

# Spark 基础知识

## 目录

1、Spark 的产生背景.....	2
1.1、MapReduce 的发展 .....	2
1.1.1、MRv1 的缺陷 .....	2
1.1.2、MRv2 的缺陷 .....	2
1.1.3、Spark 的产生.....	3
2、Spark 概念 .....	4
3、Spark 特点 .....	5
3.1、Speed：快速高效.....	5
3.2、Ease of Use：简洁易用 .....	6
3.3、Generality：全栈式数据处理 .....	6
3.4、Runs Everywhere：兼容.....	7
4、Spark 应用场景.....	7
5、Spark 集群安装.....	8
5.1、Spark 版本选择.....	8
5.2、Spark 编译 .....	9
5.3、Spark 依赖环境.....	9
5.4、安装 JDK.....	9
5.5、安装 Scala .....	10
5.6、安装 Spark .....	10
5.6.1、Spark 分布式集群.....	10
5.6.2、Spark 高可用集群.....	14
5.6.3、配置 Spark HistoryServer.....	16
6、Spark 的基本使用.....	17
6.1、执行第一个 Spark 程序.....	17
6.2、启动 Spark Shell.....	18
6.3、在 Spark Shell 中编写 WordCount 程序 .....	19
6.4、在 IDEA 中编写 WordCount 程序 .....	20
7、修改 Spark 的日志级别.....	29
7.1、永久修改.....	29
8、Spark 的 WordCount.....	29
8.1、Scala 版本的 WordCount.....	29
8.2、Java7 版本的 WordCount .....	30
8.3、Java8 Lambda 表达式版本的 WordCount.....	33

# 1、Spark 的产生背景

## 1.1、MapReduce 的发展

### 1.1.1、MRv1 的缺陷

早在 Hadoop1.x 版本，当时采用的是 MRv1 版本的 MapReduce 编程模型。MRv1 版本的实现都封装在 org.apache.hadoop.mapred 包中，MRv1 的 Map 和 Reduce 是通过接口实现的。MRv1 包括三个部分：

运行时环境（JobTracker 和 TaskTracker）

编程模型（MapReduce）

数据处理引擎（MapTask 和 ReduceTask）

MRv1 存在以下不足：

**可扩展性差：**在运行时，JobTracker 既负责资源管理又负责任务调度，当集群繁忙时，JobTracker 很容易成为瓶颈，最终导致它的可扩展性问题。

**可用性差：**采用了单节点的 Master，没有备用 Master 及选举操作，这导致一旦 Master 出现故障，整个集群将不可用。单点故障

**资源利用率低：**TaskTracker 使用“slot”等量划分本节点上的资源量。“slot”代表计算资源（CPU、内存等）。一个 Task 获取到一个 slot 后才有机会运行，Hadoop 调度器负责将各个 TaskTracker 上的空闲 slot 分配给 Task 使用。一些 Task 并不能充分利用 slot，而其他 Task 也无法使用这些空闲的资源。slot 分为 Map slot 和 Reduce slot 两种，分别供 MapTask 和 ReduceTask 使用。有时会因为作业刚刚启动等原因导致 MapTask 很多，而 Reduce Task 任务还没有调度的情况，这时 Reduce slot 也会被闲置。

**不能支持多种 MapReduce 框架：**无法通过可插拔方式将自身的 MapReduce 框架替换为其他实现，如 Spark、Storm 等。

### 1.1.2、MRv2 的缺陷

Apache 为了解决 MRv1 中的缺陷，对 Hadoop 进行了升级改造。MRv2 就诞生了。

MRv2 中，重用了 MRv1 中的编程模型和数据处理引擎。但是运行时环境被重构了。JobTracker 被拆分成了通用的

资源调度平台（ResourceManager，简称 RM）、

节点管理器（NodeManager）、

负责各个计算框架的任务调度模型（ApplicationMaster，简称 AM）。

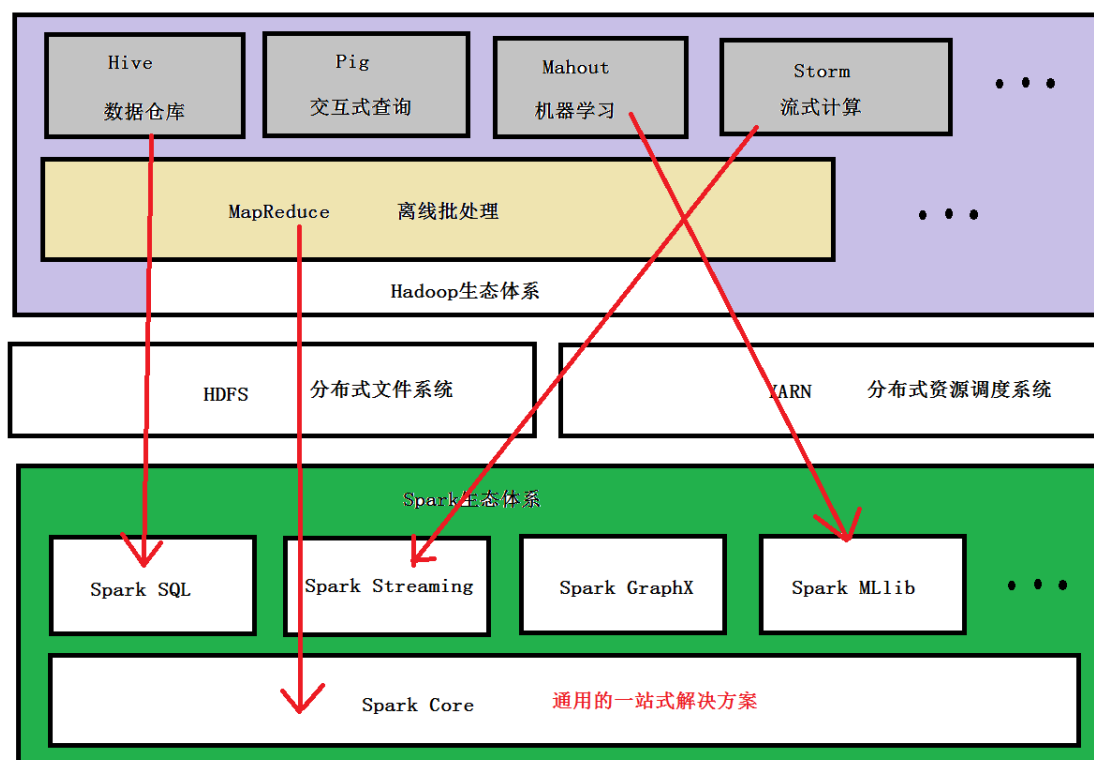
ResourceManager 依然负责对整个集群的资源管理，但是在任务资源的调度方面只负责将资源封装为 Container 分配给 ApplicationMaster 的一级调度，二级调度的细节将交给 ApplicationMaster 去完成，这大大减轻了 ResourceManager 的压力，使得 ResourceManager 更加轻量。NodeManager 负责对单个节点的资源管理，并将资源信息、Container 运行状态、健康状况等信息上报给 ResourceManager。ResourceManager 为了保证 Container 的利用率，会监控 Container，如果 Container 未在有限的时间内使用，ResourceManager 将命令 NodeManager 杀死 Container，以便于将资源分配给其他任务。MRv2 的核心不再是 MapReduce 框架，而是 YARN。在以 YARN 为核心的 MRv2 中，MapReduce 框架是可插拔的，完全可以替换为其他分布式计算模型实现，比如 Spark、Storm 等。

Hadoop MRv2 虽然解决了 MRv1 中的一些问题，但是**由于对 HDFS 的频繁操作（包括计算结果持久化、数据备份、资源下载及 Shuffle 等）导致磁盘 I/O 成为系统性能的瓶颈，因此只适用于离线数据处理或批处理，而不能支持对迭代式、交互式、流式数据的处理。**

重点概念：

离线处理，批处理，实时处理，流式处理

### 1.1.3、Spark 的产生



Spark 看到 MRv2 的问题，对 MapReduce 做了大量优化，总结如下：

**减少磁盘 I/O：**随着实时大数据应用越来越多，Hadoop 作为离线的高吞吐、低响应框架已不能满足这类需求。Hadoop MapReduce 的 map 端将中间输出和结果存储在磁盘中，reduce 端又需要从磁盘读写中间结果，势必造成磁盘 IO 成为瓶颈。Spark 允许将 map 端的中间输出和结果存储在内存中，reduce 端在拉取中间结果时避免了大量的磁盘 I/O。Hadoop YARN 中

的 ApplicationMaster 申请到 Container 后，具体的任务需要利用 NodeManager 从 HDFS 的不同节点下载任务所需的资源（如 Jar 包），这也增加了磁盘 I/O。Spark 将应用程序上传的资源文件缓冲到 Driver 本地文件服务的内存中，当 Executor 执行任务时直接从 Driver 的内存中读取，也节省了大量的磁盘 I/O。

**增加并行度**：由于将中间结果写到磁盘与从磁盘读取中间结果属于不同的环节，Hadoop 将它们简单的通过串行执行衔接起来。Spark 把不同的环节抽象为 Stage，允许多个 Stage 既可以串行执行，又可以并行执行。

**避免重新计算**：当 Stage 中某个分区的 Task 执行失败后，会重新对此 Stage 调度，但在重新调度的时候会过滤已经执行成功的分区任务，所以不会造成重复计算和资源浪费。

**可选的 Shuffle 和排序**：Hadoop MapReduce 在 Shuffle 之前有着固定的排序操作（只能对 key 排字典顺序），而 Spark 则可以根据不同场景选择在 map 端排序或者 reduce 端排序。

**灵活的内存管理策略**：Spark 将内存分为堆上的存储内存、堆外的存储内存、堆上的执行内存、堆外的执行内存 4 个部分。Spark 既提供了执行内存和存储内存之间是固定边界的实现，又提供了执行内存和存储内存之间是“软”边界的实现。Spark 默认使用“软”边界的实现，执行内存或存储内存中的任意一方在资源不足时都可以借用另一方的内存，最大限度的提高资源的利用率，减少对资源的浪费。Spark 由于对内存使用的偏好，内存资源的多寡和使用率就显得尤为重要，为此 Spark 的内存管理器提供的 Tungsten 实现了一种与操作系统的内存 Page 非常相似的数据结构，用于直接操作操作系统内存，节省了创建的 Java 对象在堆中占用的内存，使得 Spark 对内存的使用效率更加接近硬件。Spark 会给每个 Task 分配一个配套的任务内存管理器，对 Task 粒度的内存进行管理。Task 的内存可以被多个内部的消费者消费，任务内存管理器对每个消费者进行 Task 内存的分配与管理，因此 Spark 对内存有着更细粒度的管理。

基于以上所列举的优化，Spark 官网声称性能比 Hadoop 快 100 倍。即便是内存不足需要磁盘 I/O 时，其速度也是 Hadoop 的 10 倍以上。

Spark 会取代 Hadoop 么？

Spark 是 MapReduce 的替代方案，而且兼容 HDFS、Hive，可融入 Hadoop 的生态系统，以弥补 MapReduce 的不足。

## 2、Spark 概念

官网：<http://spark.apache.org/>

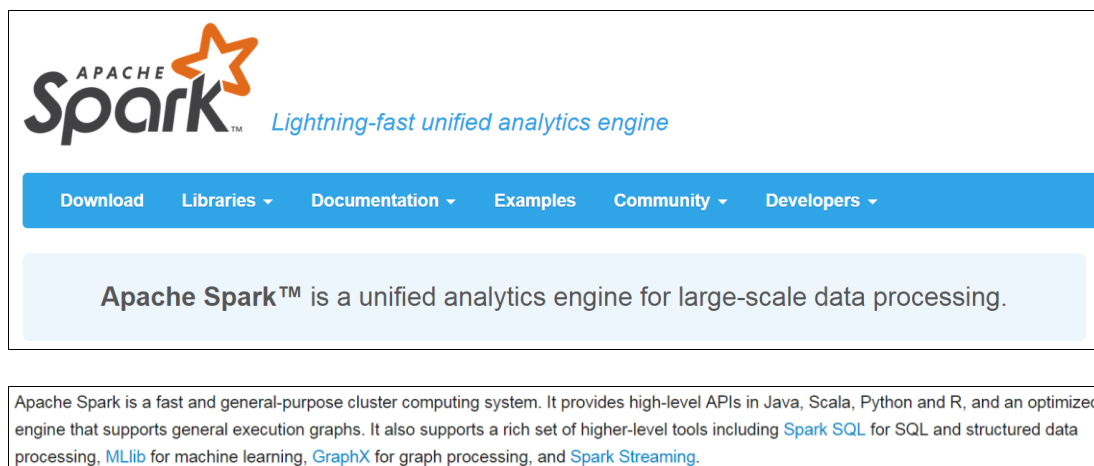
Spark 是一种快速、通用、可扩展的大数据分析引擎

2009 年诞生于加州大学伯克利分校 AMPLab

2010 年开源

2013 年 6 月成为 Apache 孵化项目

2014 年 2 月成为 Apache 顶级项目



Spark 生态圈也称为 BDAS（伯克利数据分析栈），是伯克利 APMLab 实验室打造的，力图在算法（Algorithms）、机器（Machines）、人（People）之间通过大规模集成来展现大数据应用的一个平台。伯克利 AMPLab 运用大数据、云计算、通信等各种资源以及各种灵活的技术方案，对海量不透明的数据进行甄别并转化为有用的信息，以供人们更好的理解世界。该生态圈已经涉及到机器学习、数据挖掘、数据库、信息检索、自然语言处理和语音识别等多个领域。

Spark 生态圈以 SparkCore 为核心，从 HDFS、Amazon S3 或者 HBase 等持久层读取数据，以 MESOS、YARN 和自身携带的 Standalone 为资源管理器调度 Job 完成 Spark 应用程序的计算。这些应用程序可以来自于不同的组件，如 SparkShell/SparkSubmit 的批处理、SparkStreaming 的实时处理应用、SparkSQL 的结构化数据处理/即席查询、BlinkDB 的权衡查询、MLlib/MLbase 的机器学习、GraphX 的图处理和 PySpark 的数学/科学计算和 SparkR 的数据分析等等。

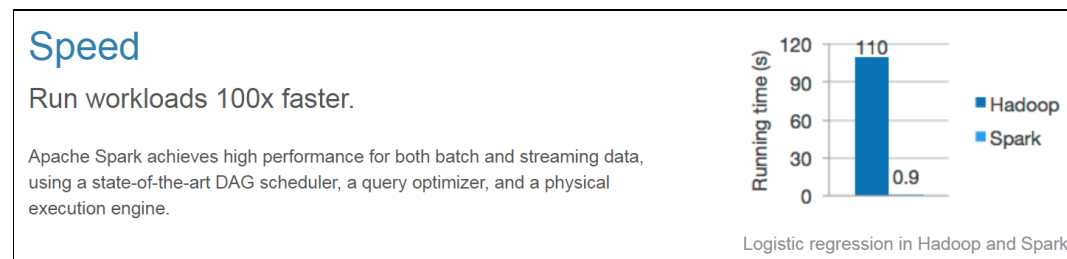
目前，Spark 生态系统已经发展成为一个包含多个子项目的集合，其中包含 Spark SQL、Spark Streaming、GraphX、MLlib 等子项目，Spark 是基于内存计算的大数据并行计算框架。Spark 基于内存计算，提高了在大数据环境下数据处理的实时性，同时保证了高容错性和高可伸缩性，允许用户将 Spark 部署在大量廉价硬件之上，形成集群。Spark 得到了众多大数据公司的支持，这些公司包括 Hortonworks、IBM、Intel、Cloudera、MapR、Pivotal、百度、阿里、腾讯、京东、携程、优酷土豆。当前百度的 Spark 已应用于凤巢、大搜索、直达号、百度大数据等业务；阿里利用 GraphX 构建了大规模的图计算和图挖掘系统，实现了很多生产系统的推荐算法；腾讯 Spark 集群达到 8000 台的规模，是当前已知的世界上最大的 Spark 集群。

## 3、Spark 特点

### 3.1、Speed：快速高效

随着实时大数据应用越来越多，Hadoop 作为离线的高吞吐、低响应框架已不能满足这类需求。Hadoop MapReduce 的 Job 将中间输出和结果存储在 HDFS 中，读写 HDFS 造成磁盘 IO 成

为瓶颈。Spark 允许将中间输出和结果存储在内存中，节省了大量的磁盘 IO。Apache Spark 使用最先进的 DAG 调度程序，查询优化程序和物理执行引擎，实现批量和流式数据的高性能。同时 Spark 自身的 DAG 执行引擎也支持数据在内存中的计算。Spark 官网声称性能比 Hadoop 快 100 倍。即便是内存不足需要磁盘 IO，其速度也是 Hadoop 的 10 倍以上。



## 3.2、Ease of Use: 简洁易用

Spark 现在支持 Java、Scala、Python 和 R 等编程语言编写应用程序，大大降低了使用者的门槛。自带了 80 多个高等级操作符，允许在 Scala, Python, R 的 shell 中进行交互式查询，可以非常方便的在这些 Shell 中使用 Spark 集群来验证解决问题的方法。

**Ease of Use**

Write applications quickly in Java, Scala, Python, R, and SQL.

```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Spark's Python DataFrame API  
Read JSON files with automatic schema inference

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python, R, and SQL shells.

## 3.3、Generality: 全栈式数据处理

Spark 提供了统一的解决方案。Spark 统一的解决方案非常具有吸引力，毕竟任何公司都想用统一的平台去处理遇到的问题，减少开发和维护的人力成本和部署平台的物力成本。

**支持批处理 (Spark Core)**。Spark Core 是 Spark 的核心功能实现，包括：SparkContext 的初始化（DriverApplication 通过 SparkContext 提交）、部署模式、存储体系、任务提交与执行、计算引擎等。

**支持交互式查询 (Spark SQL)**。Spark SQL 是 Spark 来操作结构化数据的程序包，可以让我们使用 SQL 语句的方式来查询数据，Spark 支持多种数据源，包含 Hive 表，parquet 以及 JSON 等内容。

**支持流式计算 (Spark Streaming)**。与 MapReduce 只能处理离线数据相比，Spark 还支持实时的流计算。Spark 依赖 Spark Streaming 对数据进行实时的处理。

**支持机器学习 (Spark MLlib)**。提供机器学习相关的统计、分类、回归等领域的多种算法实现。其一致的 API 接口大大降低了用户的学习成本。

**支持图计算 (Spark GraphX)**。提供图计算处理能力，支持分布式，Pregel 提供的 API 可以解决图计算中的常见问题。

**支持 Python 操作--PySpark**

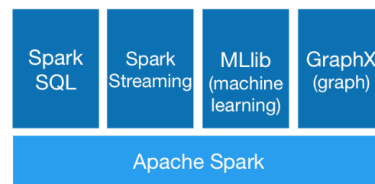
**支持 R 语言--SparkR**



## Generality

Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including [SQL and DataFrames](#), [MLlib](#) for machine learning, [GraphX](#), and [Spark Streaming](#). You can combine these libraries seamlessly in the same application.



## 3.4、Runs Everywhere: 兼容

**可用性高。** Spark 也可以不依赖于第三方的资源管理和调度器，它实现了 **Standalone** 作为其内置的资源管理和调度框架，这样进一步降低了 Spark 的使用门槛，使得所有人都可以非常容易地部署和使用 Spark，此模式下的 Master 可以有多个，解决了单点故障问题。当然，此模式也完全可以使用其他集群管理器替换，比如 **YARN、Mesos、Kubernetes、EC2** 等。

**丰富的数据源支持。** Spark 除了可以访问操作系统自身的本地文件系统和 HDFS 之外，还可以访问 Cassandra、HBase、Hive、Tachyon 以及任何 Hadoop 的数据源。这极大地方便了已经使用 HDFS、HBase 的用户顺利迁移到 Spark。

## Runs Everywhere

Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.

You can run Spark using its [standalone cluster mode](#), on [EC2](#), on [Hadoop YARN](#), on [Mesos](#), or on [Kubernetes](#). Access data in [HDFS](#), [Apache Cassandra](#), [Apache HBase](#), [Apache Hive](#), and hundreds of other data sources.



Spark 支持的几种部署方案：

**Mesos:** Spark 可以运行在 Mesos 里面（Mesos 类似于 YARN 的一个资源调度框架）

**Standalone:** Spark 自己可以给自己分配资源（Master, Worker）

**YARN:** Spark 可以运行在 Hadoop 的 YARN 上面

**Kubernetes:** Spark 接收 Kubernetes 的资源调度

## 4、Spark 应用场景

目前大数据处理场景有以下几个类型：

- 1、复杂的批量处理（Batch Data Processing），偏重点在于处理海量数据的能力，至于处理速度可忍受，通常的时间可能是在数十分钟到数小时；
- 2、基于历史数据的交互式查询（Interactive Query），通常的时间在数十秒到数十分钟之间
- 3、基于实时数据流的数据处理（Streaming Data Processing），通常在数百毫秒到数秒之间

目前对以上三种场景需求都有比较成熟的处理框架：

第一种情况可以用 Hadoop 的 MapReduce 来进行批量海量数据处理

第二种情况可以 Impala、Kylin 进行交互式查询

第三种情况可以用 Storm 分布式处理框架处理实时流式数据

以上三者都是比较独立，各自一套维护成本比较高，而 Spark 的出现能够一站式平台满足以上需求。

第一种情况使用 Spark Core 解决

第二种情况使用 Spark SQL 解决

第三种情况使用 Spark Streaming 解决

通过以上分析，总结 Spark 场景有以下几个：

- 1、Spark 是基于内存的迭代计算框架，适用于需要多次操作特定数据集的应用场合。需要反复操作的次数越多，所需读取的数据量越大，受益越大，数据量小但是计算密集度较大的场合，受益就相对较小
- 2、由于 RDD 的特性，Spark 不适用那种异步细粒度更新状态的应用，例如 web 服务的存储或者是增量的 web 爬虫和索引。就是对于那种增量修改的应用模型不适合
- 3、数据量不是特别大，但是要求实时统计分析需求

典型行业应用场景：

- 1、Yahoo 将 Spark 用在 Audience Expansion 中的应用，进行 **点击预测和即席查询**等
- 2、淘宝技术团队使用了 Spark 来解决多次迭代的 **机器学习算法、高计算复杂度的算法**等。应用于 **内容推荐、社区发现**等
- 3、腾讯大数据精准推荐借助 Spark 快速迭代的优势，实现了在“**数据实时采集、算法实时训练、系统实时预测**”的全流程实时并行高维算法，最终成功应用于广点通 pCTR 投放系统上。
- 4、优酷土豆将 Spark 应用于 **视频推荐(图计算)、广告业务**，主要实现 **机器学习、图计算**等迭代计算。
- 5、.....

## 5、Spark 集群安装

### 5.1、Spark 版本选择

三大主要版本：

Spark-0.X

Spark-1.X（主要 Spark-1.3 和 Spark-1.6）

Spark-2.X（最新 Spark-2.3）

官网首页：<http://spark.apache.org/downloads.html>



## Download Apache Spark™

1. Choose a Spark release:
2. Choose a package type:
3. Download Spark: [spark-2.3.0-bin-hadoop2.7.tgz](#)
4. Verify this release using the [2.3.0 signatures and checksums](#) and [project release KEYS](#).

*Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build with Scala 2.10 support.*

或者其他镜像站:

<https://mirrors.tuna.tsinghua.edu.cn/apache/spark/>

<https://www.apache.org/dyn/closer.lua/spark/spark-2.3.0/spark-2.3.0-bin-hadoop2.7.tgz>

<https://www.apache.org/dyn/closer.lua/spark/>

我们选择的版本: **spark-2.3.0-bin-hadoop2.7.tgz**

## 5.2、Spark 编译

自行利用搜索引擎解决, 可做可不做

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

## 5.3、Spark 依赖环境

在官网文档中有一句话:

### Downloading

Get Spark from the [downloads page](#) of the project website. This documentation is for Spark version 2.3.0. Spark uses Hadoop's client libraries for HDFS and YARN. Downloads are pre-packaged for a handful of popular Hadoop versions. Users can also download a "Hadoop free" binary and run Spark with any Hadoop version [by augmenting Spark's classpath](#). Scala and Java users can include Spark in their projects using its Maven coordinates and in the future Python users can also install Spark from PyPI.

If you'd like to build Spark from source, visit [Building Spark](#).

Spark runs on both Windows and UNIX-like systems (e.g. Linux, Mac OS). It's easy to run locally on one machine — all you need is to have `java` installed on your system `PATH`, or the `JAVA_HOME` environment variable pointing to a Java installation.

Spark runs on **Java 8+**, **Python 2.7+/3.4+** and **R 3.1+**. For the Scala API, Spark 2.3.0 uses **Scala 2.11**. You will need to use a compatible Scala version (2.11.x).

Note that support for Java 7, Python 2.6 and old Hadoop versions before 2.6.5 were removed as of Spark 2.2.0. Support for Scala 2.10 was removed as of 2.3.0.

所以总结:

Spark-2.3 需要依赖: **Java 8+** 和 **Python 2.7+/3.4+** 和 **Scala 2.11** 和 **R 3.1+**

## 5.4、安装 JDK

略

## 5.5、安装 Scala

略

## 5.6、安装 Spark

### 5.6.1、Spark 分布式集群

Spark 也是一个主从架构的分布式计算引擎。

主节点是 Master，从节点是 Worker

所以集群规划：

Server	Master	Worker
hadoop02	√	√
hadoop03		√
hadoop04		√
hadoop05		√

详细安装步骤：

1、上传下载好的 Spark 到集群中的一个节点，比如是 hadoop05  
`put c:/spark-2.3.0-bin-hadoop2.7.tgz`

2、使用之前安装 hadoop 集群相同的 hadoop 用户安装 spark 集群，现在规划安装目录 /home/hadoop/apps/，解压缩进行安装：  
`tar -zxvf spark-2.3.0-bin-hadoop2.7.tgz -apps /home/hadoop/apps/`

3、修改配置文件 spark-env.sh  
进入 SPARK\_HOME 的 conf 目录中，进行如下更改：  
`cd /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/conf`  
`mv spark-env.sh.template spark-env.sh`  
然后修改 spark-env.sh：  
`export JAVA_HOME=/usr/local/java/jdk1.8.0_73`  
`export SPARK_MASTER_HOST=hadoop02`  
`export SPARK_MASTER_PORT=7077`

4、修改配置文件 slave  
进入 SPARK\_HOME 的 conf 目录中，进行如下更改：  
`cd /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/conf`  
`mv slaves.template slaves`  
在 slaves 的最后添加所有 worker 节点的主机名  
`hadoop02`  
`hadoop03`  
`hadoop04`

## hadoop05

5、将 spark 安装包 copy 到所有安装节点

```
scp -r spark-2.3.0-bin-hadoop2.7 hadoop02:/home/hadoop/apps/
```

```
scp -r spark-2.3.0-bin-hadoop2.7 hadoop03:/home/hadoop/apps/
```

```
scp -r spark-2.3.0-bin-hadoop2.7 hadoop04:/home/hadoop/apps/
```

6、配置环境变量

```
vim ~/.bashrc
```

```
export SPARK_HOME=/home/hadoop/apps/spark-2.3.0-bin-hadoop2.7
```

```
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

```
source ~/.bashrc
```

千万注意：HADOOP\_HOME/sbin 和 SPARK\_HOME/sbin 目录中都包含 start-all.sh 和 stop-all.sh 脚本。所以会有冲突。所以在使用有冲突的命令等要千万注意。

如果区分不清楚，那么不推荐配置环境变量

7、启动 Spark 集群

```
[hadoop@hadoop02 ~]$ cd /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7
```

```
[hadoop@hadoop02 spark-2.3.0-bin-hadoop2.7]$ sbin/start-all.sh
```

```
[hadoop@hadoop02 spark-2.3.0-bin-hadoop2.7]$ sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/logs/spark-
hadoop05: starting org.apache.spark.deploy.worker.worker, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/
op05.out
hadoop02: starting org.apache.spark.deploy.worker.worker, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/
op02.out
hadoop04: starting org.apache.spark.deploy.worker.worker, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/
op04.out
hadoop03: starting org.apache.spark.deploy.worker.worker, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/
op03.out
[hadoop@hadoop02 spark-2.3.0-bin-hadoop2.7]$ jps
13936 HRegionServer
20752 Jps
3393 DataNode
3281 NameNode
20706 worker
4242 HMaster
3462 NodeManager
2537 QuorumPeerMain
3929 DFSZKFailoverController
3755 JournalNode
20621 Master
4814 RunJar
[hadoop@hadoop02 spark-2.3.0-bin-hadoop2.7]$
```

8、验证集群启动是否成功

8.1、验证每个节点上的对应进程是否都启动正常

```
hadoop03 - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Enter host address
hadoop03
[hadoop@hadoop03 ~]$ jps
2865 NodeManager
2721 NameNode
3170 DFSZKFailoverController
11235 worker
7189 HRegionServer
2502 QuorumPeerMain
3079 JournalNode
11306 Jps
2796 DataNode
[hadoop@hadoop03 ~]$

master : hadoop02

worker : hadoop02
        hadoop03
        hadoop04
        hadoop05

hadoop05
[hadoop@hadoop05 ~]$ jps
3488 HRegionServer
3286 JobHistoryServer
5542 NodeManager
2599 DataNode
6424 Jps
6362 worker
3403 ResourceManager
2831 HMaster
[hadoop@hadoop05 ~]$

hadoop02
[hadoop@hadoop02 ~]$ jps
13936 HRegionServer
3393 DataNode
3281 NameNode
20706 worker
4242 HMaster
3462 NodeManager
2537 QuorumPeerMain
3929 DFSZKFailoverController
3755 JournalNode
20621 Master
20830 Jps
4814 RunJar
[hadoop@hadoop02 ~]$

hadoop04
[hadoop@hadoop04 ~]$ jps
10017 Jps
6467 HRegionServer
2983 NodeManager
2503 QuorumPeerMain
9960 worker
2875 DataNode
2812 ResourceManager
3183 JournalNode
[hadoop@hadoop04 ~]$
```

## 8.2、验证 Spark Web UI

打开浏览器访问：<http://hadoop02:8080/>

hadoop02 就是 master 所在的服务器

Spark Master at spark://hadoop02:7077

URL: spark://hadoop02:7077  
REST URL: spark://hadoop02:6066 (cluster mode)  
Alive Workers: 4  
Cores in use: 4 Total, 0 Used  
Memory in use: 4.0 GB Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

Workers (4)

Worker Id	Address	State	Cores	Memory
worker-20180503110451-192.168.123.102-55508	192.168.123.102-55508	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503110451-192.168.123.103-56956	192.168.123.103-56956	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503110451-192.168.123.104-47404	192.168.123.104-47404	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503110451-192.168.123.105-46089	192.168.123.105-46089	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications (0)

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

Completed Applications (0)

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

## 8.3、测试能否运行一个 Spark 例子程序

提交一个 spark 程序：

**[hadoop@hadoop03 ~]\$ run-example SparkPi 10**

最后结果：

```
2018-05-03 11:15:15 INFO TaskSchedulerImpl:54 - Finished task 9/0 in stage 0.0 (r1b-0) in 19 ms on localmost executor 0
2018-05-03 11:15:15 INFO DAGScheduler:54 - ResultStage 0 (reduce at SparkPi.scala:38) finished in 1.111 s
2018-05-03 11:15:15 INFO DAGScheduler:54 - Job 0 finished: reduce at SparkPi.scala:38, took 1.220557 s
Pi is roughly 3.136863136863137
2018-05-03 11:15:15 INFO AbstractConnector:318 - Stopped Spark@b0964b2{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2018-05-03 11:15:15 INFO SparkUI:54 - Stopped Spark web UI at http://hadoop03:4040
2018-05-03 11:15:15 INFO MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMasterEndpoint stopped!
2018-05-03 11:15:15 INFO MemoryStore:54 - MemoryStore cleared
2018-05-03 11:15:15 INFO BlockManager:54 - BlockManager stopped
2018-05-03 11:15:15 INFO BlockManagerMaster:54 - BlockManagerMaster stopped
2018-05-03 11:15:15 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 - OutputCommitCoordinator stopped!
2018-05-03 11:15:15 INFO SparkContext:54 - Successfully stopped SparkContext
2018-05-03 11:15:15 INFO ShutdownHookManager:54 - Shutdown hook called
2018-05-03 11:15:15 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-a05636bb-59d3-40dd-954e-730cedb72389
2018-05-03 11:15:15 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-81c5ec3b-27e6-4a42-886a-7db767ced64d
[hadoop@hadoop03 ~]$
```

或者:

```
[hadoop@hadoop3 ~]$ ~/apps/spark-2.3.0-bin-hadoop2.7/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://hadoop02:7077 \
--executor-memory 512m \
--total-executor-cores 1 \
~/apps/spark-2.3.0-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.3.0.jar \
100
```

--master spark://hadoop02:7077	指定 Master 的地址
--executor-memory 512m	指定每个 worker 可用内存为 500m
--total-executor-cores 1	指定整个集群使用的 CPU 核数为 1 个

#### 8.4、进入 Spark Shell 提交 wordcount 程序:

数据准备:

```
[hadoop@hadoop05 ~]$ pwd
/home/hadoop
[hadoop@hadoop05 ~]$ cat words.txt
hello huangbo
hello xuzheng
hello wangbaoqiang
[hadoop@hadoop05 ~]$
```

进入 Spark Shell:

```
[hadoop@hadoop2 ~] spark-shell
```


或者

```
[hadoop@hadoop2 ~]$ ~/apps/spark-2.3.0-bin-hadoop2.7/bin/spark-shell \
> --master spark://hadoop02:7077 \
> --executor-memory 512m \
> --total-executor-cores 1
```

执行程序:

```
sc.textFile("/home/hadoop/words.txt").flatMap(_.split(" "))
.map(_._1).reduceByKey(_+_).foreach(println)
```

```
[hadoop@hadoop05 ~]$ spark-shell
2018-05-03 11:17:57 WARN NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin
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://hadoop05:4040
spark context available as 'sc' (master = local[*], app id = local-1525317486789).
spark session available as 'spark'.
welcome to

 version 2.3.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)
Type in expressions to have them evaluated.
Type :help for more information.

scala> sc.textFile("/home/hadoop/words.txt").flatMap(_.split(" ")).map(_._1).reduceByKey(_+_).foreach(println)
(wangbaoqiang,1)
(xuzheng,1)
(huangbo,1)
(hello,3)

scala>
```

如果是 Spark-1.6.3, 那么启动的 spark-shell 如下:

```
[hadoop@hadoop02 ~]$ apps/spark-1.6.3-bin-hadoop2.6/bin/spark-shell
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's repl log4j profile: org/apache/spark/log4j-defaults-repl.properties
To adjust logging level use sc.setLogLevel("INFO")
Welcome to

  SPARK  version 1.6.3

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.
18/05/04 08:48:15 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:48:16 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:48:19 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification
18/05/04 08:48:19 WARN ObjectStore: failed to get database default, returning NoSuchObjectException
18/05/04 08:48:22 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:48:23 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
SQL context available as sqlContext.

scala> sc.textFile("/home/hadoop/words.txt").flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).foreach(println)
(wangbaoqiang,1)
(xuzheng,1)
(huangbo,1)
(hello,3)

scala>
```

注意：

如果启动 Spark Shell 时没有指定 master 地址，但是也可以正常启动 Spark Shell 和执行 Spark Shell 中的程序，其实是启动了 Spark 的 local 模式，该模式仅在本机启动一个进程，没有与集群建立联系。

Spark Shell 中已经默认将 SparkContext 类初始化为对象 sc。用户代码如果需要用到，则直接应用 sc 即可。

Spark Shell 中已经默认将 Spark Session 类初始化为对象 spark。用户代码如果需要用到，则直接应用 spark 即可。

注意 Spark2 和 Spark1 的区别

## 5.6.2、Spark 高可用集群

在上面的 4.6.1 中的安装的 Spark 集群是一个普通的分布式集群，存在 master 节点的单点故障问题。Hadoop 在 2.X 版本开始，已经利用 ZooKeeper 解决了单点故障问题。同样的策略，Spark 也利用 ZooKeeper 解决 Spark 集群的单点故障问题

集群规划：

Server	Master	Worker
hadoop02	√	√
hadoop03		√
hadoop04	√	√
hadoop05		√

具体步骤：

1、停止 Spark 集群

```
[hadoop@hadoop02 ~]$ cd /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7
```

```
[hadoop@hadoop02 ~]$ sbt/bin/stop-all.sh
```

2、配置 ZooKeeper 集群，并且启动好 ZooKeeper 集群



3、修改 SPARK\_HOME/conf 目录中的 spark-env.sh 配置文件：

删掉：

```
export SPARK_MASTER_HOST=hadoop02
```

增加一行：

```
export SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -  
Dspark.deploy.zookeeper.url=hadoop02,hadoop03,hadoop04 -  
Dspark.deploy.zookeeper.dir=/spark"
```

解释：

**-Dspark.deploy.recoveryMode=ZOOKEEPER**

说明整个集群状态是通过 zookeeper 来维护的，整个集群状态的恢复也是通过 zookeeper 来维护的。就是说用 zookeeper 做了 Spark 的 HA 配置，Master(Active)挂掉的话，Master(standby)要想变成 Master (Active) 的话，Master(Standby)就要像 zookeeper 读取整个集群状态信息，然后进行恢复所有 Worker 和 Driver 的状态信息，和所有的 Application 状态信息

**-Dspark.deploy.zookeeper.url=hadoop2:hadoop03:hadoop04**

#将所有配置了 zookeeper，并且在这台机器上有可能做 master(Active)的机器都配置进来（我用了 3 台，就配置了 3 台）

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

这里的 dir 和 zookeeper 配置文件 zoo.cfg 中的 dataDir 的区别？？

-Dspark.deploy.zookeeper.dir 是保存 spark 的元数据，保存了 spark 的作业运行状态；zookeeper 会保存 spark 集群的所有的状态信息，包括所有的 Workers 信息，所有的 Applactions 信息，所有的 Driver 信息,如果集群

4、同步配置文件

```
[hadoop@hadoop02 conf]$ scp -r spark-env.sh hadoop03:$PWD
```

```
[hadoop@hadoop02 conf]$ scp -r spark-env.sh hadoop04:$PWD
```

```
[hadoop@hadoop02 conf]$ scp -r spark-env.sh hadoop05:$PWD
```

5、启动集群

在 hadoop02 上执行：

```
[hadoop@hadoop02 ~]$ cd /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7
```

```
[hadoop@hadoop02 spark-2.3.0-bin-hadoop2.7]$ sbin/start-all.sh
```

```
[hadoop@hadoop02 ~]$ cd /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7  
[hadoop@hadoop02 spark-2.3.0-bin-hadoop2.7]$ sbin/start-all.sh  
starting org.apache.spark.deploy.master.Master, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/logs/spa  
hadoop02: starting org.apache.spark.deploy.worker.Worker, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.  
op02.out  
hadoop03: starting org.apache.spark.deploy.worker.Worker, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.  
op03.out  
hadoop05: starting org.apache.spark.deploy.worker.Worker, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.  
op05.out  
hadoop04: starting org.apache.spark.deploy.worker.Worker, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.  
op04.out  
[hadoop@hadoop02 spark-2.3.0-bin-hadoop2.7]$
```

此时，通过观察启动日志，或者检查 hadoop04 上是否包含有 master 进程等都可以得知 hadoop04 上的 master 并不会自动启动，所以需要手动启动

那么在 hadoop04 执行命令进行启动：

```
[hadoop@hadoop04 ~]$ cd /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7
[hadoop@hadoop04 spark-2.3.0-bin-hadoop2.7]$ sbin/start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7
[hadoop@hadoop04 spark-2.3.0-bin-hadoop2.7]$ jps
10449 Jps
6467 HRegionServer
10323 Worker
10390 Master
2983 NodeManager
2503 QuorumPeerMain
2875 DataNode
2812 ResourceManager
3183 JournalNode
[hadoop@hadoop04 spark-2.3.0-bin-hadoop2.7]$
```

## 6、验证高可用

这是正常情况：

Hadoop02 是 spark 集群的 active master 节点

Hadoop04 是 spark 集群的 standby master 节点

Spark Master at spark://hadoop02:7077

URL: spark://hadoop02:7077  
REST URL: spark://hadoop02:6066 (cluster mode)  
Alive Workers: 4  
Cores in use: 4 Total, 0 Used  
Memory in use: 4.0 GB Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

Workers (4)

Worker Id
worker-20180503113800-192.168.123.102-35802
worker-20180503113800-192.168.123.103-49913
worker-20180503113800-192.168.123.104-60481
worker-20180503113800-192.168.123.105-58298

Running Applications (0)

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

Completed Applications (0)

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

Spark Master at spark://hadoop04:7077

URL: spark://hadoop04:7077  
REST URL: spark://hadoop04:6066 (cluster mode)  
Alive Workers: 0  
Cores in use: 0.0 B Total, 0 Used  
Memory in use: 0.0 B Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: STANDBY

Workers (0)

Worker Id	Address
-----------	---------

Running Applications (0)

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

Completed Applications (0)

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

通过杀掉 active master 观察是否 hadoop04 能启动切换为 active 状态。

结果：

Spark Master at spark://hadoop04:7077

URL: spark://hadoop04:7077  
REST URL: spark://hadoop04:6066 (cluster mode)  
Alive Workers: 4  
Cores in use: 4.0 GB Total, 0 Used  
Memory in use: 4.0 GB Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

Workers (4)

Worker Id	Address	State	Cores	Memory
worker-20180503113800-192.168.123.102-35802	192.168.123.102:35802	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503113800-192.168.123.103-49913	192.168.123.103:49913	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503113800-192.168.123.104-60481	192.168.123.104:60481	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503113800-192.168.123.105-58298	192.168.123.105:58298	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications (0)

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

Completed Applications (0)

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

## 5.6.3、配置 Spark HistoryServer

详细步骤：

第一步:

```
cd /home/hadoop/apps/spark-2.3.0-bin-hadoop2.7/conf
```

```
cp spark-defaults.conf.template spark-defaults.conf
```

在文件里面添加如下内容:

```
spark.eventLog.enabled
```

```
true
```

```
spark.eventLog.dir
```

```
hdfs://myha01/sparklog
```

```
# Example:
# spark.master spark://master:7077
# spark.eventLog.enabled true
# spark.eventLog.dir hdfs://namenode:8021/directory
# spark.serializer org.apache.spark.serializer.KryoSerializer
# spark.driver.memory 5g
# spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"

spark.eventLog.enabled true
spark.eventLog.dir hdfs://myha01/sparklog
```

第二步:

在 spark-env.sh 的文件里面添加如下内容:

```
export SPARK_HISTORY_OPTS="-Dspark.history.ui.port=18080" -
```

```
Dspark.history.retainedApplications=30 -
```

```
Dspark.history.fs.logDirectory=hdfs://myha01/sparklog"
```

```
export JAVA_HOME=/usr/local/java/jdk1.8.0_73
export SPARK_MASTER_PORT=7077

export SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.deploy.zookeeper.url=hadoop02,hadoop03,hadoop04 -Dspark.deploy.zookeeper.dir=/spark"
export SPARK_HISTORY_OPTS="-Dspark.history.ui.port=18080 -Dspark.history.retainedApplications=30 -Dspark.history.fs.logDirectory=hdfs://myha01/sparklog"
```

第三步:

在启动 HistoServer 服务之前 hdfs://myha01/sparklog 目录要提前创建

```
hadoop fs -mkdir -p hdfs://myha01/sparklog
```

第四步: 启动 Spark HistoryServer

```
[hadoop@hadoop02 ~] $SPARK_HOME/sbin/start-history-server.sh
```

第五步: 访问 Spark History WebUI

```
http://hadoop02:18080/
```

## 6、Spark 的基本使用

### 6.1、执行第一个 Spark 程序

利用 Spark 自带的例子程序执行一个求 PI (蒙特卡洛算法) 的程序:

```
$SPARK_HOME/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master spark://hadoop02:7077 \
--executor-memory 512m \
```

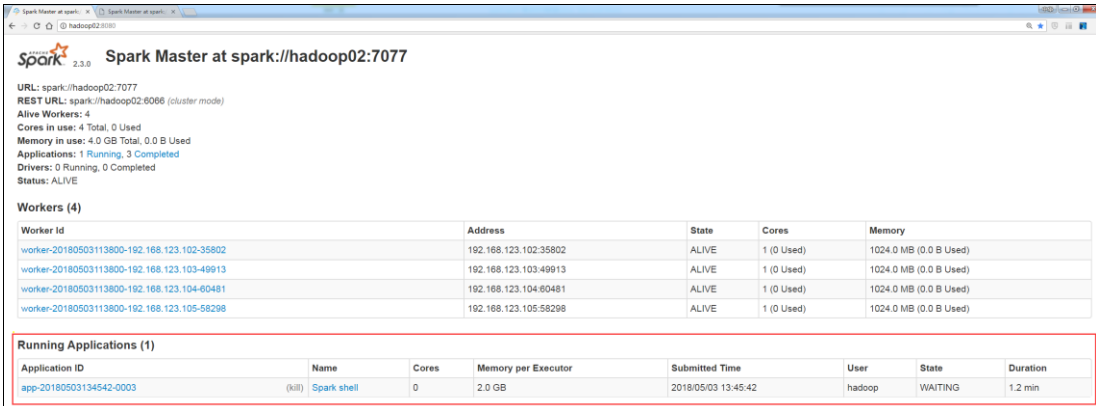
```
--total-executor-cores 2 \  
$SPARK_HOME/examples/jars/spark-examples_2.11-2.3.0.jar \  
100
```

## 6.2、启动 Spark Shell

启动命令：

```
$SPARK_HOME/bin/spark-shell \  
--master spark://hadoop02:7077,hadoop04:7077 \  
--executor-memory 512M \  
--total-executor-cores 2
```

```
[hadoop@hadoop05 ~]$ $SPARK_HOME/bin/spark-shell \  
> --master spark://hadoop02:7077 \  
> --executor-memory 2g \  
> --total-executor-cores 2  
2018-05-03 13:45:32 WARN NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using  
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://hadoop05:4040  
Spark context available as 'sc' (master = spark://hadoop02:7077, app id = app-20180503134542-0003).  
Spark session available as 'spark'.  
Welcome to  
 version 2.3.0  
using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)  
Type in expressions to have them evaluated.  
Type :help for more information.  
scala>
```



Spark Master at spark://hadoop02:7077

URL: spark://hadoop02:7077  
REST URL: spark://hadoop02:6066 (cluster mode)  
Alive Workers: 4  
Cores in use: 4 Total, 0 Used  
Memory in use: 4.0 GB Total, 0.0 B Used  
Applications: 1 Running, 3 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

Workers (4)

Worker Id	Address	State	Cores	Memory
worker-20180503113800-192.168.123.102-35802	192.168.123.102:35802	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503113800-192.168.123.103-49913	192.168.123.103:49913	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503113800-192.168.123.104-60481	192.168.123.104:60481	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20180503113800-192.168.123.105-58298	192.168.123.105:58298	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20180503134542-0003	(kill) Spark shell	0	2.0 GB	2018/05/03 13:45:42	hadoop	WAITING	1.2 min

注意上图中的 cores 参数，是 0，那么以后这个 spark shell 中运行的代码是不能执行成功的。千万注意。必要要把 cpu cores 和 memory 设置合理

- 1、executor memory 不能超过虚拟机的内存
- 2、cpu cores 不要超过 spark 集群能够提供的总 cpu cores，否则会使用全部。最好不要使用全部。否则其他程序由于没有 cpu core 可用，就不能正常运行

参数说明：

```
--master spark://hadoop02:7077    指定 Master 的地址  
--executor-memory 2G              指定每个 worker 可用内存为 2G  
--total-executor-cores 2          指定整个集群使用的 cup 核数为 2 个
```

注意：

如果启动 **spark shell** 时没有指定 **master** 地址，但是也可以正常启动 **spark shell** 和执行 **spark shell** 中的程序，其实是启动了 **spark** 的 **local** 模式，该模式仅在本机启动一个进程，没有与集群建立联系。

Spark-2.X:

Spark Shell 中已经默认将 **SparkContext** 类初始化为对象 **sc**。

Spark Shell 中已经默认将 **SparkSession** 类初始化为对象 **spark**。

用户代码如果需要用到，则直接应用 **sc**，**spark** 即可

Spark-1.X:

Spark Shell 中已经默认将 **SparkContext** 类初始化为对象 **sc**。

Spark Shell 中已经默认将 **SQLContext** 类初始化为对象 **sqlContext**。

用户代码如果需要用到，则直接应用 **sc**，**sqlContext** 即可

```
hadoop02 x | hadoop03 | hadoop04 | hadoop05
[hadoop@hadoop02 ~]$ apps/spark-1.6.3-bin-hadoop2.6/bin/spark-shell
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's repl log4j profile: org/apache/spark/log4j-defaults-repl.properties
To adjust logging level use sc.setLogLevel('INFO')
Welcome to

  Spark
  version 1.6.3

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.
18/05/04 08:48:15 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:48:16 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:48:19 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification
18/05/04 08:48:19 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
18/05/04 08:48:22 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
18/05/04 08:48:23 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
SQL context available as sqlContext.
```

## 6.3、在 Spark Shell 中编写 WordCount 程序

在提交 **WordCount** 程序之前，先在 **HDFS** 集群中的准备一个文件用于做单词统计：

**words.txt** 内容如下：

```
hello huangbo
hello xuzheng
hello wangbaoqiang
```

把该文件上传到 **HDFS** 文件系统中：

```
[hadoop@hadoop05 ~]$ hadoop fs -mkdir -p /spark/wc/input
[hadoop@hadoop05 ~]$ hadoop fs -put words.txt /spark/wc/input
```

在 **Spark Shell** 中提交 **WordCount** 程序：

```
sc.textFile("hdfs://myha01/spark/wc/input/words.txt").flatMap(_.split("
")).map(_._1).reduceByKey(_+_).saveAsTextFile("hdfs://myha01/spark/wc/output")
```

```
welcome to
SPARK version 2.3.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_73)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
scala> sc.textFile("hdfs://myha01/spark/wc/input/words.txt").flatMap(_.split(" ")).map(_._1).reduceByKey(_+_).foreach(println)
scala> sc.textFile("hdfs://myha01/spark/wc/input/words.txt").flatMap(_.split(" ")).map(_._1).reduceByKey(_+_).foreach(println)
scala> sc.textFile("hdfs://myha01/spark/wc/input/words.txt").flatMap(_.split(" ")).map(_._1).reduceByKey(_+_).saveAsTextFile("hdfs://myha01/spark/wc/output")
scala>
```

执行最后的结果：

```
[hadoop@hadoop05 conf]$ hadoop fs -cat hdfs://myha01/spark/wc/output/p*
(huangbo,1)
(hello,3)
(wangbaoqiang,1)
(xuzheng,1)
[hadoop@hadoop05 conf]$ hadoop fs -ls hdfs://myha01/spark/wc/output/
Found 3 items
-rw-r--r--  2 hadoop supergroup          0 2018-05-03 14:03 hdfs://myha01/spark/wc/output/_SUCCESS
-rw-r--r--  2 hadoop supergroup    22 2018-05-03 14:03 hdfs://myha01/spark/wc/output/part-00000
-rw-r--r--  2 hadoop supergroup    29 2018-05-03 14:03 hdfs://myha01/spark/wc/output/part-00001
[hadoop@hadoop05 conf]$
```

说明：

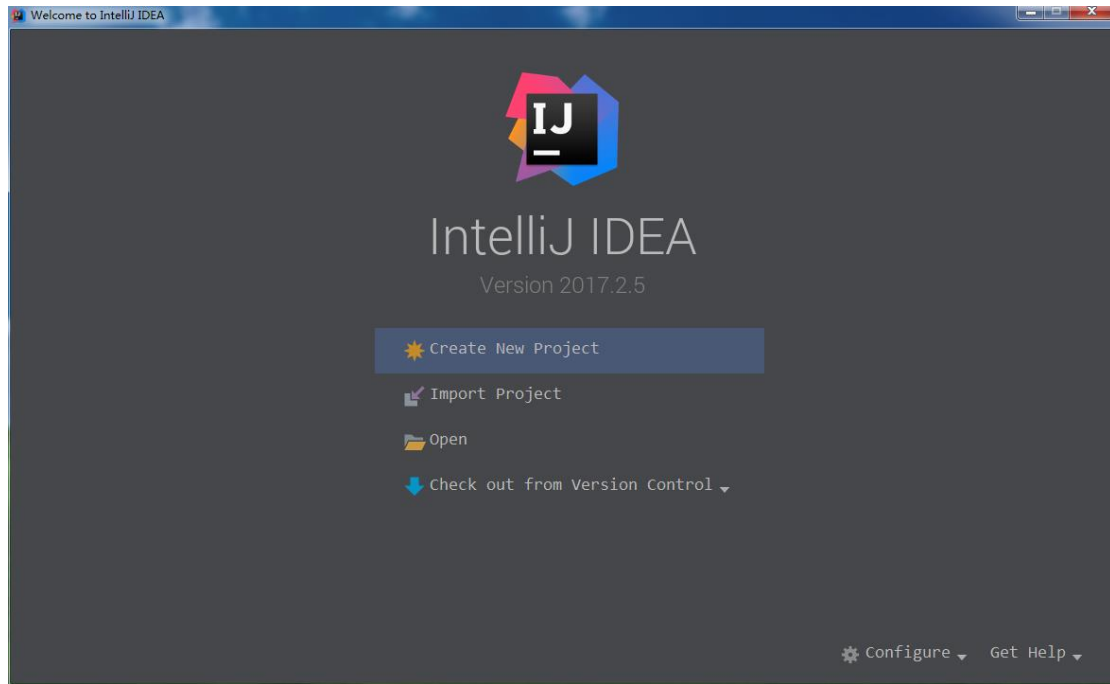
**sc** 是 **SparkContext** 对象，该对象是提交 **spark** 程序的入口  
**textFile("hdfs://myha01/spark/wc/input/words.txt")** 是从 **HDFS** 中读取数据  
**flatMap(\_.split(" "))** 先 **map** 在压平  
**map(\_.\_1)** 将单词和 1 构成元组  
**reduceByKey(\_+\_)** 按照 **key** 进行 **reduce**，并将 **value** 累加  
**saveAsTextFile("hdfs://myha01/spark/wc/output")** 将结果写入到 **HDFS** 中

## 6.4、在 IDEA 中编写 WordCount 程序

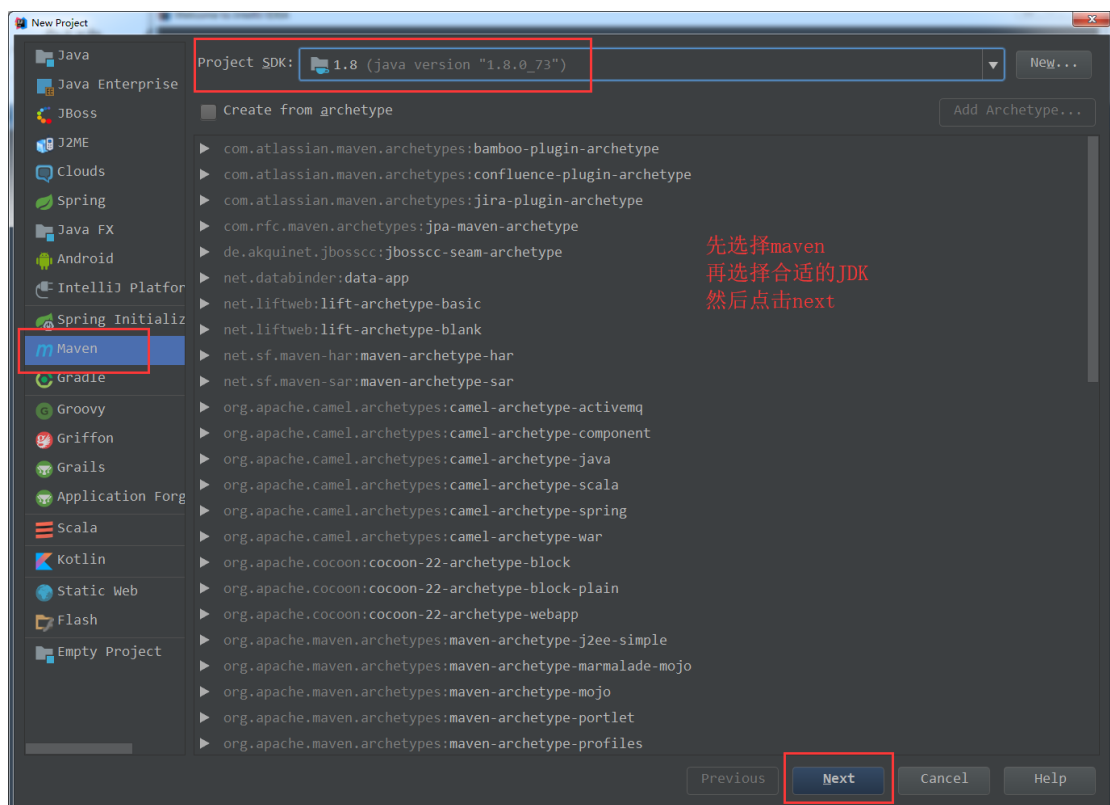
Spark Shell 仅在测试和验证我们的程序时使用的较多，在生产环境中，通常会在 IDEA 中编制程序，然后打成 jar 包，然后提交到集群，最常用的是创建一个 Maven 项目，利用 Maven 来管理 jar 包的依赖。

### 1、创建一个 IDEA 的 maven 项目

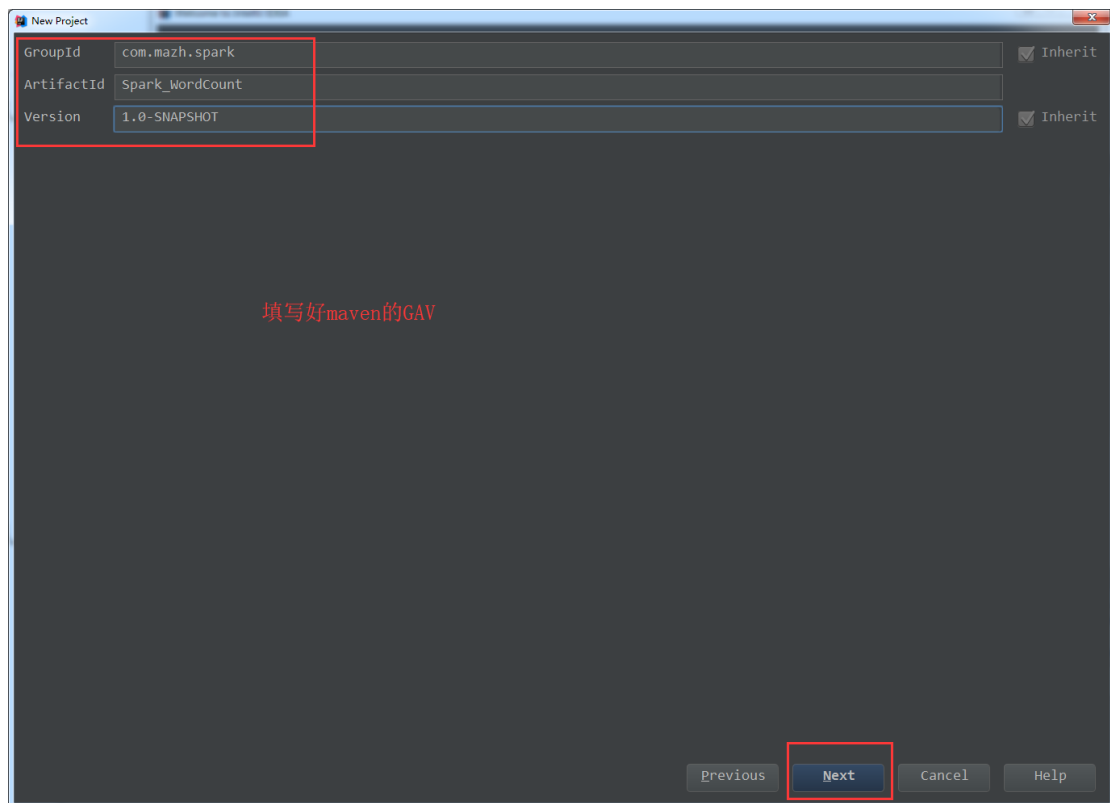




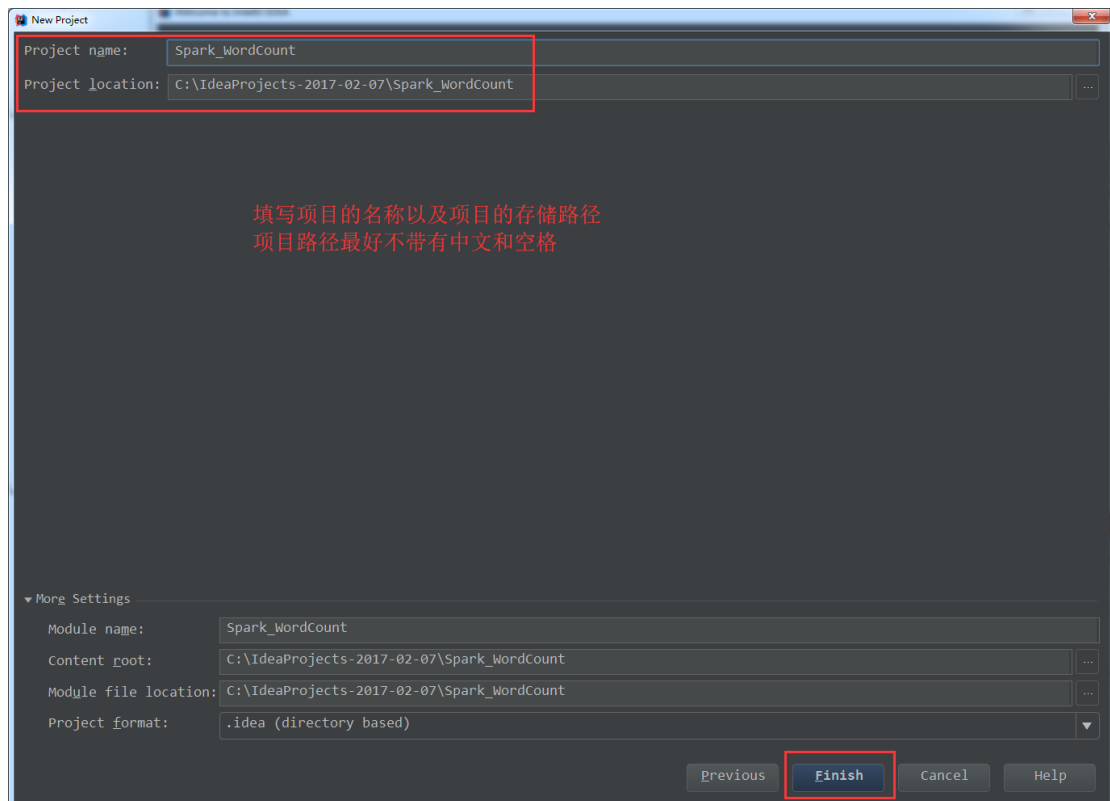
2、选择 Maven 项目，然后点击 next



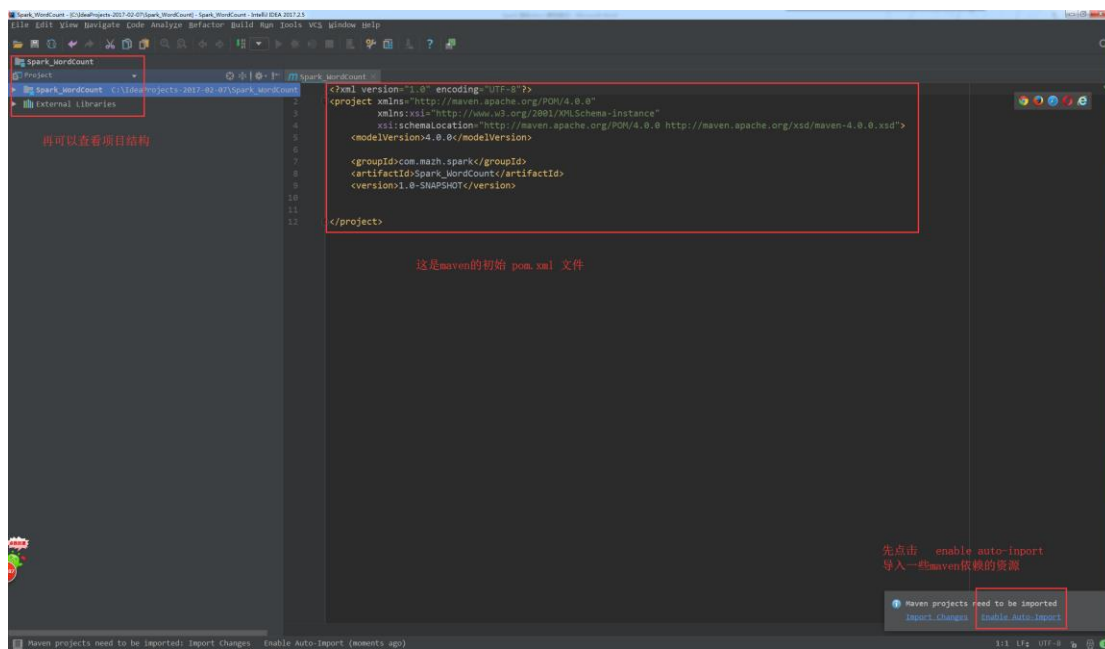
3、填写 maven 的 GAV，然后点击 next



4、填写项目名称，然后点击 finish



5、创建好 maven 项目后，点击 Enable Auto-Import



## 6、配置 maven 的 pom.xml 文件

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mazh.spark</groupId>
  <artifactId>Spark_WordCount</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <encoding>UTF-8</encoding>
    <scala.version>2.11.8</scala.version>
    <spark.version>2.3.0</spark.version>
    <hadoop.version>2.7.5</hadoop.version>
    <scala.compat.version>2.11</scala.compat.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.scala-lang</groupId>
      <artifactId>scala-library</artifactId>
      <version>${scala.version}</version>
    </dependency>
  </dependencies>

```

```
</dependency>

<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.11</artifactId>
  <version>${spark.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>${spark.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming_2.11</artifactId>
  <version>${spark.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-client</artifactId>
  <version>${hadoop.version}</version>
</dependency>
</dependencies>

<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>net.alchim31.maven</groupId>
        <artifactId>scala-maven-plugin</artifactId>
        <version>3.2.2</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
      </plugin>
    </plugins>
  </pluginManagement>
  <plugins>
    <plugin>
```

```

<groupId>net.alchim31.maven</groupId>
<artifactId>scala-maven-plugin</artifactId>
<executions>
  <execution>
    <id>scala-compile-first</id>
    <phase>process-resources</phase>
    <goals>
      <goal>add-source</goal>
      <goal>compile</goal>
    </goals>
  </execution>
  <execution>
    <id>scala-test-compile</id>
    <phase>process-test-resources</phase>
    <goals>
      <goal>testCompile</goal>
    </goals>
  </execution>
</executions>
</plugin>

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <executions>
    <execution>
      <phase>compile</phase>
      <goals>
        <goal>compile</goal>
      </goals>
    </execution>
  </executions>
</plugin>

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>2.4.3</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

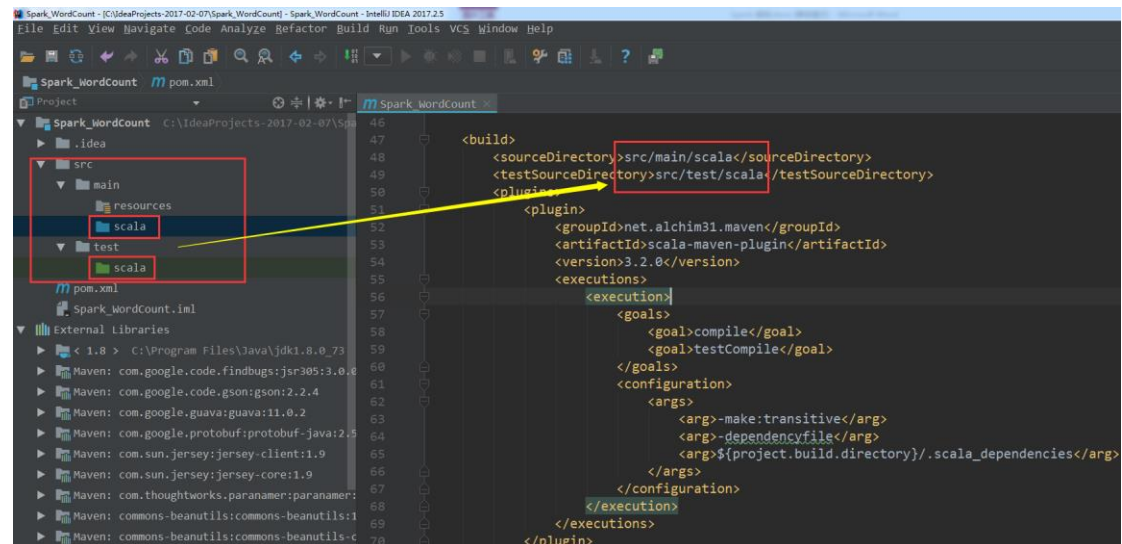
```

        <configuration>
            <filters>
                <filter>
                    <artifact>*:*/artifact>
                    <excludes>
                        <exclude>META-INF/*.SF</exclude>
                        <exclude>META-INF/*.DSA</exclude>
                        <exclude>META-INF/*.RSA</exclude>
                    </excludes>
                </filter>
            </filters>
        </configuration>
    </execution>
</executions>
</plugin>
</plugins>
</build>

</project>

```

7、将 `src/main/java` 和 `src/test/java` 分别修改成 `src/main/scala` 和 `src/test/scala`，与 `pom.xml` 中的配置保持一致



8、新建一个 Scala Class 类型为 Object，编写 WordCount 程序

```

package com.mazh.spark

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

object WordCount {

```



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

    // 创建一个SparkConf对象，并设置程序的名称
    val conf = new SparkConf().setAppName("WordCount")

    // 创建一个SparkContext对象
    val sc = new SparkContext(conf)

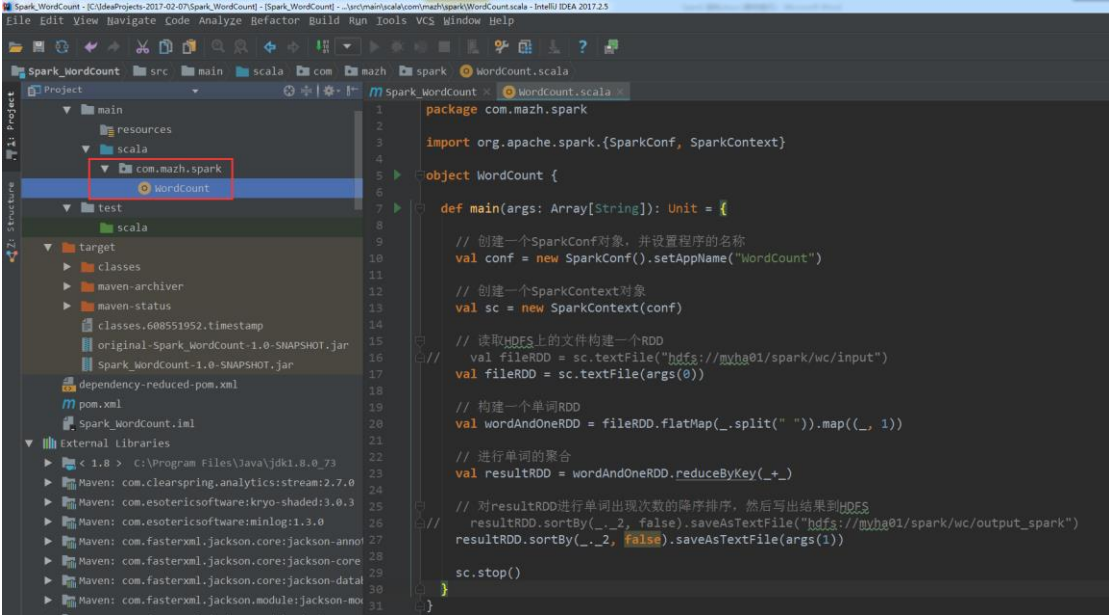
    // 读取HDFS上的文件构建一个RDD
    val fileRDD = sc.textFile(args(0))

    // 构建一个单词RDD
    val wordAndOneRDD = fileRDD.flatMap(_.split(" ")).map((_, 1))

    // 进行单词的聚合
    val resultRDD = wordAndOneRDD.reduceByKey(_+_ )

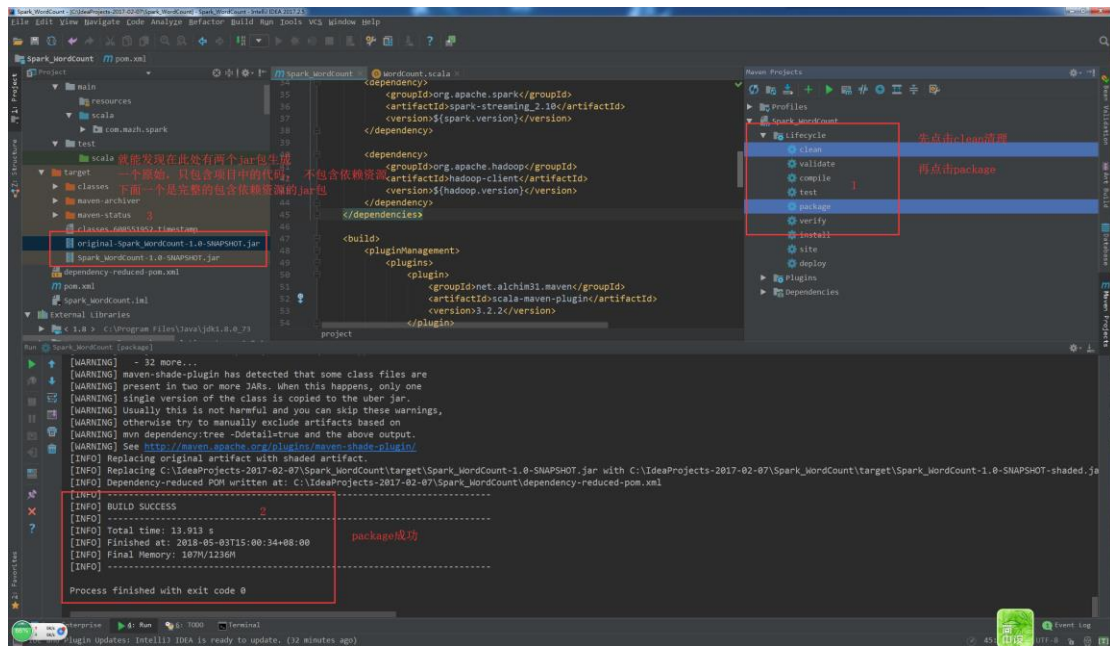
    // 对resultRDD进行单词出现次数的降序排序，然后写出结果到HDFS
    resultRDD.sortBy(_._2, false).saveAsTextFile(args(1))

    sc.stop()
}
}
```

The screenshot shows an IDE window with two panes. The left pane displays the project structure of 'Spark\_WordCount'. It shows a 'main' directory containing 'resources', 'scala', and 'test' subdirectories. The 'scala' directory contains a 'com.mazh.spark' package, which includes a 'WordCount' class. The right pane shows the source code of 'WordCount.scala'. The code defines a 'main' method that takes an array of strings as input. It creates a 'SparkConf' object with the application name 'WordCount', then a 'SparkContext' object. It reads a file from HDFS (specified by 'args(0)') into an RDD, splits the words, and maps each word to a tuple (word, 1). These are then reduced by key to get the word counts. The results are sorted by count in descending order and saved to HDFS (specified by 'args(1)'). Finally, the 'SparkContext' is stopped.

9、使用 maven 进行打包

点击右侧的 maven project 选项。先点击 clean 再点击 package 进行打包



## 10、启动 HDFS 集群和 Spark 集群

### 启动操作略

## 11、上传打好的 jar 包到 spark 集群中的用来提交任务的节点

**put c:/Spark\_WordCount-1.0-SNAPSHOT.jar**

执行命令：

```
$SPARK_HOME/bin/spark-submit \
--class com.mazh.spark.WordCount \
--master spark://hadoop02:7077 \
--executor-memory 512m \
--total-executor-cores 4 \
/home/hadoop/Spark_WordCount-1.0-SNAPSHOT.jar \
hdfs://myha01/spark/wc/input \
hdfs://myha01/spark/wc/output_11
```

## 12、验证结果

```
2018-05-03 15:13:45 INFO MapOutputTrackerMasterEndpoint:54 - Asked to send map output locations for shuffle 1 to 192.168.123.105:56930
2018-05-03 15:13:45 INFO TaskSetManager:54 - Finished task 1.0 in stage 4.0 (TID 6) in 562 ms on 192.168.123.102 (executor 0) (1/2)
2018-05-03 15:13:45 INFO TaskSetManager:54 - Finished task 0.0 in stage 4.0 (TID 7) in 617 ms on 192.168.123.105 (executor 1) (2/2)
2018-05-03 15:13:45 INFO TaskSchedulerImpl:54 - Removed TaskSet 4.0, whose tasks have all completed, from pool
2018-05-03 15:13:45 INFO DAGScheduler:54 - ResultStage 4 (runJob at SparkHadoopWriter.scala:78) finished in 0.668 s
2018-05-03 15:13:45 INFO DAGScheduler:54 - Job 1 finished: runJob at SparkHadoopWriter.scala:78, took 0.816621 s
2018-05-03 15:13:45 INFO SparkHadoopWriter:54 - Job job_20180503151344_0010 committed.
2018-05-03 15:13:45 INFO AbstractConnector:318 - stopped spark@72/c4c0r[http://1.1][http://1.1][0.0.0.0:4041]
2018-05-03 15:13:45 INFO sparkUI:54 - Stopped spark web UI at http://hadoop05:4041
2018-05-03 15:13:45 INFO StandaloneSchedulerBackend:54 - Shutting down all executors
2018-05-03 15:13:45 INFO CoarseGrainedSchedulerBackend$DriverEndpoint:54 - Asking each executor to shut down
2018-05-03 15:13:45 INFO MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMasterEndpoint stopped!
2018-05-03 15:13:45 INFO MemoryStore:54 - MemoryStore cleared
2018-05-03 15:13:45 INFO BlockManager:54 - BlockManager stopped
2018-05-03 15:13:45 INFO BlockManagerMaster:54 - BlockManagerMaster stopped
2018-05-03 15:13:45 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 - outputCommitCoordinator stopped!
2018-05-03 15:13:45 INFO sparkContext:54 - Successfully stopped sparkContext
2018-05-03 15:13:45 INFO ShutdownHookManager:54 - Shutdown hook called
2018-05-03 15:13:45 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-7f537a1b-368f-4218-b28b-7e8c6f666654
2018-05-03 15:13:45 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-80bf20b8-73b0-4e3a-b13f-86bf08b889f4
[hadoop@hadoop05 ~]$ hadoop fs -ls hdfs://myha01/spark/wc/output_11
Found 3 items
-rw-r--r-- 2 hadoop supergroup 0 2018-05-03 15:13 hdfs://myha01/spark/wc/output_11/_SUCCESS
-rw-r--r-- 2 hadoop supergroup 10 2018-05-03 15:13 hdfs://myha01/spark/wc/output_11/part-00000
-rw-r--r-- 2 hadoop supergroup 41 2018-05-03 15:13 hdfs://myha01/spark/wc/output_11/part-00001
[hadoop@hadoop05 ~]$ hadoop fs -cat hdfs://myha01/spark/wc/output_11/p*
(hello,3)
(wangbaoliang,1)
(xuzheng,1)
(huanbo,1)
[hadoop@hadoop05 ~]$
```

## 7、修改 Spark 的日志级别

### 7.1、永久修改

从我们运行的 spark 程序运行的情况来看，可以看到大量的 INFO 级别的日志信息。淹没了我们需要运行输出结果。可以通过修改 Spark 配置文件来 Spark 日志级别。

以下是详细步骤：

第一步：先进入 conf 目录

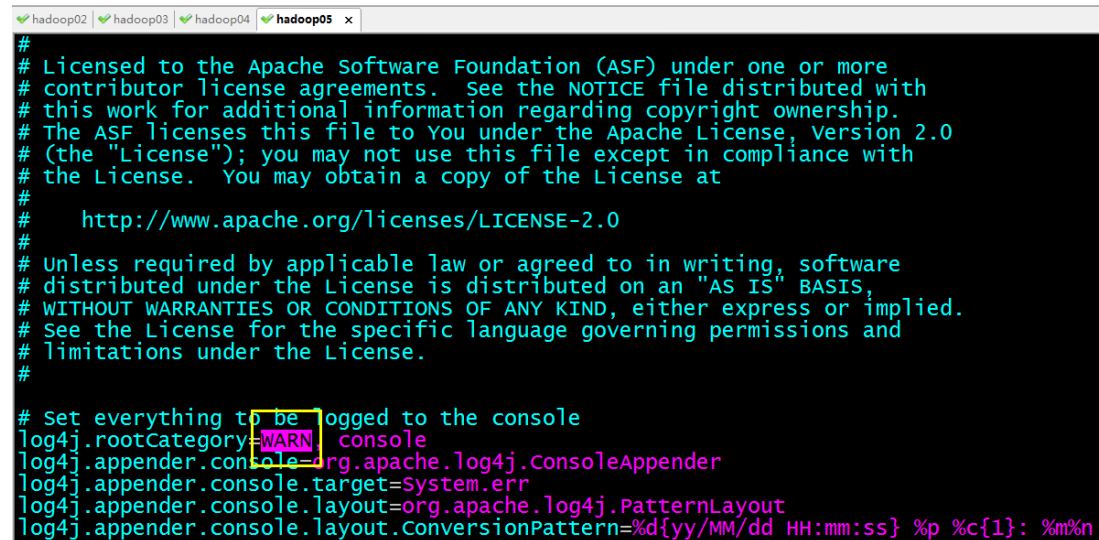
```
[hadoop@hadoop05 conf]$ cd $SPARK_HOME/conf
```

第二步：准备 log4j.properties

```
[hadoop@hadoop05 conf]$ cp log4j.properties.template log4j.properties
```

第三步：配置日志级别：

把 INFO 改成你想要的级别：主要有 ERROR, WARN, INFO, DEBUG 几种



```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Set everything to be logged to the console
log4j.rootCategory=WARN console
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
```

## 8、Spark 的 WordCount

### 8.1、Scala 版本的 WordCount

```
package com.mazh.spark
```

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

```

/**
 * 作者: 马中华: http://blog.csdn.net/zhongqi2513
 * 日期: 2018 年 4 月 22 日 上午 8:30:47
 *
 * 描述: Scala 版本的 WordCount
 */
object WordCount {

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

        // 创建一个 SparkConf 对象, 并设置程序的名称
        val conf = new SparkConf().setAppName("WordCount")
        conf.setMaster("local")

        // 创建一个 SparkContext 对象
        val sc = new SparkContext(conf)

        // 读取 HDFS 上的文件构建一个 RDD
        val fileRDD = sc.textFile("hdfs://myha01/spark/wc/input")
        // val fileRDD = sc.textFile(args(0))

        // 构建一个单词 RDD
        val wordAndOneRDD = fileRDD.flatMap(_.split(" ")).map((_, 1))

        // 进行单词的聚合
        val resultRDD = wordAndOneRDD.reduceByKey(_+_ )

        // 对 resultRDD 进行单词出现次数的降序排序, 然后写出结果到 HDFS
        resultRDD.sortBy(_._2,
false).saveAsTextFile("hdfs://myha01/spark/wc/output_spark33")
        // resultRDD.sortBy(_._2, false).saveAsTextFile(args(1))

        sc.stop()
    }
}

```

## 8.2、Java7 版本的 WordCount

```

package com.mazh.spark.wc;

import org.apache.spark.SparkConf;

```

```

import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;
import scala.Tuple2;

import java.util.Arrays;
import java.util.Iterator;

/**
 * 作者: 马中华; http://blog.csdn.net/zhongqi2513
 * 日期: 2018 年 04 月 22 日 上午 8:49:10
 *
 * 描述: Java 版本的 WordCount
 */
public class JavaWordCount {
    public static void main(String[] args){
        if(args.length!=2){
            System.out.println("Usage:JavaWordCount<input><output>");
            System.exit(1);
        }
        SparkConf conf = new SparkConf();
        conf.setMaster("local");
        conf.setAppName(JavaWordCount.class.getSimpleName());
        JavaSparkContext jsc = new JavaSparkContext(conf);
        JavaRDD<String> line = jsc.textFile(args[0]);
        //切割压平 flatMap() 两个参数, 一个输入类型, 一个输出类型
        JavaRDD<String> jrdd1 = line.flatMap(new FlatMapFunction<String, String>() {
            @Override
            public Iterator<String> call(String s) throws Exception {
                //该方法的返回值类型是 Iterator, 需要把 Array 类型的结果转换为迭代器类型的
                return Arrays.asList(s.split(" ")).iterator();
            }
        });
        //和 1 组合成元组 mapToPair() 第一个参数, 输入数据类型, 第二个参数是元组的 key 类型, 第三个参数是元组的 value 类型
        JavaPairRDD<String, Integer> javaPairRDD = jrdd1.mapToPair(new
        PairFunction<String, String, Integer>() {
            @Override
            public Tuple2<String, Integer> call(String s) throws Exception {
                return new Tuple2<String, Integer>(s, 1);
            }
        });
    }
}

```

```

    }
    });
    //分组合并 reduceByKey() (a,b)=>a+b 第三个参数: 返回值的类型
    JavaPairRDD<String, Integer> result = javaPairRDD.reduceByKey(new
Function2<Integer, Integer, Integer>() {
        @Override
        public Integer call(Integer v1, Integer v2) throws Exception {
            return v1 + v2;
        }
    });
    //先在本地图试一下
    /* result.foreach(new VoidFunction<Tuple2<String, Integer>>() {
        @Override
        public void call(Tuple2<String, Integer> tuple) throws Exception {
            System.out.println(tuple);
        }
    });*/
    //可以进行排序
    JavaPairRDD<Integer, String> res1 = result.mapToPair(new
PairFunction<Tuple2<String, Integer>, Integer, String>() {
        @Override
        public Tuple2<Integer, String> call(Tuple2<String, Integer> t) throws
Exception {
            return t.swap();
        }
    });
    //排序, 默认是升序, 如果需要降序, 参数 false
    JavaPairRDD<Integer, String> res2 = res1.sortByKey(false);
    JavaPairRDD<String, Integer> finalRes = res2.mapToPair(new
PairFunction<Tuple2<Integer, String>, String, Integer>() {
        @Override
        public Tuple2<String, Integer> call(Tuple2<Integer, String> t) throws
Exception {
            return t.swap();
        }
    });
    //保存
    finalRes.saveAsTextFile(args[1]);
    //释放资源
    jsc.close();
}
}

```



## 8.3、Java8 Lambda 表达式版本的 WordCount

```
package com.mazh.spark.wc;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import scala.Tuple2;

import java.util.Arrays;

/**
 * 作者: 马中华: http://blog.csdn.net/zhongqi2513
 * 日期: 2018 年 04 月 22 日 上午 9:27:17
 *
 * 描述: Java 版本的 WordCount -- 使用 Lambda 表达式
 */
public class JavaLambdaWordCount {
    public static void main(String[] args){
        if(args.length!=2){
            System.out.println("Usage JavaLambdaWordCount<input><output>");
            System.exit(1);
        }

        SparkConf conf = new SparkConf();
        conf.setMaster("local");
        conf.setAppName(JavaLambdaWordCount.class.getSimpleName());
        JavaSparkContext jsc = new JavaSparkContext(conf);
        // 读取数据
        JavaRDD<String> jrdd = jsc.textFile(args[0]);
        // 切割压平
        JavaRDD<String> jrdd2 = jrdd.flatMap(t -> Arrays.asList(t.split("
")).iterator());
        // 和 1 组合
        JavaPairRDD<String, Integer> jprdd = jrdd2.mapToPair(t -> new
        Tuple2<String, Integer>(t, 1));
        // 分组聚合
        JavaPairRDD<String, Integer> res = jprdd.reduceByKey((a, b) -> a + b);
        // 保存
        res.saveAsTextFile(args[1]);
        // 释放资源
        jsc.close();
    }
}
```

