课程实验三：Spark Core Scala 单词计数

一、实验描述

本实验使用 Scala 语言编写 Spark 程序，完成单词计数任务。首先，在华为云购买 3 台服务器，然后搭建 Hadoop 集群和 Spark 集群(YARN 模式)，接着使用 Scala 语言利用 Spark Core 编写程序，最后将程序打包在集群上运行程序。

二、实验目的

1.了解服务器配置的过程；

2.熟悉使用 Scala 编写 Spark 程序；

3.了解 Spark RDD 的工作原理；

4.掌握在 Spark 集群上运行程序的方法。

三、实验环境

1.服务器节点数量：3；

2.系统版本：Centos 7.5；

3.Hadoop 版本：Apache Hadoop 2.7.3；

4.Spark 版本：Apache Spark 2.1.1；

5.JDK 版本：1.8.0_131；

6.Scala 版本：scala2.11.11；

7.IDEA 版本：ideaIC-2017.2.7。

四、实验步骤

4.1 服务器购买与配置

首先需要在华为云完成服务器和 OBS 对象存储的购买与配置，需要购买 3 个服务器，其中一个作为主节点，两个作为从节点。

步骤 1：打开华为云地址：https://www.huaweicloud.com/，点击“登录”，

输入用户名、密码，如下图所示；

图 1

步骤 2：点击“控制台”，选择“服务列表”—>“弹性云服务器 ECS”—>“购买弹性云服务器”，选择“按需计费”，“可用区 2”，CPU 架构“鲲鹏计算”，选择“鲲鹏通用计算增强型”，2vCPUs|4GB，如下图：

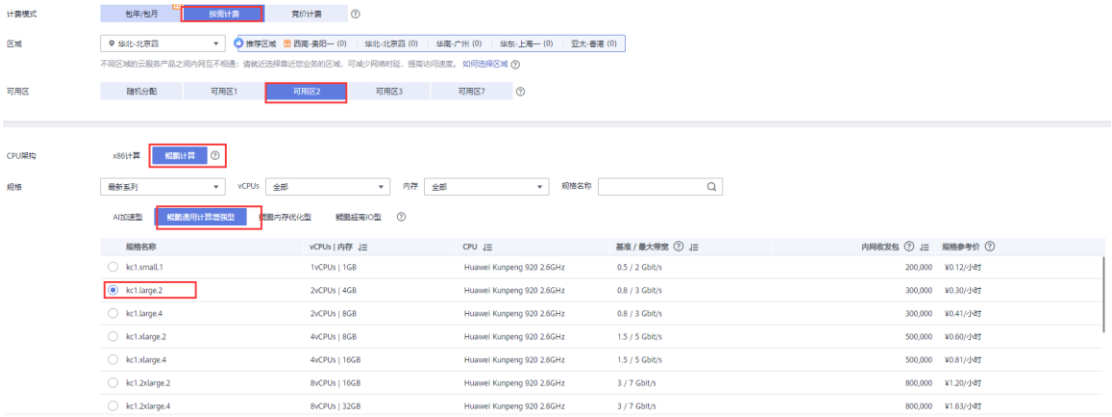


图 2

步骤 3：选择“公共镜像”，CentOS7.6，系统盘建议配置 40GB，购买数量 3 台，点击“网络配置”，如下图：

图 3

步骤 4：配置网络，网络选择“vpc-default”，安全组选择“Sys-default”，“现在购买”，选择“全动态 BGP”，“按流量计算”，“5M”，点击“高级配置”，如下图：

图 4

步骤 5：配置密码，自定义云服务器名称，自行设置 root 登录密码，云备份选择“暂不购买”，点击“确认配置”，点中“我已经阅读并同意”，点击“立即购买”，如下图：

图 5

注：创建过程需等待几分钟

4.2 Hadoop 集群搭建

Hadoop 是基础，其中的 HDFS 提供文件存储，Yarn 进行资源管理。可以运行 MapReduce、Spark、Tez 等计算框架。因此，搭建 Hadoop 集群是分布式运行 Spark 程序的关键环节。

步骤 1：使用 PUTTY 登录各个节点，节点 IP 见图 13；

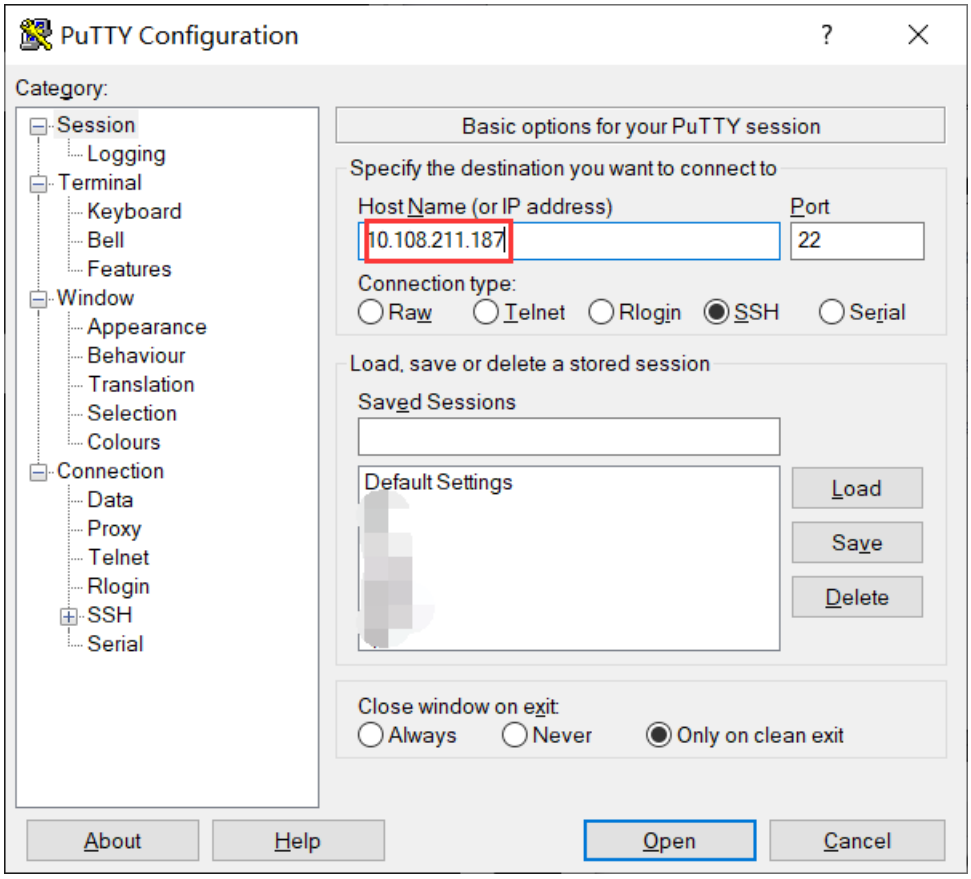


图 6

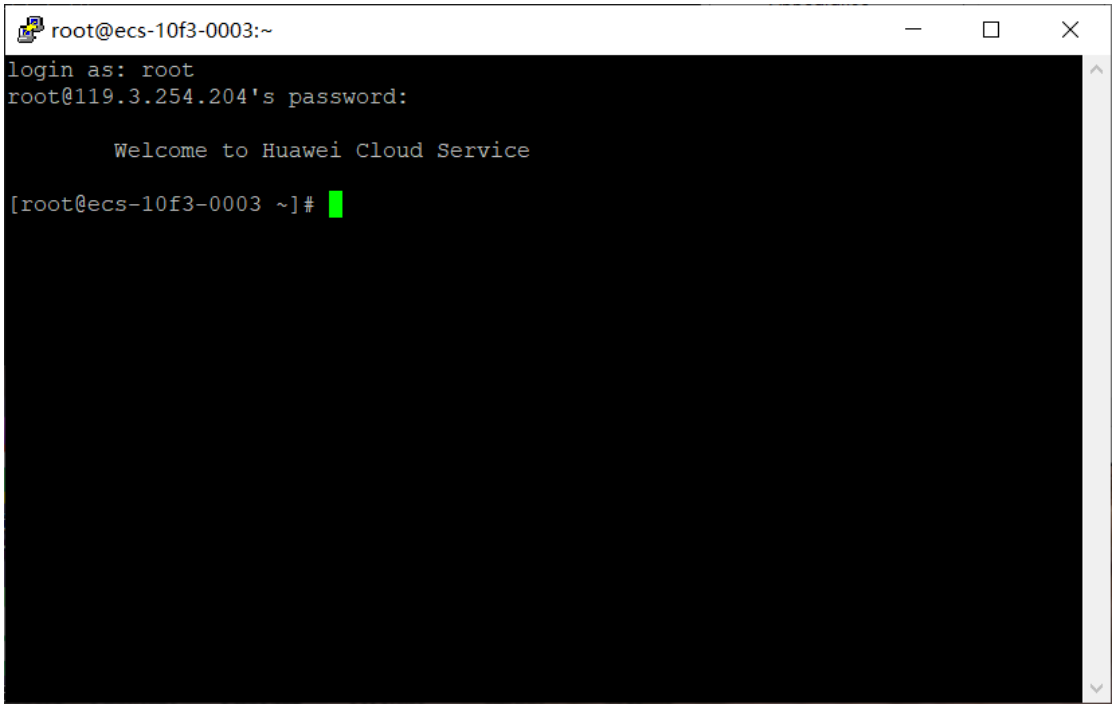


图 7

步骤 2：配置主机名（各节点分别操作此步骤，主机名分别设置为 master, slave01, slave02）。使用 vim /etc/hostname 编辑主机名：

1)移动光标至末尾，按‘i’(编辑模式开启为：插入)键，左下键出现“—INSERT—”后，删除原名称，输入对应的名称：

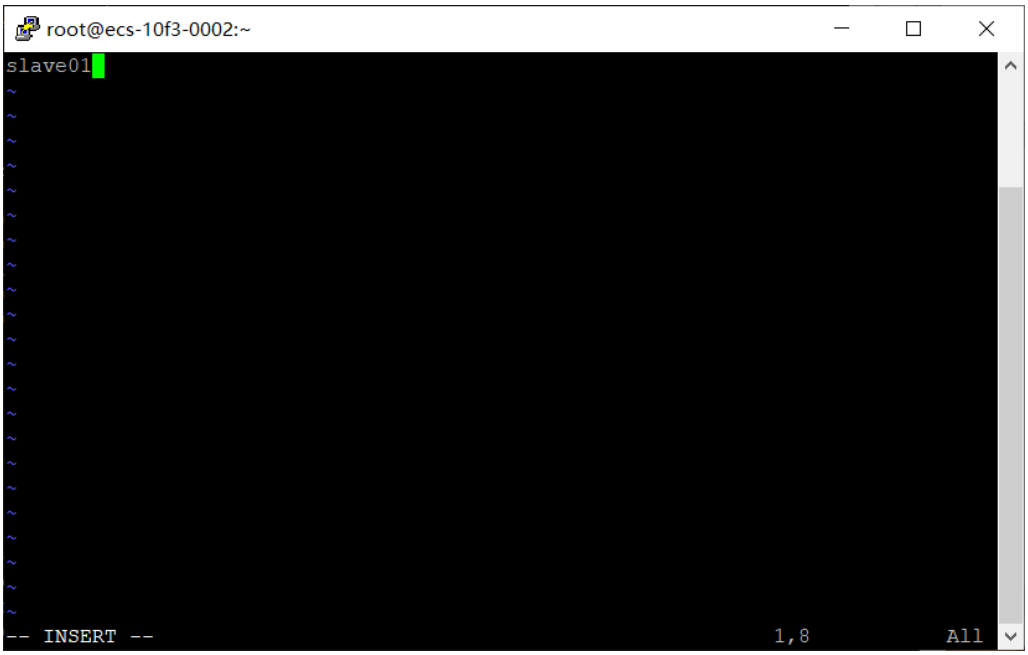


图 8

2)按下‘ESC’(退出 vim 的编辑模式，进入命令模式)，输入‘:wq’(表示写入)，按回车。

步骤 3：将主机名替换为对应名字(master 节点下就替换为 master，其他同理),并检测是否替换成功(重启 PUTTY 后，命令提示符就会发生改变)：

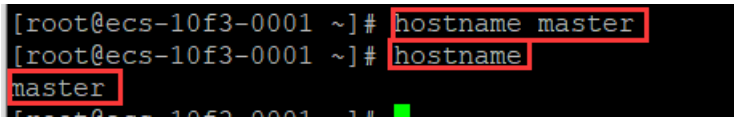


图 9

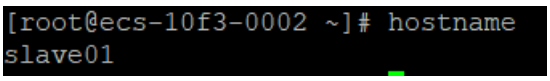


图 10

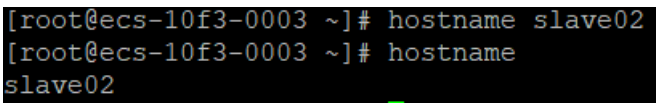


图 11

步骤 4：关闭防火墙，在中断分别输入如下两条命令，分别是临时关闭防火墙和禁止开机启动防火墙(3 个节点都需要处理)：

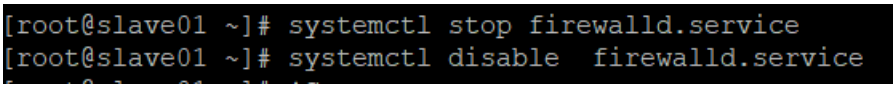


图 12

步骤 5：配置 hosts 列表(3 个节点都要配置，可以用 scp 复制到其他节点)

1)3 个节点分别运行 ifconfig 命令，获取当前节点的 ip 地址：

```
[root@master ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.138  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::f816:3eff:fe28:412  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:28:04:12  txqueuelen 1000  (Ethernet)
        RX packets 13617  bytes 17312859 (16.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2192  bytes 242000 (236.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 13

2)使用 vim /etc/hosts 命令编辑列表文件;

3)将下面三行添加到/etc/hosts 文件中，保存退出。这里 master 节点对应 IP 地址是 192.168.0.138，slave01 对应的 IP 是 192.168.0.245，slave02 对应的 IP 是 192.168.0.61,而自己在做配置时,需要将 IP 地址改成自己的 master、slave01 和 slave02 对应的 IP 地址。

```
192.168.0.138 master
192.168.0.245 slave01
192.168.0.61 slave02
```

图 14

4)测试是否能 ping 通：

```
[root@master ~]# ping slave01 -c 3
PING slave01 (192.168.0.245) 56(84) bytes of data.
64 bytes from slave01 (192.168.0.245): icmp_seq=1 ttl=64 time=1.14 ms
64 bytes from slave01 (192.168.0.245): icmp_seq=2 ttl=64 time=0.285 ms
64 bytes from slave01 (192.168.0.245): icmp_seq=3 ttl=64 time=0.152 ms

--- slave01 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 0.152/0.527/1.146/0.441 ms
[root@master ~]# ping slave02 -c 3
PING slave02 (192.168.0.61) 56(84) bytes of data.
64 bytes from slave02 (192.168.0.61): icmp_seq=1 ttl=64 time=0.905 ms
64 bytes from slave02 (192.168.0.61): icmp_seq=2 ttl=64 time=0.296 ms
64 bytes from slave02 (192.168.0.61): icmp_seq=3 ttl=64 time=0.168 ms

--- slave02 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2105ms
rtt min/avg/max/mdev = 0.168/0.456/0.905/0.322 ms
```

图 15

步骤 6：配置互信节点

1)三个 node 节点分别执行如下命令：

ssh-keygen -t rsa

```
[root@master ~]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:i696vAzlefk10CHa3zo3dtp2oMTtnUAhfEh586ugXTA root@master
The key's randomart image is:
+---[RSA 2048]----+
|          o.o    |
|           = =   |
|          . = +  |
|         o E o . |
|      . S o.=.  .|
|     o o o +o+o. |
|    ..+ + o.*o+o.|
|     ooo o oo+=o=|
|    .o+o. . .+.=o|
+----[SHA256]-----+
```

图 16

2)三个节点分别执行 cat /root/.ssh/id_rsa.pub 命令：

```
[root@slave01 ~]# cat /root/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC/UvfUfXPdHjue1sMEHilE9q4tE7+jygVORouoIKrW
8iGb1W3Cf9IoKesDl8mgV+A6QKpBEjusvw+p71ETocn7NlXmg09Y+7sEp1e03salLITdAIXS/kqhDiQv
mlIxcqtPiN1juLyiEuJK03mLDh79Hq3bss1eArd58n4RHpQQG7vJ8nIgmZnPKfltWc7SCDDONoulWYwu
71TW5r4X6jn14Rae5quq3OV7aF2cRwdc8ZdqgYn9HlzAQVbPi8w8a3yuJgkPP3KZoEePk/BNVpjt9h8C
WgWBx7o3aaEZmbuOMYuriTTpxrYPc0pcQiarstHVq6sw2kXK8nvJkfIYDBKN root@slave01
[root@slave01 ~]#
```

图 17

使用 vim /root/.ssh/authorized_keys 命令(每个节点都要使用这个命令，也可以一个节点使用，然后在该节点使用 scp 命令将文件拷贝到其他两个节点上，scp 命令操作如下图)将上述 3 个节点的执行结果进行粘贴。

```
[root@slave01 .ssh]# scp ./authorized_keys root@121.36.94.37:/root/.ssh
The authenticity of host '121.36.94.37 (121.36.94.37)' can't be established.
ECDSA key fingerprint is SHA256:vH6V82haEOM5AbvI89iQvrIb6GPaWNUiVUTBApMB9q8.
ECDSA key fingerprint is MD5:76:0f:74:dc:61:38:68:b4:d8:51:fd:18:51:a7:75:6d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '121.36.94.37' (ECDSA) to the list of known hosts.
root@121.36.94.37's password:
authorized_keys                               100% 1180   448.7KB/s   00:00
[root@slave01 .ssh]# scp ./authorized_keys root@119.3.254.204:/root/.ssh
The authenticity of host '119.3.254.204 (119.3.254.204)' can't be established.
ECDSA key fingerprint is SHA256:Dd2Z22LOwJjEGcoW9X5RT/7z5a5sKB97/QZ41ROJH7E.
ECDSA key fingerprint is MD5:43:c1:65:19:7d:e3:cb:e0:8b:67:5a:fa:82:19:a6:dd.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '119.3.254.204' (ECDSA) to the list of known hosts.
root@119.3.254.204's password:
authorized_keys                               100% 1180   445.7KB/s   00:00
[root@slave01 .ssh]#
```

图 18

步骤 7：安装 JDK(在三个节点分别执行此操作)

1)删除系统自带的 JDK

yum remove java-1.*

```
[root@master ~]# yum remove java-1.*
Loaded plugins: fastestmirror
Resolving Dependencies
--> Running transaction check
---> Package java-1.8.0-openjdk.aarch64 1:1.8.0.232.b09-0.el7_7 will be erased
---> Package java-1.8.0-openjdk-devel.aarch64 1:1.8.0.232.b09-0.el7_7 will be er
ased
---> Package java-1.8.0-openjdk-headless.aarch64 1:1.8.0.232.b09-0.el7_7 will be
 erased
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                     Arch     Version                 Repository
                                                                            Size
```

图 19

2)创建存放 JDK 文件目录：

mkdir /usr/java

3)将 JDK 压缩包(JDK 压缩包可通过 WINSCP 上传，或使用 wget 下载 https://github.com/AdoptOpenJDK/openjdk8-binaries/releases/download/jdk8u191-b12/OpenJDK8U-jdk_aarch64_linux_hotspot_8u191b12.tar.gz)解压到/usr/java：

```
[root@master ~]# tar -xzvf ./OpenJDK8U-jdk_aarch64_linux_hotspot_8u191b12.tar.gz
 -C /usr/java
```

图 20

4)配置环境变量(三个节点都要，可以复制到其他两个节点中)：

vim ./.bash_profile

在末尾添加：

export JAVA_HOME=/usr/java/jdk8u191-b12

export PATH=$JAVA_HOME/bin:$PATH

5)使环境变量生效：

source ./.bash_profile

6)查看 java 是否配置成功

java -version

```
[root@master ~]# java -version
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_191-b12)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.191-b12, mixed mode)
```

图 21

注：其他两个节点的 jdk 压缩包可以使用 scp 指令进行拷贝：scp 文件地址 服务器地址

```
[root@master ~]# scp ./OpenJDK8U-jdk_aarch64_linux_hotspot_8u191b12.tar.gz slave
01:/root
OpenJDK8U-jdk_aarch64_linux_hotspot_8u191b12. 100%   72MB 127.5MB/s   00:00
[root@master ~]# scp ./OpenJDK8U-jdk_aarch64_linux_hotspot_8u191b12.tar.gz slave
02:/root
OpenJDK8U-jdk_aarch64_linux_hotspot_8u191b12. 100%   72MB 122.2MB/s   00:00
[root@master ~]#
```

图 22

步骤 8：安装部署 Hadoop 集群

1)下载 Hadoop 安装包，执行命令在线下载安装包 wget https://archive.apache.org/dist/hadoop/ common/hadoop-2.7.3/hadoop-2.7.3.tar.gz(该方法比较慢，预计等待 40 分钟)或通过 WinSCP 上传至服务器，访问 https://winscp.net/eng/docs/lang:chs 下载安装 WinSCP(预计 2 分钟左右)。

2)上传完成后，使用 tar -xzvf 命令解压文件：

```
[root@master ~]# tar -xzvf hadoop-2.7.3.tar.gz
```

图 23

3)使用 vim 命令修改/root/hadoop-2.7.3/etc/hadoop 目录下的 hadoop-env.sh 文件：

```
vim hadoop-env.sh
```

图 24

在末尾追加:

export JAVA_HOME=/usr/java/jdk8u191-b12

4)使用 vim 指令修改 yarn-env.sh(/root/hadoop-2.7.3/etc/hadoop)文件：

```
vim yarn-env.sh
```

图 25

在开头添加：

export JAVA_HOME=/usr/java/jdk8u191-b12

5)用所给代码替换 core-site.xml(/root/hadoop-2.7.3/etc/hadoop)中的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
        <property>
```

```
                <name>fs.defaultFS</name>
                <value>hdfs://master:9000</value>
        </property>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/root/hadoopdata</value>
        </property>
</configuration>
```

6)用所给代码替换 hdfs-site.xml (/root/hadoop-2.7.3/etc/hadoop)中的内容：

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.namenode.secondary.http-address</name>
        <value>master:50090</value>
    </property>
    <property>
        <name>dfs.namenode.secondary.https-address</name>
        <value>master:50091</value>
    </property>
    <property>
        <name>dfs.permissions</name>
        <value>false</value>
        <description>
            If "true", enable permission checking in HDFS.
            If "false", permission checking is turned off,
            but all other behavior is unchanged.
            Switching from one parameter value to the other does not change the
mode,
```

```
                owner or group of files or directories.
            </description>
        </property>
</configuration>
```

7) 配置 yarn-site.xml(/root/hadoop-2.7.3/etc/hadoop)文件，用所给代码替换 yarn-site.xml 中的内容。

```
<?xml version="1.0"?>
<configuration>
        <property>
                <name>yarn.nodemanager.aux-services</name>
                <value>mapreduce_shuffle</value>
        </property>
        <property>
                <name>yarn.resourcemanager.address</name>
                <value>master:18040</value>
        </property>
        <property>
                <name>yarn.resourcemanager.scheduler.address</name>
                <value>master:18030</value>
        </property>
        <property>
                <name>yarn.resourcemanager.resource-tracker.address</name>
                <value>master:18025</value>
        </property>
        <property>
                <name>yarn.resourcemanager.admin.address</name>
                <value>master:18141</value>
        </property>
        <property>
                <name>yarn.resourcemanager.webapp.address</name>
                <value>master:18088</value>
```

```
    </property>
</configuration>
```

8)复制 mapred-site-template.xml 文件：

```
[root@master hadoop]# cp ~/hadoop-2.7.3/etc/hadoop/mapred-site.xml.template  ~/hado
op-2.7.3/etc/hadoop/mapred-site.xml
```

图 26

9)替换 mapred-site.xml 文件中的内容：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

10)使用 vim 编辑 slaves(/root/hadoop-2.7.3/etc/hadoop)文件：

vim slaves

修改内容如下(删除 localhost)：

slave01

slave02

11)创建 hadoop 数据目录：

/root/hadoopdata

12)将配置好的 hadoop(/root/)文件复制到从节点：

scp -r hadoop-2.7.3 slave01:/root/

scp -r hadoop-2.7.3 slave02:/root/

13)配置 Hadoop 环境变量：

使用 vim 打开.bash_profile(/root/)文件，在末尾添加如下内容：

#HADOOP

export HADOOP_HOME=/root/hadoop-2.7.3

export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH

14)使环境变量生效：

source ./.bash_profile

15)格式化 Hadoop 文件目录：

hdfs namenode -format

```
[root@master ~]# hdfs namenode -format
21/04/11 16:10:33 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = master/192.168.0.138
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.7.3
STARTUP_MSG:   classpath = /root/hadoop-2.7.3/etc/hadoop:/root/hadoop-2.7.3/share/h
adoop/common/lib/httpcore-4.2.5.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/jsch
-0.1.42.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/htrace-core-3.1.0-incubating
.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/jetty-6.1.26.jar:/root/hadoop-2.7.3
/share/hadoop/common/lib/curator-recipes-2.7.1.jar:/root/hadoop-2.7.3/share/hadoop/
common/lib/jackson-xc-1.9.13.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/servlet
-api-2.5.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/hamcrest-core-1.3.jar:/root
/hadoop-2.7.3/share/hadoop/common/lib/commons-cli-1.2.jar:/root/hadoop-2.7.3/share/
hadoop/common/lib/java-xmlbuilder-0.4.jar:/root/hadoop-2.7.3/share/hadoop/common/li
b/netty-3.6.2.Final.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/protobuf-java-2.
5.0.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/jets3t-0.9.0.jar:/root/hadoop-2.
7.3/share/hadoop/common/lib/jackson-core-asl-1.9.13.jar:/root/hadoop-2.7.3/share/ha
doop/common/lib/jsp-api-2.1.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/jackson-
jaxrs-1.9.13.jar:/root/hadoop-2.7.3/share/hadoop/common/lib/junit-4.11.jar:/root/ha
```

图 27

16)启动 hadoop 集群，运行如下指令：

start-all.sh

```
[root@master ~]# start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
21/04/11 16:26:32 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-
java classes where applicable
Starting namenodes on [master]
master: starting namenode, logging to /root/hadoop-2.7.3/logs/hadoop-root-namenode-master.out
slave01: starting datanode, logging to /root/hadoop-2.7.3/logs/hadoop-root-datanode-slave01.out
slave02: starting datanode, logging to /root/hadoop-2.7.3/logs/hadoop-root-datanode-slave02.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /root/hadoop-2.7.3/logs/hadoop-root-secondarynamenode-master.out
21/04/11 16:26:47 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-
java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /root/hadoop-2.7.3/logs/yarn-root-resourcemanager-master.out
slave01: starting nodemanager, logging to /root/hadoop-2.7.3/logs/yarn-root-nodemanager-slave01.out
slave02: starting nodemanager, logging to /root/hadoop-2.7.3/logs/yarn-root-nodemanager-slave02.out
[root@master ~]#
```

图 28

17) 在 master、slave01、slave02 的终端执行 jps 命令:

```
[root@master hadoop]# jps
23379 Jps
1316 WrapperSimpleApp
18742 NameNode
10219 SecondaryNameNode
19022 ResourceManager
[root@master hadoop]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.138  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::f816:3eff:fe28:412  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:28:04:12  txqueuelen 1000  (Ethernet)
        RX packets 1211937  bytes 1275654776 (1.1 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 544512  bytes 873614163 (833.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 29 实验截图

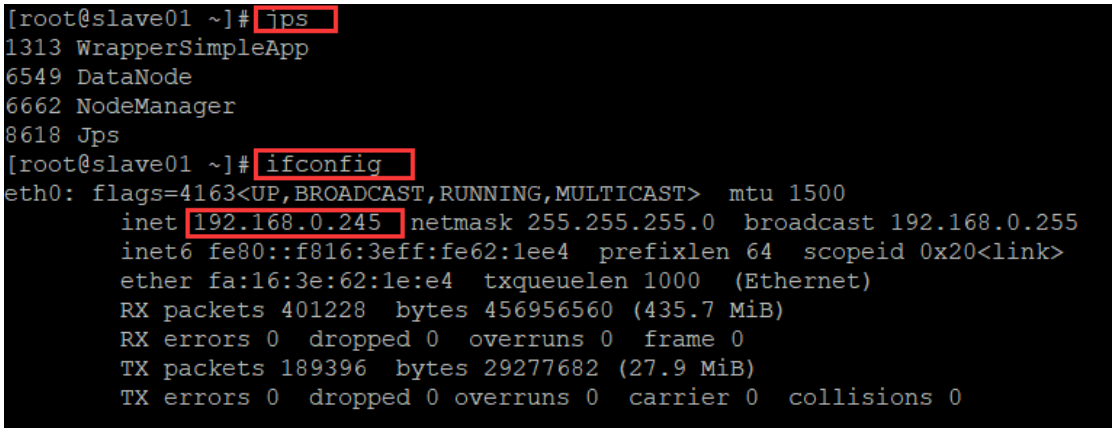
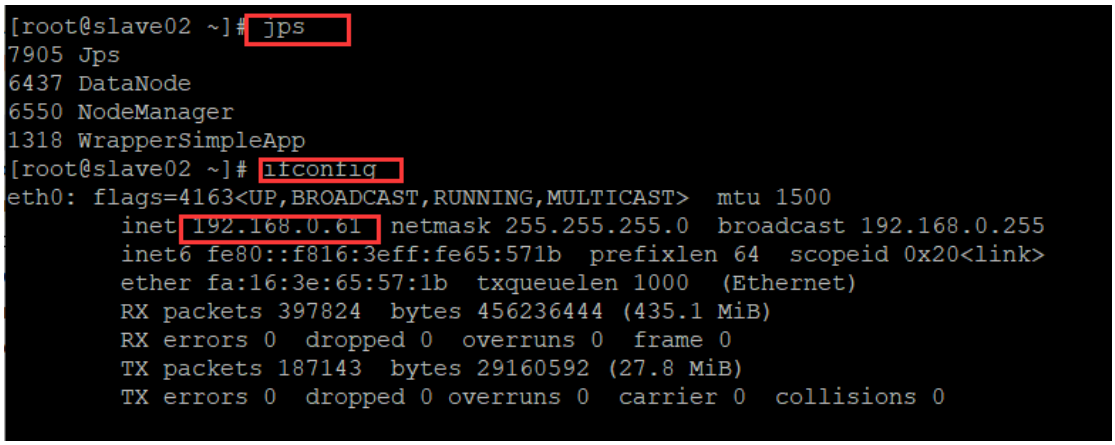


图 30 实验截图



注：显示的 ip(查重用)

图 31 实验截图

4.2 Spark 集群搭建(On Yarn 模式)

步骤 1：在 master 节点上解压 spark 压缩包：

1)使用 WinsCP 上传 spark 压缩包 spark-2.1.1-bin-hadoop2.7.tgz；

2)解压 spark 压缩包；

```
tar -xzvf spark-2.1.1-bin-hadoop2.7.tgz -C ./
```

步骤 2：配置环境变量：

1)返回 root 目录；

```
cd /root/
```

2)用 vim 编辑.bash_profile 文件：

```
vim .bash_profile
```

添加如下内容：

```
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

```
export HDFS_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
export PATH=$PATH:/root/spark-2.1.1-bin-hadoop2.7/bin
```

3) 运行 source 命令，重新编译.bash_profile，使添加变量生效:

source .bash_profile

步骤 3：配置 yarn-site.xml 文件

1)进入/root/hadoop-2.7.3/etc/hadoop 目录：

cd ./hadoop-2.7.3/etc/Hadoop

2)使用 vim 编辑 yarn-site.xml 添加如下内容：

```
<property>
    <name>yarn.nodemanager.pmem-check-enabled</name>
    <value>false</value>
</property>
<property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>false</value>
</property>
```

3)将 yarn-site.xml 文件发送到从节点：

scp yarn-site.xml slave01:/root/hadoop-2.7.3/etc/hadoop

yarn-site.xml

scp yarn-site.xml slave02:/root/hadoop-2.7.3/etc/hadoop

yarn-site.xml

步骤 4：重启 Hadoop 集群：

1)stop-all.sh(如果你已经启动了 hadoop 需要停止)；

2)start-all.sh

3)使用 jps 查看集群是否启动成功(结果应与图 29-图 31 一致)；

步骤 5：运行如下指令检验 spark 是否部署成功：

```
spark-submit --class org.apache.spark.examples.SparkPi --master yarn --num-executors 3 --driver-memory 1g --executor-memory 1g --executor-cores 1 spark-2.1.1-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.1.1.jar 10
```

输出如下结果，集群部署成功：

```
21/04/14 10:37:52 INFO scheduler.DAGSchedule
Pi is roughly 3.143815143815144
21/04/14 10:37:52 INFO server.ServerConnecto
21/04/14 10:37:52 INFO handler.ContextHandle
```

4.3 Scala 程序编写

步骤 1：创建项目，打开 IDEA(IDEA 需要在自己电脑上安装)，创建工程：

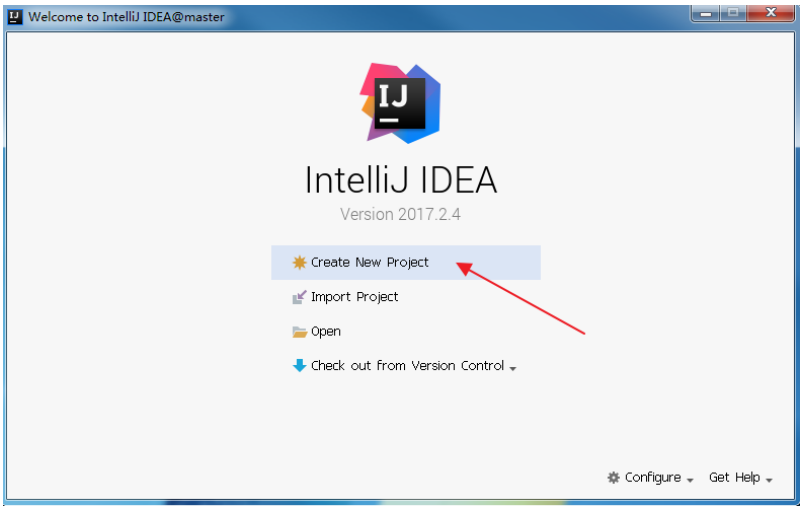


图 32

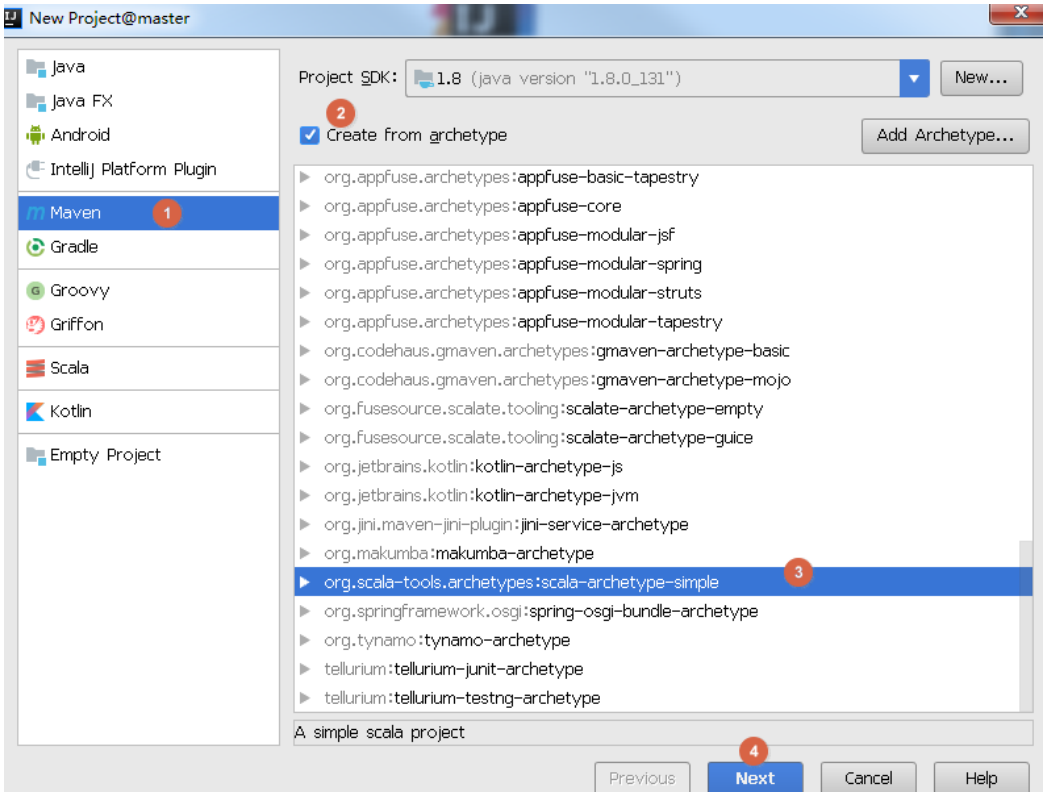


图 33

依次输入 GroupId、ArtifactId 和 Version 的值，然后点击 next；

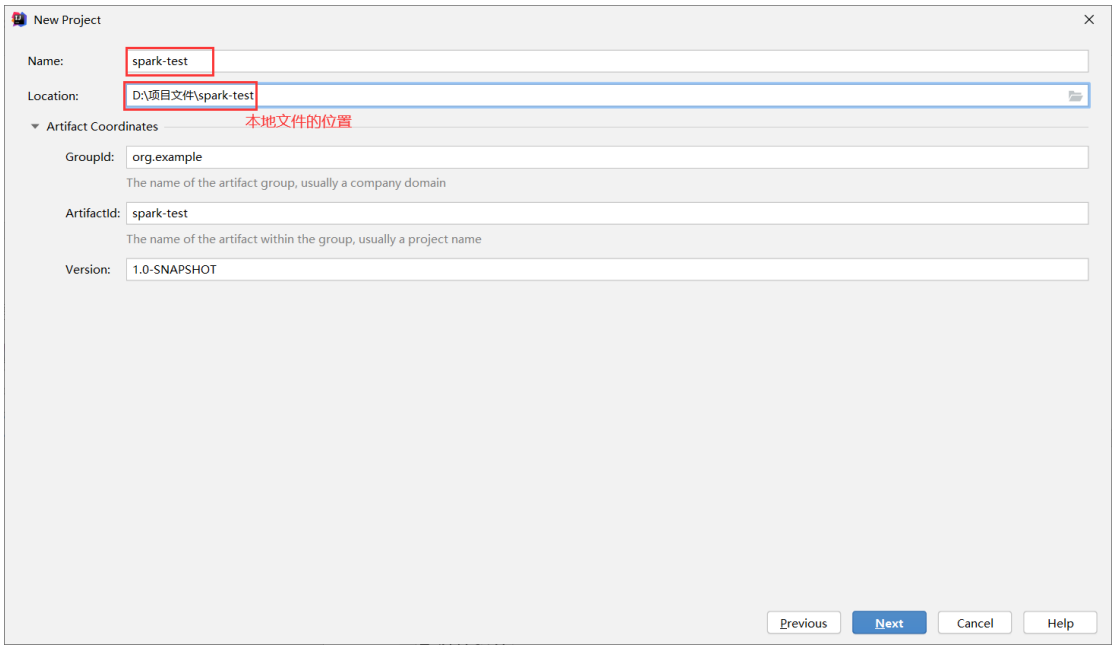


图 34

进入如下界面表示工程创建成功：

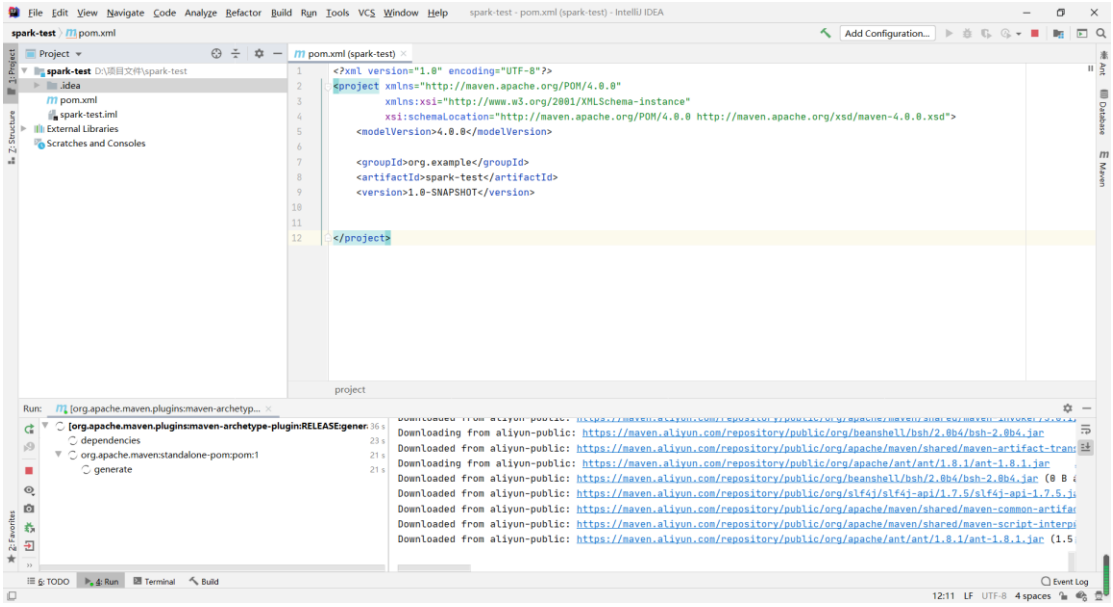


图 35

步骤 2：依赖设置：

1)在 pom.xml 文件中找到 properties 配置项，修改 scala 版本号（此处对应 scala 安装版本），并添加 spark 版本号（此处对应 spark 安装版本）；

```
<groupId>org.zkpk.tab</groupId>
<artifactId>zkpk</artifactId>
<version>1.0</version>
<inceptionYear>2008</inceptionYear>
<properties>
  <scala.version>2.11.11</scala.version>
  <spark.version>2.1.1</spark.version>
</properties>

<repositories>
  <repository>
    <id>scala-tools.org</id>
```

图 36

2)找到 dependency 配置项，添加如下图标红部分的配置，分别是 scala 依赖和 spark 依赖，\${scala.version}表示上述配置的 scala.version 变量；

```
25      </pluginRepository>
26    </pluginRepositories>
27
28    <dependencies>
29      <dependency>
30        <groupId>org.scala-lang</groupId>
31        <artifactId>scala-library</artifactId>
32        <version>${scala.version}</version>
33      </dependency>
34      <dependency>
35        <groupId>org.apache.spark</groupId>
36        <artifactId>spark-core_2.11</artifactId>
37        <version>${spark.version}</version>
38      </dependency>
39      <dependency>
40        <groupId>org.apache.spark</groupId>
41        <artifactId>spark-sql_2.11</artifactId>
42        <version>${spark.version}</version>
43      </dependency>
44
45    </dependencies>
```

图 37

一般修改 pom.xml 文件后，会提示 enable auto-import，点击即可，如果没有提示，则可以点击工程名，依次选择 Maven—->Reimport，即可根据 pom.xml 文件导入依赖包；

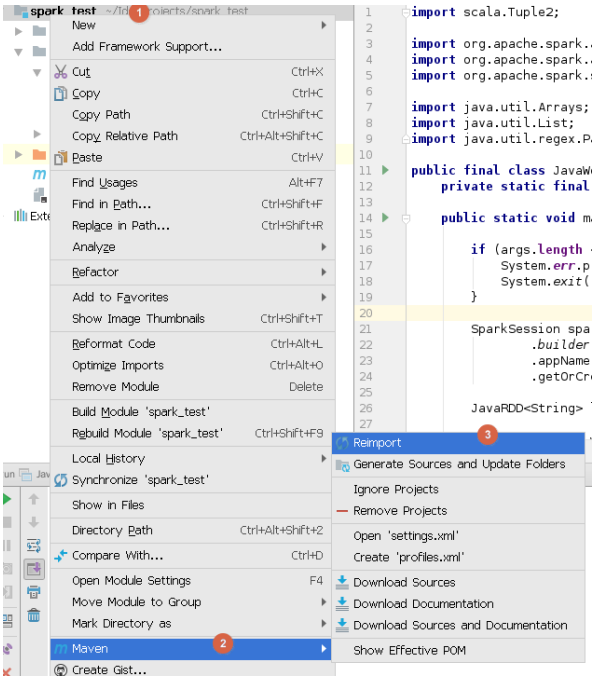


图 38

步骤 3：设置语言环境：

1）设置语言环境 language level，点击菜单栏中的 file，选择 Project Structure；

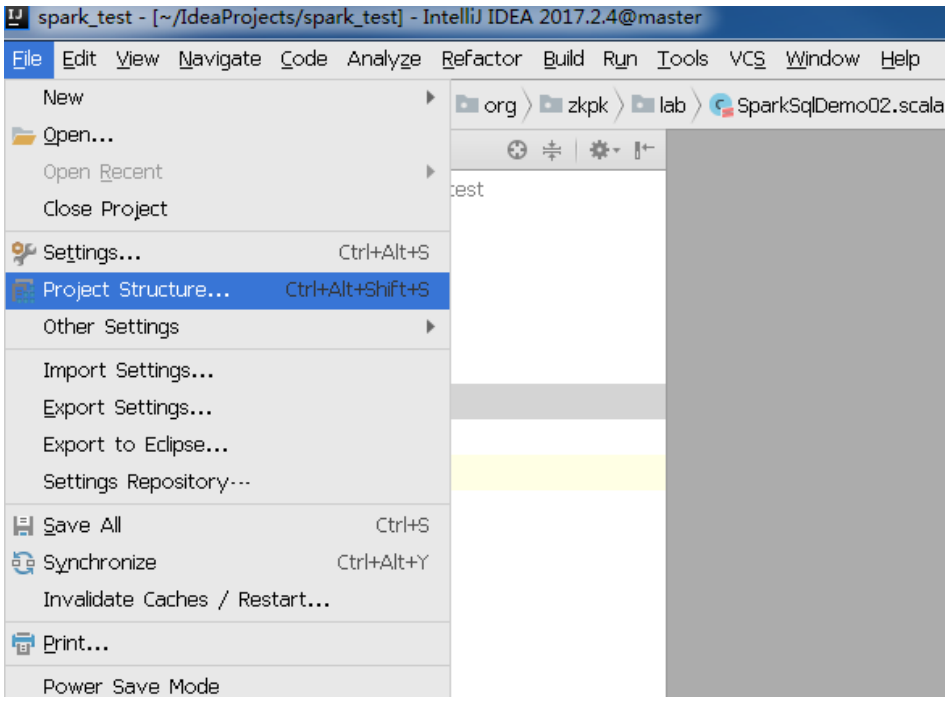


图 40

弹出如下对话框，选择 Modules，选择 Language level 为 8，然后点击 Apply，点击 OK；

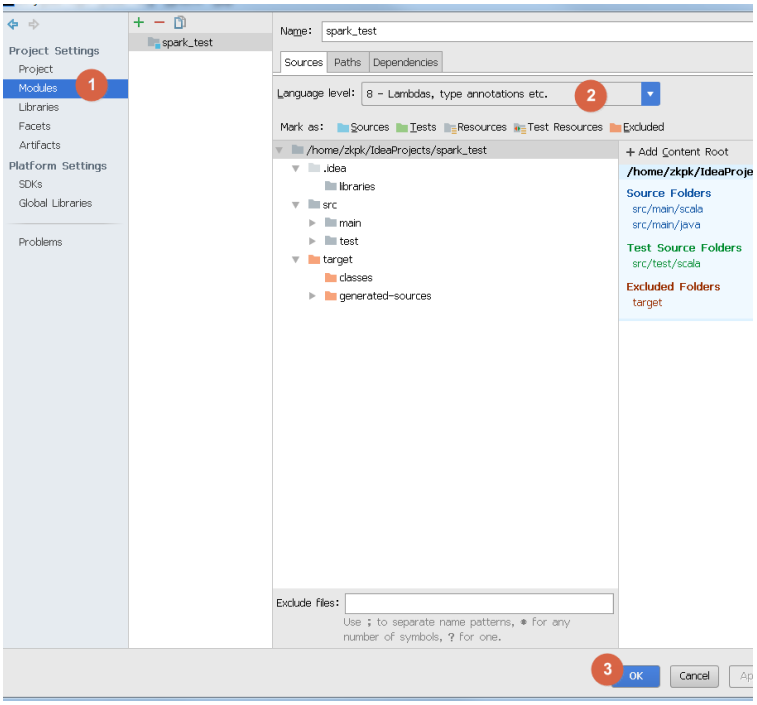


图 41

步骤 4：设置 java Compiler 环境：

1)点击菜单栏中的 file，选择 Setting；

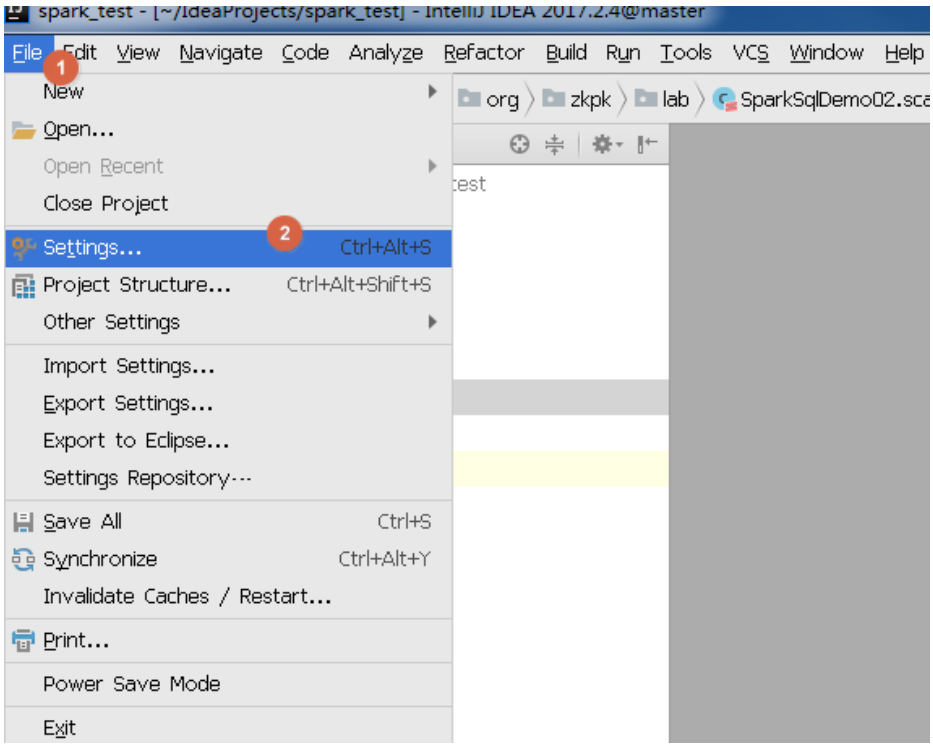


图 42

2）弹出如下对话框，依次选择 Build，Execution—->Compiler—->Java Compiler，设置图中的 Project bytecode version 为 1.8，设置图中的 Target bytecode version 为 1.8，然后依次点击 Apply 和 OK；

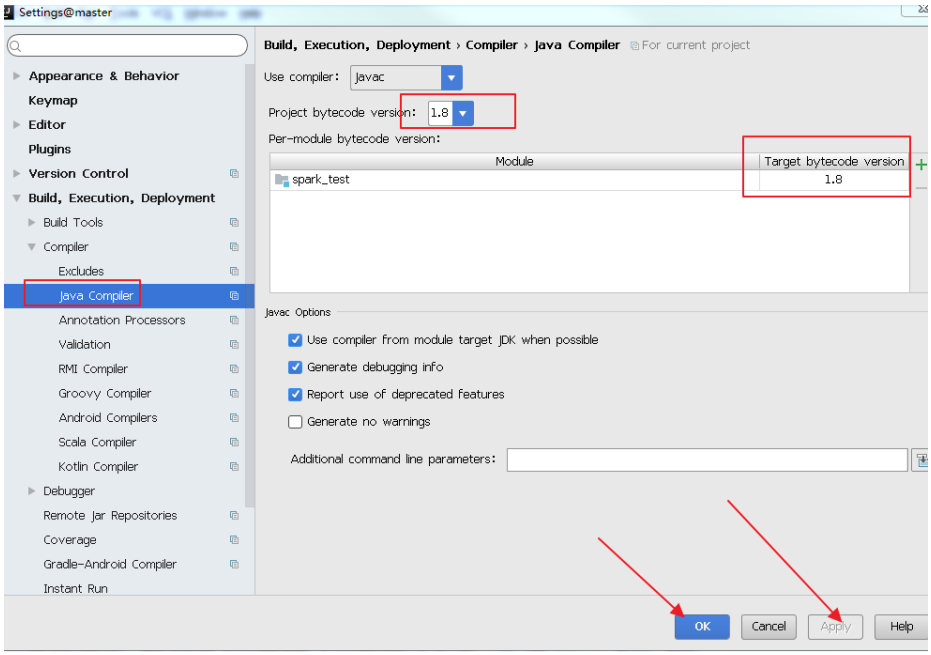


图 43

步骤 5：文件配置：

1）如下图删除测试环境 test 中的测试类；

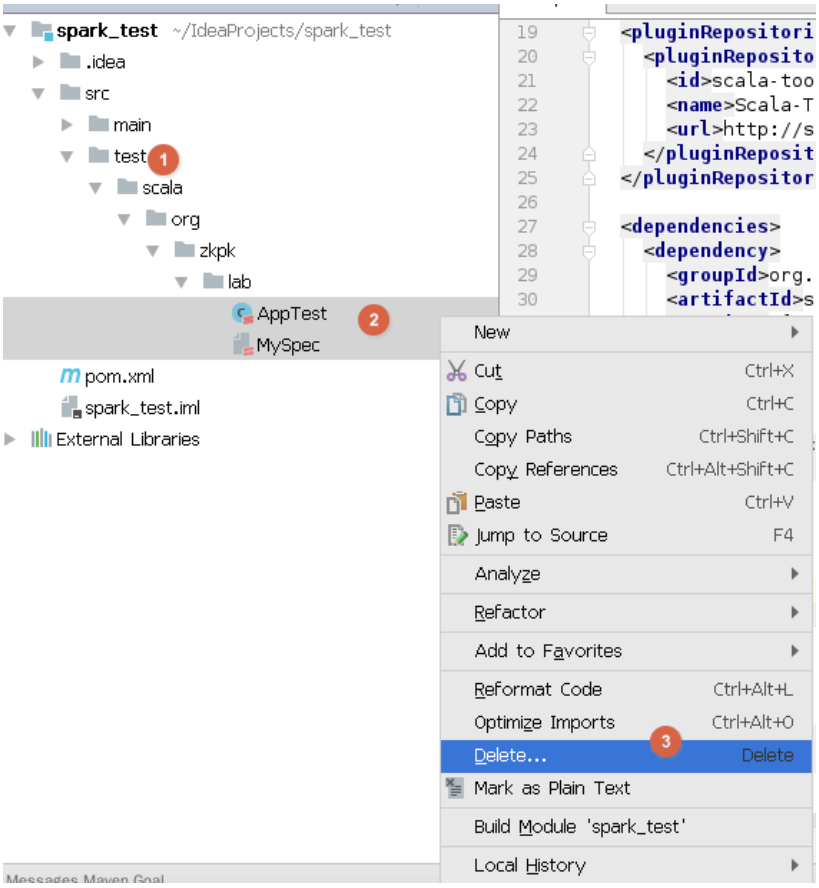


图 44

2）如下图删除，main 文件夹中，包名下的 App 文件；

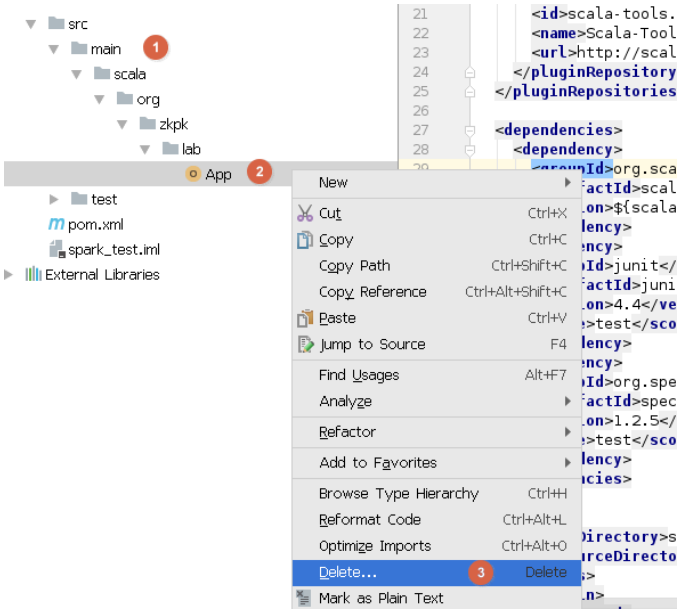


图 45

步骤 6：程序编写：

1) 6.3.1 如下图依次打开 src—>main—>scala，在 org.zkpk.lab 上点击右键，创建 Scala Class；

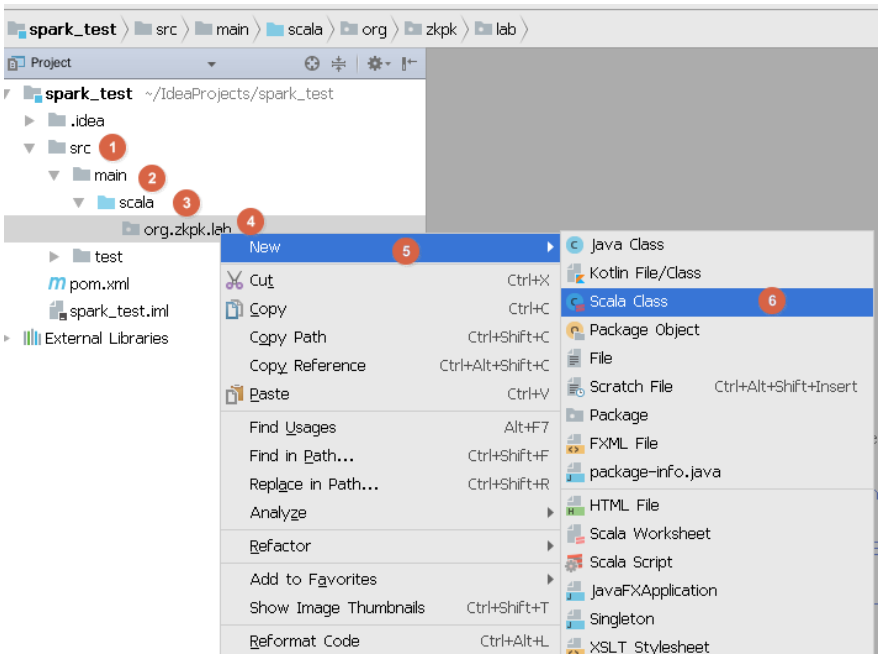


图 46

2) 弹出如下对话框，输入类名 ScalaWordCount，点击 ok

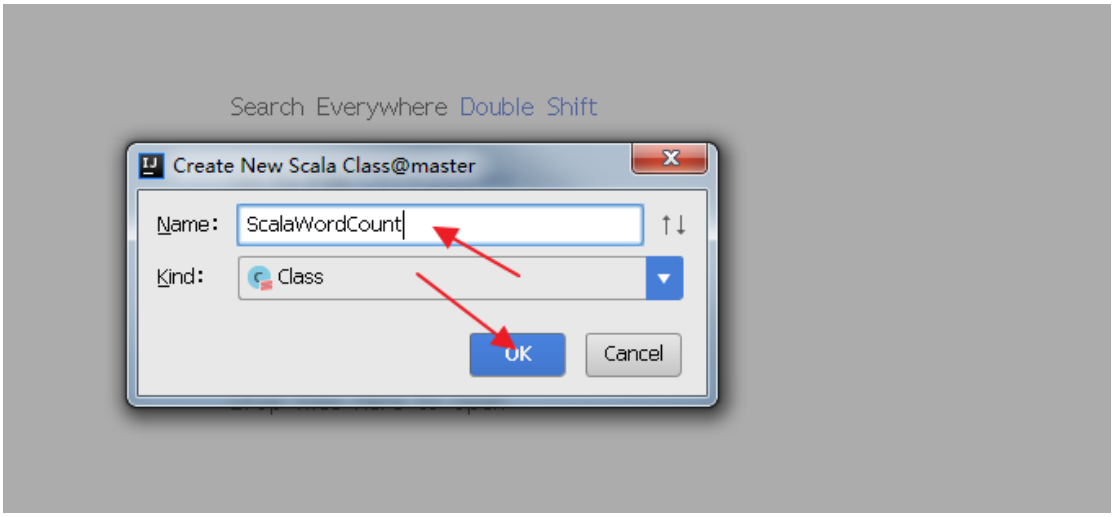


图 47

3)在类 ScalaWordCount 中新建伴生对象 object ScalaWordCount；

```
3  class ScalaWordCount{
4
5  }
6  object ScalaWordCount {
```

图 48

4) 在伴生对象 object ScalaWordCount 中创建 main 方法；

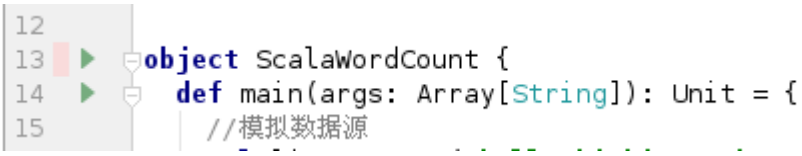


图 49

5）在 main 方法中创建列表 List 对象并赋值给常量 list，列表中包含 4 个元素，分别是：“hello hi hi spark”，“hello spark hello hi sparksql”，“hello hi hi sparkstreaming”，“hello hi sparkgraphx”；

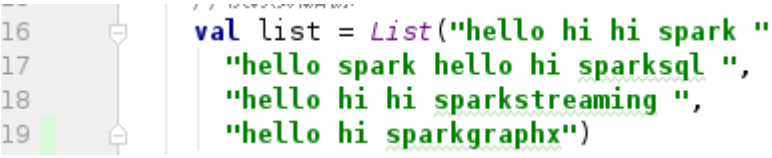


图 50

6）创建 SparkConf 对象，对 Spark 运行属性进行配置，调用该对象的 setAppName 方法设置 Spark 程序的名称为“word-count”，调用 setMaster 方法设置 Spark 程序运行模式，一般分为两种：本地模式和 yarn 模式，这里我们采用本地模式，参数为“local[*]”，属性设置完成后赋值给常量 sparkConf；

7）创建 SparkContext，参数为 sparkconf，赋值给常量 sc，该对象是 Spark 程序的入口；

```
val sparkConf = new SparkConf().setAppName("word-count").setMaster("yarn")
val sc = new SparkContext(sparkConf)
```

图 51

8）调用 SparkContext 对象 sc 的方法 parallelize，参数为列表对象 list，该方法使用现成的 scala 集合生成 RDD lines，类型为 String（RDD 为 Spark 计算中的最小数据单元），该 RDD 存储元素是字符串语句；

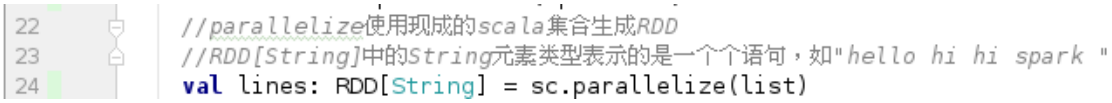


图 52

9）调用 RDD 对象 lines 的 flatMap 方法按照空格切分 RDD 中的字符串元素，并存入新的 RDD 对象 words 中，参数类型为 String，该 RDD 存储的元素是每一个单词；

```
25        //RDD[String]中的元素类型String表示的是一个个的单词
26        val words: RDD[String] =
27            lines.flatMap((line: String) => {line.split(" ")})
```

图 53

10）调用 RDD 对象 words 的 map 方法，将 RDD 中的每一个单词转换为 kv 对，key 是 String 类型的单词，value 是 Int 类型的 1，并赋值给新的 RDD 对象 wordAndOne，参数为（String，Int）类型键值对；

```
28        //将word变成kv对,(word, 1)
29        val wordAndOne: RDD[(String, Int)] =
30            words.map((word: String) => {(word, 1)})
```

图 54

11）调用 RDD 对象 wordAndOne 的 reduceByKey 方法，传入的参数为两个 Int 类型变量，该方法将 RDD 中的元素按照 Key 进行分组，将同一组中的 value 值进行聚合操作，得到 valueRet，最终返回（key，valueRet）键值对，并赋值给新的 RDD 对象 wordAndNum，参数为（String，Int）类型键值对；

```
31        //reduceByKey就按照key进行分组，将同一组中的value进行聚合操作
32        //得到值valueRet,最终返回(key,valueRet)的键值对
33        val wordAndNum: RDD[(String, Int)] =
34            wordAndOne.reduceByKey((count1: Int, count2: Int) => {count1 + count2})
```

图 55

12）调用 RDD 对象 wordAndNum 的 sortBy 方法，第一个参数为 kv 对中的 value，即单词出现次数，第二个参数为 boolean 类型，true 表示升序，false 表示降序；

```
36
37        val ret = wordAndNum.sortBy(kv => kv._2, false)
38
```

图 56

13）调用 ret 对象的 collect 方法，获取集合中的元素，再调用 mkString 方法，参数为“，”，将集合中的元素用逗号连接成字符串，调用 println 方法打印输出在控制台；

```
39
40        println(ret.collect().mkString(","))
```

图 57

14）调用 ret 对象的 saveAsTextFile，该方法的参数为运行时指定的参数，此方法的用处是将 Spark 程序运行的结果保存到指定路径，一般是把结果保存到 HDFS 中，所以这里的参数定义为；

hdfs://master:9000/spark_test

HDFS 根目录中不存在 spark_test 此目录，spark 程序会自动创建该目录；调用 SparkContext 对象 sc 的 stop 方法，释放 spark 程序所占用的资源；

```
ret.saveAsTextFile( path = "hdfs://master:9000/spark_test")
sc.stop()
```

图 58

15)完整程序如下：

```
package org.example
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
class ScalaWordCount {
}
object ScalaWordCount{
  def main(args: Array[String]): Unit = {
    val list = List("hello hi hi spark",
      "hello spark hello hi sparksql",
      "hello hi hi sparkstreaming",
      "hello hi sparkkgraphx")
    val sparkConf = new SparkConf().setAppName("word-count").setMaster("yarn")
    val sc = new SparkContext(sparkConf)
    val lines:RDD[String] = sc.parallelize(list)
    val words:RDD[String] = lines.flatMap((line:String) => {line.split( regex = " ")})
    val wordAndOne:RDD[(String, Int)] = words.map((word:String)=>{(word, 1)})
    val wordAndNum:RDD[(String, Int)] = wordAndOne.reduceByKey((count1:Int, count2:Int)=>{count1 + count2})
    val ret = wordAndNum.sortBy(kv => kv._2, ascending = false)
    print(ret.collect().mkString(","))
    ret.saveAsTextFile( path = "hdfs://master:9000/spark_test")
    sc.stop()
  }
}
```

图 59

4.5 程序打包与运行

步骤 1：打开 File->Project Structure：

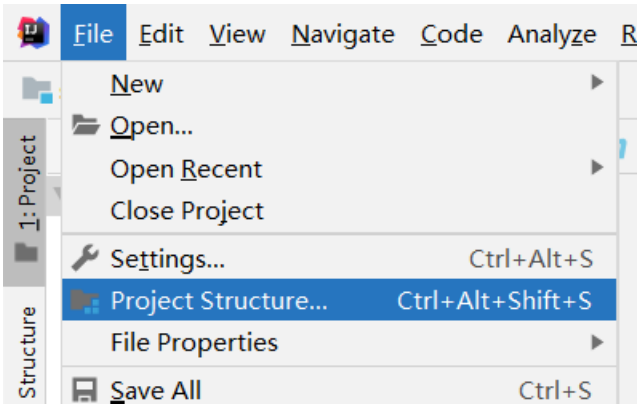


图 60

步骤 2：Project Settings 栏下的 Artifacts，点击“+”，选择 JAR->From modules with dependencies…:

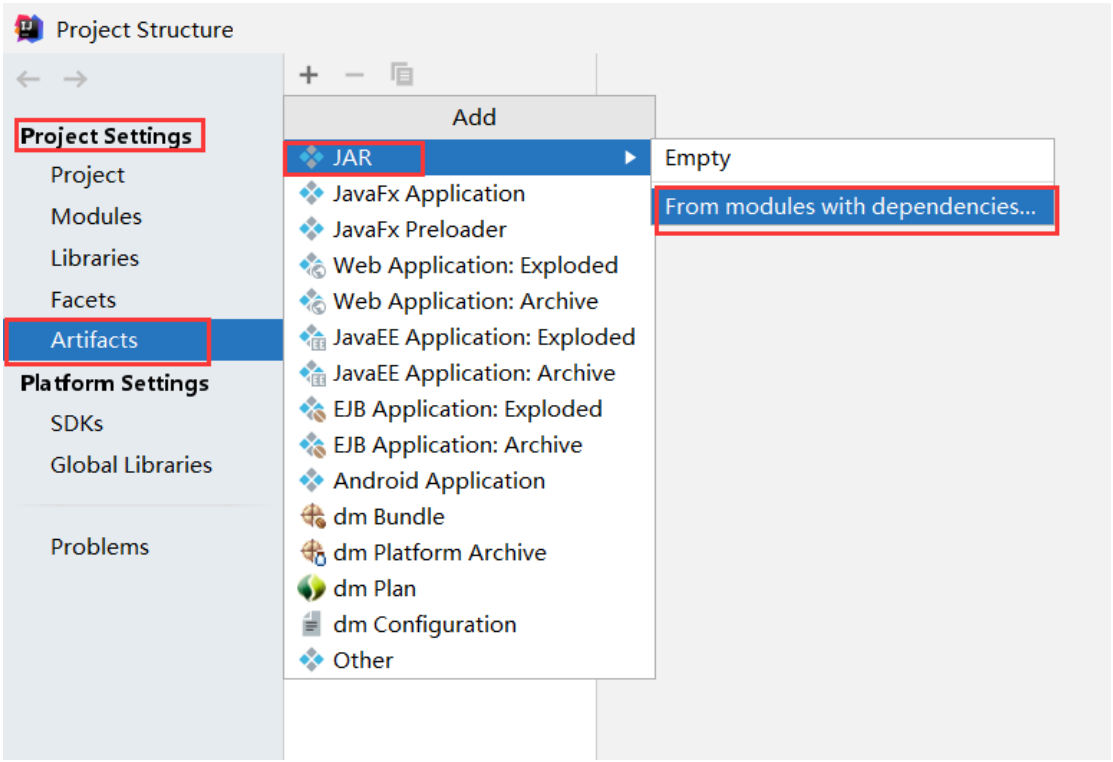


图 61

步骤 3：填写主类名称：

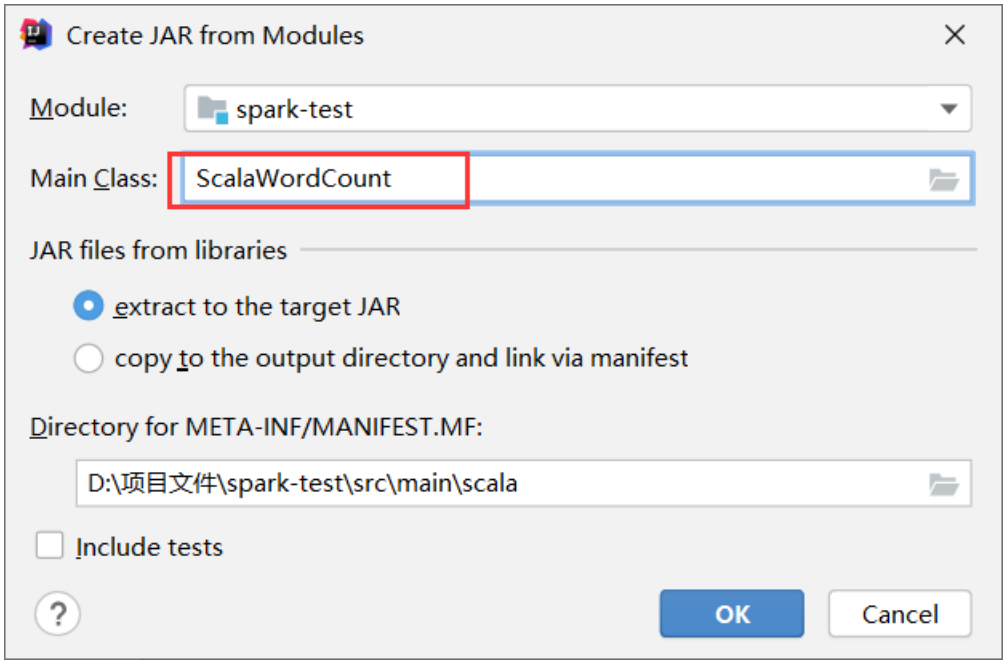


图 62

步骤 4：选择 Build->Artifacts:

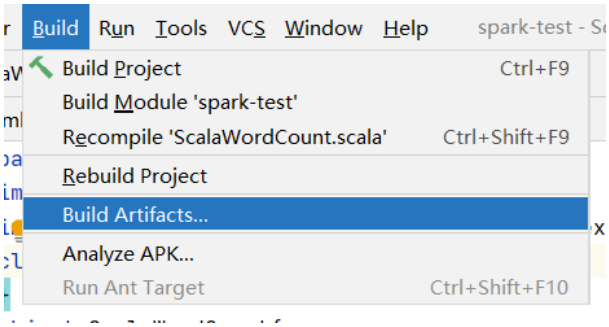


图 63

步骤 5：选择 Build：

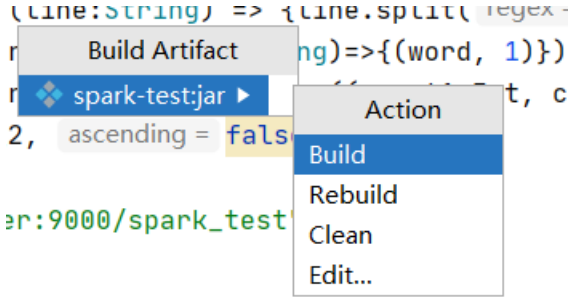


图 64

建立完成：

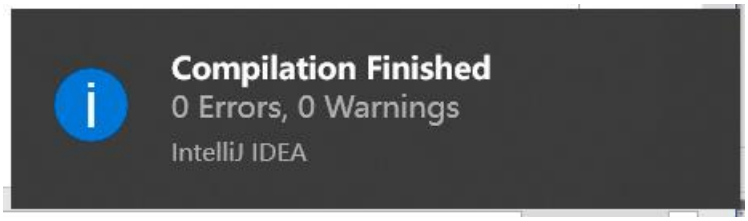


图 65

步骤 6：使用压缩软件打开生成的 jar 包：

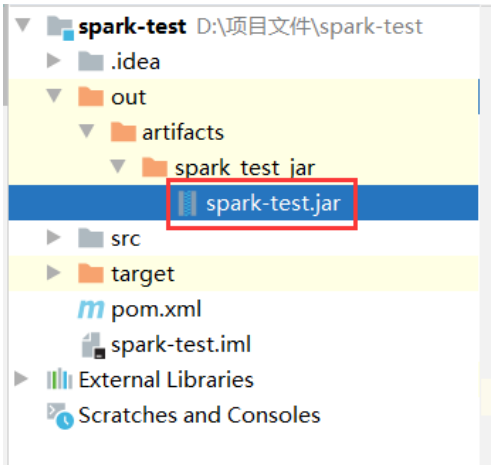


图 66

步骤 7：找到 META-INF 目录

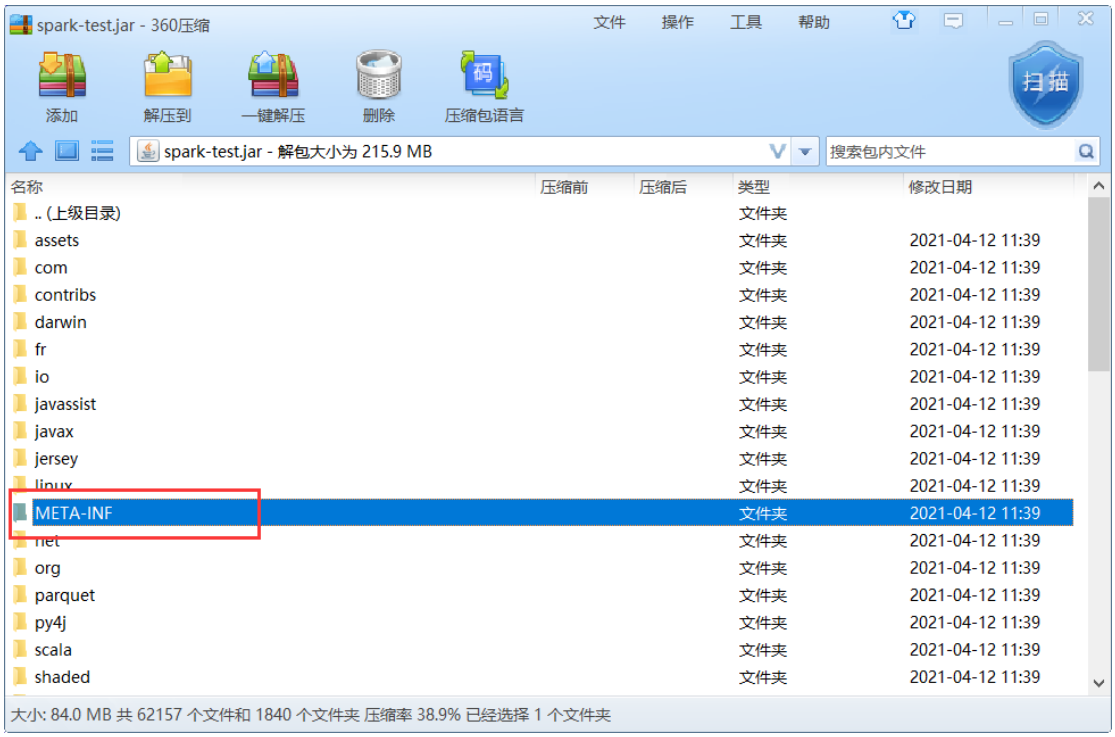


图 67

步骤 8：删除 MANIFEST.MF 文件

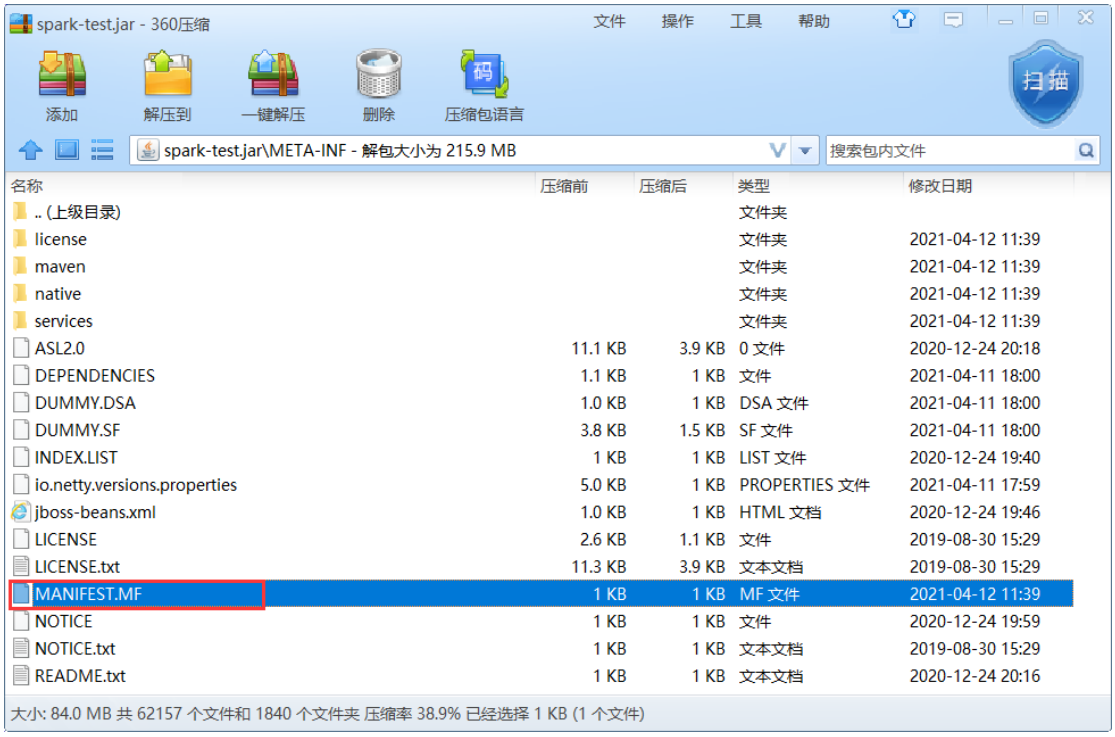


图 68

步骤 9：使用 WinSCP 上传处理后的 jar 包到服务器：

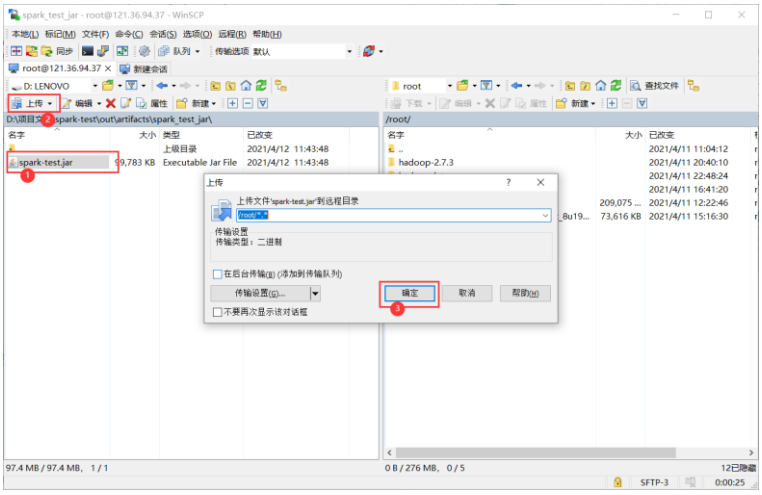


图 69

步骤 10：使用 spark-submit 命令，在 hadoop 运行程序：

```
spark-submit --class org.example.ScalaWordCount --master yarn --num-executors 3 --driver-memory 1g --executor-memory 1g --executor-cores 1 spark-test.jar
```

图 70

得到如下结果：

```
21/04/14 10:46:04 INFO scheduler.DAGScheduler: Job 1 finished: collect at ScalaWordCount.scala:19, took 0.080027
(hi,6),(hello,5),(spark,2),(sparkkgraphx,1),(sparkstreaming,1),(sparksql,1)21/04/14 10:46:04 INFO storage.BlockMa
t 3 piece0 on 192.168.0.138:44381 in memory (size: 2023.0 B, free: 366.3 MB)
```

图 71

步骤 11：在 hdfs 上查看程序的输出：

```
[root@master ~]# hadoop fs -ls /
21/04/12 11:51:17 WARN util.NativeCodeLoader: Unable to load native-hado
re applicable
Found 3 items
drwxr-xr-x   - root supergroup          0 2021-04-12 11:49 /spark_test
drwx------   - root supergroup          0 2021-04-11 22:49 /tmp
drwxr-xr-x   - root supergroup          0 2021-04-11 22:49 /user
[root@master ~]#
```

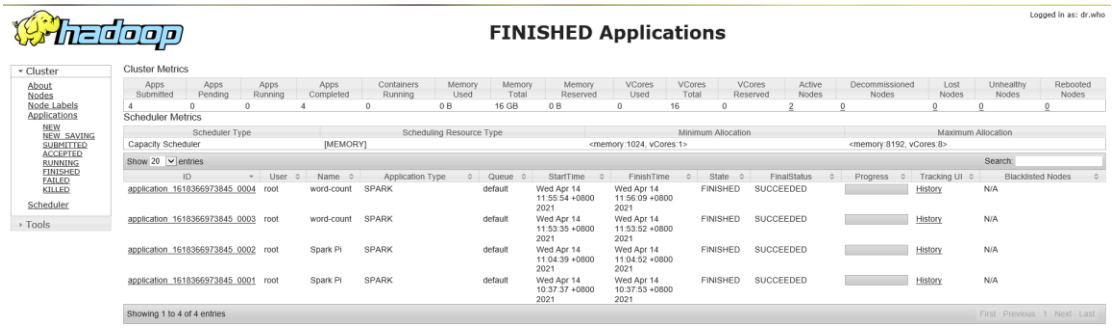
图 72

```
[root@master ~]# hadoop fs -cat /spark_test/part-00000
21/04/12 11:51:39 WARN util.NativeCodeLoader: Unable to load native-hadoop lib
re applicable
(hi,6)
(hello,5)
(spark,2)
[root@master ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.138  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::f816:3eff:fe28:412  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:28:04:12  txqueuelen 1000  (Ethernet)
        RX packets 1209538  bytes 1275250010 (1.1 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
```

注：第二步需要打印主服务器的 ip 地址(查重用)

图 73 实验截图

步骤 12(选做)：登录 http://master:18088，可以查看任务完成情况：



注：master 应替换为 master 节点的 ip 地址，18088 需要在华为云配置安全组，开放 18088 接口。

图 74

五、实验结果与分析

请按照实验要求撰写实验报告。

六、需要注意的问题

序号	问题	解决方案
1	关于环境	在客户端打包程序的时候，使用实验提供的环境，要求 jdk 版本为 1.8，否则会出错；
2	新建类，无 Scala.class	https://blog.csdn.net/qq_16410733/article/details/85038832
3	关于防火墙	一定要检查三个节点是否关闭了防火墙；
4	文件拷贝	注意主节点配置的文件是否全部拷贝给了数据节点，主要包括(Hadoop 文件夹、hosts、.bash_profile)
5	Hadoop 需要进一步验证是否能够正常	运行：hadoop jar ./hadoop-2.7.3/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar pi 10 10 输出结果：3.2000

	使用	
6	关于 web 页面	web 页面默认在本地端是无法打开的，需要配置华为云的安全组，开放指定的端口，如：50070