

# Day31\_Spark的安装及简单操作

大数据-张军锋

Day31

Spark的安装

Spark计算WordCount

## Day31\_Spark的安装及简单操作

### Spark安装及配置

Spark安装

Spark配置

配置高可用

启动spark客户端

### spark

#### 前言

数据处理应用场景

数据处理方式

#### Spark介绍

什么是Spark?

Spark比Hadoop快的两个原因

#### Spark运行模式（四种）

Spark内核之RDD的五大特性

Spark运行机制

Spark运行时

#### spark编程模型

spark应用程序编程模型

RDD来源-集合数据

RDD来源-Hadoop数据集

#### 计算WordCount

处理过程

#### Spark算子-Transformations || Actions

常见的算子

本地程序运行在Spark集群上

### 作业

# Spark安装及配置

下载地址：<https://www.apache.org/dyn/closer.lua/spark/spark-2.2.0/spark-2.2.0-bin-hadoop2.7.tgz>

**注意：**下载的时候选择自己的hadoop版本

## Spark安装

解压spark：`tar -zxvf spark-2.2.0-bin-hadoop2.7.tgz`

配置环境变量

```
#spark
export SPARK_HOME=/opt/Software/Spark/spark-2.2.0-bin-hadoop2.7
export PATH=$PATH:$SPARK_HOME/bin
```

是环境变量生效：`source /etc/profile`

## Spark配置

进入spark的conf目录

```
cd /opt/Software/Spark/spark-2.2.0-bin-hadoop2.7/conf
cp spark-env.sh.template spark-env.sh
cp log4j.properties.template log4j.properties
cp slaves.template slaves
```

编辑spark-env.sh

```
export SCALA_HOME=/opt/Software/Scala/scala-2.11.11
export JAVA_HOME=/opt/Software/Java/jdk1.8.0_141
export SPARK_WORKER_MEMORY=1G
export HADOOP_CONF_DIR=/opt/Software/Hadoop/hadoop-2.7.3/etc/hadoop
```

编辑slaves

**注意：**删掉之前的localhost

```
master
slaver1
slaver2
```

将spark拷贝到子节点上然后配置环境变量并使其生效

```
scp -r /opt/Software/Spark/spark-2.2.0-bin-hadoop2.7/ root@slaver1:/opt/Software/Spark/
scp -r /opt/Software/Spark/spark-2.2.0-bin-hadoop2.7/ root@slaver2:/opt/Software/Spark/
```

进入主节点的sbin目录运行：`start-all.sh`


主节点上Master Worker两个进程，子节点上Worker一个进程

```
[root@master ~]# jps
3130 Jps
2955 Master      主节点
3019 Worker
```

```
[root@slaver1 ~]# jps
3188 Jps
3095 Worker      子节点
```

```
[root@slaver2 ~]# jps
3196 Jps
3084 Worker
```

浏览器访问：<http://master:8080/>

2.2.0

Spark Master at spark://master:7077

URL: spark://master:7077

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

Alive Workers: 3

Cores in use: 3 Total, 0 Used

Memory in use: 3.0 GB Total, 0.0 B Used

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
<a href="#">worker-20171118000824-192.168.89.200-58690</a>	192.168.89.200:58690	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
<a href="#">worker-20171118000834-192.168.89.201-49374</a>	192.168.89.201:49374	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
<a href="#">worker-20171118000834-192.168.89.202-40179</a>	192.168.89.202:40179	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications

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

Completed Applications

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

# 配置高可用

Spark-env.sh里面添加配置完成

```
export SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.deploy.zookeeper.url=master:2181,slaver1:2181,slaver2:2181 -Dspark.deploy.zookeeper.dir=/opt/Software/Spark/spark-2.2.0-bin-hadoop2.7/zookeeper"
```

## 检查高可用配置

- 启动Zookeeper
- 主节点启动：`start-all.sh`
- 其中一个子节点启动：`start-master.sh`

```
[root@slaver1 /]# jps
3445 Jps
3338 Worker
3261 QuorumPeerMain
3391 Master
```

- 主节点杀掉master进程

```
[root@master /]# jps
3223 QuorumPeerMain
3431 Jps
3290 Master
3357 Worker
[root@master /]# kill -9 3290
```

- 访问子节点8080端口：<http://slaver1:8080/>



Spark Master at spark://slaver1:7077

URL: spark://slaver1:7077  
REST URL: spark://slaver1:6066 (cluster mode)  
Alive Workers: 3  
Cores in use: 3 Total, 0 Used  
Memory in use: 3.0 GB Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

### Workers

Worker Id	Address	State	Cores	Memory
worker-20171118005820-192.168.89.201-34189	192.168.89.201:34189	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20171118005820-192.168.89.202-49113	192.168.89.202:49113	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20171118005822-192.168.89.200-57051	192.168.89.200:57051	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

## 启动spark客户端

- 1.先启动hadoop集群：`start-dfs.sh`
- 启动zookeeper集群：`zkServer.sh start`
- 启动spark服务器：`start-all.sh`
- 2.启动spark客户端：`spark-shell --master spark://master:7077`
- 3.启动成功显示，输入sc和spark都有显示

```
Welcome to
  ____  _
 / ___|| | | |
| |___| |_| |
 \___|_____|_|_|
version 2.2.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_141)
Type in expressions to have them evaluated.
Type :help for more information.

scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@25dacc5a

scala> spark
res1: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@8e100e7
```

#### • 4.计算wordcount测试

- `val file = sc.textFile("hdfs://master:9000/README.txt")`
- `file.foreach(println)`
- `val result = file.flatMap(x=>x.split("\\s")).map(x=>(x,1)).reduceByKey((x1,x2)=>x1+x2)`
- 输出到hdfs上：`result.saveAsTextFile("/sparkwcresult")`
- 查看hdfs上面生成的wc：`hdfs dfs -cat /sparkwcresult/part-00000`

# spark

## 前言

### 数据处理应用场景

Item	Value
批处理	bi分析、olap
流处理	实时需求
交互式处理	开发调试、数据挖掘、模型计算

### 数据处理方式

函数式数据处理，使用某种语言的api，通过方法的调用完成数据的处理

sql

```
result = ax + by + cz + c
```

## Spark介绍

# 什么是Spark?

Apache Spark是一个开源的集群计算框架，使数据计算更快（高效运行，快速开发）

## Spark比Hadoop快的两个原因

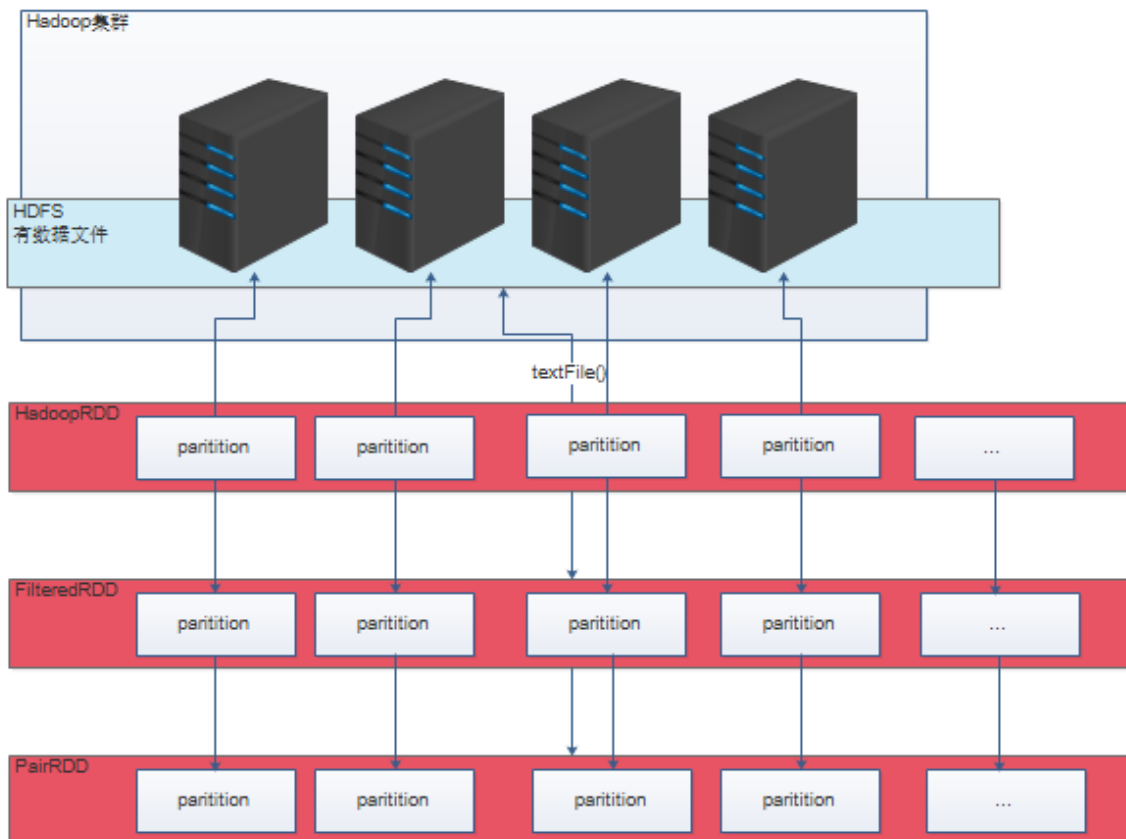
第一，内存计算

第二，DAG（有向无环图）

## Spark运行模式（四种）

Item	Value
Local	多用于测试
Standalone	Spark自带的资源调度器（默认情况下就跑在这里面）
MeSOS	资源调度器，同Hadoop中的YARN
YARN	最具前景，公司里大部分都是 Spark on YARN

## Spark内核之RDD的五大特性



**Resilient Distributed Dataset**

**RDD是基础→弹性分布式数据集**

**第一大特性：**RDD由一系列的partitions组成（如果数据源在HDFS上，默认partition的数量与block的个数一致，Spark并没有读取HDFS的方法，它是沿用MR的方法，MR读取HDFS上的数据时首先会进行split,RDD中每一个partition与split对应，split默认与block的大小一致，所以默认partition的数量与block的个数一致）

**第二大特性：**每一个函数实际上是作用在RDD中的每一个partition上

**第三大特性：**RDD是由一系列的依赖关系的（这里体现出了RDD的弹性，弹性一，数据容错；弹性二，partition可大可小）

**第四大特性：**partitioner（分区器）是作用在KV格式的RDD上（RDD执行聚合类函数的时候会产生shuffle,Spark产生shuffle肯定会有partitioner,而partitioner是作用在KV格式的RDD上，推测出聚合类函数必须作用在KV格式的RDD上）

**第五大特性：**每一个RDD提供了最佳的计算位置，告诉我们每一个partition所在的节点，然后相对应的task就会移动到该节点进行计算（移动计算，而不是移动数据）

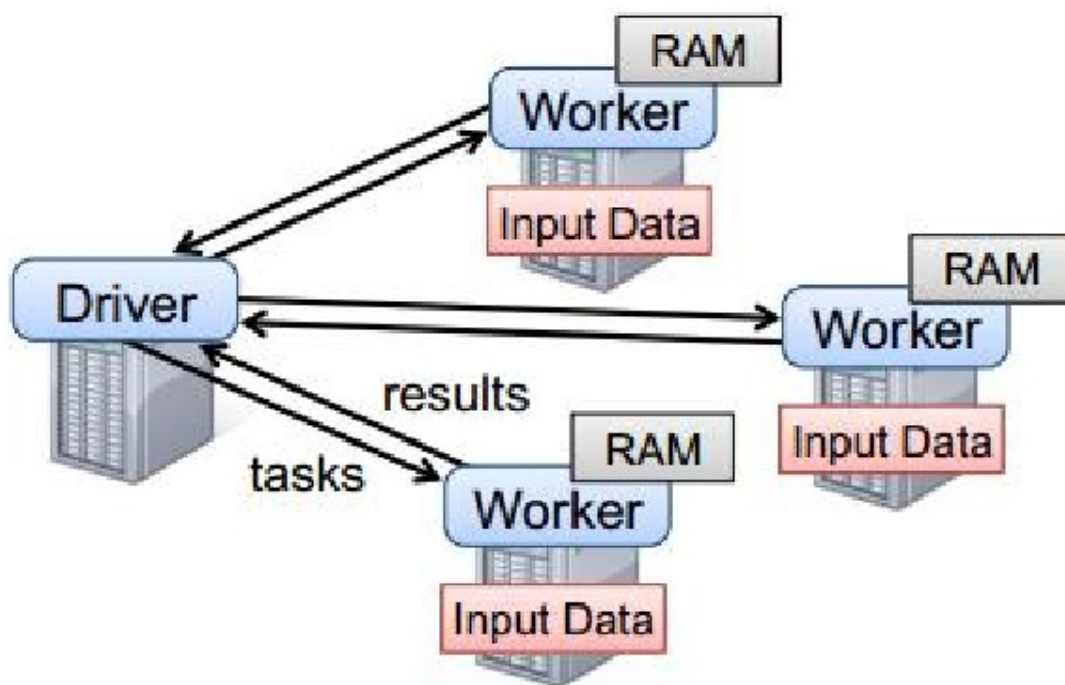
## Spark运行机制

开机启动时 **Driver**、**Worker**、和 **Application** 会将自己的资源信息注册到 **Master** 中，当初初始化的时候，**Master** 先为 **Driver** 分配资源然后启动 **Driver**。

**Driver** 运行时先从 main()方法开始，任务在 **Worker** 上执行，**Worker** 可以是一台真实的物理机，也可以是虚拟机，拥有 RAM 和 Core。然后将 **Task** 移动到本地的数据上执行运算。**最优计算位置 Inputdata 和 Task 在一起(避免了网络间的信息传输)**。实际情况很少会这样，有可能存在当前那个计算节点的**计算资源**和**计算能力**都满了，默认配置不变的情况下 Spark 计算框架会等待

3s(spark.locality.wait 设置的，在 **SparkConf()**可以修改)，默认重试 5 次。如果均失败了，会选择一个比较差的本地节点；Spark 分配算法会将其分配到计算数据附近的节点，**Task** 会通过其所在节点的 **BlockManager** 来获取数据，**BlockManager** 发现自己本地没有数据，会通过**getRemote()**方法，通过 **TransferService**（网络数据传输组件）从原 task 所在节点的 **BlockManager** 中获取数据，通过网络传输回 Task 所在节点——>(性能大幅度下降，大量的网络 IO 占用资源) 计算后的结果会返回到 **Driver** 上

## Spark运行时



**Driver** ( SparkContext运行所在的节点可以看做一个Driver ) 作用：

- 分发task给对应的Worker,可以和其他节点(Worker)进行通信
- 接收task的计算结果

**Worker**作用：

- **Worker** 可以是一台真实的物理机，也可以是虚拟机，拥有 RAM 和 Core，执行运算

## spark编程模型

**spark应用程序有两部分组成：Driver和Executor**

**Application**：基于spark的用户程序，包含一个Driver和多个Executor

**Driver**：运行Application的main函数，一般在其中创建SparkContext

**Executor**：某个Application运行在某个节点上的一个进程，该进程负责运行Task，并且负责将数据存储在内存或者磁盘上。每个Application都有自己独立的Executors

**Cluster Manager**：在分布式集群上获取资源的外部服务（ Standalone、Mesos、Yarn ）

**Worker Node**：集群中任意可以执行Application代码的节点

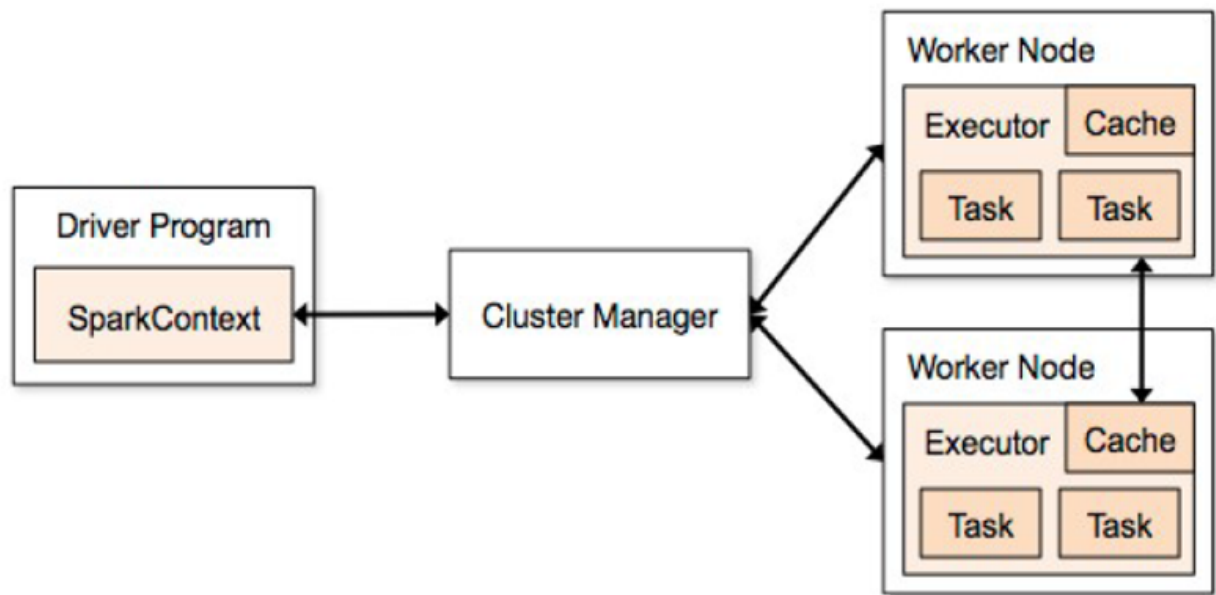
**Job**：包含多个Task组成的并行计算，往往由Spark Action催生，日志中会经常看到

**Task**：被发送到某个Executor上去执行的工作单元

**Stage**：每个Job会被拆分成多组Task，每组Task被称作一个Stage，往往以shuffle过程作为划分边界

**RDD**：Spark的计算单元，可以通过一系列算子的调用来进行操作，其内代表着分布式的待处理的数据，算子主要包括Transformation和Action





## spark应用程序编程模型

### Driver

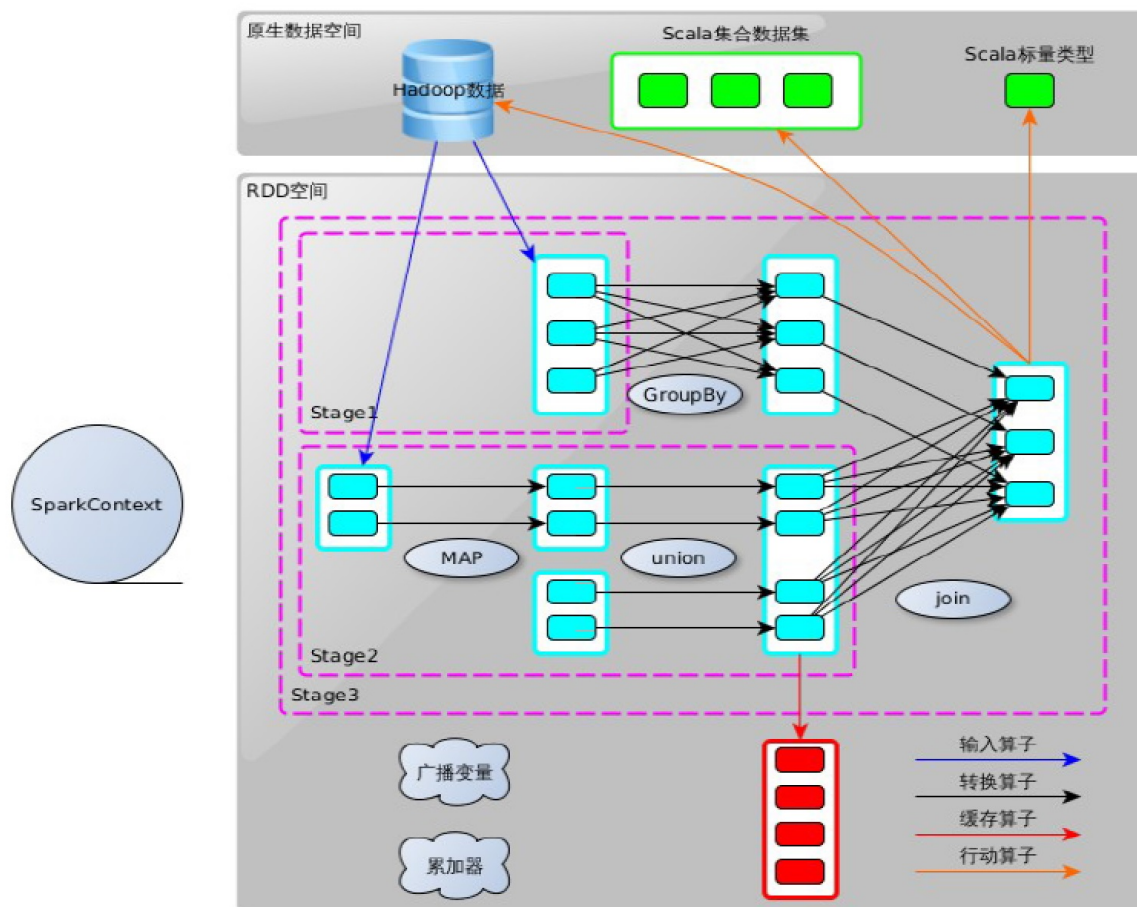
- 导入Spark的类和隐式转换
- 构建spark应用程序的运行环境配置SparkConf
- 初始化SparkContext
- 关闭SparkContext

### Executor

- 输入Base->RDD
- Transformation RDD->RDD
- Action RDD->driver or Base
- 缓存Persist或者cache

### 共享变量

- Broadcast variables
- Accumulators



## RDD来源-集合数据

Spark使用parallelize方法转换成RDD

```
val rdd1 = sc.parallelize(Array(1,2,3,4,5,6))
val rdd2 = sc.parallelize(List(0 to 10),5)
```

参数slice是对数据集切片，每个slice启动一个Task处理

## RDD来源-Hadoop数据集

- Spark可以将任何hadoop所支持的存储资源转化成RDD，如本地文件、HDFS、HBase等
- Spark支持text files，SequenceFiles和任何Hadoop InputFormat格式
- 使用textFile()方法可以将本地或HDFS文本文件转换成RDD
- 如果读取本地文件，个节点都要有该文件，或者使用网络共享文件
- 支持整个文件目录读取如 `textFile("/my/directory")`
- 压缩文件读取，如 `textFile("/my/directory/*.gz")`
- 通配符文件读取，如 `textFile("/my/directory/*.txt")`
- 
- 其第二个参数slice，默认情况下为每个block创建一个分片，用户也可以通过slice指定更多分片，但不能使用少于block的分片数
- 使用 `wholeTextFiles()` 读取目录里面的小文件，返回（文件名，内容）对
- 使用 `sequenceFile[K,V]()` 方法可以将SequenceFile转换成RDD
- 使用 `hadoopRDD()` 方法可以将其他任何Hadoop的输入类型转换成RDD

# 计算WordCount

```
object WordCount {  
  
  def main(args: Array[String]): Unit = {  
    //创建SparkConf对象  
    //设置分布式运行平台和AppName  
    //使用master指定运行平台,yarn,standalone,mesos,local  
    //local local[N] local[*]  
    //val conf = new SparkConf().setMaster("local[2]").setAppName("word count")  
  }  
}
```

生产环境

开发调试环境

单线程

n个线程

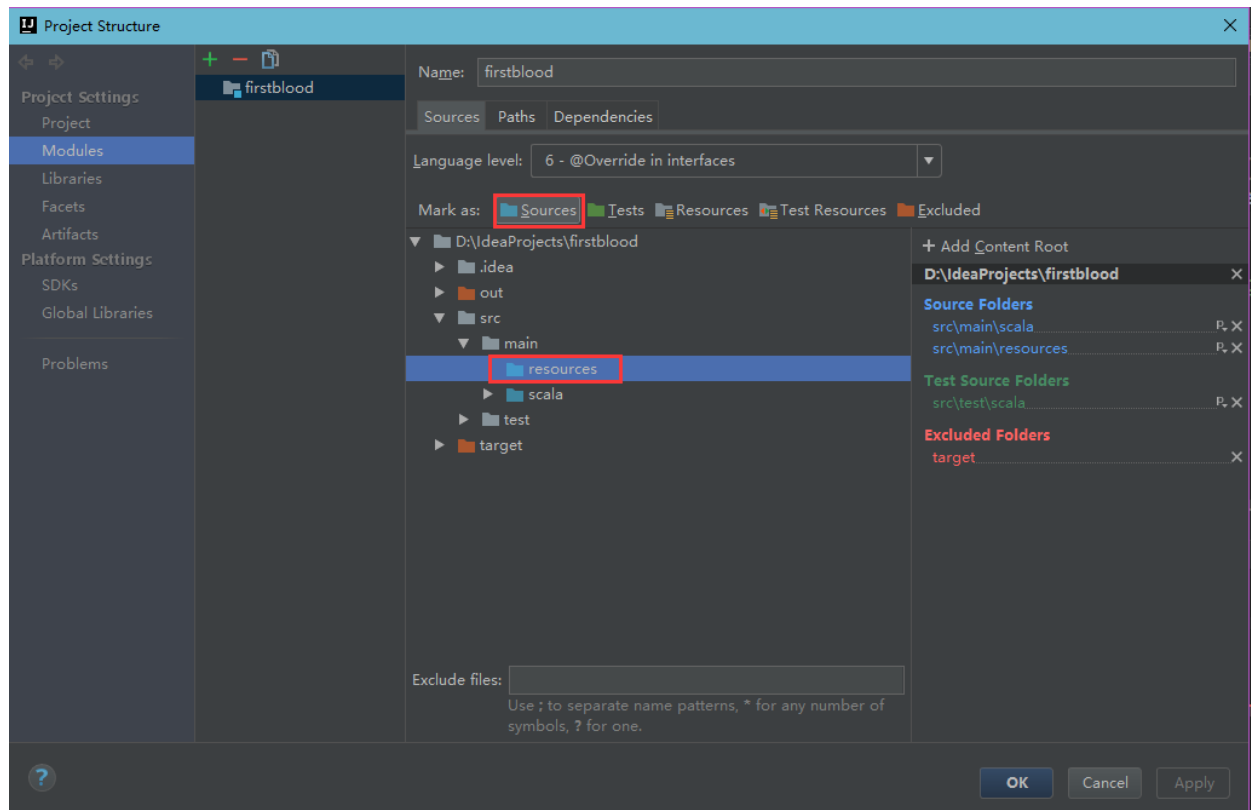
本地cpu有多少核心就有多少线程

```
package com.bd14.zjf.sparktest  
  
import org.apache.spark.{SparkConf, SparkContext}  
  
object WordCount {  
  
  def main(args: Array[String]): Unit = {  
    //创建SparkConf对象  
    //设置分布式运行平台和AppName  
    //使用master指定运行平台,yarn,standalone,mesos,local  
    //local local[N] local[*]  
    val conf = new SparkConf().setMaster("local[2]").setAppName("word count")  
    //构建SparkContext对象  
    val sc = new SparkContext(conf)  
  
    //加载数据源,获取RDD对象  
    //file:///c:\\user_info.txt  
    //hdfs://master:9000/user_info.txt  
    val fileRdd = sc.textFile("c:\\user_info.txt")  
    //数据处理开始  
    val wordRdd = fileRdd.flatMap(line => line.split("\\s"))  
    val result = wordRdd.map(x => (x, 1)).reduceByKey((v1, v2) => v1 + v2)  
    result.saveAsTextFile("c:\\user_info_spark")  
    //省略写法  
    //fileRdd.flatMap(_.split("\\s"))  
    //      .map((_, 1))  
    //      .reduceByKey(_ + _)  
    //      .saveAsTextFile("/bd14/user_info_spark")  
  }  
}
```

**注意：**如果要读取hdfs上的文件，保存解析后的文件到hdfs，需要添加core-site.xml

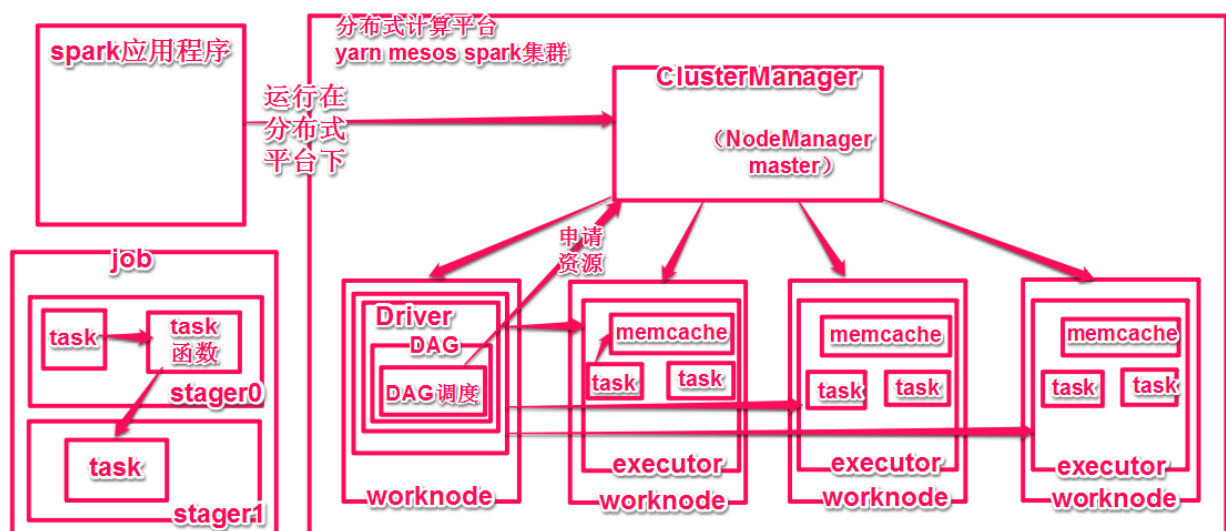
### 创建sources Directory

new Directory—>Project Structure—>Modules—>选择文件夹，点击Structure

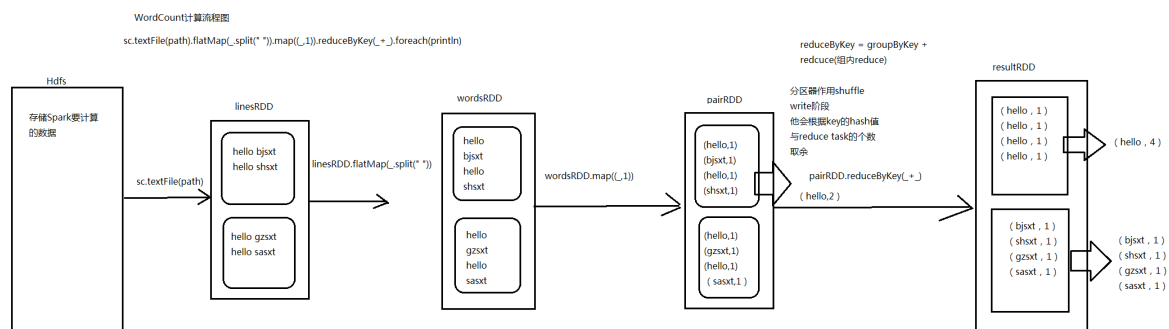
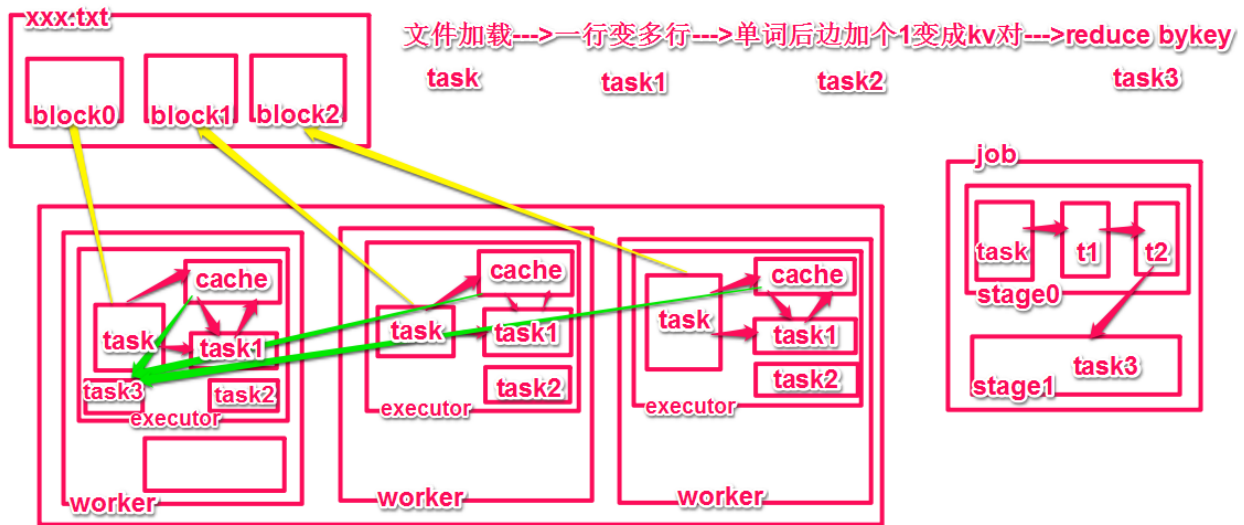


添加core-site.xml，默认加载路径为hdfs，加载文件 `val fileRdd = sc.textFile("/user_info.txt")` 或者 `hdfs://master:9000/user_info.txt`，如果此时想要加载windows本地文件 `file:///c:\\user_info.txt`

## 处理过程



Spark中WordCount演变流程图



spark程序的数据处理过程，是在driver上来进行解析编译，在具体对数据处理的时候需要一个触发执行计算过程，这个触发点就是rdd的action方法

## Spark算子-Transformations || Actions

rdd的处理方法（函数）有两种

Transformations    Actions 这两类算子的区别	
Transformations	Transformations类的算子会返回一个新的RDD，懒执行
Actions	Actions类的算子会返回基本类型或者一个集合，能够触发一个job的执行，代码里面有多少个action类算子，那么就有多少个job

## transformation

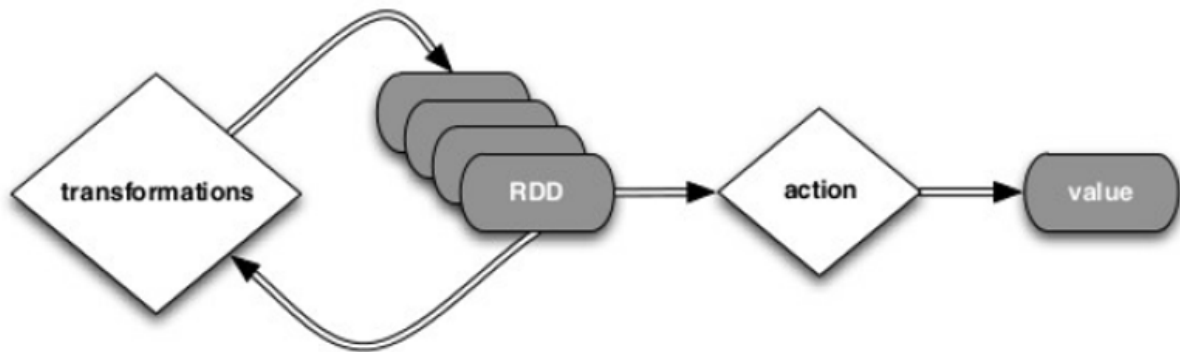
转换 对数据进行计算转换抽取，只记录处理过程而不真正计算数据，例如：一行变多行，一个单词变kv对，把一个key下的所有value累加

特点：在一个已有的RDD上创建一个新的RDD、延迟执行（Lazy）

## action

行为 对数据进行读值，触发对数据的计算过程，触发后会把action前面的transformation启动执行，一般不改变数据，只是对数据的结果值进行读取

特点：对RDD数据集上运行计算后，将结果传给驱动程序或写入文件系统、出发Job

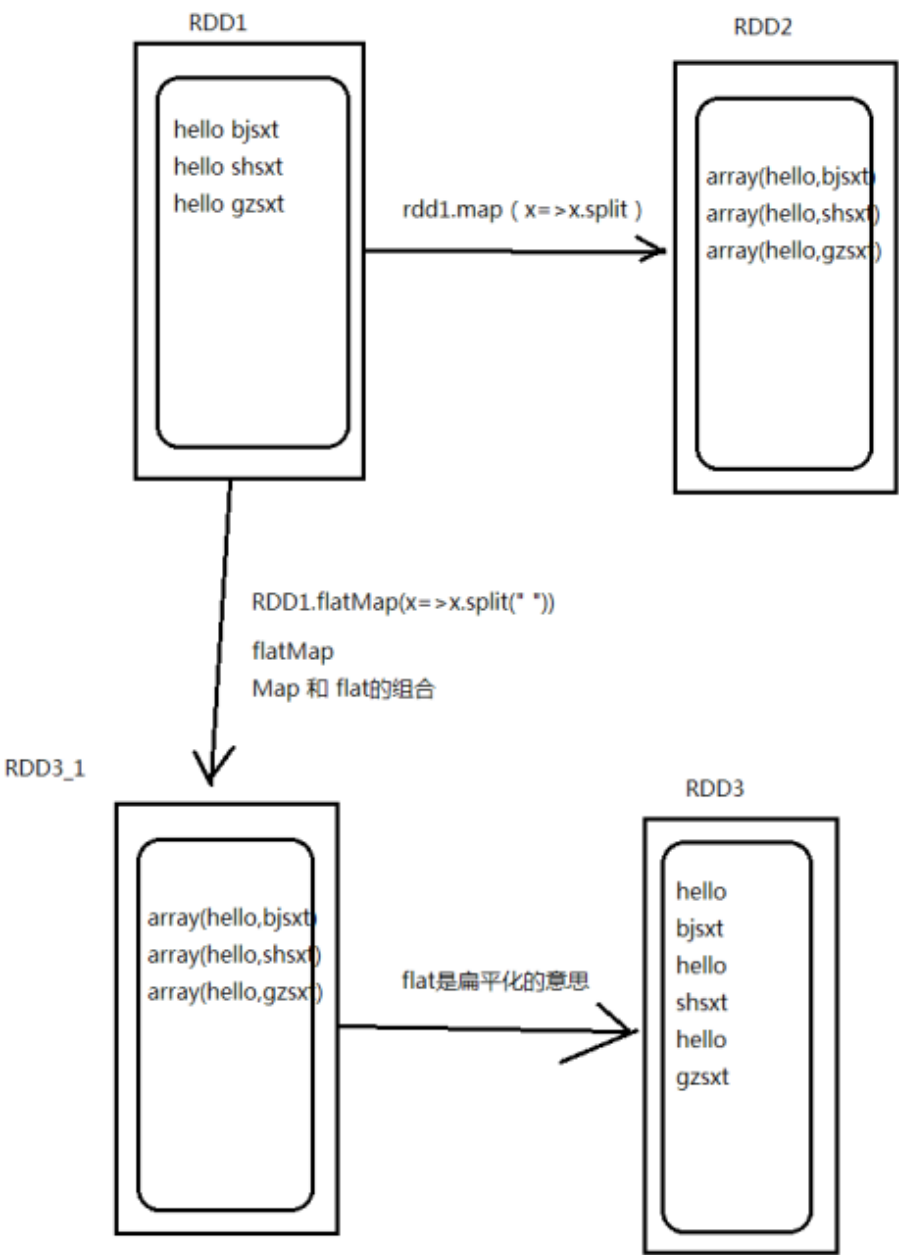


## 常见的算子

Transformation类算子	map	输入一条，输出一条 将原来 RDD 的每个数据项通过 map 中的用户自定义函数映射转变为一个新的 元素。输入一条输出一条；
	flatMap	输入一条输出多条 先进行map后进行flat
	mapPartitions	与 map 函数类似，只不过映射函数的参数由 RDD 中的每一个元素变成了 RDD 中每一个分区的迭代器。将 RDD 中的所有数据通过 JDBC 连接写入数据库，如果使用 map 函数，可能要为每一个元素都创建一个 connection，这样开销很大，如果使用 mapPartitions，那么只需要针对每一个分区建立一个 connection。
	mapPartitionsWithIndex	
	filter	依据条件过滤的算子
	join	聚合类的函数，会产生shuffle，必须作用在KV格式的数据上 join 是将两个 RDD 按照 Key 相同做一次聚合；而 leftouterjoin 是依据左边的 RDD 的 Key 进行聚
	union	不会进行数据的传输，只不过将这两个的RDD标识一下 （代表属于一个RDD）
	reduceByKey	先分组groupByKey，后聚合根据传入的匿名函数聚合，适合在 map 端进行 combiner
	sortByKey	依据 Key 进行排序，默认升序，参数设为 false 为降序
	mapToPair	进行一次 map 操作，然后返回一个键值对的 RDD。（所有的带 Pair 的算子返回值均为键值对）
	sortBy	根据后面设置的参数排序
	distinct	对这个 RDD 的元素或对象进行去重操作

Actions类算子	foreach	foreach 对 RDD 中的每个元素都应用函数操作，传入一条处理一条数据，返回值为空
	collect	返回一个集合 ( RDD[T] => Seq[T] ) collect 相当于 toArray， collect 将分布式的 RDD 返回为一个单机的 Array 数组。
	count	一个 action 算子，计数功能，返回一个 Long 类型的对象
	take(n)	取前N条数据
	save	将RDD的数据存入磁盘或者HDFS
	reduce	返回T和原来的类型一致 ( RDD[T] => T )
	foreachPartition	foreachPartition 也是根据传入的 function 进行处理，但不同处在于 function 的传入参数是一个 partition 对应数据的 iterator，而不是直接使用 iterator 的 foreach。

map和flatMap者两个算子的区别



本地程序运行在Spark集群上

intelliJ中的程序直接运行在spark的standalone模式下

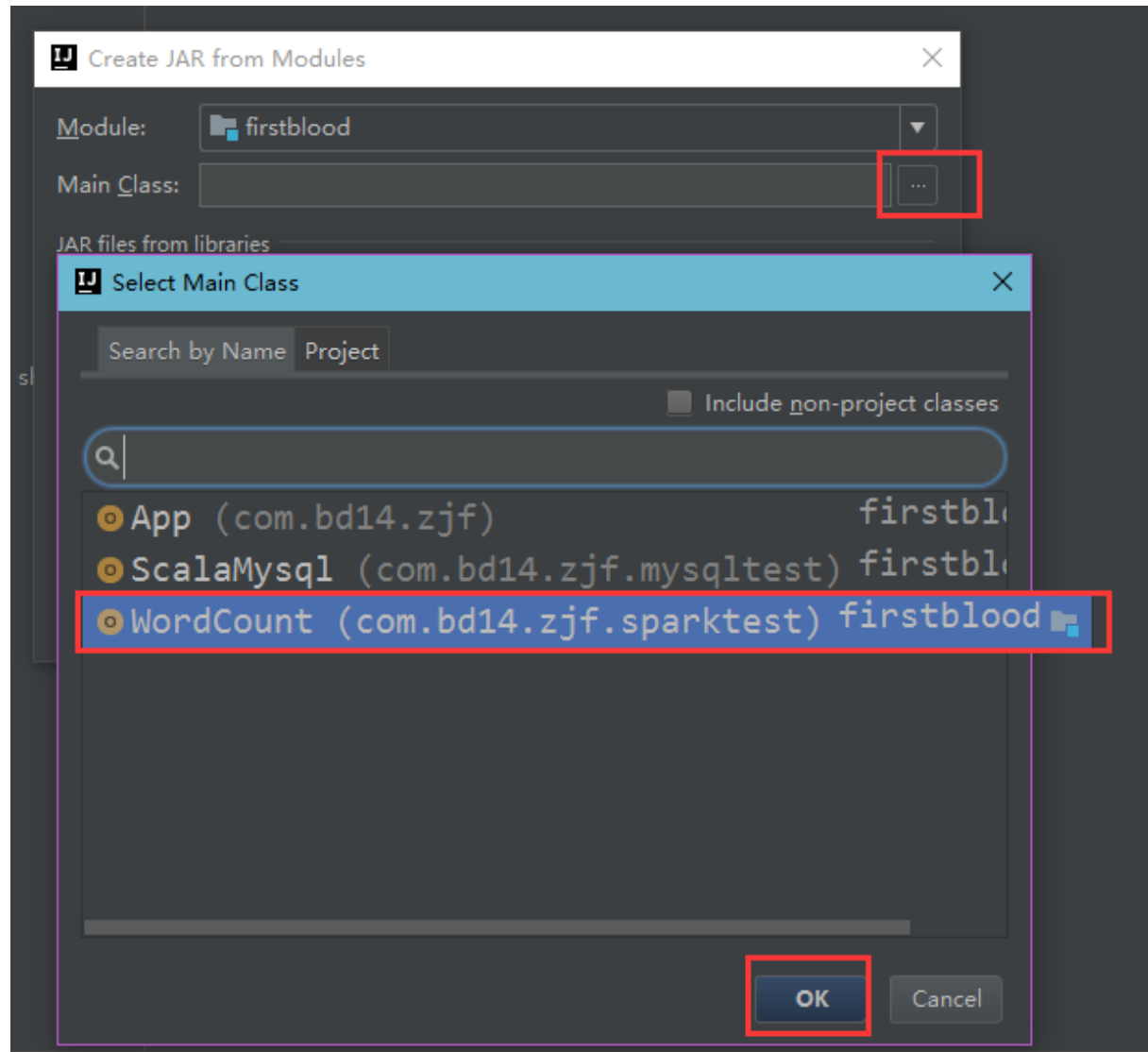
1.创建artifact (项目编译打包)

2.修改conf的master设置: `val conf = new`

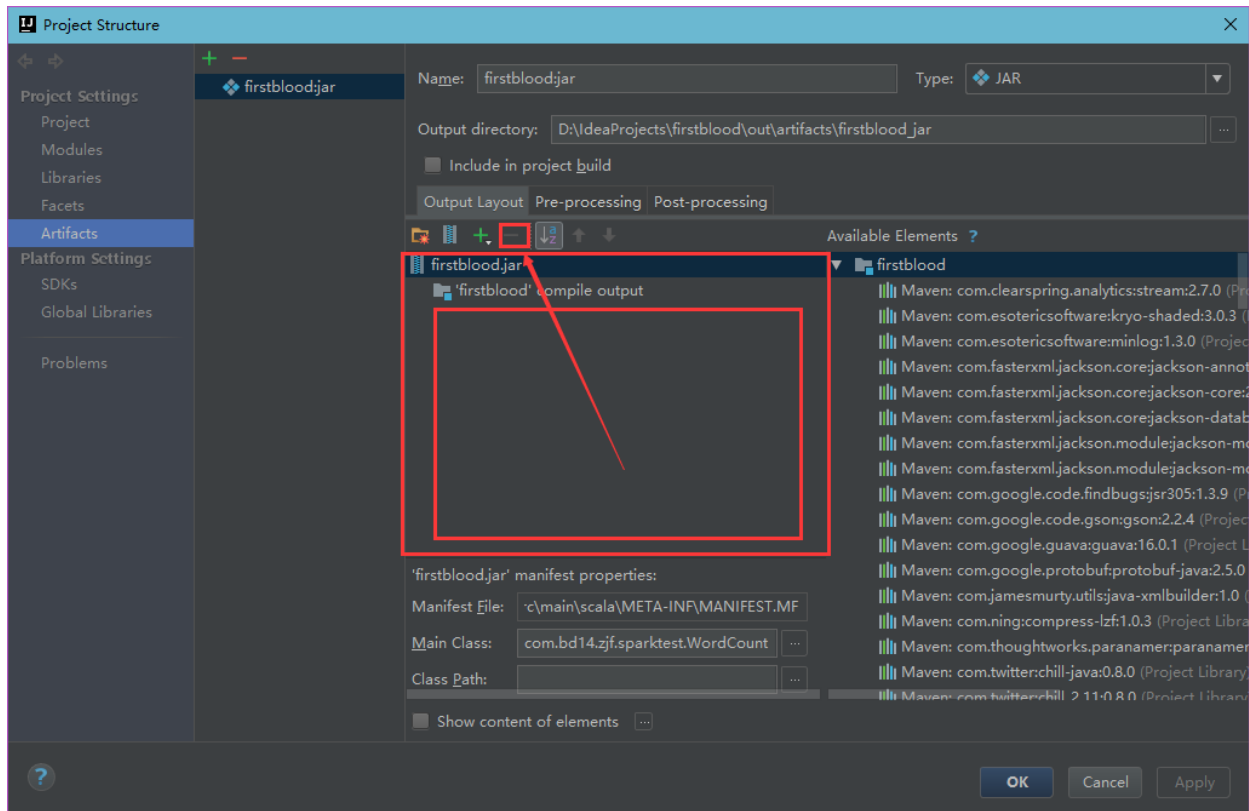
`SparkConf().setMaster("spark://master:7077").setAppName("word count")`

添加退出: `sc.stop()`

3.设置打包Project Structure—>Artifacts—>+—>JAR—>from modules.....—>添加类—>删除除  
compile output其他的jar包 (选择点击“-号删除)



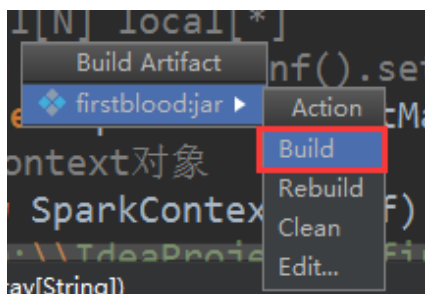
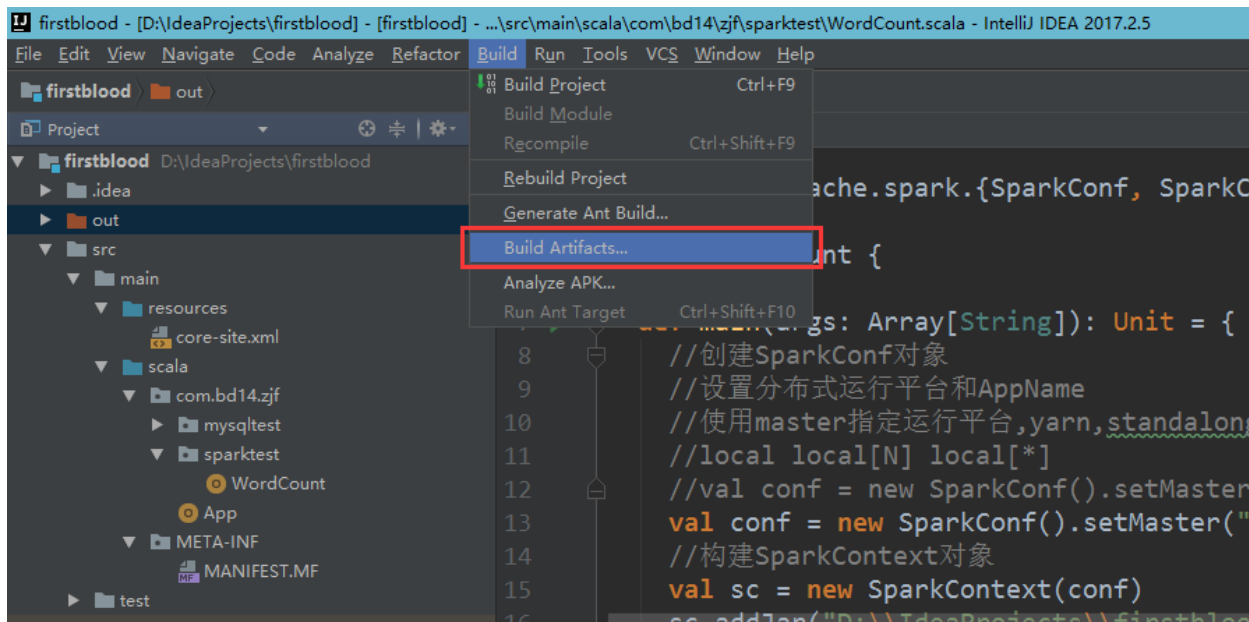


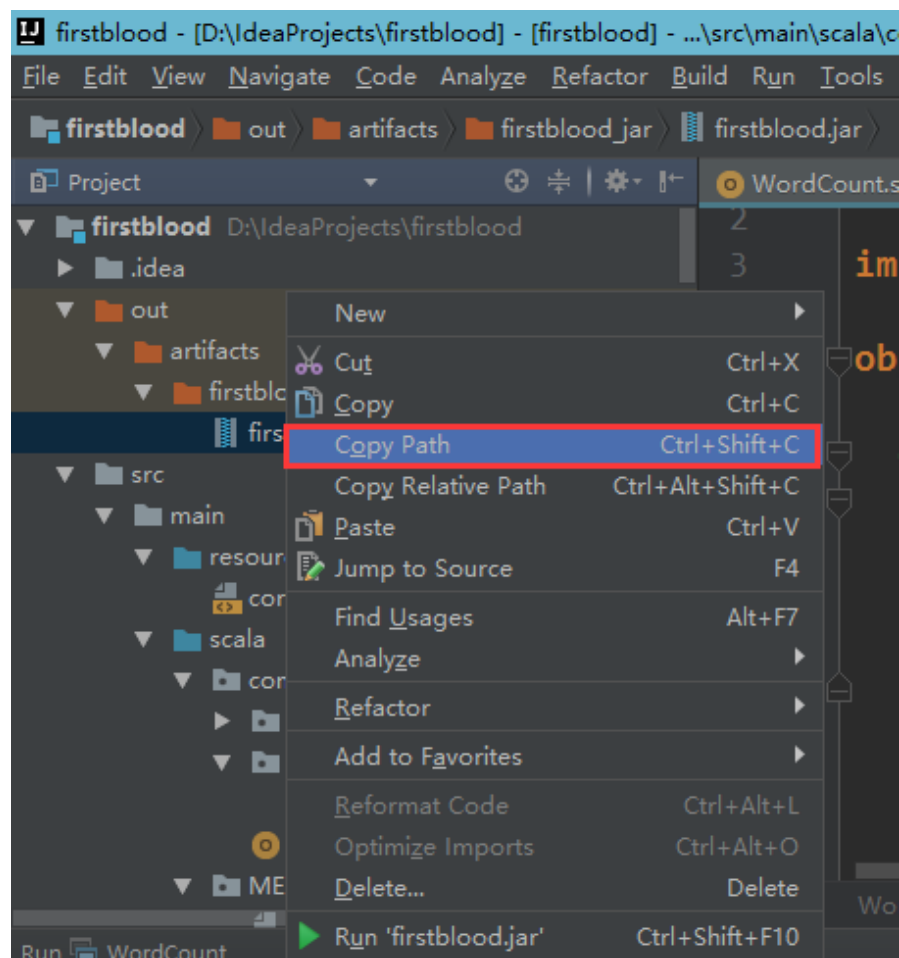


4.在sc下添加我们intelliJ打的包，打包过程，build—>build artifacts—>build：

拷贝jar包路径，添加代码：

```
sc.addJar("D:\\IdeaProjects\\firstblood\\out\\artifacts\\firstblood.jar")
```





完整代码

```

package com.bd14.zjf.sparktest

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

object WordCount {

    def main(args: Array[String]): Unit = {
        //创建SparkConf对象
        //设置分布式运行平台和AppName
        //使用master指定运行平台,yarn,standalone,mesos,local
        //local local[N] local[*]
        val conf = new SparkConf().setMaster("spark://master:7077").setAppName("word count")
        //构建SparkContext对象
        val sc = new SparkContext(conf)
        sc.addJar("D:\\IdeaProjects\\firstblood\\out\\artifacts\\firstblood_jar\\firstblood.jar")
        //加载数据源,获取RDD对象
        val fileRdd = sc.textFile("/user_info.txt")
        fileRdd.foreach(println)
        //数据处理开始
        fileRdd.flatMap(_.split("\\s"))
                    .map((_, 1))
                    .reduceByKey(_ + _)
                    .saveAsTextFile("/bd14/user_info_spark")

        sc.stop()
    }

}

```

**注意：**执行的之后关闭spark-shell，不然会执行不成功，一直警告：`TaskSchedulerImpl: Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient resources`

## 作业

数据如下图所示

```
1 jim logout 93.24.237.12
2 mike new_tweet 87.124.79.252
3 bob new_tweet 58.133.120.100
4 mike logout 55.237.104.36
5 jim new_tweet 93.24.237.12
6 marie view_user 122.158.130.90
7 jim login 198.184.237.49
8 marie login 58.133.120.100
9 jim logout 58.133.120.100
10 jim logout 93.24.237.12
11 bob login 198.184.237.49
12 mike view_user 198.184.237.49
13 marie new_tweet 231.112.6.146
14 mary view_user 235.200.255.154
15 jim new_tweet 7.157.53.156
16 jim view_user 55.237.104.36
17 bob logout 87.124.79.252
18 marie view_user 55.237.104.36
19 jim view_user 87.124.79.252
20 mike new_tweet 198.184.237.49
21 jim login 93.24.237.12
22 bob view_user 141.45.47.131
23 marie new_tweet 87.124.79.252
24 bob new_tweet 235.200.255.154
25 marie new_tweet 7.157.53.156
26 bob login 231.112.6.146
```

要求：计算出每人登录的总次数

参考代码

```

package com.bd14.zjf

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

object homework1120 {

    def main(args: Array[String]): Unit = {
        //创建SparkConf对象
        val conf = new SparkConf().setMaster("local[2]").setAppName("user login")
        //构建SparkContext对象
        val sc = new SparkContext(conf)
        //加载数据源,获取RDD对象
        val fileRdd = sc.textFile("file:///D:\\svn\\user-logs-large.txt")
        //数据处理开始
        //val wordRdd = fileRdd.flatMap(line =>
        //{ val infos= line.split("\\s")
        //  if (infos(1).equals("login")) Array(infos(0)) else Nil})
        //val result = wordRdd.map((_, 1)).reduceByKey(_ + _)
        val wordRdd = fileRdd.flatMap(line => {
            val infos = line.split("\\s")
            if (infos(1).equals("login")) Map(infos(0) -> 1) else Nil
        })
        val result = wordRdd.reduceByKey(_ + _)
        result.saveAsTextFile("file:///D:\\svn\\user-logs-large_intellij_spark")
    }

}

```

## 输出结果

```

(alison,42)
(jim,872)
(dave,2)
(mike,332)
(dude,1)
(joe,9)
(marie,864)
(bob,311)
(mary,59)

```

## 分析易错点

```

val wordRdd = fileRdd.flatMap(line =>
{ val infos= line.split("\\s")
  if (infos(1).equals("login")) infos(0) else Nil})
val result = wordRdd.map((_, 1)).reduceByKey(_ + _)

```

这样写的话会把得到的字符串再度分割为单个字母.....

