Spark入门

[1. 课程目标](#_Toc22790)

[1.1. 目标1：熟悉Spark相关概念](#_Toc18985)

[1.2. 目标2：搭建Spark集群](#_Toc27759)

[1.3. 目标3：编写简单的Spark应用程序](#_Toc8322)

[2. Spark概述](#_Toc32712)

[2.1. 什么是Spark（官网：http://spark.apache.org）](#_Toc19173)

[2.2. 为什么要学Spark](#_Toc23663)

[2.3. Spark特点](#_Toc1566)

[2.3.1. 快](#_Toc2395)

[2.3.2. 易用](#_Toc27960)

[2.3.3. 通用](#_Toc28050)

[2.3.4. 兼容性](#_Toc19107)

[3. Spark集群安装](#_Toc13118)

[3.1. 安装](#_Toc27509)

[3.1.1. 机器部署](#_Toc22634)

[3.1.2. 下载Spark安装包](#_Toc19885)

[3.1.3. 配置Spark](#_Toc691)

[4. 执行Spark程序](#_Toc2470)

[4.1. 执行第一个spark程序](#_Toc5486)

[4.2. 启动Spark Shell](#_Toc6177)

[4.2.1. 启动spark shell](#_Toc17940)

[4.2.2. 在spark shell中编写WordCount程序](#_Toc24968)

[4.3. 在IDEA中编写WordCount程序](#_Toc2290)

# 课程目标

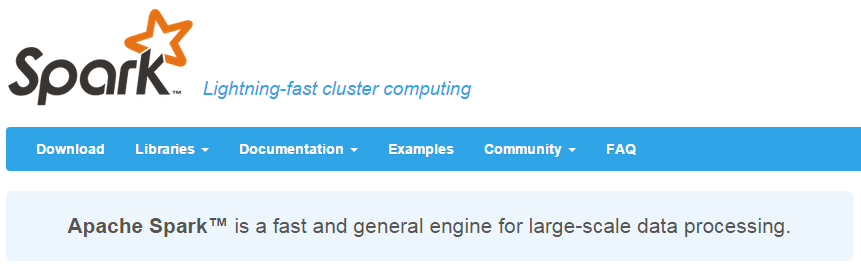
## 目标1：熟悉Spark相关概念

## 目标2：搭建Spark集群

## 目标3：编写简单的Spark应用程序

# Spark概述

## 什么是Spark（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。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SparkSQL、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。

## 为什么要学Spark

中间结果输出：基于MapReduce的计算引擎通常会将中间结果输出到磁盘上，进行存储和容错。出于任务管道承接的，考虑，当一些查询翻译到MapReduce任务时，往往会产生多个Stage，而这些串联的Stage又依赖于底层文件系统（如HDFS）来存储每一个Stage的输出结果

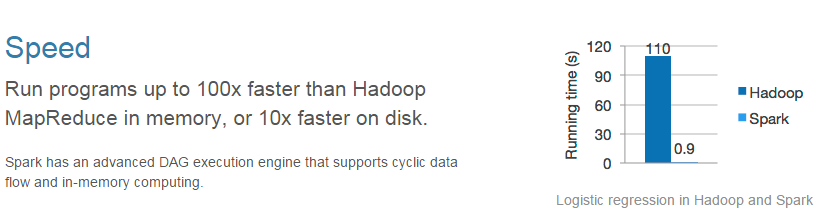
|  |  |
| --- | --- |
| **Hadoop** | **Spark** |
|  |  |

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。

## Spark特点

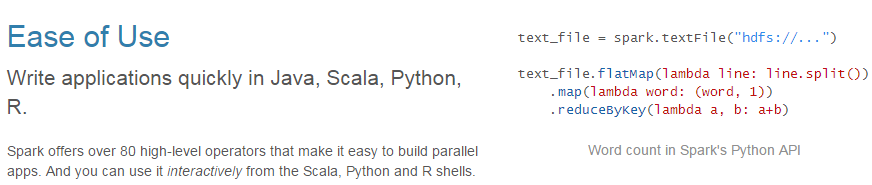
### 快

与Hadoop的MapReduce相比，Spark基于内存的运算要快100倍以上，基于硬盘的运算也要快10倍以上。Spark实现了高效的DAG执行引擎，可以通过基于内存来高效处理数据流。



### 易用

Spark支持Java、Python和Scala的API，还支持超过80种高级算法，使用户可以快速构建不同的应用。而且Spark支持交互式的Python和Scala的shell，可以非常方便地在这些shell中使用Spark集群来验证解决问题的方法。

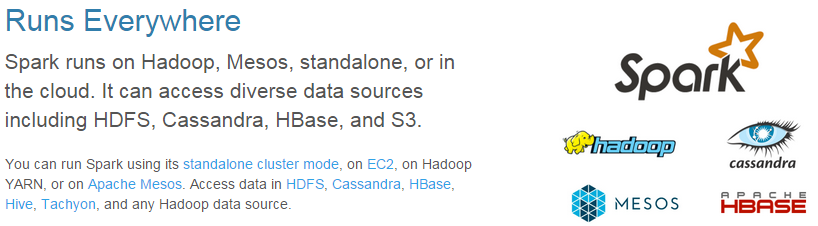


### 通用

Spark提供了统一的解决方案。Spark可以用于批处理、交互式查询（Spark SQL）、实时流处理（Spark Streaming）、机器学习（Spark MLlib）和图计算（GraphX）。这些不同类型的处理都可以在同一个应用中无缝使用。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。

### 兼容性

Spark可以非常方便地与其他的开源产品进行融合。比如，Spark可以使用Hadoop的YARN和Apache Mesos作为它的资源管理和调度器，器，并且可以处理所有Hadoop支持的数据，包括HDFS、HBase和Cassandra等。这对于已经部署Hadoop集群的用户特别重要，因为不需要做任何数据迁移就可以使用Spark的强大处理能力。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。此外，Spark还提供了在EC2上部署Standalone的Spark集群的工具。



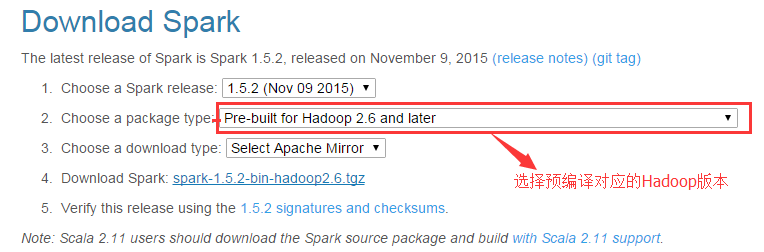
# Spark集群安装

## 安装

### 机器部署

准备两台以上Linux服务器，安装好JDK1.7

### 下载Spark安装包



http://www.apache.org/dyn/closer.lua/spark/spark-1.5.2/spark-1.5.2-bin-hadoop2.6.tgz

上传解压安装包

上传spark-1.5.2-bin-hadoop2.6.tgz安装包到Linux上

解压安装包到指定位置

tar -zxvf spark-1.5.2-bin-hadoop2.6.tgz -C /usr/local

### 配置Spark

进入到Spark安装目录

cd /usr/local/spark-1.5.2-bin-hadoop2.6

进入conf目录并重命名并修改spark-env.sh.template文件

cd conf/

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

vi spark-env.sh

在该配置文件中添加如下配置

export JAVA\_HOME=/usr/java/jdk1.7.0\_45

export SPARK\_MASTER\_IP=node1.itcast.cn

export SPARK\_MASTER\_PORT=7077

保存退出

重命名并修改slaves.template文件

mv slaves.template slaves

vi slaves

在该文件中添加子节点所在的位置（Worker节点）

node2.itcast.cn

node3.itcast.cn

node4.itcast.cn

保存退出

将配置好的Spark拷贝到其他节点上

scp -r spark-1.5.2-bin-hadoop2.6/ node2.itcast.cn:/usr/local/

scp -r spark-1.5.2-bin-hadoop2.6/ node3.itcast.cn:/usr/local/

scp -r spark-1.5.2-bin-hadoop2.6/ node4.itcast.cn:/usr/local/

Spark集群配置完毕，目前是1个Master，3个Work，在node1.itcast.cn上启动Spark集群

/usr/local/spark-1.5.2-bin-hadoop2.6/sbin/start-all.sh

启动后执行jps命令，主节点上有Master进程，其他子节点上有Work进行，登录Spark管理界面查看集群状态（主节点）：<http://node1.itcast.cn:8080/>



到此为止，Spark集群安装完毕，但是有一个很大的问题，那就是Master节点存在单点故障，要解决此问题，就要借助zookeeper，并且启动至少两个Master节点来实现高可靠，配置方式比较简单：

Spark集群规划：node1，node2是Master；node3，node4，node5是Worker

安装配置zk集群，并启动zk集群

停止spark所有服务，修改配置文件spark-env.sh，在该配置文件中删掉SPARK\_MASTER\_IP并添加如下配置

export SPARK\_DAEMON\_JAVA\_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.deploy.zookeeper.url=zk1,zk2,zk3 -Dspark.deploy.zookeeper.dir=/spark"

1.在node1节点上修改slaves配置文件内容指定worker节点

2.在node1上执行**sbin/start-all.sh**脚本，然后在node2上执行**sbin/start-master.sh**启动第二个Master

# 执行Spark程序

## 执行第一个spark程序

/usr/local/spark-1.5.2-bin-hadoop2.6/bin/spark-submit \

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

--master spark://node1.itcast.cn:7077 \

--executor-memory 1G \

--total-executor-cores 2 \

/usr/local/spark-1.5.2-bin-hadoop2.6/lib/spark-examples-1.5.2-hadoop2.6.0.jar \

100

该算法是利用蒙特·卡罗算法求PI

## 启动Spark Shell

spark-shell是Spark自带的交互式Shell程序，方便用户进行交互式编程，用户可以在该命令行下用scala编写spark程序。

### 启动spark shell

/usr/local/spark-1.5.2-bin-hadoop2.6/bin/spark-shell \

--master spark://node1.itcast.cn:7077 \

--executor-memory 2g \

--total-executor-cores 2

参数说明：

--master spark://node1.itcast.cn: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 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

### 在spark shell中编写WordCount程序

1. 首先启动hdfs
2. 向hdfs上传一个文件到hdfs://node1.itcast.cn:9000/words.txt
3. 在spark shell中用scala语言编写spark程序

sc.textFile("hdfs://node1.itcast.cn:9000/words.txt").flatMap(\_.split(" "))

.map((\_,1)).reduceByKey(\_+\_).saveAsTextFile("hdfs://node1.itcast.cn:9000/out")

1. 使用hdfs命令查看结果

hdfs dfs -ls hdfs://node1.itcast.cn:9000/out/p\*

说明：

sc是SparkContext对象，该对象时提交spark程序的入口

textFile(hdfs://node1.itcast.cn:9000/words.txt)是hdfs中读取数据

flatMap(\_.split(" "))先map在压平

map((\_,1))将单词和1构成元组

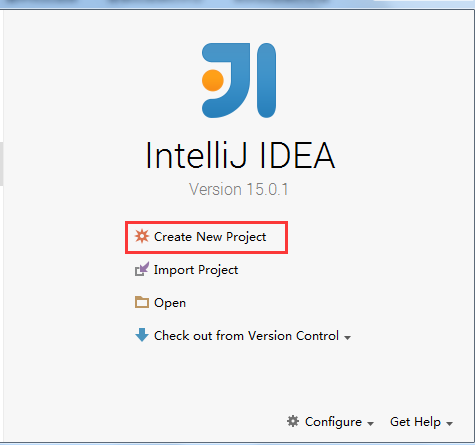
reduceByKey(\_+\_)按照key进行reduce，并将value累加

saveAsTextFile("hdfs://node1.itcast.cn:9000/out")将结果写入到hdfs中

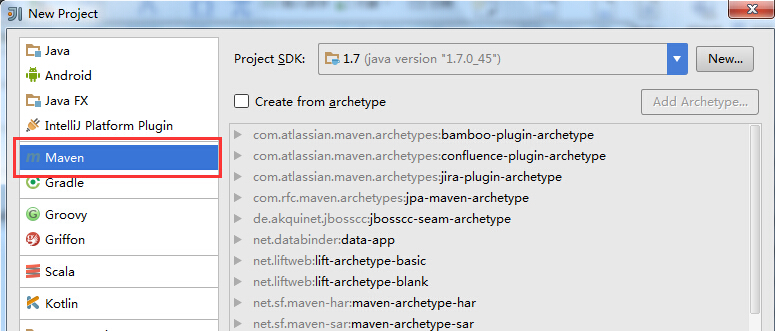
## 在IDEA中编写WordCount程序

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IDE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。

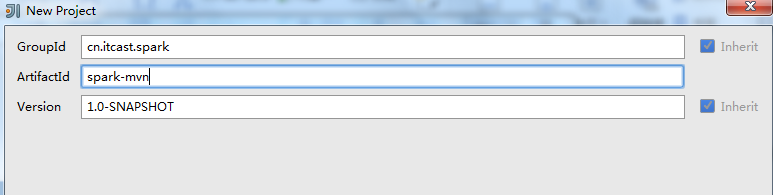
1.创建一个项目



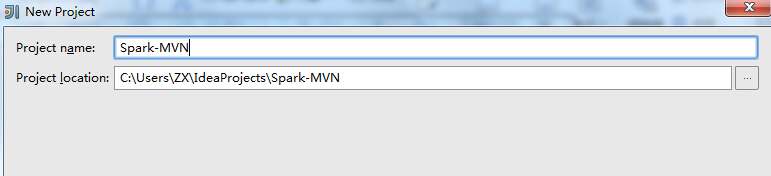
2.选择Maven项目，然后点击next



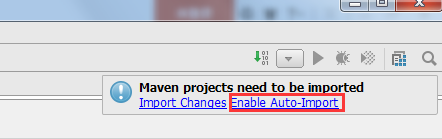
3.填写maven的GAV，然后点击next



1. 填写项目名称，然后点击finish



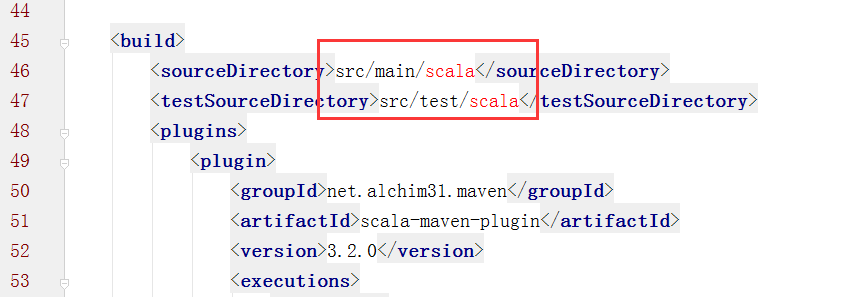
5.创建好maven项目后，点击Enable Auto-Import

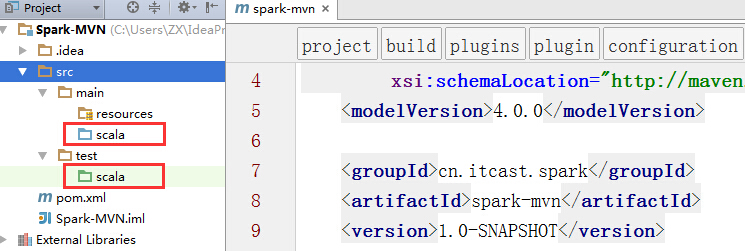


1. 配置Maven的pom.xml

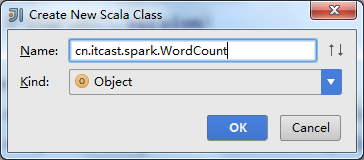
|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>cn.itcast.spark</**groupId**>  <**artifactId**>spark-mvn</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>   <**properties**>  <**maven.compiler.source**>1.7</**maven.compiler.source**>  <**maven.compiler.target**>1.7</**maven.compiler.target**>  <**encoding**>UTF-8</**encoding**>  <**scala.version**>2.10.6</**scala.version**>  <**scala.compat.version**>2.10</**scala.compat.version**>  </**properties**>   <**dependencies**>  <**dependency**>  <**groupId**>org.scala-lang</**groupId**>  <**artifactId**>scala-library</**artifactId**>  <**version**>${scala.version}</**version**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-core\_2.10</**artifactId**>  <**version**>1.5.2</**version**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.spark</**groupId**>  <**artifactId**>spark-streaming\_2.10</**artifactId**>  <**version**>1.5.2</**version**>  </**dependency**>   <**dependency**>  <**groupId**>org.apache.hadoop</**groupId**>  <**artifactId**>hadoop-client</**artifactId**>  <**version**>2.6.2</**version**>  </**dependency**>  </**dependencies**>   <**build**>  <**sourceDirectory**>src/main/scala</**sourceDirectory**>  <**testSourceDirectory**>src/test/scala</**testSourceDirectory**>  <**plugins**>  <**plugin**>  <**groupId**>net.alchim31.maven</**groupId**>  <**artifactId**>scala-maven-plugin</**artifactId**>  <**version**>3.2.0</**version**>  <**executions**>  <**execution**>  <**goals**>  <**goal**>compile</**goal**>  <**goal**>testCompile</**goal**>  </**goals**>  <**configuration**>  <**args**>  <**arg**>-make:transitive</**arg**>  <**arg**>-dependencyfile</**arg**>  <**arg**>${project.build.directory}/.scala\_dependencies</**arg**>  </**args**>  </**configuration**>  </**execution**>  </**executions**>  </**plugin**>  <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-surefire-plugin</**artifactId**>  <**version**>2.18.1</**version**>  <**configuration**>  <**useFile**>false</**useFile**>  <**disableXmlReport**>true</**disableXmlReport**>  <**includes**>  <**include**>\*\*/\*Test.\*</**include**>  <**include**>\*\*/\*Suite.\*</**include**>  </**includes**>  </**configuration**>  </**plugin**>   <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-shade-plugin</**artifactId**>  <**version**>2.3</**version**>  <**executions**>  <**execution**>  <**phase**>package</**phase**>  <**goals**>  <**goal**>shade</**goal**>  </**goals**>  <**configuration**>  <**filters**>  <**filter**>  <**artifact**>\*:\*</**artifact**>  <**excludes**>  <**exclude**>META-INF/\*.SF</**exclude**>  <**exclude**>META-INF/\*.DSA</**exclude**>  <**exclude**>META-INF/\*.RSA</**exclude**>  </**excludes**>  </**filter**>  </**filters**>  <**transformers**>  <**transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"**>  <**mainClass**>cn.itcast.spark.WordCount</**mainClass**>  </**transformer**>  </**transformers**>  </**configuration**>  </**execution**>  </**executions**>  </**plugin**>  </**plugins**>  </**build**> </**project**> |

1. 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





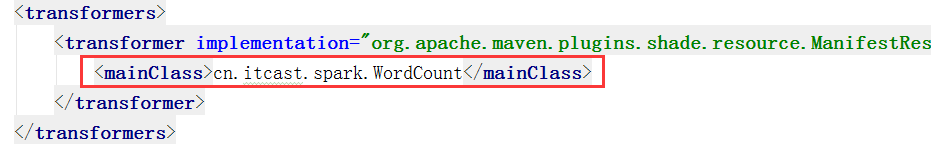
1. 新建一个scala class，类型为Object



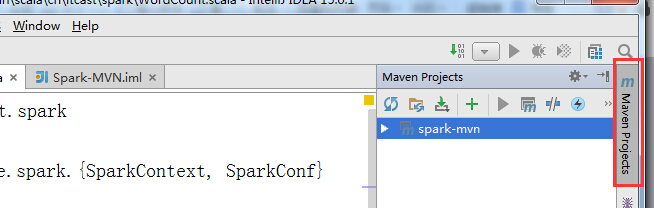
1. 编写spark程序

|  |
| --- |
| **package** cn.itcast.spark  **import** org.apache.spark.{SparkContext, SparkConf}  **object** WordCount {  **def** main(args: Array[String]) {  *//创建SparkConf()并设置App名称* **val** conf = **new** SparkConf().setAppName(**"WC"**)  *//创建SparkContext，该对象是提交spark App的入口* **val** sc = **new** SparkContext(conf)  *//使用sc创建RDD并执行相应的transformation和action* sc.textFile(args(0)).flatMap(\_.split(**" "**)).map((\_, 1)).reduceByKey(\_+\_, 1).sortBy(\_.\_2, **false**).saveAsTextFile(args(1))  *//停止sc，结束该任务* sc.stop()  } } |

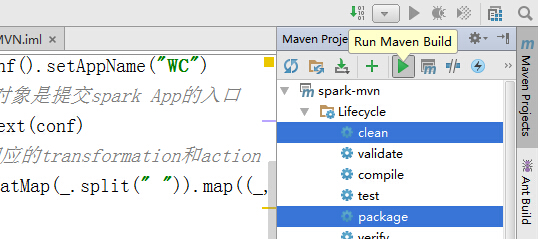
1. 使用Maven打包：首先修改pom.xml中的main class



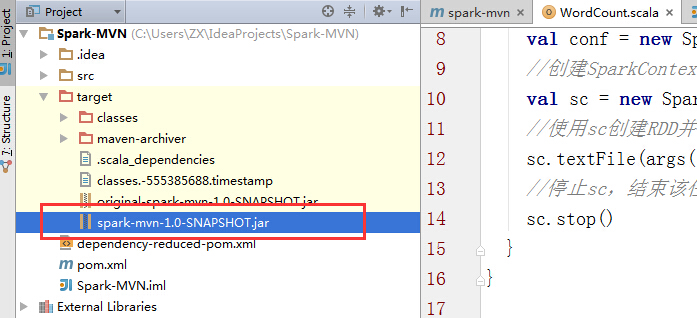
点击idea右侧的Maven Project选项



点击Lifecycle,选择clean和package，然后点击Run Maven Build



1. 选择编译成功的jar包，并将该jar上传到Spark集群中的某个节点上



1. 首先启动hdfs和Spark集群

启动hdfs

/usr/local/hadoop-2.6.1/sbin/start-dfs.sh

启动spark

/usr/local/spark-1.5.2-bin-hadoop2.6/sbin/start-all.sh

1. 使用spark-submit命令提交Spark应用（注意参数的顺序）

/usr/local/spark-1.5.2-bin-hadoop2.6/bin/spark-submit \

--class cn.itcast.spark.WordCount \

--master spark://node1.itcast.cn:7077 \

--executor-memory 2G \

--total-executor-cores 4 \

/root/spark-mvn-1.0-SNAPSHOT.jar \

hdfs://node1.itcast.cn:9000/words.txt \

hdfs://node1.itcast.cn:9000/out

查看程序执行结果

hdfs dfs -cat hdfs://node1.itcast.cn:9000/out/part-00000

(hello,6)

(tom,3)

(kitty,2)

(jerry,1)