大数据平台搭建验证方案

# 所有用到的环境

VMware® Workstation 12 Pro 12.5.2 build-4638234

CentOS release 6.5 (Final)

jdk-8u162-linux-x64

hadoop-2.7.6

spark-2.3.0-bin-hadoop2.7

scala-2.12.6

apache-hive-2.3.0-bin

python3.6.6

pyspark==2.3.0

py4j 0.10.6

pycharm professional 2.5

sqoop-1.4.7.bin\_\_hadoop-2.6.0

# Hadoop安装

## 固定IP和hostname先

先把Redhat的NetworkManager停掉先

service NetworkManager stop

把Redhat的NetworkManager永久停掉

chkconfig NetworkManager off

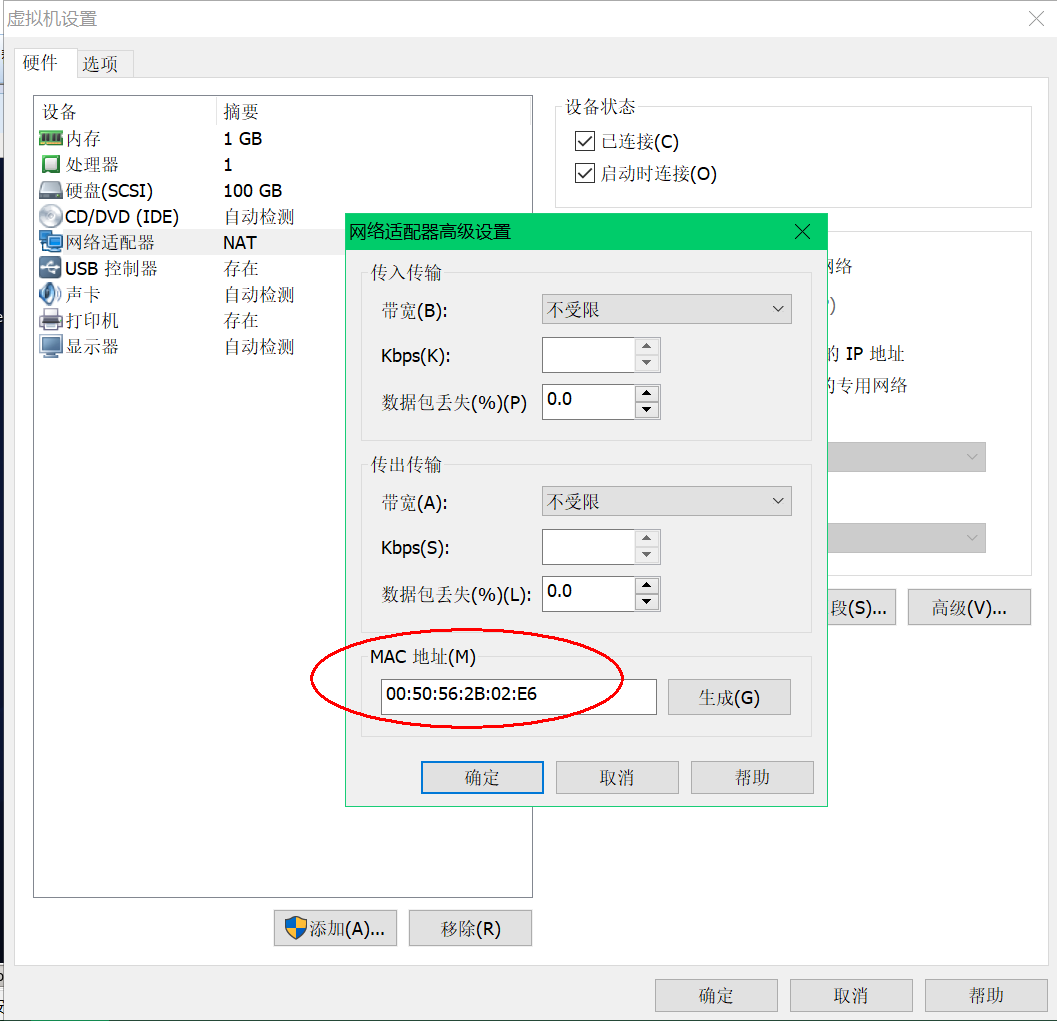
ifconfig –a看看当前生效的是哪张网卡

vim /etc/udev/rules.d/70-persistent-net.rules

SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?\*", ATTR{address}=="00:50:56:2B:02:E6", ATTR{type}=="1", KERNEL=="eth\*", NAME="eth0"

如果最后一条不是eth0，要将其他的记录删掉，保留那条激活的。将这条激活的MAC地址更新到VMWare的配置上。然后将NAME改为eth0

从VMWare界面查看网卡的MAC地址



为网卡产生一个UUID

uuidgen eth0

f2e44eb5-1fe2-4bc6-975c-961789ff0311

将上面的eth0和MAC地址更新到/etc/sysconfig/network-scripts里的ifcfg-eth0里

| DEVICE=eth0 HWADDR=00:0c:29:d1:04:ce TYPE=Ethernet UUID= 6e98108d-0a36-47b0-adcc-3335c9558a98 ONBOOT=yes NM\_CONTROLLED=yes BOOTPROTO=static IPADDR=10.25.0.103 NETMASK=255.255.255.0 GATEWAY=10.25.0.2 DNS1=10.25.0.2 DNS2=10.25.0.2 |
| --- |
|  |

|  |
| --- |

重启网络service network restart

重启主机 reboot

设置hostname

vi /etc/hosts

10.25.0.101 Hadoop-master

10.25.0.102 Hadoop-node1

10.25.0.103 Hadoop-node2

vi /etc/sysconfig/network

NETWORKING=yes

HOSTNAME=Hadoop-master

## 关掉防火墙

chkconfig iptables off

vi /etc/selinux/config

SELINUX=enforcing改为SELINUX=disabled

## 配置免密认证

每一台机器都执行

确定ssh已安装

rpm -qa|grep ssh

ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa // 在~/.ssh/下生成公钥/私钥对 -P '' 密码为空

cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys // 把公钥添加到authorzied\_keys文件里

chmod 0600 ~/.ssh/authorized\_keys // 修改authorized\_keys的权限

ssh localhost // 验证ssh是否可以访问

在Hadoop-master上执行：

scp .ssh/id\_rsa.pub Hadoop-node1:~/.ssh/master\_rsa.pub

在Hadoop-node1上执行

cat ~/.ssh/master\_rsa.pub >> ~/.ssh/authorized\_keys

在Hadoop-master上执行：

ssh Hadoop-node1

在Hadoop-master上执行：

scp .ssh/id\_rsa.pub Hadoop-node2:~/.ssh/master\_rsa.pub

在Hadoop-node2上执行

cat ~/.ssh/master\_rsa.pub >> ~/.ssh/authorized\_keys

---------------------------

在Hadoop-node1上执行

scp .ssh/id\_rsa.pub Hadoop-master:~/.ssh/node1\_rsa.pub

在Hadoop-master上执行：

cat ~/.ssh/node1\_rsa.pub >> ~/.ssh/authorized\_keys

在Hadoop-node1上执行

scp .ssh/id\_rsa.pub Hadoop-node2:~/.ssh/node1\_rsa.pub

在Hadoop-node2上执行：

cat ~/.ssh/node1\_rsa.pub >> ~/.ssh/authorized\_keys

---------------------------

在Hadoop-node2上执行

scp .ssh/id\_rsa.pub Hadoop-master:~/.ssh/node2\_rsa.pub

在Hadoop-master上执行：

cat ~/.ssh/node2\_rsa.pub >> ~/.ssh/authorized\_keys

在Hadoop-node2上执行

scp .ssh/id\_rsa.pub Hadoop-node1:~/.ssh/node2\_rsa.pub

在Hadoop-node1上执行：

cat ~/.ssh/node2\_rsa.pub >> ~/.ssh/authorized\_keys

## 三台机器安装JDK1.8

mkdir –p /java

cd /java

tar -zxvf jdk-8u162-linux-x64.tar.gz

vim /etc/profile

export JAVA\_HOME=/java/jdk1.8.0\_162

export CLASSPATH=.:$JAVA\_HOME/jre/lib/rt.jar:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

export PATH=.:$JAVA\_HOME/bin:$PATH

source /etc/profile

## 三台机器安装hadoop

mkdir –p /hadoop/

cp hadoop-2.7.6.tar.gz /hadoop/

cd /hadoop

tar –zxvf hadoop-2.7.6.tar.gz

vi /etc/profile

export HADOOP\_HOME=/hadoop/hadoop-2.7.6

export PATH=.:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin:$JAVA\_HOME/bin:$PATH

source /etc/profile

hadoop version

## 三台机器配置hadoop

参照：http://hadoop.apache.org/docs/r2.7.6/

三台机器配置成一模一样的就可以：

/hadoop/hadoop-2.7.6/etc/hadoop

建立三个目录

mkdir -p /hadoop/hadoop-2.7.6/tmp

### (1) 配置hadoop-env.sh

vim hadoop-env.sh

export JAVA\_HOME=/java/jdk1.8.0\_162

### (2) 配置core-site.xml

vim core-site.xml

fs.defaultFS ： 这个属性用来指定namenode的hdfs协议的文件系统通信地址，可以指定一个主机+端口，也可以指定为一个namenode服务（这个服务内部可以有多台namenode实现ha的namenode服务

hadoop.tmp.dir : hadoop集群在工作的时候存储的一些临时文件的目录

| <configuration> <!-- 指定HDFS中NameNode的地址 --> <property> <name>fs.defaultFS</name> <value>hdfs://Hadoop-master:9000</value> </property> <!-- 指定hadoop运行时产生文件的存储目录 --> <property> <name>hadoop.tmp.dir</name> <value>/hadoop/hadoop-2.7.6</value> </property> </configuration> |
| --- |
|  |

|  |
| --- |

### (3) 配置hdfs-site.xml

vim hdfs-site.xml

 dfs.namenode.name.dir：namenode数据的存放地点。也就是namenode元数据存放的地方，记录了hdfs系统中文件的元数据。

 dfs.datanode.data.dir： datanode数据的存放地点。也就是block块存放的目录了。

dfs.replication：hdfs的副本数设置。也就是上传一个文件，其分割为block块后，每个block的冗余副本个数，默认配置是3。

dfs.secondary.http.address：secondarynamenode 运行节点的信息，和 namenode 不同节点

| <configuration> <!-- 设置dfs副本数，不设置默认是3个 --> <property> <name>dfs.replication</name> <value>2</value> </property> <!-- 设置secondname的端口 --> <property> <name>dfs.namenode.secondary.http-address</name> <value>Hadoop-node1:50090</value> </property> </configuration> |
| --- |
|  |

|  |
| --- |

### (4) 配置mapred-env.sh

vim mapred-env.sh

export JAVA\_HOME=/java/jdk1.8.0\_162

### (5) 配置mapred-site.xml

cp /hadoop/hadoop-2.7.6/etc/hadoop/mapred-site.xml.template /hadoop/hadoop-2.7.6/etc/hadoop/mapred-site.xml

vim mapred-site.xml

 mapreduce.framework.name：指定mr框架为yarn方式,Hadoop二代MP也基于资源管理系统Yarn来运行 。

| <configuration> <property> <name>mapreduce.framework.name</name> <value>yarn</value> </property> </configuration> |
| --- |
|  |

|  |
| --- |

### (6) 配置yarn-evn.sh

vim yarn-env.sh

export JAVA\_HOME=/java/jdk1.8.0\_162

### (7) 配置yarn-site.xml

vim yarn-site.xml

 yarn.resourcemanager.hostname：yarn总管理器的IPC通讯地址

 yarn.nodemanager.aux-services：

| <configuration> <!-- reducer获取数据的方式 --> <property> <name>yarn.nodemanager.aux-services</name> <value>mapreduce\_shuffle</value> </property> <!-- 指定YARN的ResourceManager的地址 --> <property> <name>yarn.resourcemanager.hostname</name> <value>Hadoop-node2</value> </property> </configuration> |
| --- |
|  |

|  |
| --- |

### (8) 配置slaves

vim slaves

Hadoop-master

Hadoop-node1

Hadoop-node2

PS：在master上配置完，再传到另外两个node上也是可以的

scp -r /hadoop/hadoop-3.0.2 Hadoop-node1:/hadoop/hadoop-3.0.2

scp -r /hadoop/hadoop-3.0.2 Hadoop-node2:/hadoop/hadoop-3.0.2

## 初始化HDFS

只能在Hadoop-master上对HDFS进行初始化：

hdfs namenode -format

## 启动HDFS

在Hadoop-master执行start-dfs.sh

在Hadoop-node2执行start-yarn.sh

## 验证启动情况

在Hadoop-master执行jps

2532 NameNode

3015 Jps

2633 DataNode

2893 NodeManager

在Hadoop-node1执行jps

2632 Jps

2346 DataNode

2396 SecondaryNameNode

2510 NodeManager

在Hadoop-node2执行jps

2499 ResourceManager

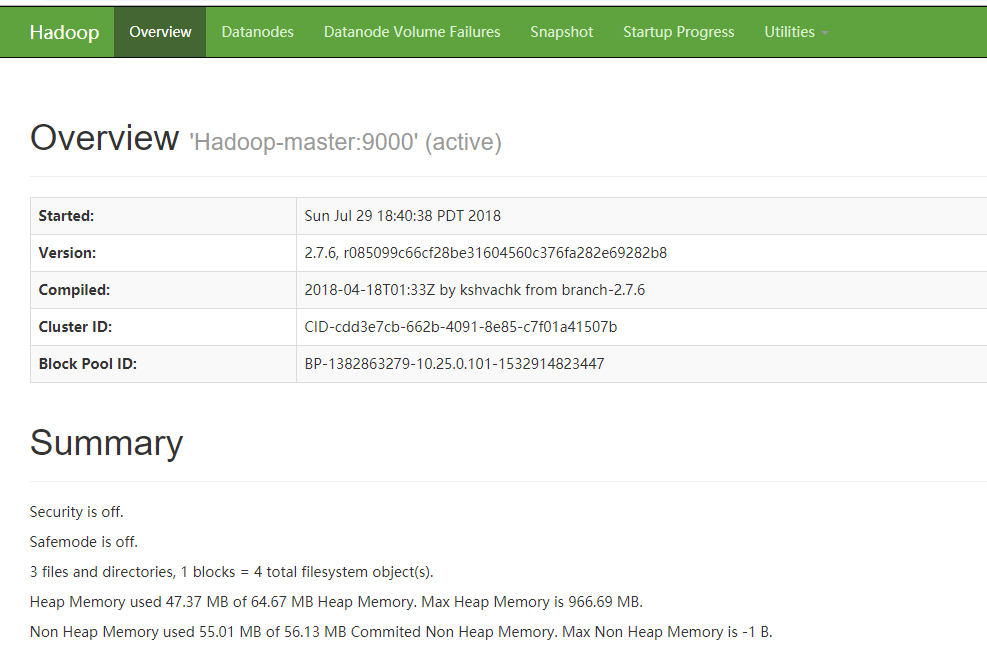
2649 Jps

2602 NodeManager

2381 DataNode

## Web页面查看Hadoop启动情况

<http://10.25.0.101:50070>



## NTP时间同步

### (1) 查看ntp包是否已安装

[root@hadoop-master ~]#rpm -qa |grep ntp

[root@hadoop-master ~]#ntp-4.2.6p5-1.el6.centos.x86\_64

fontpackages-filesystem-1.41-1.1.el6.noarch

ntpdate-4.2.6p5-1.el6.centos.x86\_64

### (2) 把Hadoop-master作为NTP Server

查看亚洲大部分时区和国家城市的时间文件位置

ls -F /usr/share/zoneinfo/Asia

查看当前某一时区的时间

zdump Hongkong

设置系统时间的时区

ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

查看系统时间和硬件时间是否一致(RTC即系统停止还在计算时间一般写在BIOS里头)

date

hwclock –r

设置系统时间和硬件时间与当前实际时间保持一致

date -s "dd/mm/yyyy hh:mm:ss"，这里按格式填入具体的时间

hwclock -w

编辑/etc/ntp.conf

//找到restrict字段并在中间加上如下信息

restrict 10.25.0.1/24 mask 255.255.255.0 nomodify noquery

//找到server 0.centos.pool.ntp.org iburst 并把相关的给注释了，添加如下

server ntp.sjtu.edu.cn

server 127.127.1.0 # local clock

fudge 127.127.1.0 stratum 10

//如果第二部配置的server ntp.sjtu.edu.cn 无效时，则NTP服务器会根据这里的配置，把自己的时间做为NTP服务器的时间，即和自己同步。考虑到有的局域网里不可以访问外网，所有这里需要把这个配置项用上

设置ntp服务开机启动

[root@hadoop-master ~]#service ntpd start

[root@hadoop-master ~]#chkconfig ntpd on

**(1) 把Hadoop-node1和Hadoop-node2作为NTP Client**

编辑/etc/ntp.conf

//找到server 0.centos.pool.ntp.org iburst 并把相关的给注释了，添加如下

server 10.25.0.101 #ip为时间服务器地址

[root@hadoop-master ~]#service ntpd start

[root@hadoop-master ~]#chkconfig ntpd on

## HDFS的常用命令

[root@hadoop-master ~]# hadoop fs -mkdir /NaiveBayesData

[root@hadoop-master ~]# hadoop fs -ls

[root@hadoop-master ~]# hadoop fs -copyFromLocal /root/a.txt /helloHDFS/a.txt

[root@hadoop-master ~]# hadoop fs -ls /helloHDFS/

[root@hadoop-master ~]# hadoop fs -cat /helloHDFS/a.txt

[root@hadoop-master ~]# hadoop dfs -put /alphabeta.txt /NaiveBayesData

# Spark安装

## 三台机器安装scala和spark

[root@hadoop-master ~]# vim /etc/profile

|  |
| --- |
| export SPARK\_HOME=/spark/spark-2.3.0-bin-hadoop2.7/  export PATH=$PATH:$SPARK\_HOME/bin  export SCALA\_HOME=/scala/scala-2.12.6  export PATH=$PATH:$SCALA\_HOME/bin |

使得profile生效

[root@hadoop-master ~]# source /etc/profile

测试scala是否安装成功

[root@hadoop-master ~]# scala –version

|  |
| --- |
| Scala code runner version 2.12.6 -- Copyright 2002-2018, LAMP/EPFL and Lightbend, Inc |

## 三台机器配置spark

### (1) 配置spark-env.sh

[root@hadoop-master ~]# cp /spark/spark-2.3.0-bin-hadoop2.7/conf/spark-env.sh.template /spark/spark-2.3.0-bin-hadoop2.7/conf/spark-env.sh

[root@hadoop-master ~]# vim /spark/spark-2.3.0-bin-hadoop2.7/conf/spark-env.sh

|  |
| --- |
| export JAVA\_HOME=/java/jdk1.8.0\_162  export SCALA\_HOME=/scala/scala-2.12.6  export HADOOP\_HOME=/hadoop/hadoop-2.7.6  export HADOOP\_CONF\_DIR=/hadoop/hadoop-2.7.6/etc/hadoop  #定义管理端口  export SPARK\_MASTER\_WEBUI\_PORT=8088  #定义master域名和端口  export SPARK\_MASTER\_HOST=Hadoop-master  export SPARK\_MASTER\_PORT=7077  #定义master的地址slave节点使用  export SPARK\_MASTER\_IP=Hadoop-master  #定义work节点的管理端口.work节点使用  export SPARK\_WORKER\_WEBUI\_PORT=8088  #每个worker节点能够最大分配给exectors的内存大小  export SPARK\_WORKER\_MEMORY=4g |

### (2) 配置slaves

[root@hadoop-master ~]# cp /spark/spark-2.3.0-bin-hadoop2.7/conf/slaves.template /spark/spark-2.3.0-bin-hadoop2.7/conf/slaves

|  |
| --- |
| vim /spark/spark-2.3.0-bin-hadoop2.7/conf/slaves  Hadoop-node1  Hadoop-node2 |

### (3) 配置spark-defaults.conf

[root@hadoop-master ~]# cp /spark/spark-2.3.0-bin-hadoop2.7/conf/spark-defaults.conf.template /spark/spark-2.3.0-bin-hadoop2.7/conf/spark-defaults.conf

|  |
| --- |
| vim /spark/spark-2.3.0-bin-hadoop2.7/conf/spark-defaults.conf  spark.eventLog.enabled=true  spark.eventLog.compress=true  #保存在本地  #spark.eventLog.dir=file://hadoop/hadoop-2.7.6/logs/userlogs  #spark.history.fs.logDirectory=file://hadoop/hadoop-2.7.6/logs/userlogs  #保存在hdfs上  spark.eventLog.dir=hdfs://Hadoop-master:9000/tmp/logs/root/logs  spark.history.fs.logDirectory=hdfs://Hadoop-master:9000/tmp/logs/root/logs  spark.yarn.historyServer.address=Hadoop-master:18080 |

Hadoop-Master上执行

[root@hadoop-master bin]# hadoop fs -mkdir -p /tmp/logs/root/logs

## 三台机器配置pyspark兼容python3.6.6

安装编译python3.6.6所需要的包

[root@hadoop-X ~]# yum -y install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel gdbm-devel db4-devel libpcap-devel xz-devel gcc

下载python3.6.6

[root@hadoop- X ~]# wget https://www.python.org/ftp/python/3.6.6/Python-3.6.6.tgz

[root@hadoop- X ~]# cd /Python-3.6.6

配置编译后的目录

[root@hadoop- X ~]# ./configure --prefix=/usr/bin/python3.6.6

安装

[root@hadoop- X ~]# make && make install

配置软连接，测试

[root@hadoop- X ~]# ln -s /usr/bin/python3.6.6/bin/python3 /usr/bin/python3

[root@hadoop- X ~]# python3 –V

配置pip

[root@hadoop- X ~]# cp /usr/bin/python3.6.6/bin/pip3 /usr/bin/

[root@hadoop- X ~]# cp /usr/bin/python3.6.6/bin/pip3.6 /usr/bin/

修改/spark/spark-2.3.0-bin-hadoop2.7/conf/spark-env.sh文件，在末尾添加export PYSPARK\_PYTHON=/usr/bin/python3

修改/spark/spark-2.3.0-bin-hadoop2.7/bin/pyspark，将原来PYSPARK\_PYTHON=python改成PYSPARK\_PYTHON=python3

|  |
| --- |
| # Determine the Python executable to use for the executors:  if [[ -z "$PYSPARK\_PYTHON" ]]; then  if [[ $PYSPARK\_DRIVER\_PYTHON == \*ipython\* && ! $WORKS\_WITH\_IPYTHON ]]; then  echo "IPython requires Python 2.7+; please install python2.7 or set PYSPARK\_PYTHON" 1>&2  exit 1  else  PYSPARK\_PYTHON=python3  fi  fi  export PYSPARK\_PYTHON |

启动pyspark



## 启动spark

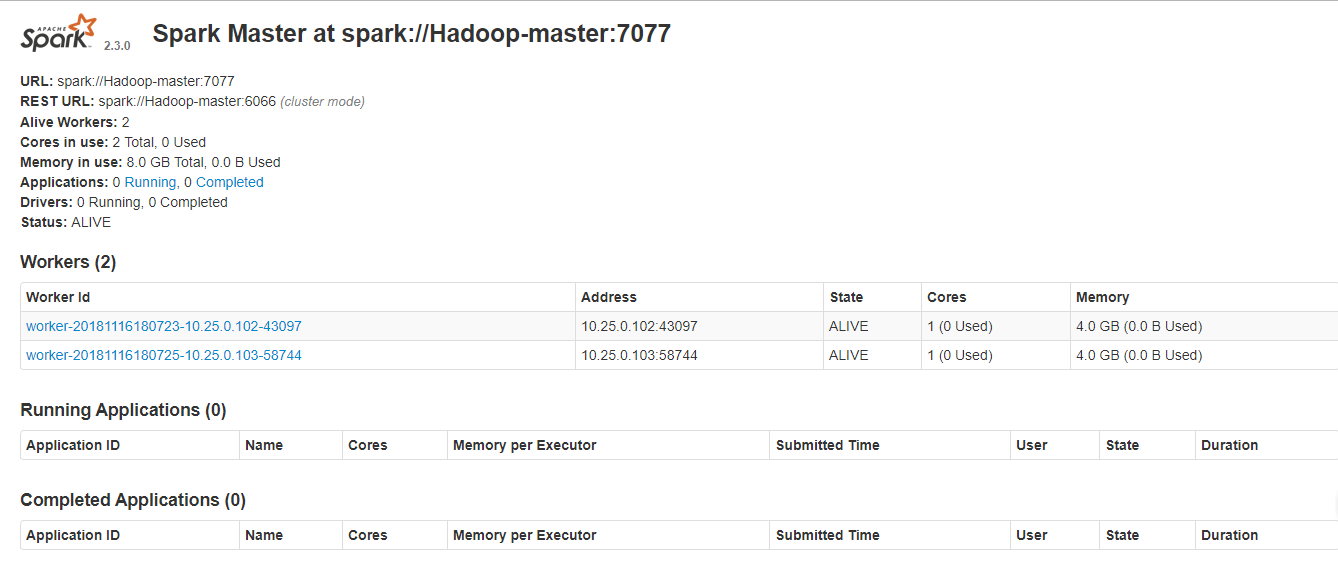
在Hadoop-master上

[root@hadoop-master ~]# /spark/spark-2.3.0-bin-hadoop2.7/sbin/start-all.sh

|  |
| --- |
| starting org.apache.spark.deploy.master.Master, logging to /spark/spark-2.3.0-bin-hadoop2.7/logs/spark-root-org.apache.spark.deploy.master.Master-1-hadoop-master.out  Hadoop-node2: starting org.apache.spark.deploy.worker.Worker, logging to /spark/spark-2.3.0-bin-hadoop2.7/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-Hadoop-node2.out  Hadoop-node1: starting org.apache.spark.deploy.worker.Worker, logging to /spark/spark-2.3.0-bin-hadoop2.7/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-Hadoop-node1.out |

## Web页面查看Spark启动情况

http://10.25.0.101:8088/



# Hive安装

## Hadoop-master安装Hive

[root@hadoop-master ~]# vim /etc/profile

|  |
| --- |
| export HIVE\_HOME=/hive/apache-hive-2.3.0-bin  export PATH=$PATH:$HIVE\_HOME/bin |

[root@hadoop-master ~]# source /etc/profile

## Hadoop-master配置mysql驱动

[root@hadoop-master ~]# mv /mysql-connector-java-5.1.13-bin.jar /hive/apache-hive-2.3.0-bin/lib/mysql-connector-java-5.1.13-bin.jar

## Hadoop-master配置hive

### (1) 配置hive-env.sh

[root@hadoop-master ~]# cp /hive/apache-hive-2.3.0-bin/conf/hive-env.sh.template /hive/apache-hive-2.3.0-bin/conf/hive-env.sh

[root@hadoop-master ~]# vim /hive/apache-hive-2.3.0-bin/conf/hive-env.sh

|  |
| --- |
| export JAVA\_HOME=/java/jdk1.8.0\_162  export HADOOP\_HOME=/hadoop/hadoop-2.7.6  export HIVE\_HOME=/hive/apache-hive-2.3.0-bin  export HIVE\_CONF\_DIR=/hive/apache-hive-2.3.0-bin/conf |

### (2) 启动hadoop并创建相关目录，授权，用于存储文件

[root@hadoop-master ~]# hdfs dfs -mkdir -p /user/hive/warehouse

[root@hadoop-master ~]# hdfs dfs -mkdir -p /user/hive/tmp

[root@hadoop-master ~]# hdfs dfs -mkdir -p /user/hive/log

[root@hadoop-master ~]# hdfs dfs -chmod -R 777 /user/hive/warehouse

[root@hadoop-master ~]# hdfs dfs -chmod -R 777 /user/hive/tmp

[root@hadoop-master ~]# hdfs dfs -chmod -R 777 /user/hive/log

[root@hadoop-master ~]# hdfs dfs -chmod -R 777 /user

[root@hadoop-master ~]# mkdir -p /hive/apache-hive-2.3.0-bin/tmp

[root@hadoop-master ~]# chmod -R 777 /hive/apache-hive-2.3.0-bin/tmp

### (3) 配置hive-site.xml

[root@hadoop-master ~]# cp /hive/apache-hive-2.3.0-bin/conf/hive-default.xml.template /hive/apache-hive-2.3.0-bin/conf/hive-site.xml

[root@hadoop-master ~]# vim /hive/apache-hive-2.3.0-bin/conf/hive-site.xml

|  |
| --- |
| <property>  <name>hive.exec.scratchdir</name>  <value>/user/hive/tmp</value>  </property>  <property>  <name>hive.metastore.warehouse.dir</name>  <value>/user/hive/warehouse</value>  </property>  <property>  <name>hive.querylog.location</name>  <value>/user/hive/log</value>  </property>  <property>  <name>javax.jdo.option.ConnectionURL</name>  <value>jdbc:mysql://10.25.0.141:3306/hive?createDatabaseIfNotExist=true&amp;characterEncoding=UTF-8&amp;useSSL=false</value>  </property>    <property>  <name>javax.jdo.option.ConnectionDriverName</name>  <value>com.mysql.jdbc.Driver</value>  </property>  <property>  <name>javax.jdo.option.ConnectionUserName</name>  <value>root</value>  </property>  <property>  <name>javax.jdo.option.ConnectionPassword</name>  <value>Workhard\_1234</value>  </property> |

将hive-site.xml文件中的${system:java.io.tmpdir}替换为hive的临时目录 /hive/apache-hive-2.3.0-bin/tmp

把 {system:user.name} 改成 {user.name}

### (3) 配置hive-log4j2.properties

[root@hadoop-master ~]# cp /hive/apache-hive-2.3.0-bin/conf/hive-log4j2.properties.template /hive/apache-hive-2.3.0-bin/conf/hive-log4j2.properties

### (4) 配置hive-exec-log4j2.properties

[root@hadoop-master ~]# cp /hive/apache-hive-2.3.0-bin/conf/hive-exec-log4j2.properties.template /hive/apache-hive-2.3.0-bin/conf/hive-exec-log4j2.properties

## Hadoop-master上初始化Hive

[root@hadoop-master ~]# schematool -dbType mysql -initSchema

|  |
| --- |
| SLF4J: Class path contains multiple SLF4J bindings.  SLF4J: Found binding in [jar:file:/hive/apache-hive-2.3.0-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]  SLF4J: Found binding in [jar:file:/hadoop/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]  SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.  SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]  Metastore connection URL: jdbc:mysql://10.25.0.141:3306/hive?createDatabaseIfNotExist=true&characterEncoding=UTF-8&useSSL=false  Metastore Connection Driver : com.mysql.jdbc.Driver  Metastore connection User: root  Starting metastore schema initialization to 2.3.0  Initialization script hive-schema-2.3.0.mysql.sql  Initialization script completed  schemaTool completed |

## 启动Hive

[root@hadoop-master ~]# hive

|  |
| --- |
| which: no hbase in (.:/hadoop/hadoop-2.7.6/bin:/hadoop/hadoop-2.7.6/sbin:/java/jdk1.8.0\_162/bin:.:/java/jdk1.8.0\_162/bin:.:/hadoop/hadoop-2.7.6/bin:/hadoop/hadoop-2.7.6/sbin:/java/jdk1.8.0\_162/bin:.:/java/jdk1.8.0\_162/bin:.:/hadoop/hadoop-2.7.6/bin:/hadoop/hadoop-2.7.6/sbin:/java/jdk1.8.0\_162/bin:.:/java/jdk1.8.0\_162/bin:.:/hadoop/hadoop-2.7.6/bin:/hadoop/hadoop-2.7.6/sbin:/java/jdk1.8.0\_162/bin:.:/java/jdk1.8.0\_162/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/spark/spark-2.3.0-bin-hadoop2.7/bin:/scala/scala-2.12.6/bin:/spark/spark-2.3.0-bin-hadoop2.7/bin:/scala/scala-2.12.6/bin:/spark/spark-2.3.0-bin-hadoop2.7/bin:/scala/scala-2.12.6/bin:/hive/apache-hive-2.3.0-bin/bin)  SLF4J: Class path contains multiple SLF4J bindings.  SLF4J: Found binding in [jar:file:/hive/apache-hive-2.3.0-bin/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]  SLF4J: Found binding in [jar:file:/hadoop/hadoop-2.7.6/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]  SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.  SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]  Logging initialized using configuration in jar:file:/hive/apache-hive-2.3.0-bin/lib/hive-common-2.3.0.jar!/hive-log4j2.properties Async: true  Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.  hive> |

## Hive常用操作

### (1) 创建数据库

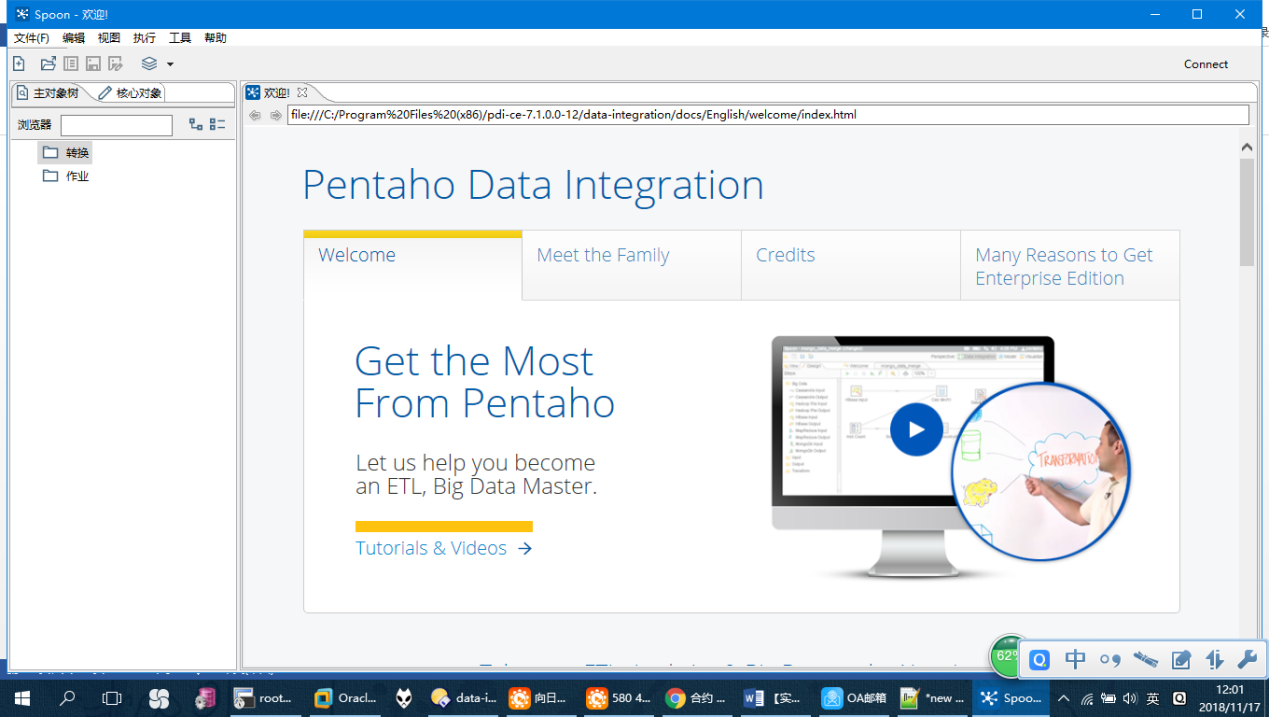
hive> create database myhive;

### (2) 查看数据库

hive> show databases;

hive> use myhive;

# Kettle安装



# sqoop安装

https://www.cnblogs.com/harrymore/p/9056863.html

# 汇总整个集群的启停操作

### (1) 停止整个集群

|  |
| --- |
| [root@hadoop-master /]# /spark/spark-2.3.0-bin-hadoop2.7/sbin/stop-all.sh  [root@hadoop-master /]# /hadoop/hadoop-2.7.6/sbin/stop-all.sh |

### (1) 启动整个集群

|  |
| --- |
| [root@hadoop-master /]# /hadoop/hadoop-2.7.6/sbin/start-all.sh  [root@hadoop-master /]# /spark/spark-2.3.0-bin-hadoop2.7/sbin/start-all.sh |

# ETL过程

## 准备一份实验数据



各个字段对应的中文含义为：

ID|客户年龄|客户性别|终端品牌|主资费套餐|流量资费编码|细分市场|终端机龄|终端制式|在网网龄（月）|近三月月均ARPU|近三月月均缴费|近三月月均主叫时长|近三月月均流量|近三月月均上网天数|宽带流量|是否乡音网|是否校园网|是否朋友圈|是否合家欢|是否V网|是否4G终端|是否4G卡|当月是否4G主资费|是否wap套餐|平均换机周期|换机次数|存送合约到期|是否绑定宽带|是否集团客户|是否办理流量|是否换机客户

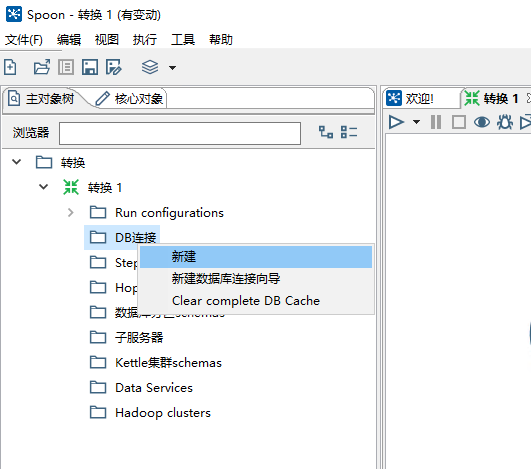
## 将Excel文件通过kettle迁移到Oracle库表

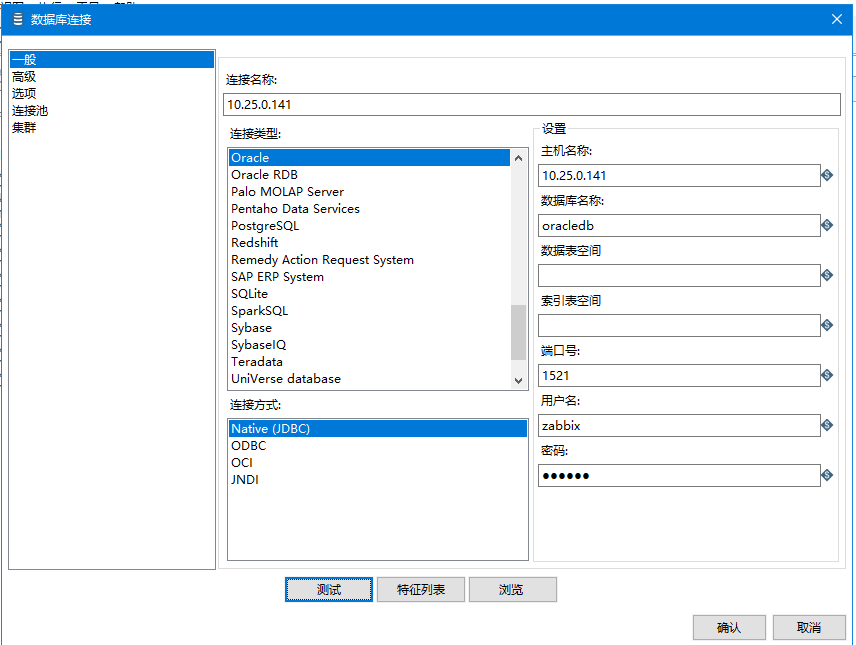
在10.25.0.141上zabbix用户建立表CMCC

|  |
| --- |
| CREATE TABLE CMCC  (  ID varchar2(100),  age float,  sex float,  terminal varchar2(100),  mainprodid varchar2(100),  dataprodid varchar2(100),  market varchar2(100),  terminalage float,  terminalstandar varchar2(100),  subsage float,  arpyper3 float,  payper3 float,  calltimeper3 float,  dataper3 float,  netplayper3 float,  databroadband float,  iscountrynet float,  iscampusnet float,  isfriendnet float,  isfamilynet float,  isvnet float,  is4gterminal varchar2(100),  is4gcard varchar2(100),  is4gmain varchar2(100),  iswap varchar2(100),  phonechangeT float,  phonechangetimes float,  iscontractdated float,  isbindbroadband varchar2(100),  isgroupcust varchar2(100),  isattenddata varchar2(100),  isphonechangecust varchar2(100)  ); |

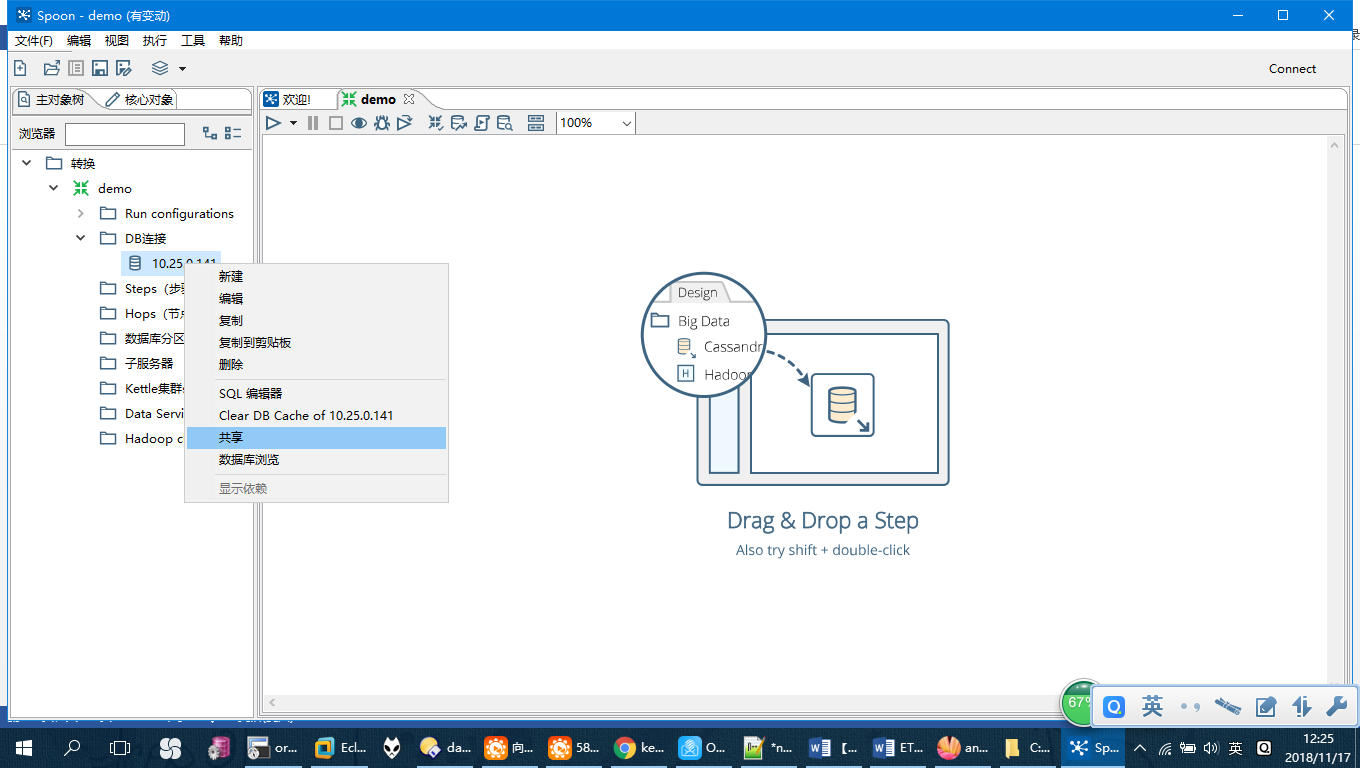
将oracle的jdbc保存在kettle的lib目录下：

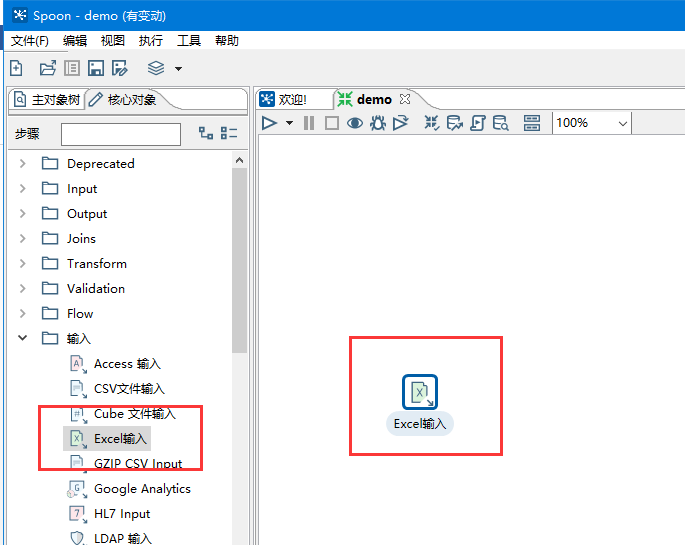
首先要建立到Oracle库的数据库连接

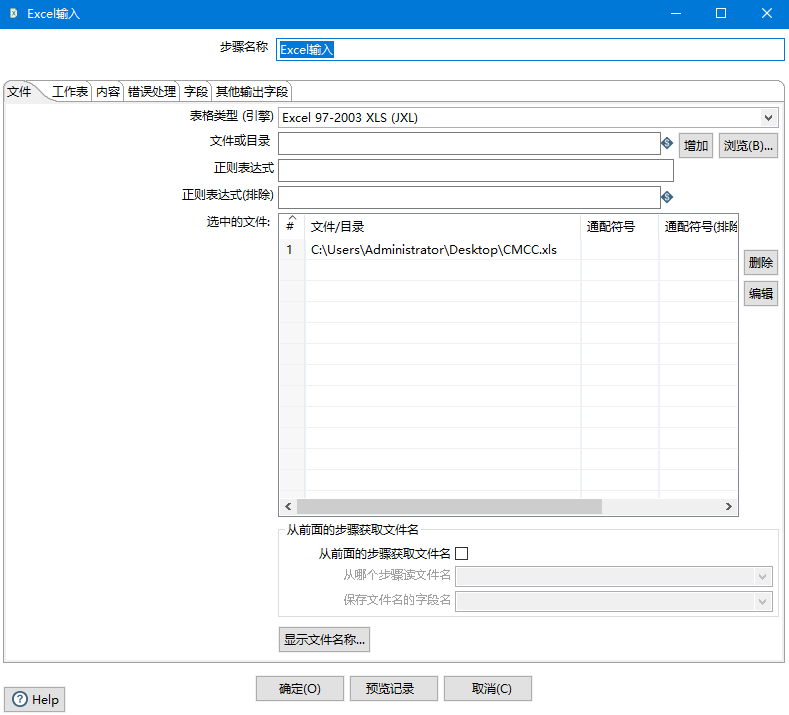


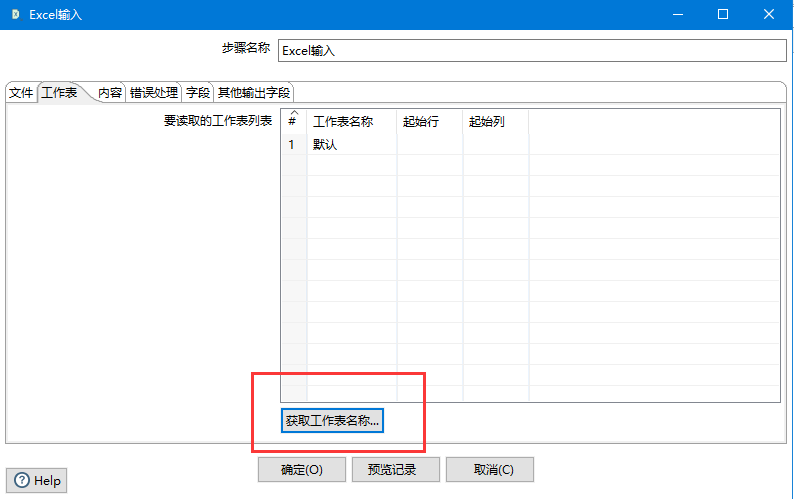


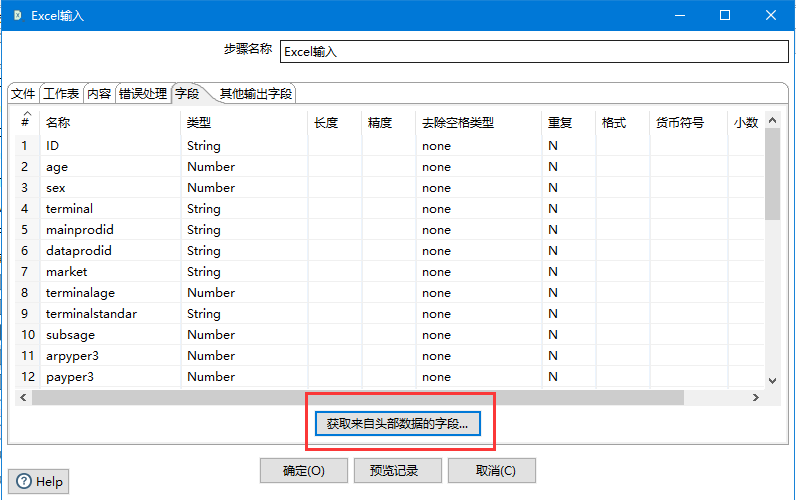
一定要将两个DB连接设置为共享才行。记得及时保存

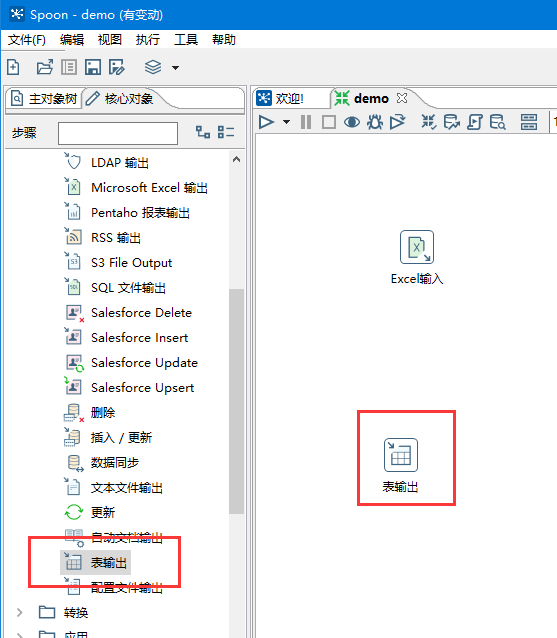


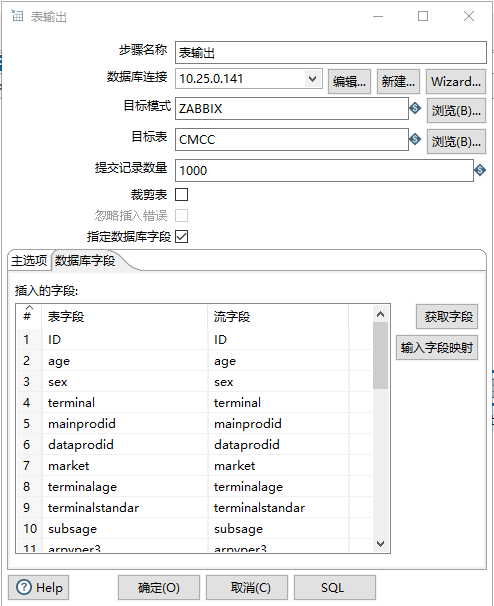


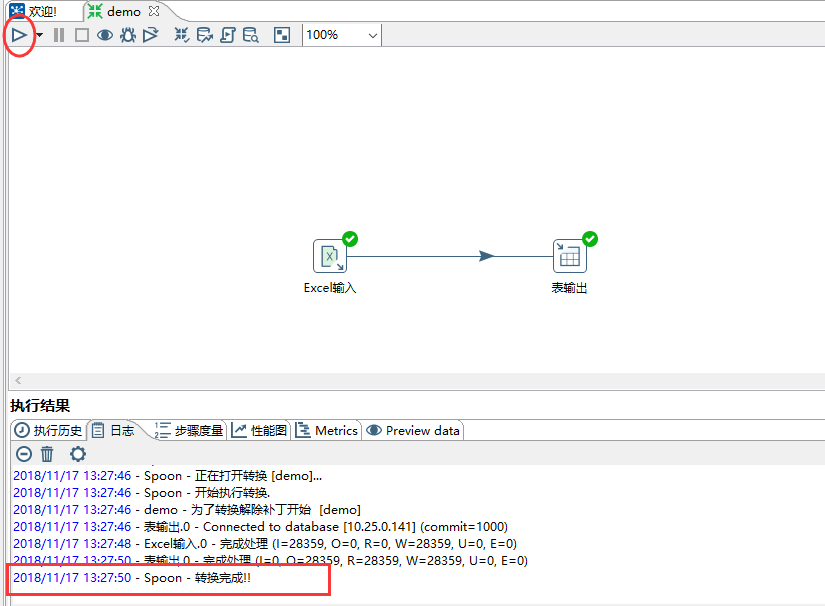












## 将Oracle库表通过sqoop迁移至Hadoop的HDFS

# 配置pycharm使用pyspark

## 安装专业版的pycharm(破解)



## 三台机器安装相关的python的包

pip3 install pyspark==2.3.0

pip3 install py4j 0.10.6

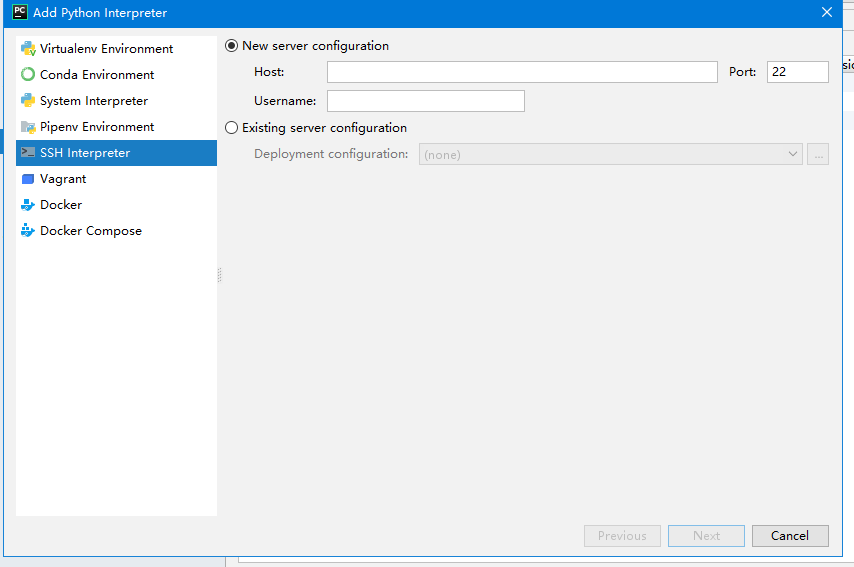
这个pyspark和py4j的版本是跟spark的版本有兼容性要求的。换个高一点的版本就不兼容了。

## 三台机器配置sudoers

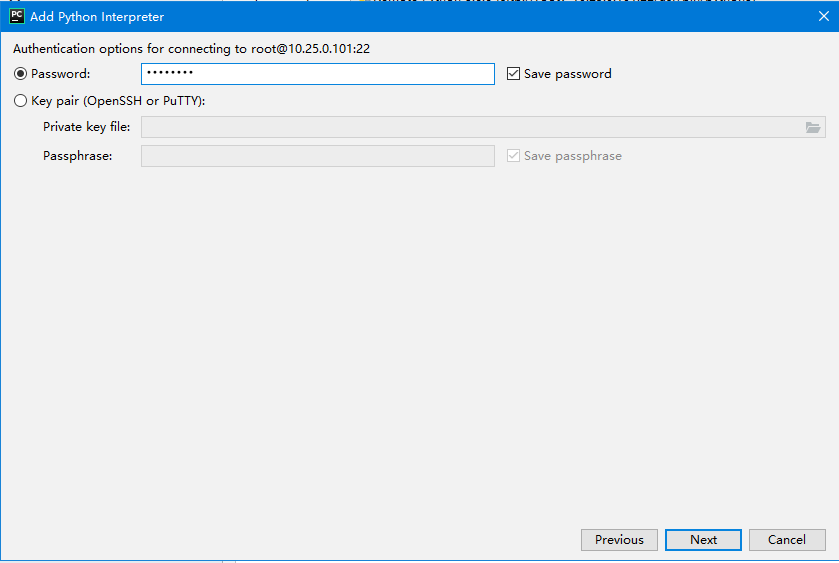
vi /etc/sudoers (最好用visudo命令)

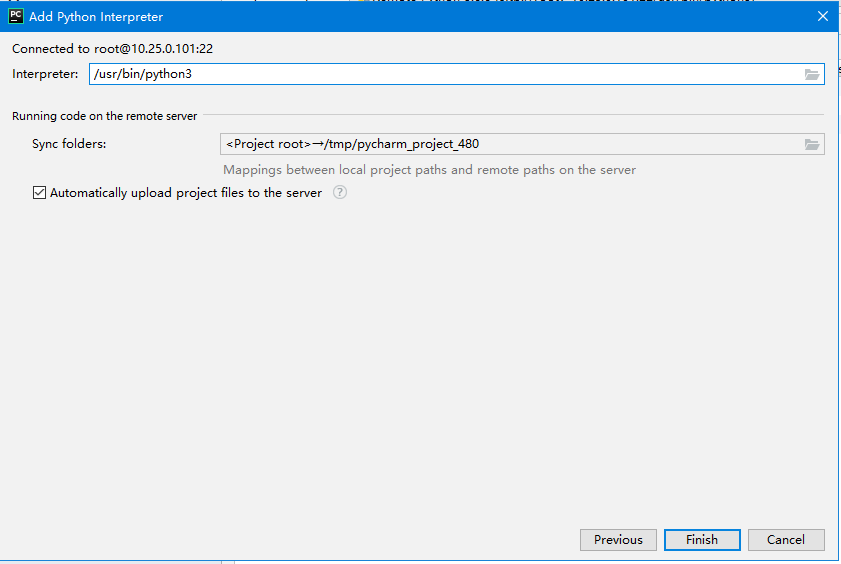
注释掉 Default requiretty 一行

## 专业版Pycharm配置远程python解析器









## 三台机器配置环境变量

export PYTHONPATH=$SPARK\_HOME/python/:$PYTHONPATH

export PYTHONPATH=$SPARK\_HOME/python/lib/py4j-0.10.6-src.zip:$PYTHONPATH

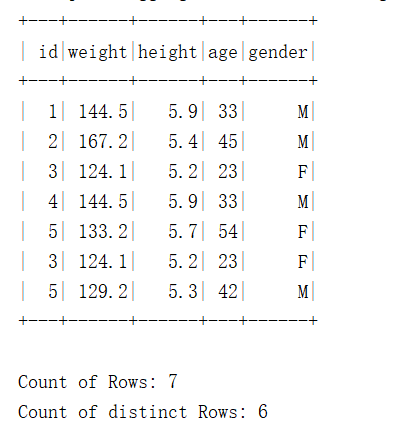
export PYSPARK\_DRIVER\_PYTHON=/usr/bin/python3

export PYSPARK\_PYTHON=/usr/bin/python3

## 在专业版pycharm测试将代码提交到spark集群上跑

|  |
| --- |
| **import** pyspark **from** pyspark.sql **import** SparkSession **import** os  ***# 这两句一定是要的，否则就报错*** os.environ[**"SPARK\_HOME"**] = **"/spark/spark-2.3.0-bin-hadoop2.7/"** os.environ[**"PYSPARK\_PYTHON"**]=**"/usr/bin/python3"** spark = SparkSession.builder \  .master(**"local"**) \  .appName(**"Word Count"**) \  .getOrCreate()  df = spark.createDataFrame([  (1, 144.5, 5.9, 33, **'M'**),  (2, 167.2, 5.4, 45, **'M'**),  (3, 124.1, 5.2, 23, **'F'**),  (4, 144.5, 5.9, 33, **'M'**),  (5, 133.2, 5.7, 54, **'F'**),  (3, 124.1, 5.2, 23, **'F'**),  (5, 129.2, 5.3, 42, **'M'**),  ], [**'id'**, **'weight'**, **'height'**, **'age'**, **'gender'**])  df.show() print(**'Count of Rows: {0}'**.format(df.count())) print(**'Count of distinct Rows: {0}'**.format((df.distinct().count())))  spark.stop() |

跑出来的结果如下：



# 贝叶斯定理与朴素贝叶斯分类器

## 贝叶斯定理的推导

[条件概率](http://link.zhihu.com/?target=https%3A//zh.wikipedia.org/wiki/%25E6%259D%25A1%25E4%25BB%25B6%25E6%25A6%2582%25E7%258E%2587" \t "_blank)，描述的是事件在另一个事件已经发生条件下的概率，记作 ， 和可能是相互独立的两个事件，也可能不是。



表示和事件同时发生的概率，如果和是相互独立的两个事件，那么：



上面的反过来证明了如果和是相互独立的事件，那么事件 发生的概率与无关。

稍微改变一下，



考虑到先验条件的多种可能性，引入全概率公式：

，这里是的补集。

根据条件概率和全概率公式，很容易推导出贝叶斯公式：



特别地，

如果是的一个划分，



当然，从逻辑上，和也可以无类别上的关系。仅仅是观察到的两个因素。

## 先验概率和后验概率

如果从先验和后验去理解：



对于贝叶斯公式，本质上是用先验概率去修正后验概率，其中，

是先验概率，是不考虑任何方面的因素。

是标准化常量

是似然度，是观察到的内容，是相关的证据。

后验概率 = (似然度×先验概率) / 标准化常量。也就是说，后验概率与先验概率和似然度的乘积成正比。

## 机器学习的视角理解贝叶斯



## 朴素贝叶斯分类器的原理

已有的样本数据：

，有个样本，每个样本有个特征，特征输出有个类别，定义未

从样本中可以学到朴素贝叶斯的先验分布，接着学习到条件概率分布：



根据贝叶斯定理：



其中，**朴素贝叶斯的假设是：各个特征之间是独立无关的(如果各个特征之间是相关的，应考虑使用贝叶斯信念网络)**。基于这一假设，



是根据样本数据的各个特征取值和对应的类别出现的频数可以构造出来。作为极大似然。

是作为一个先验。实际上，这个分类的概率应该是一个先验知识，而不是根据样本数据的类别频数构造出来。这里朴素贝叶斯分类器用样本的频数作为先验了。

有了和，分类器就可以构造了。

分类器构造完毕，就可以对一个新样本进行分类了。

对于样本，对于，计算

，







哪个大，就认为新样本属于哪个分类。至于实际上是不用计算的，因为每个公式里都有它，作为一个相同值的因子不影响计算结果。

## 朴素贝叶斯的参数估计

直接根据训练样本的频数计算即可。

但对于测试样本或者验证样本的计算就比较复杂了。

如果第个特征是离散的值，如果出现在训练样本中，，其中，为样本类别的样本的个数，为样本类别第个特征取值为的样本个数。如果没有出现在训练样本中，需要引入拉普拉斯平滑，此时有，其中为一个大于0的常数，常常取为1。为第个特征的取值个数。如果是非常稀疏的离散值，即各个特征出现的概率很低，这时可以假设符合伯努利分布，即特征出现记为1，不出现记为0。即只要出现即可，不用关注的次数。是在样本类别中，出现的频率。

，取值为0或者1。

如果第个特征是连续的值，通常认为的先验分布为正态分布，即在样本类别为的样本中，的值符合正态分布，对于取值为的概率为

，是取值为的样本中所有的方差， 是取值为的样本中所有的均值。

# 实现朴素贝叶斯分类器预测用户是否换机

## 基于Java+Mathout+HDFS 实现朴素贝叶斯分类

## 基于Hive SQL+sklearn实现朴素贝叶斯分类

## 基于pyspark+Mlib实现朴素贝叶斯分类

## 基于SPSS实现朴素贝叶斯分类

# 大数据量场景的性能测试