Spark 入门

1. 课程目标

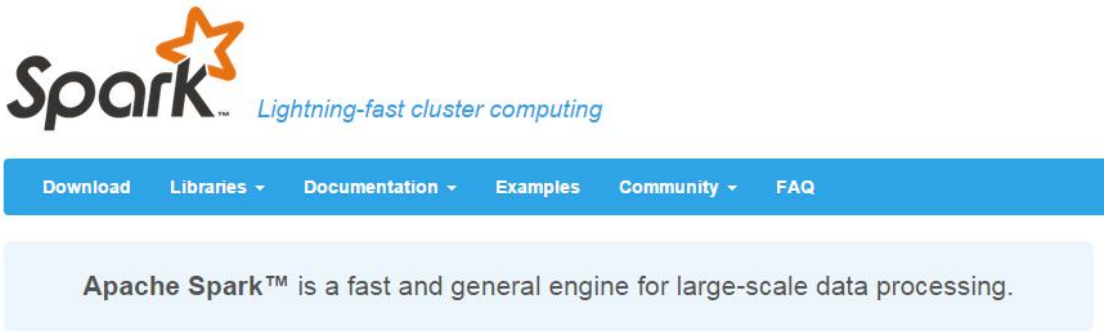
1.1. 目标 1：熟悉 Spark 相关概念

1.2. 目标 2：搭建 Spark 集群

1.3. 目标 3：编写简单的 Spark 应用程序

2. Spark 概述

2.1. 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。

2.2. 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。

2.3. Spark 特点

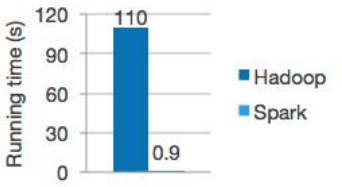
2.3.1. 快

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。

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

2.3.2. 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。

Ease of Use

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

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 and R shells.

```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

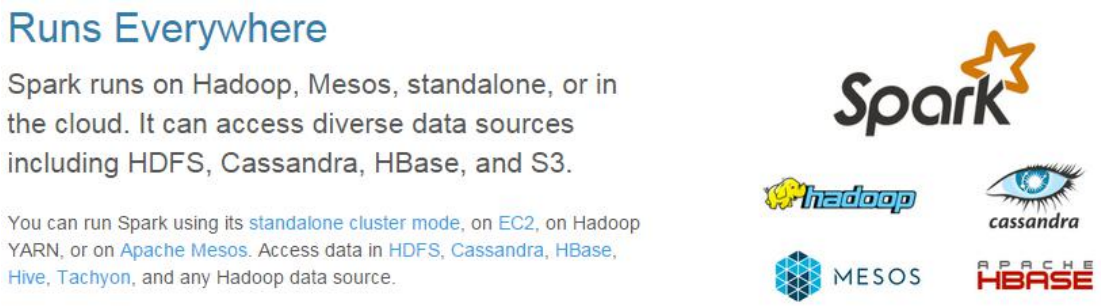
Word count in Spark's Python API

2.3.3. 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。

2.3.4. 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。



3. Spark 集群安装

3.1. 安装

3.1.1. 机器部署

准备两台以上 Linux 服务器，安装好 JDK1.7

3.1.2. 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

3.1.3. 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

4. 执行 Spark 程序

4.1. 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

4.2. 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。

4.2.1. 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

4.2.2. 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")
```

4.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

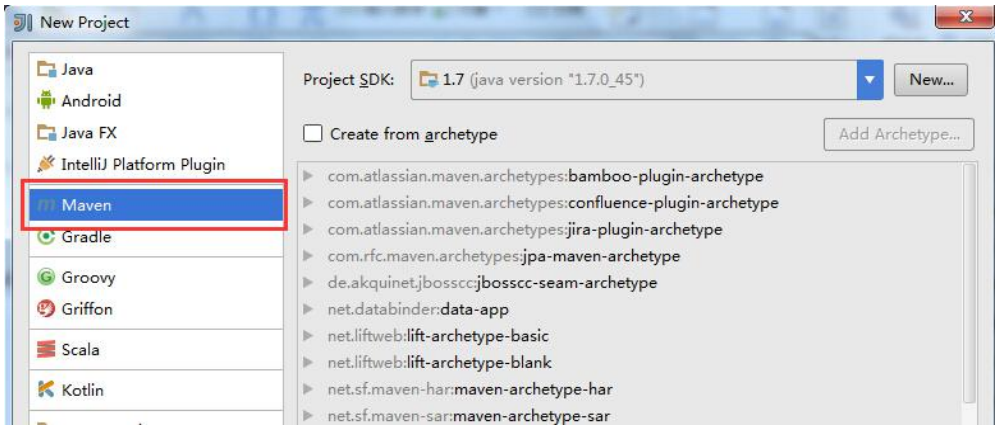
saveAsTextFile("hdfs://node1.itcast.cn:9000/out")将结果写入到 hdfs 中

4.3. 在 IDEA 中编写 WordCount 程序

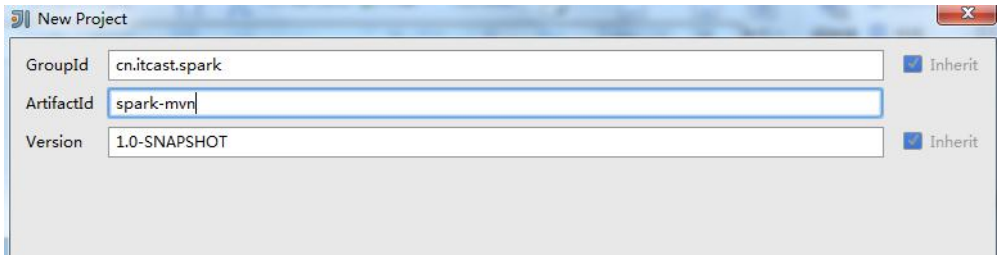
spark shell 仅在测试和验证我们的程序时使用的较多，在生产环境中，通常会在 IDE 中编制程序，然后打成 jar 包，然后提交到集群，最常用的是创建一个 Maven 项目，利用 Maven 来管理 jar 包的依赖。

1.创建一个项目

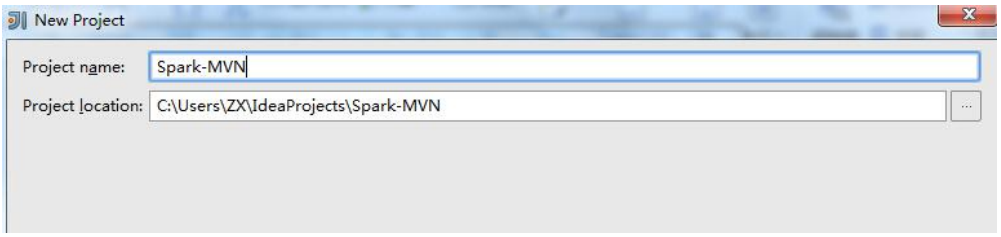
2.选择 Maven 项目，然后点击 next



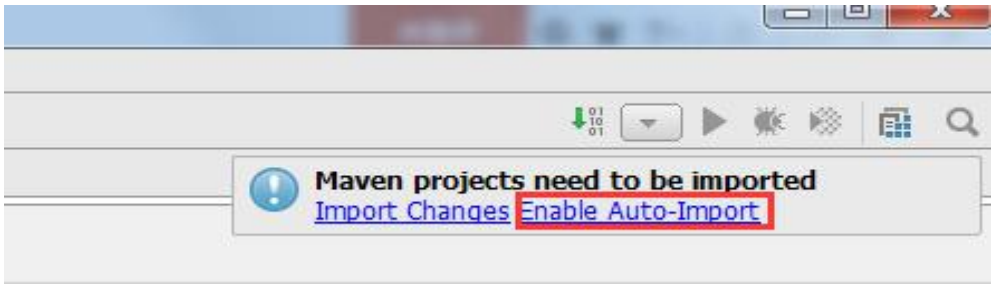
3.填写 maven 的 GAV，然后点击 next



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

```
<?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>
```

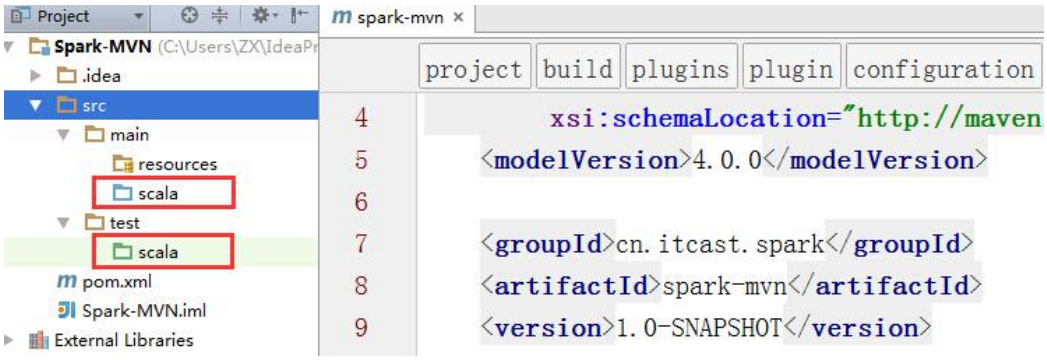
```
<dependencies>
    <dependency>
        <groupId>org.scala-lang</groupId>
        <artifactId>scala-library</artifactId>
        <version>${scala.version}</version>
    </dependency>

    <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
```

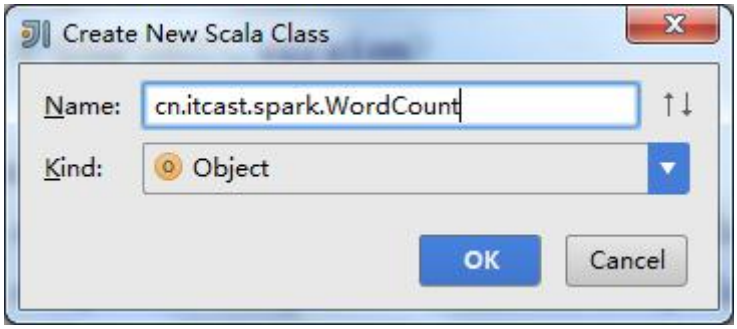
```
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>cn.itcast.spark.WordCount</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

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

```
44
45      <build>
46          <sourceDirectory>src/main/scala</sourceDirectory>
47          <testSourceDirectory>src/test/scala</testSourceDirectory>
48          <plugins>
49              <plugin>
50                  <groupId>net.alchim31.maven</groupId>
51                  <artifactId>scala-maven-plugin</artifactId>
52                  <version>3.2.0</version>
53                  <executions>
```



8.新建一个 scala class，类型为 Object



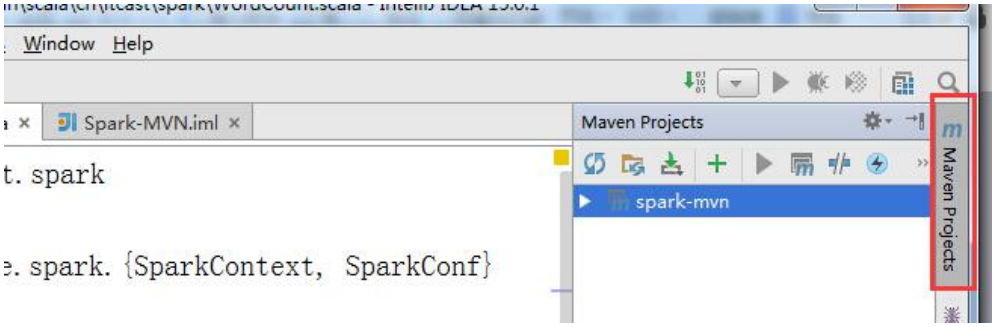
9.编写 spark 程序

```
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()
  }
}
```

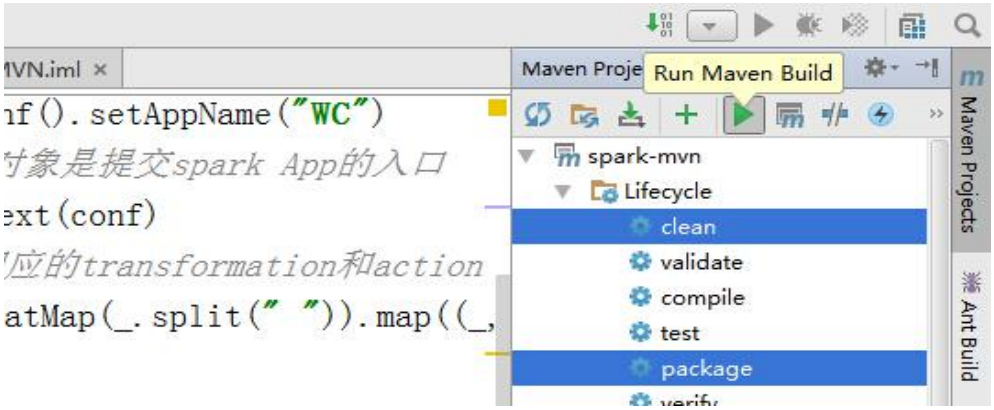
10.使用 Maven 打包：首先修改 pom.xml 中的 main class

```
<transformers>
    <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestRes
      <mainClass>cn.itcast.spark.WordCount</mainClass>
    </transformer>
</transformers>
```

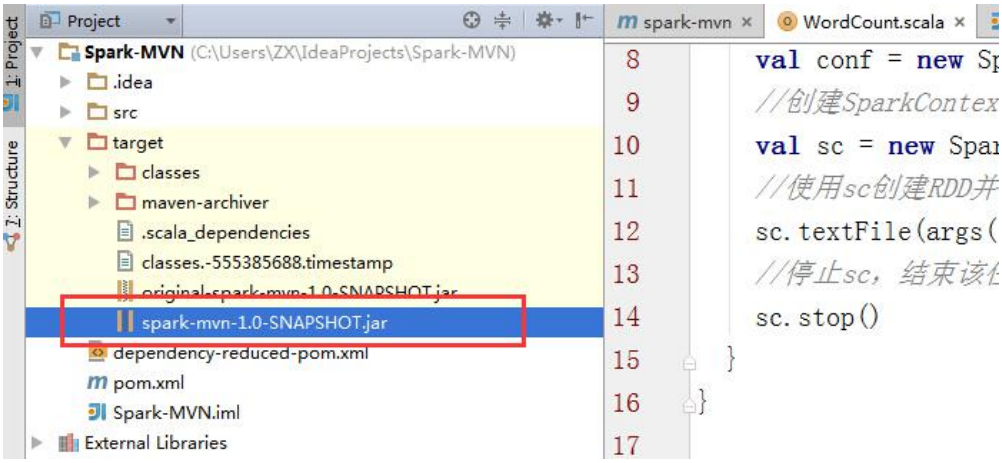
点击 idea 右侧的 Maven Project 选项



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



11.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



12.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
```

13.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)