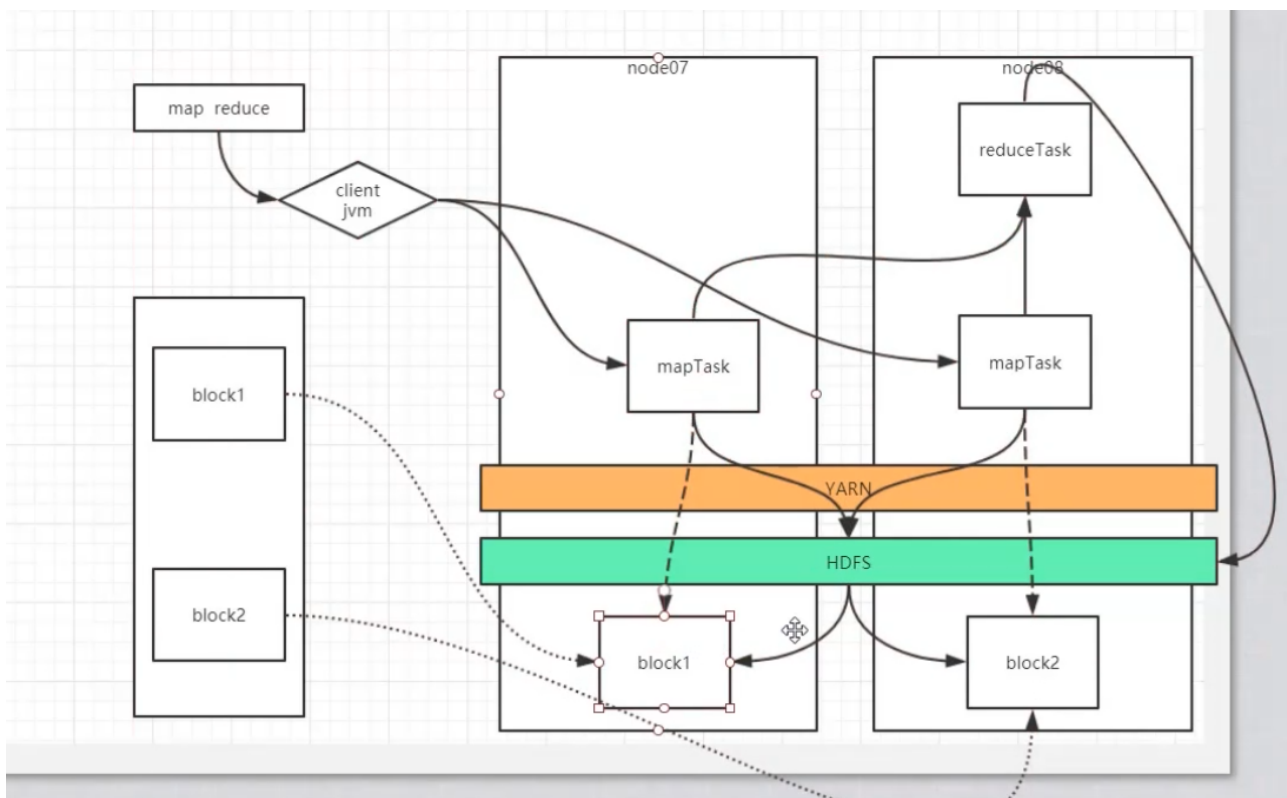
---

spark位于大数据生态中的位置



## 一、结合hadoop对比理解spark

---



1. 应用 application: client jvm
2. 步骤 stage: map-stage, reduce-stage  
每个stage里会有一些并行的任务mapTask, reduceTask
3. job

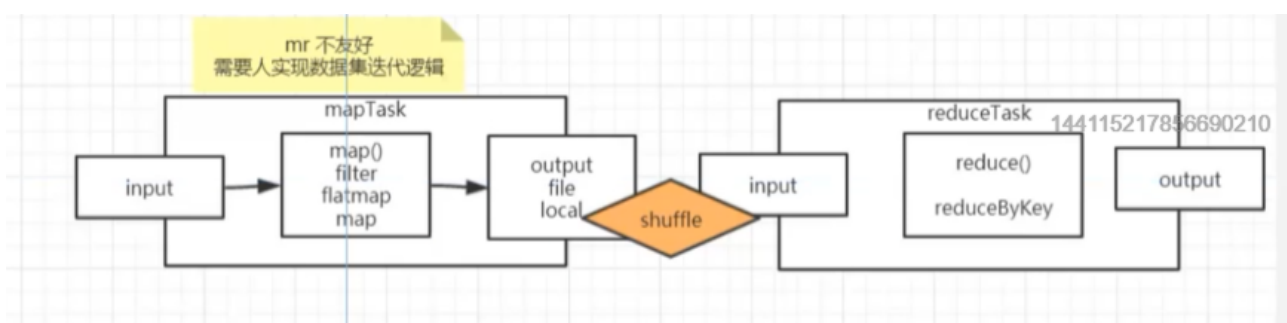
比例: 1个app: 1个job

1个job: 1-2 stage

1个stage: N个task (map、reduce)

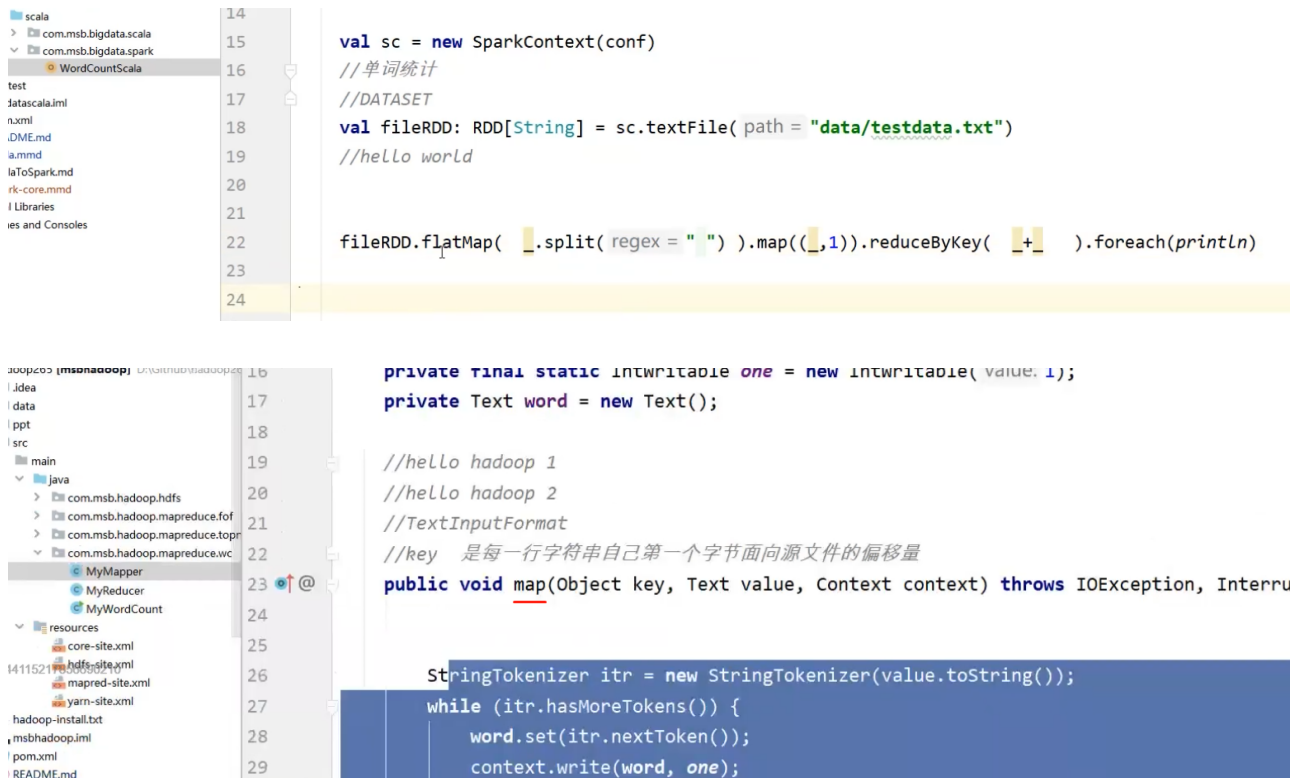
注意: 多个job可以组成作业链。

## MapTask 执行步骤:



缺点:

1. 冷启动: 复杂业务需要多次动态启动/关闭MR的jvm。
2. maptask 输出需要写磁盘, 而且不能多被复用。业务复杂可能多次执行。
3. 需要自己实现 那些迭代逻辑, 而spark自带了很多函数如 flatMap等。



```
14
15
16 val sc = new SparkContext(conf)
17 // 单词统计
18 // DATASET
19 val fileRDD: RDD[String] = sc.textFile( path = "data/testdata.txt")
20 // hello world
21
22 fileRDD.flatMap( _.split( regex = " ") ).map((_,1)).reduceByKey( _+_ ).foreach(println)
23
24
```

```
16 private final static IntWritable one = new IntWritable( value: 1);
17 private Text word = new Text();
18
19 //hello hadoop 1
20 //hello hadoop 2
21 //TextInputFormat
22 //key 是每一行字符串自己第一个字节面向源文件的偏移量
23 public void map(Object key, Text value, Context context) throws IOException, Interru
24
25
26 StringTokenizer itr = new StringTokenizer(value.toString());
27 while (itr.hasMoreTokens()) {
28     word.set(itr.nextToken());
29     context.write(word, one);
30 }
```

期望:

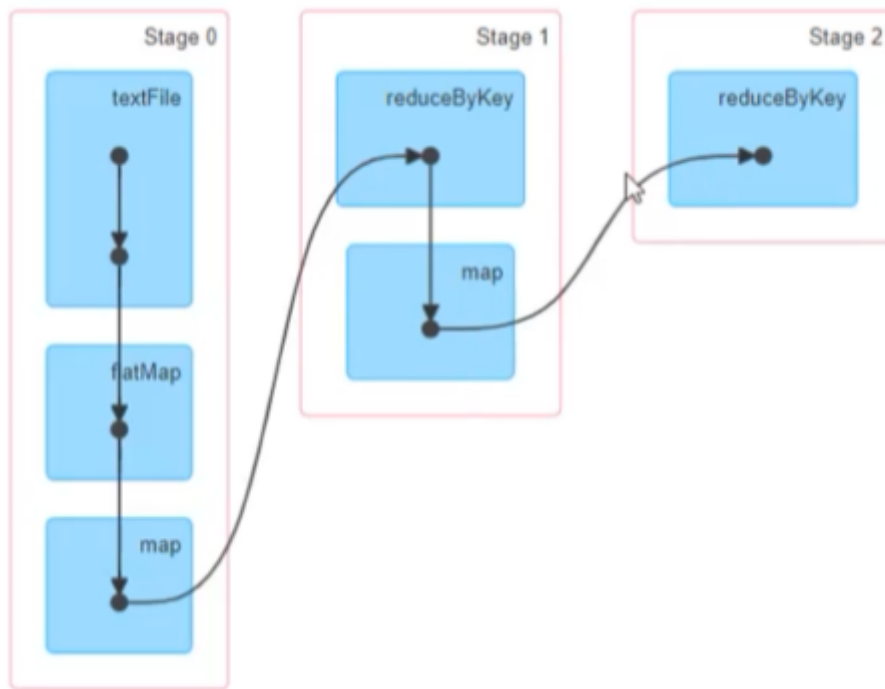
1. 1个 app -> N 个 Job ,复用资源
2. 1个job -> N个stage : 避免多次启动jvm
3. 1个stage -> N个Task (与hadoop一样)

一个stage的1个Task: 可以在一台机器完成的计算。

reduceByKey 需要从多个机器拿取数据聚合计算, 所以会新开一个stage。

所以stage的边界 是 shuffle。

## ▼ DAG Visualization



复用资源job0的资源如下：



## Spark Jobs (?)

User: root  
Total Uptime: 36 s  
Scheduling Mode: FIFO  
Completed Jobs: 2

▶ Event Timeline

### Completed Jobs (2)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1	foreach at WordCountScala.scala:42 foreach at WordCountScala.scala:42	2019/10/10 20:56:26	60 ms	2/2 (1 skipped)	2/2 (1 skipped)
0	foreach at WordCountScala.scala:41 foreach at WordCountScala.scala:41	2019/10/10 20:56:26	0.3 s	2/2	2/2

↶

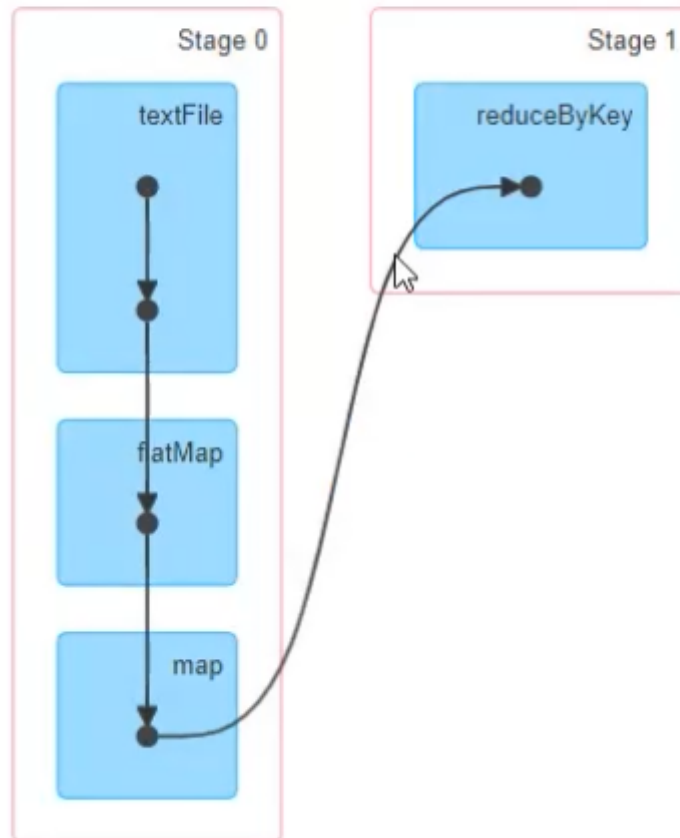
# Details for Job 0

**Status:** SUCCEEDED

**Completed Stages:** 2

► [Event Timeline](#)

▼ [DAG Visualization](#)



Completed Stages: 2

# Details for Job 1

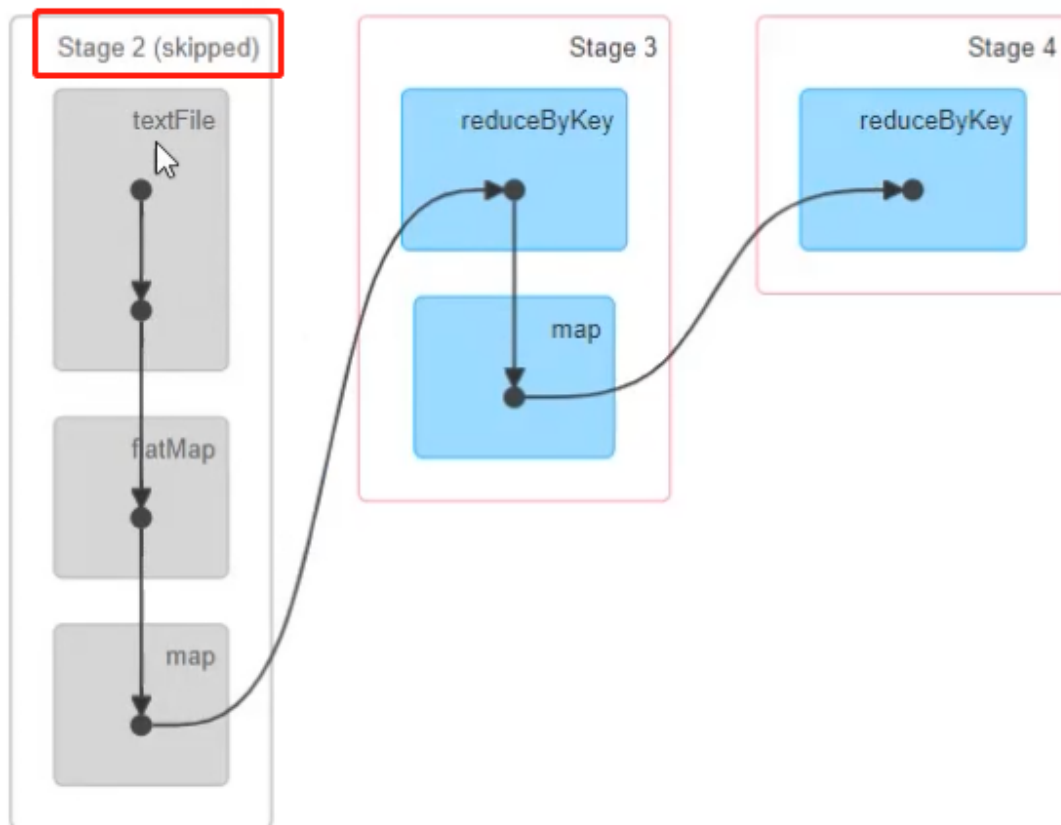
Status: SUCCEEDED

Completed Stages: 2

Skipped Stages: 1

► Event Timeline

▼ DAG Visualization



## 二、spark集群总结

spark 可以依托于hadoop生态如yarn, hdfs, 也可以独立

spark 有自己的资源层(master/worker), 也可以有自己的临时的dfs。

spark具备启动计算程序后, 为自己维护一个临时的分布式存储系统的能力, 程序执行完后, 临时分布式存储系统就销毁了。

spark onYarn/standalone 集群中部署模式分两种:

根据Driver(SparkContext)运行在client里还是集群里来区分

1. 客户端模式（clientJvm）
2. 集群模式（跟随ApplicationMaster进程一起）

ps: 客户端模式优点：分布式计算 结果汇总到driver里后，客户端模式可以直观的看到结果，如果是集群模式则看不到。

资源申请：

hadoop:

1. 每到一步时再，动态申请资源。
2. 任务是jvm级别。

因为hadoop计算逻辑是已知的。

而spark 程序的计算逻辑是未知的，job，stage数量不一,必须运行时确定(DAG:任务切割，每一个job都会进行DAG任务切割)。

所以为了避免运行时资源申请失败，spark 采用预先抢占资源。

spark集群一启动应用就抢占每个节点的cpu、内存（Executor），此时rdd那些都还没创建/执行，未来任务是以线程级别在每个节点执行。

spark driver 和 excutor最好再一个局域网，提交通讯效率。

## 三、单机搭建

---

### (一)本地开发环境搭建

win+ idea 需要按如下步骤部署，否则idea 运行Hadoop，spark会报错如下：

*spark Failed to locate the winutils binary in the hadoop binary path*

## 1. 在win的系统中部署我们的hadoop:

D:\codes\hadoop-2.6.5\

2. hadoop-install\soft\bin 文件覆盖到D:\codes\hadoop-2.6.5\bin目录下  
还要将hadoop.dll 复制到 c:\windwos\system32\

3. 设置环境变量: HADOOP\_HOME C:\usr\hadoop-2.6.5\hadoop-2.6.5

## (二)单机搭建

此时 spark使用自己的资源层master/worker，不需要其他的资源层如yarn。

#上传spark 到node1解压

```
tar xf spark-2.3.4-bin-hadoop2.6.tgz
```

执行cli进行spark交互式编程学习

ps:

1. 可以用tab补全

2. Spark context available as 'sc' (master = local[\*], //单进程多线程运行

```
cd spark-2.3.4-bin-hadoop2.6/bin/  
./spark-shell
```

尝试用spark分析本机文件 /tmp/data.txt:

```
[root@node1 tmp]# cat data.txt  
hello hadoop  
hello spark  
hello bigdata
```

```
scala> sc.textFile("/tmp/data.txt").flatMap(_.split("  
"))).map((_,1)).reduceByKey((_+_)).foreach(println)
```

输出:



```
(hello,3)
(bigdata,1)
(spark,1)
(hadoop,1)
```

ps: 在spark-shell终端进行scala编程，也会先编译字节码然后执行。

## 三、standalone集群搭建

---

<https://spark.apache.org/docs/latest/spark-standalone.html#installing-spark-standalone-to-a-cluster>

基于已有的hadoop hdfs环境搭建 spark-standalone 集群

### 前置条件

```
#1. hadoop基础设施:
jdk: 1.8.xxx
节点的配置
部署hadoop: hdfs  zookeeper
#2. 免密
node1 ssh 免密到node1,2,3,4
```

### 环境变量

```
#node1,2,3,4
vi /etc/profile
export SPARK_HOME=/opt/bigdata/spark-2.3.4-bin-hadoop2.6
source /etc/profile
```

### 修改配置

```
#1. node1上操作，添加从节点信息（node1主，其他从）
cp slaves.template slaves
vi slaves
node2
```

```
node3
node4

#2. node1 添加hadoop dfs有关的配置信息
cd conf/
mv spark-env.sh.template spark-env.sh
vi spark-env.sh
export SPARK_MASTER_HOST=node1
export SPARK_MASTER_PORT=7077
export SPARK_MASTER_WEBUI_PORT=8080
export SPARK_WORKER_CORES=4
export SPARK_WORKER_MEMORY=2g
```

```
#3. 分发到其他node2, 3, 4
scp -r spark-2.3.4-bin-hadoop2.6/ node2: `pwd`
scp -r spark-2.3.4-bin-hadoop2.6/ node3: `pwd`
scp -r spark-2.3.4-bin-hadoop2.6/ node4: `pwd`
```

## 启动Hadoop集群

```
#node2,3,4
zkServer.sh start
#检查zk集群状态
zkServer.sh status

#node1
start-dfs.sh
```

验证：访问**Hdfs web UI**控制台,主/备 node1/node2:

```
#node1
http://192.168.56.103:50070/dfshealth.html#tab-overview
#node2
http://192.168.56.104:50070/dfshealth.html#tab-overview
```

## 启动spark集群

#node执行

```
[root@node1 bigdata]# cd spark-2.3.4-bin-hadoop2.6/sbin/
```

```
[root@node1 sbin]# ./start-all.sh
```

#此时在node1有master进程，node2，3，4各有一个worker进程

访问node1的8080webUI验证: <http://192.168.56.103:8080/>



### Spark Master at spark://node1:7077

URL: spark://node1:7077

REST URL: spark://node1:6066 (cluster mode)

Alive Workers: 3

Cores in use: 12 Total, 0 Used

Memory in use: 6.0 GB Total, 0.0 B Used

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

#### Workers (3)

Worker Id	Address
<a href="#">worker-20230122235350-192.168.130.4-39086</a>	192.168.130.4:39086
<a href="#">worker-20230122235351-192.168.130.6-42213</a>	192.168.130.6:42213
<a href="#">worker-20230122235351-192.168.130.7-42548</a>	192.168.130.7:42548

#### Running Applications (0)

Application ID	Name	Cores	Memory per Executor
----------------	------	-------	---------------------

#### Completed Applications (0)

Application ID	Name	Cores	Memory per Executor
----------------	------	-------	---------------------

## hdfs准备

```
[root@node1 sbin]# cd /tmp/
```

```
[root@node1 tmp]# hdfs dfs -mkdir /sparktest
```

```
[root@node1 tmp]# hdfs dfs -put data.txt /sparktest
```

## cli连接集群

```
#这个master信息在 sparkwebUI 上有。  
#这样在cli里就不是本地单机执行了(local)  
./spark-shell --master spark://node1:7077
```

此时webUI能看到cli这个应用如下：

← → ↺

⚠ 不安全 | 192.168.56.103:8080

译 翻译

翻 翻译

后端学习

开发资源

易联众易惠

业余


云原生

python

大数据

web前端

网关

 2.3.4

# Spark Master at spark://node1:7077

**URL:** spark://node1:7077  
**REST URL:** spark://node1:6066 (cluster mode)  
**Alive Workers:** 3  
**Cores in use:** 12 Total, 12 Used  
**Memory in use:** 6.0 GB Total, 3.0 GB Used  
**Applications:** 1 Running, 0 Completed  
**Drivers:** 0 Running, 0 Completed  
**Status:** ALIVE

### Workers (3)

Worker Id	Address
<a href="#">worker-20230122235350-192.168.130.4-39086</a>	192.168.130.4:39086
<a href="#">worker-20230122235351-192.168.130.6-42213</a>	192.168.130.6:42213
<a href="#">worker-20230122235351-192.168.130.7-42548</a>	192.168.130.7:42548

### Running Applications (1)

Application ID	Name	Cores	Memory per Executor
<a href="#">app-20230123001152-0000</a>	(kill) <a href="#">Spark shell</a>	12	1024.0 MB

### Completed Applications (0)

Application ID	Name	Cores	Memory per Executor
----------------	------	-------	---------------------

点进去能看到每个worker默认都尽量榨干cpu，内存。

测试分析hdfs里的文件

#hdfs搭建了HA，所以要写集群id: mycluster，不能写具体的node1/node2

scala>

```
sc.textFile("hdfs://mycluster/sparktest/data.txt").flatMap(_.split(
" ")).map(_._1).reduceByKey(_+_).foreach(println)
```

#发现执行后

没结果输出，因为结果输出在集群里，没在cli。

#加一个

collect()算子，把结果回收到cli进程，即可显示结果

#而且速度很快，因为刚才的计算结果缓存过了

scala>

```
sc.textFile("hdfs://mycluster/sparktest/data.txt").flatMap(_.split(
" ")).map(_._1).reduceByKey(_+_).collect.foreach(println)
```

(hello,3)

(bigdata,1)

(spark,1)

(hadoop,1)