# 整体的课程体系

spark-core

最核心，最基础的阶段

spark集群安装

任务的提交

RDD 抽象概念 弹性分布式数据集

方法

spark任务提交的机制

spark中的高级特性 ：宽窄依赖，缓存，持久化，checkpoint,广播变量,累加器

spark on yarn

spark HA

spark-sql

mysql hive

处理结构化数据的模块

spark-streaming

实时数据处理

# spark简介

## spark简介

官网： <http://spark.apache.org/>

中文官网： <http://spark.apachecn.org/docs/cn/2.2.0/>



下定义：

spark 是基于内存的，分布式的大数据计算引擎。（处理框架）

优先使用内存，还会使用磁盘。 内存+ 磁盘

hdfs mapreduce hbase

09年产生，10年开源，2014.2 正式成员apache的顶级项目。 可以应用于企业级应用。

## spark和hadoop-mapreduce的区别

hadoop: hdfs mapreduce yarn

spark 和 mapreduce :

1,spark优先使用内存，mapreduce中间结果落地到磁盘中,spark DAG 提升运行效率。

2， spark的容错性更高，RDD

3， mapreduce 只有map reduce 两个算子，spark有非常丰富的算子（方法）

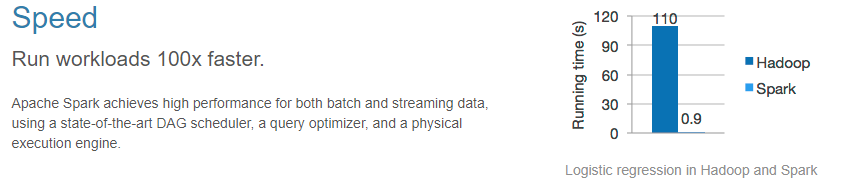
4，spark 通用，spark支持4种语言

spark 仅仅是mapreduce的替代方案，对mapreduce进行优化，以弥补mapreduce的不足。

## spark的优点：

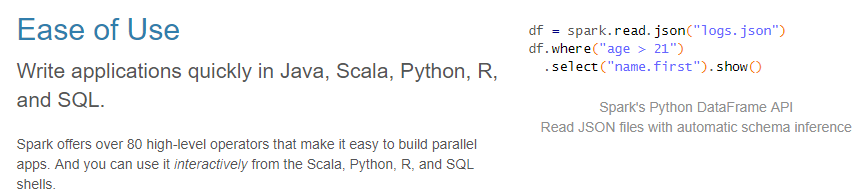
### 速度快

基于内存 快100倍，基于磁盘 ，10倍以上



### 易用性

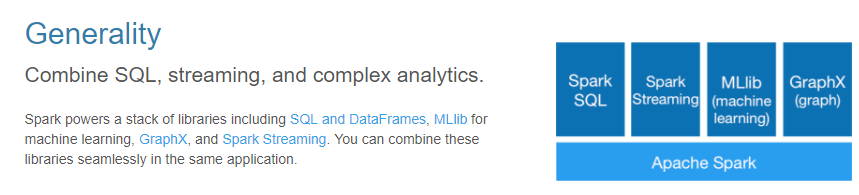
spark scala python java R



### 通用性

一站式解决方案：

离线数据处理，结构化的数据处理，实时的数据处理

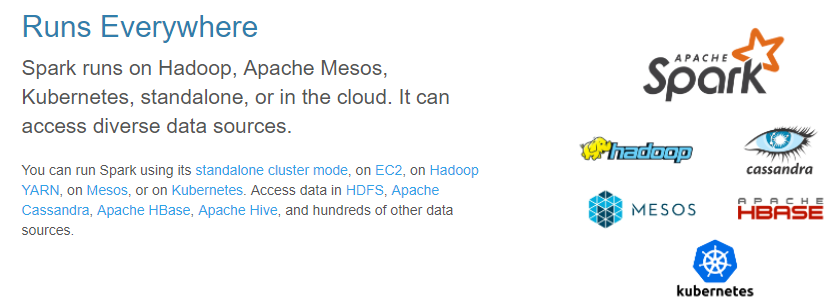


### 兼容性

spark 可以融合hadoop生态圈中的各种框架

hdfs:

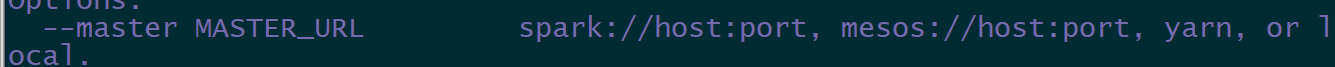
运行spark程序：多种运行方式



# spark部署模式

spark有4种部署模式：

都是通过master 参数来指定的。



## local

本地模式 开箱即用

不指定master

或者

--master local local[N] local[\*]

local: 只是用一个cores

local[N] : 使用N个cores

local[\*] : 使用当前机器的所有的可以用的cores

## standalone

spark本身提供的集群模式

--master spark://host:port

## yarn

统一的资源调度平台

--master yarn

## mesos

类似于yarn，资源调度平台

--master mesos://host:port

# 搭建spark的集群

特指 standalone集群。

框架使用的套路：

下载

上传

解压

配置文件

启动

使用

## 集群机器的准备工作：

类似于hdfs

准备好4台机器：

hdp-01：192.168.8.11

hdp-02：192.168.8.12

hdp-03: 192.168.8.13

hdp-04: 192.168.8.14

NAT: 网关： 192.168.8.2 windows本地的ip地址： 192.168.8.1

Master: hdp-01

Worker: hdp-02 hdp-03 hdp-04

master 和worker可以部署在同一台机器中。

## 集群的环境准备：

### jdk必须安装

java -version 1.8+

部署spark的时候，是否需要安装scala呢？**不需要**

### 防火墙

大数据集群，基本上都是内网环境。 会有一到两台机器，部署有公网环境。

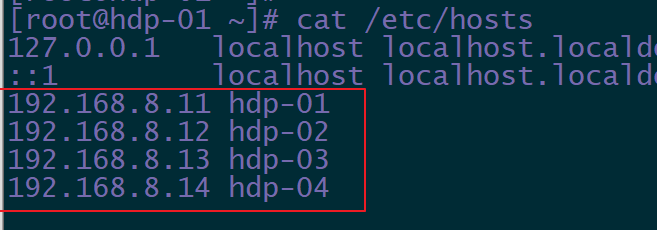
|  |
| --- |
| service iptables status  service iptables stop  chkconfig iptables off |

免密登录

Master节点到worker节点的免密登录。

|  |
| --- |
| ssh-keygen  ssh-copy-id 节点 |

映射关系



至少2台节点。

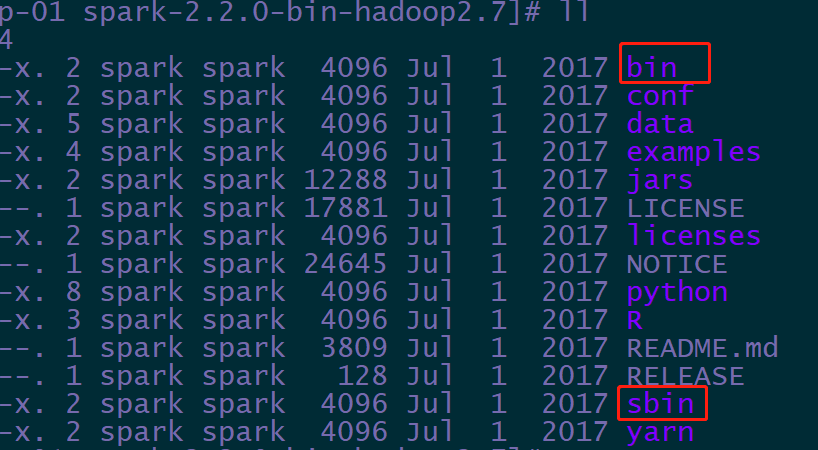
## 开始部署集群



上传并解压：

tar -zxvf spark-2.2.0-bin-hadoop2.7.tgz -C apps/

spark安装包的目录结构：



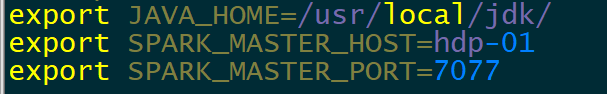
bin: spark的命令

sbin：集群启停的命令

conf: 配置文件

## 修改配置文件：

### spark-env.sh

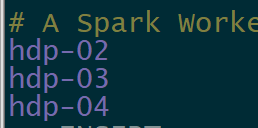


: r! echo “/usr/local/jdk/”

### slaves

批量启动脚本使用。

去掉localhost



### 把安装包分发到其他的节点中：

for i in 2 3 4;do scp -r spark-2.2.0-bin-hadoop2.7/ hdp-0$i:$PWD ;done

## 集群启停操作

### 单独启停

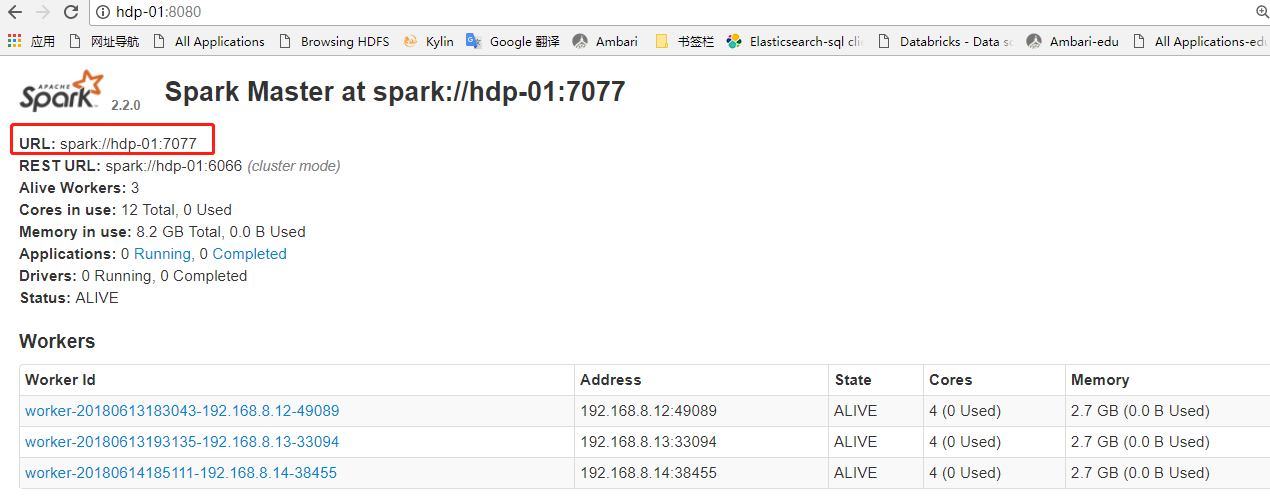
单独启动master:

[root@hdp-01 sbin]# ./start-master.sh

启动所有的workers:

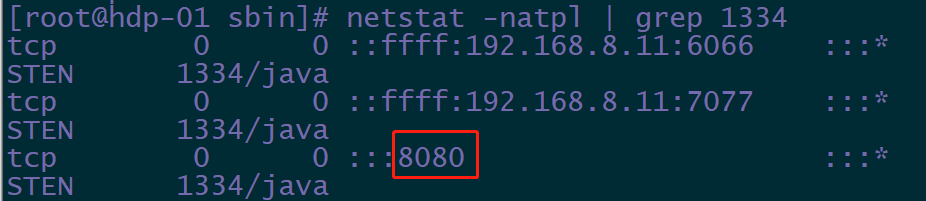
[root@hdp-01 sbin]# ./start-slaves.sh

启动之后的web监控界面：



netstat -antpl | grep master的进程

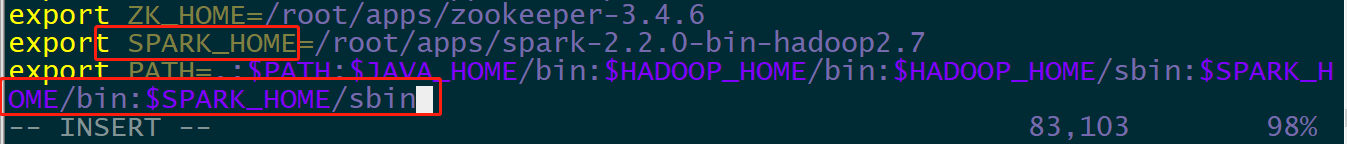
8080是默认的web端口，8081 。。。。



### 直接一键启停：

start-all.sh stop-all.sh

### 配置环境变量：



分发到其他的节点中：

for i in 2 3 4 ;do scp /etc/profile hdp-0$i:/etc/ ;done

重新的再每一台节点汇总，source /etc/profile

### 如何解决命令冲突的问题？

1. 修改启动脚本的名称。

hadoop中的start-all.sh -🡪 start-all-hdp.sh

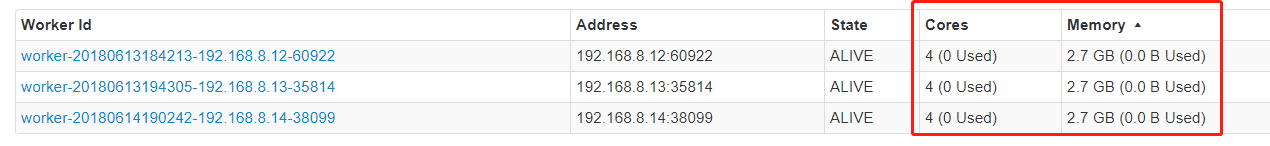
stop-all.sh -🡪 stop-all-hdp.sh

2，直接把其他一个框架的sbin目录的配置，删了，再启动时，就用绝对路径。

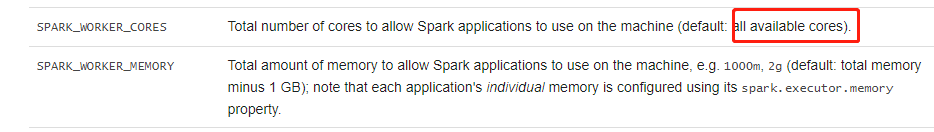
start-all.sh stop-all.sh

## 查看spark的监控界面：

## worker的默认资源分配

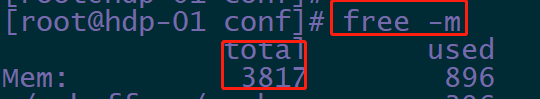
worker的cores和memory的官方配置：

<http://spark.apache.org/docs/latest/spark-standalone.html>



cores: 使用所有的。

memory: 当前机器中可以的内存 - 1g



# 如何提交spark的任务

## 提交任务的两个命令

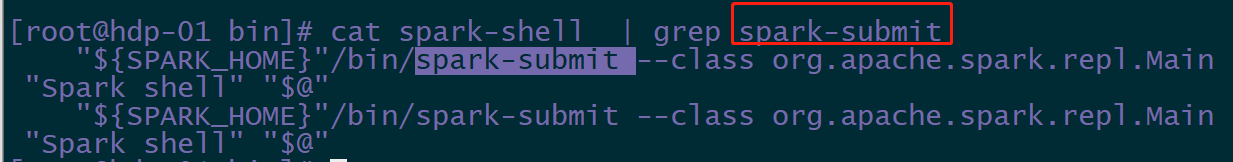
spark-submit 程序执行之后，application就会退出。

spark-shell 会一直占有一个application，手动退出。 ctrl + c

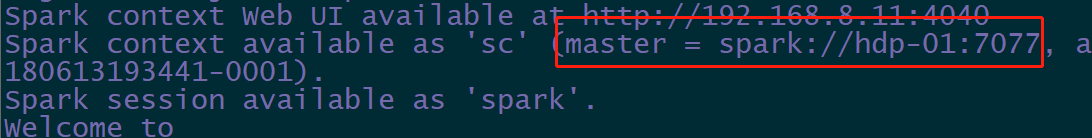
### spark-shell

是一个交互式的命令行,主要用于测试。

spark-shell脚本，实际上调用的是spark-submit脚本：

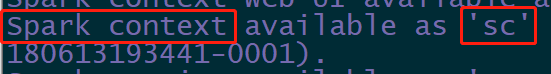


spark-shell --master spark://hdp-01:7077



在spark-shell中，已经为我们初始化好了一个SparkContext，名称是sc。

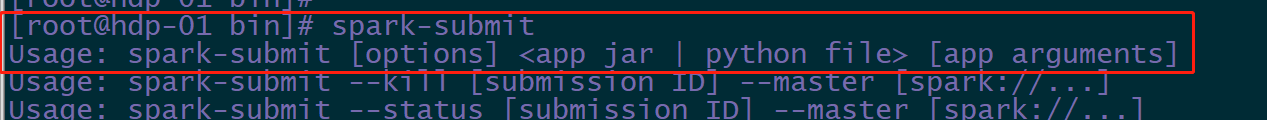
在写spark程序之前，必须创建SparkContext实例对象。



### spark-submit

最常用于提交spark任务的命令。 需要有jar包。 工作中常用。

sparkSubmit的使用：



spark-submit 选项 jar包 参数列表

spark-submit –master xx –class 程序运行的主类 xxx.jar input output

当我们提交spark-submit或者spark-shell时，就会生成一个Application

## spark的第一个示例程序：

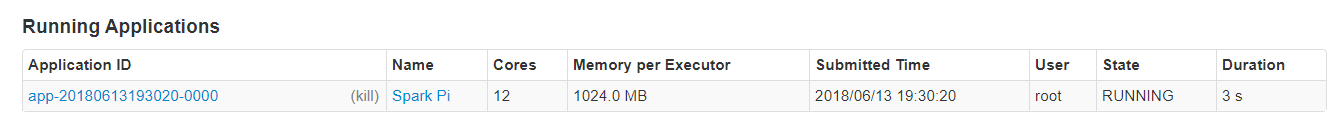
求圆周率： SparkPi

local模式运行程序，不能再spark 监控界面中进行查看。

|  |
| --- |
| spark-submit --class org.apache.spark.examples.SparkPi /root/apps/spark-2.2.0-bin-hadoop2.7/examples/jars/spark-examples\_2.11-2.2.0.jar 100 |

指定master，任务提交到standalone集群运行：

|  |
| --- |
| spark-submit **--master spark://hdp-01:7077** --class org.apache.spark.examples.SparkPi /root/apps/spark-2.2.0-bin-hadoop2.7/examples/jars/spark-examples\_2.11-2.2.0.jar 1000 |



# Spark编程（wordcount）

scala, java , java lambda

jdk1.8的新特性 ，lambda 实现了函数式编程

## 利用spark-shell来编程

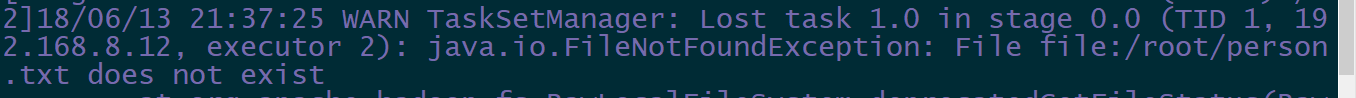
# spark-shell --master spark://hdp-01:7077

已经初始化好了SparkContext sc

回顾wordcount的思路：

读数据，切分并压平，组装，分组聚合， 排序

当我们使用spark-shell以集群模式读取本地的数据的时候，报错：文件不存在：



如果以local模式运行，没有问题。

把文件发送到所有的worker节点中去。

读取分布式的文件：

spark中的RDD上的方法都称之为算子。

分为两类：

转换 transformation lazy执行。 当遇到action算子的时候，才开始真正的运行。

转换类的算子，都会生成新的rdd。

行动类的算子 action

### [parallelize函数和makeRDD函数的区别](http://xiaotutu365.iteye.com/blog/2379890)

<http://xiaotutu365.iteye.com/blog/2379890>

当调用parallelize()方法的时候，不指定分区数的时候，使用系统给出的分区数，而调用makeRDD()方法的时候，会为每个集合对象创建最佳分区，而这对后续的调用优化很有帮助。

textFile: 转换算子

collect : action

|  |
| --- |
| scala> val rdd1 = sc.textFile("hdfs://hdp-01:9000/wordcount/input/")  rdd1: org.apache.spark.rdd.RDD[String] = hdfs://hdp-01:9000/wordcount/input/ MapPartitionsRDD[7] at textFile at <console>:24  scala> val rdd2 = rdd1.flatMap(\_.split(" "))  rdd2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[8] at flatMap at <console>:26  scala> rdd2.collect  [Stage 1:> (0 + 3) / [Stage 1:=======================================> (2 + 1) / res3: Array[String] = Array(hello, spark, hello, tom, hello, jim, hello, hello, tom, tom, jim, jim, hello, spark, hello, tom, hello, jim, hello, hello, tom, tom, jim, jim, hello, spark, hello, tom, hello, jim, hello, hello, tom, tom, jim, jim)  scala> val rdd3 = rdd2.map((\_,1))  rdd3: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[9] at map at <console>:28  scala> val rdd4 = rdd3.reduceByKey(\_+\_)  rdd4: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[10] at reduceByKey at <console>:30  scala> rdd4.collect  res4: Array[(String, Int)] = Array((tom,9), (hello,15), (jim,9), (spark,3))  scala> val rdd4 = rdd3.reduceByKey((a,b)=>a + b)  rdd4: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[11] at reduceByKey at <console>:30  scala> val rdd5 = rdd4.sortBy(-\_.\_2)  rdd5: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[16] at sortBy at <console>:32  scala> rdd5.saveAsTextFile("hdfs://hdp-01:9000/wordcount/output1")  [Stage 7:> |

collect saveAsTextFile 是action算子

map flatMap reduceByKey sortBy 都是转换算子

转换算子，是lazy执行的。

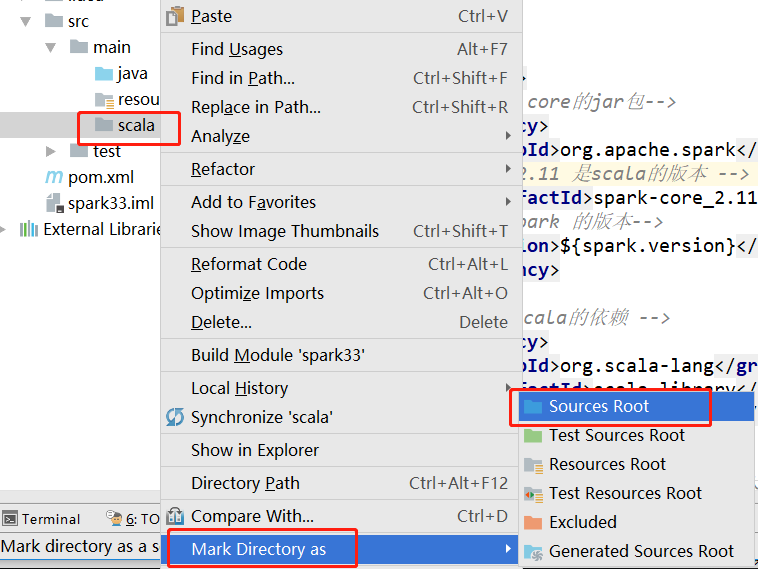
## maven工程

maven工程。

pom.xml文件中的配置：详细参考<pom.xml>

创建一个源码包：

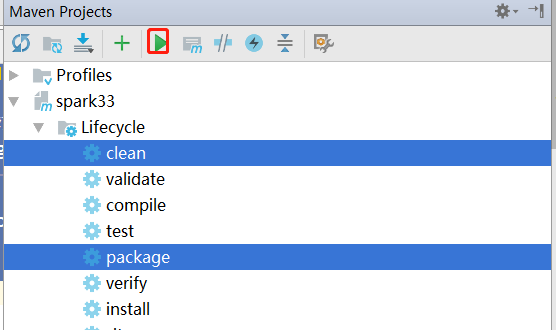
在main目录下，创建普通目录scala,然后右键，make xxx



## spark版本的wordcount

|  |
| --- |
| **def** main(args: Array[String]): Unit = {   **if** (args.length != 2) {  *println*(**"Usage :cn.huge.spark33.day01.ScalaWordCount <input> <output>"**)  sys.*exit*(1)  }   *// 参数接收* **val** *Array*(input, output) = args    **val** conf: SparkConf = **new** SparkConf()  *// 创建SparkContext* **val** sc: SparkContext = **new** SparkContext(conf)   *// 理论可以一行搞定，实际不推荐  // sc.textFile("").flatMap(\_.split(" ")).map((\_,1)).reduceByKey(\_+\_).saveAsTextFile("")   // 读取数据* **val** file: RDD[String] = sc.textFile(input)   *// 切分并压平* **val** words: RDD[String] = file.flatMap(\_.split(**" "**))   *// 组装* **val** wordAndOne: RDD[(String, Int)] = words.map((\_, 1))  *// 分组聚合* **val** result: RDD[(String, Int)] = wordAndOne.reduceByKey(\_ + \_)   *// 排序 降序  // 第一种 - 第二种 ： 第二个参数* **val** finalRes: RDD[(String, Int)] = result.sortBy(\_.\_2, **false**)   *// 直接存储到hdfs中* finalRes.saveAsTextFile(output)   *// 释放资源* sc.stop() |

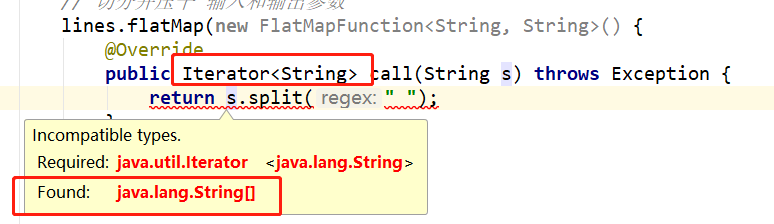
把程序打成jar包，提交到集群中运行：



|  |
| --- |
| spark-submit --master spark://hdp-01:7077 --class cn.huge.spark33.day01.ScalaWordCount /root/spark33-1.0-SNAPSHOT.jar hdfs://hdp-01:9000/wordcount/input hdfs://hdp-01:9000/wordcount/output2 |

## java编写spark的wordcount

API不同

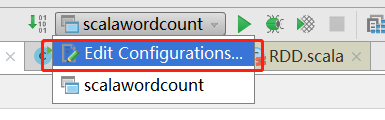


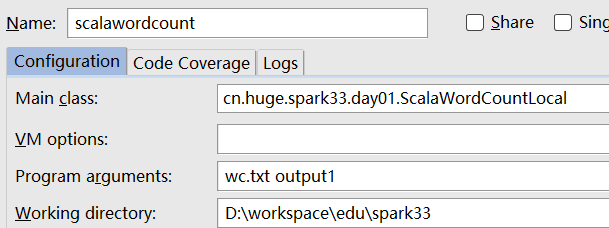
|  |
| --- |
| SparkConf conf = **new** SparkConf();  *// java的程序 一定是使用JavaAPI 去实现* JavaSparkContext sc = **new** JavaSparkContext(conf);   *// 读数据* JavaRDD<String> lines = sc.textFile(args[0]);   *// 切分并压平 输入和输出参数* JavaRDD<String> words = lines.flatMap(**new** FlatMapFunction<String, String>() {  @Override  **public** Iterator<String> call(String s) **throws** Exception {  *// 调用Arrays 生成指定类型* **return** Arrays.*asList*(s.split(**" "**)).iterator();  }  });   *// 组装 第一个：输入参数 输出参数（String，Int）* JavaPairRDD<String, Integer> wordAndOne = words.mapToPair(**new** PairFunction<String, String, Integer>() {  @Override  **public** Tuple2<String, Integer> call(String s) **throws** Exception {  *// 组装元组 // return Tuple2.apply(s,1);* **return new** Tuple2<>(s, 1);  }  });   *// 分组聚合 两个输入参数 一返回值类型* JavaPairRDD<String, Integer> result = wordAndOne.reduceByKey(**new** Function2<Integer, Integer, Integer>() {  @Override  **public** Integer call(Integer v1, Integer v2) **throws** Exception {  **return** v1 + v2;  }  });   *// 排序 先把k -v 互换 然后调用 sortByKey 然后再调换回来* JavaPairRDD<Integer, String> beforeSwap = result.mapToPair(**new** PairFunction<Tuple2<String, Integer>, Integer, String>() {  @Override  **public** Tuple2<Integer, String> call(Tuple2<String, Integer> tp) **throws** Exception {  *// 元素交换* **return** tp.swap();  }  });   *// 默认是升序 传false的参数* JavaPairRDD<Integer, String> sortedTp = beforeSwap.sortByKey(**false**);    JavaPairRDD<String, Integer> finalRes = sortedTp.mapToPair(**new** PairFunction<Tuple2<Integer, String>, String, Integer>() {  @Override  **public** Tuple2<String, Integer> call(Tuple2<Integer, String> integerStringTuple2) **throws** Exception {  **return** integerStringTuple2.swap();  }  });   *// 存储 写到hdfs中* finalRes.saveAsTextFile(args[1]);   sc.stop(); |

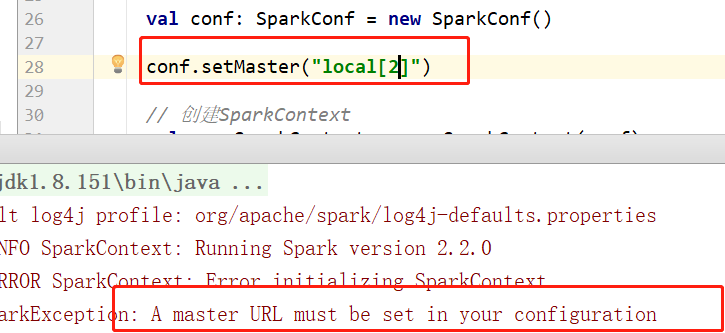
把程序提交到集群中运行

## 以local模式运行程序

本地模式下，输入参数：

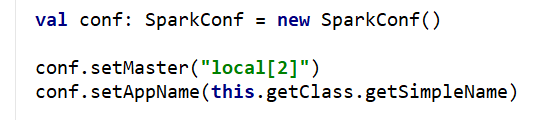






如果是local模式运行spark程序：

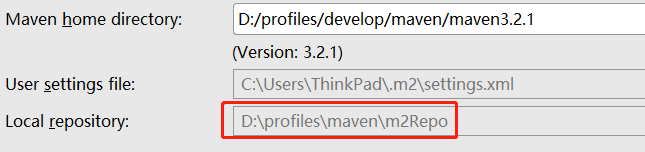
master地址，appname必须在Conf上设置。



## java lambda

|  |
| --- |
| SparkConf conf = **new** SparkConf(); *// java的程序 一定是使用JavaAPI 去实现* conf.setMaster(**"local[\*]"**); conf.setAppName(JavaLambdaWordCount.**class**.getSimpleName());  JavaSparkContext sc = **new** JavaSparkContext(conf);  *// 读数据* JavaRDD<String> lines = sc.textFile(args[0]);  *// t => t ->* JavaRDD<String> words = lines.flatMap(t -> Arrays.*asList*(t.split(**" "**)).iterator());  JavaPairRDD<String, Integer> wordAndOne = words.mapToPair(word -> **new** Tuple2<>(word, 1));  JavaPairRDD<String, Integer> result = wordAndOne.reduceByKey((a, b) -> a + b);  JavaPairRDD<Integer, String> beforeSwap = result.mapToPair(tp -> tp.swap());  JavaPairRDD<Integer, String> sorted = beforeSwap.sortByKey(**false**);  JavaPairRDD<String, Integer> finalRes = sorted.mapToPair(tp -> tp.swap());  finalRes.saveAsTextFile(args[1]);  sc.stop(); |

## maven工程的本地maven配置补充：



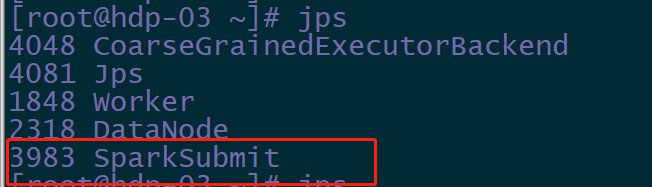
# spark中的基本机制

## spark中有哪些角色：

Master： 常驻进程，守护进程。 管理worker，接收提交的任务，进行任务的分配调度。

Worker: 常驻进程，守护进程。 报活；管理自己节点上的executor





当执行spark任务的时候，在哪里提交spark-submit命令，就在哪台机器上启动SparkSubmit进程。

CoarseGrainedExecutorBackend 简称为 executor

当程序执行的时候，会在worker中，启动executor进程。

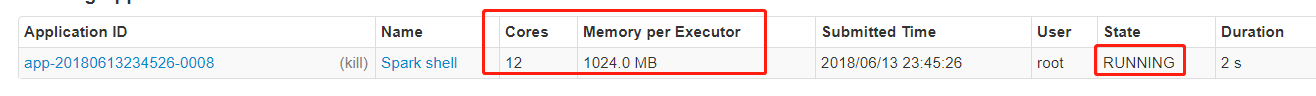
当spark程序执行完成之后，SparkSubmit，executor都会退出。

Driver: 在哪里创建SparkContext实例，哪里就是Driver。初始化的工作。

在当前的standalone的模式下，Driver就在客户端。和执行sparkSubmit进程在一起。

spark on yarn 的是，Driver，可能运行在集群中。

# 任务资源的默认值：



worker: 4cores 2.7g

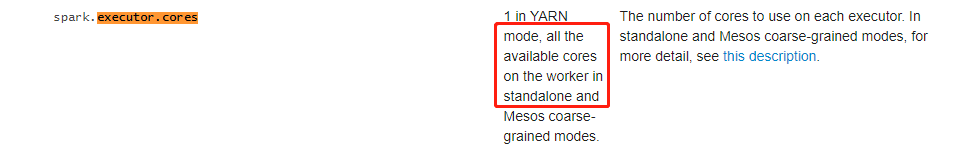
默认情况下，每一个worker启动一个executor

每一个executor占用所有的cores, 占用1024m内存。

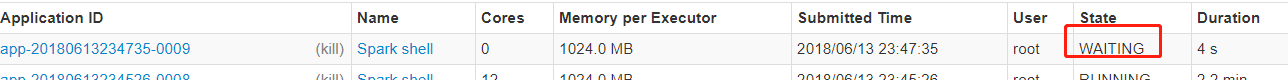
默认的配置文件：

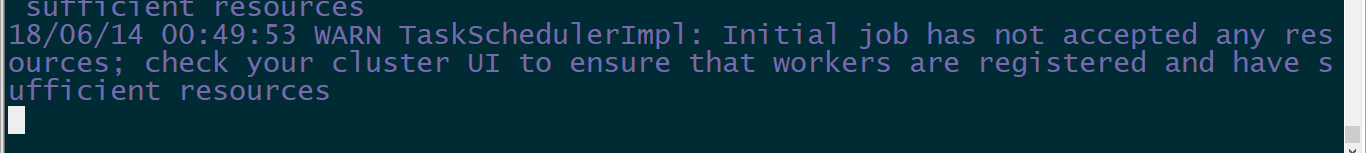
<http://spark.apache.org/docs/latest/configuration.html>





如果没有资源（cores,memory）,那么我们的程序，不能正常运行。





# 今日总结：

spark的集群

spark任务的提交

编程

spark中任务运行的基本机制

executor的默认资源

SparkCore中除了HashPartitioner分区器外，另外一个比较重要的已经实现的分区器，主要用于RDD的数据排序相关API中，比如sortByKey底层使用的数据分区器就是RangePartitioner分区器；该分区器的实现方式主要是通过两个步骤来实现的，第一步：先重整个RDD中抽取出样本数据，将样本数据排序，计算出每个分区的最大key值，形成一个Array[KEY]类型的数组变量rangeBounds；第二步：判断key在rangeBounds中所处的范围，给出该key值在下一个RDD中的分区id下标；该分区器要求RDD中的KEY类型必须是可以排序的