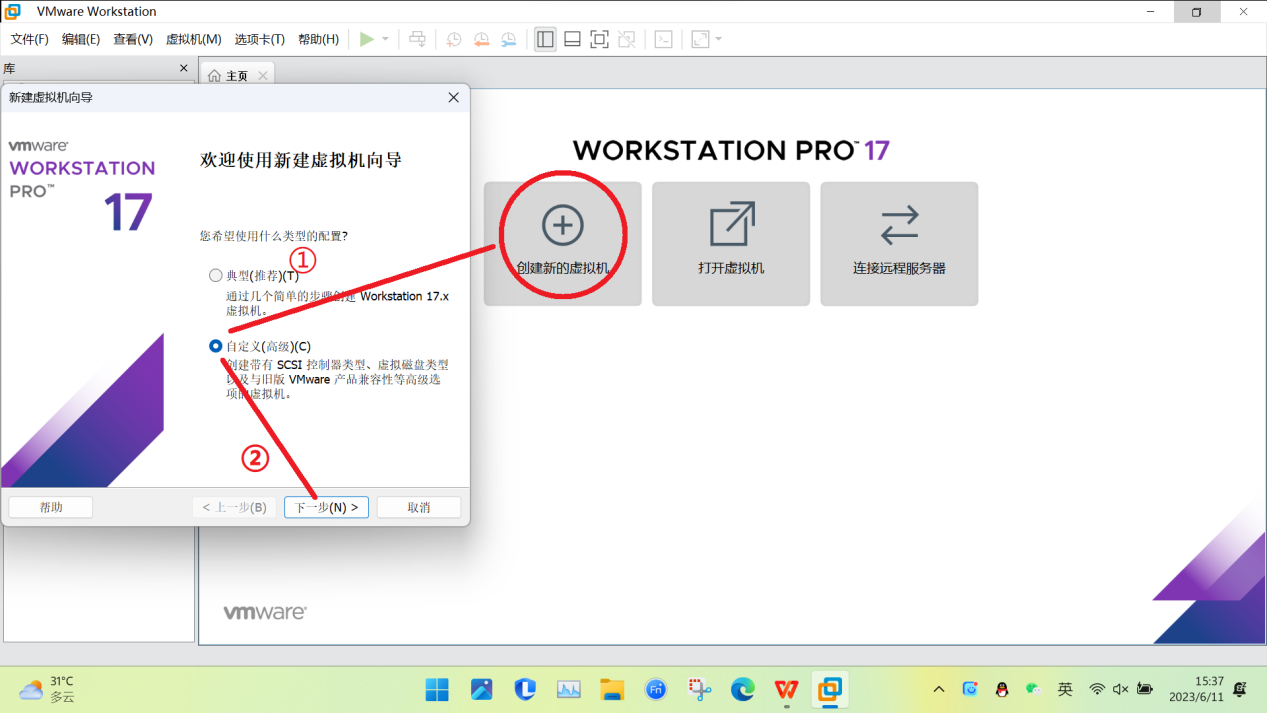
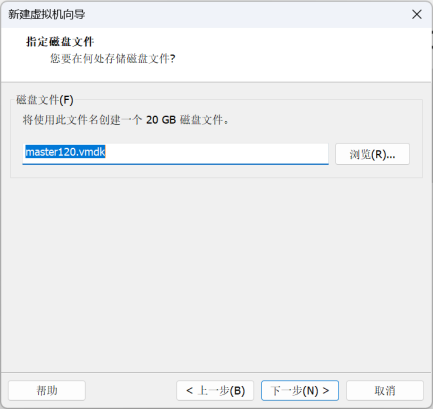
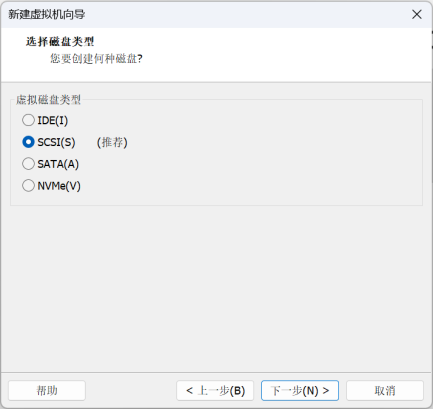
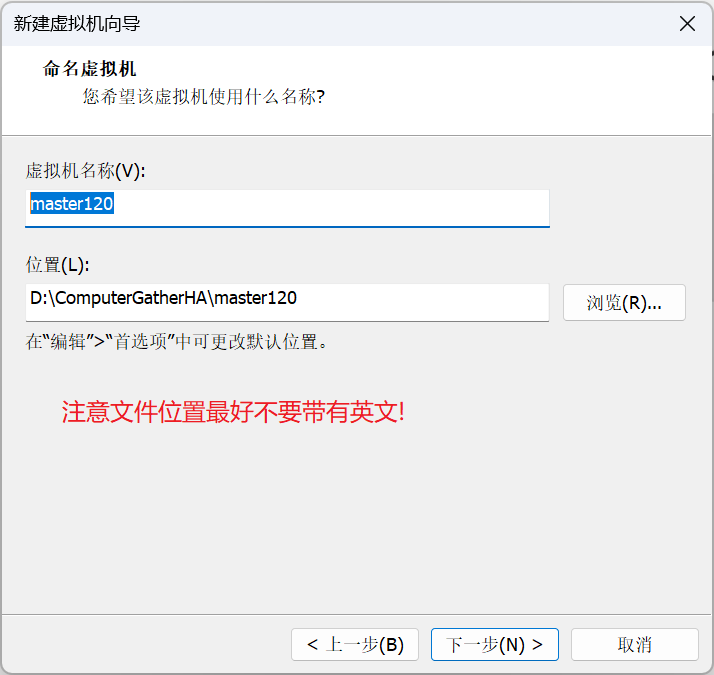
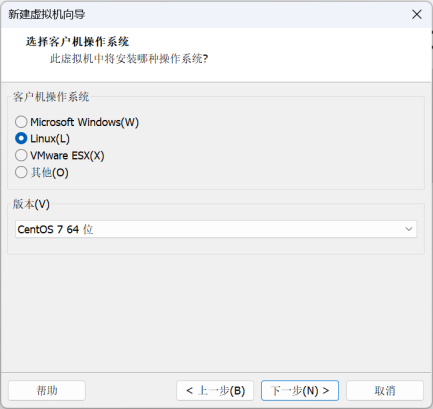
# Hadoop高可用教程

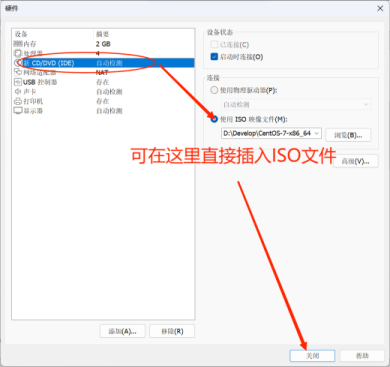
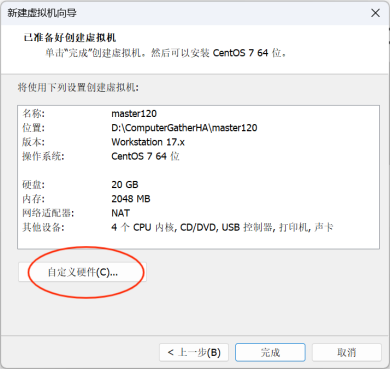
——责任编辑: Mochi 琥珀光-

## 第一章. 创建虚拟机

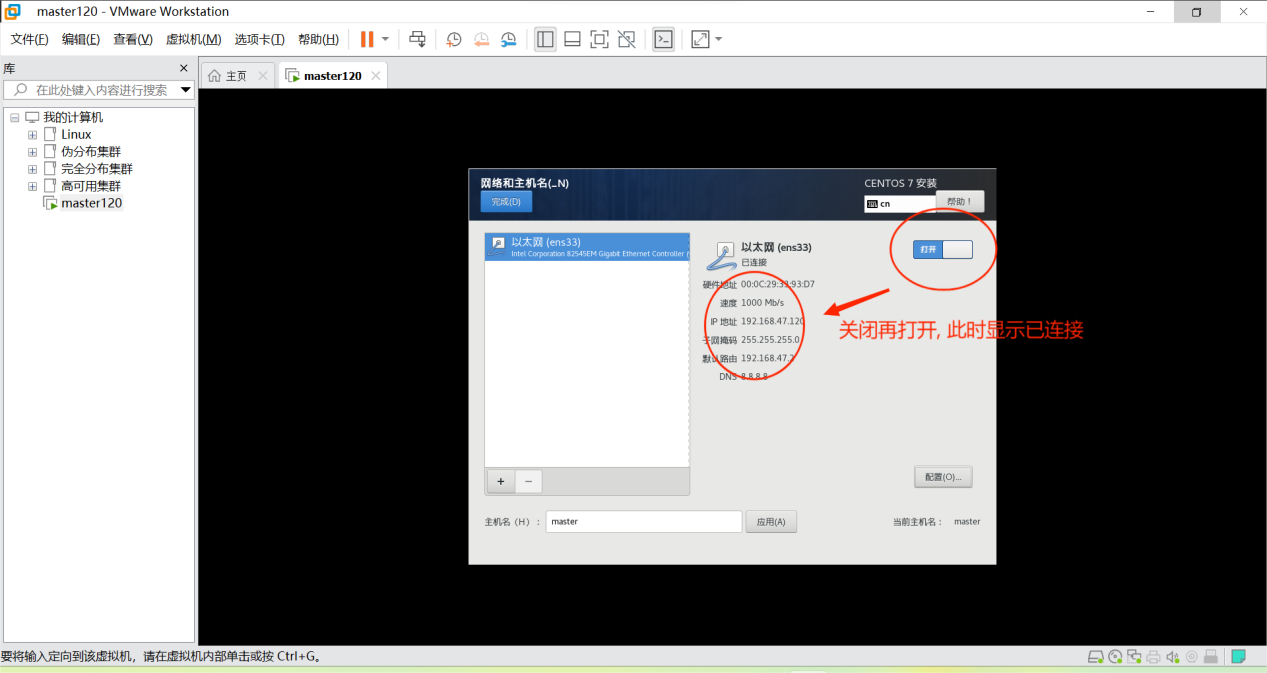
### 1. 打开VMware(17版本为例)从创建一台虚拟机开始

剩下的步骤如下图所示:



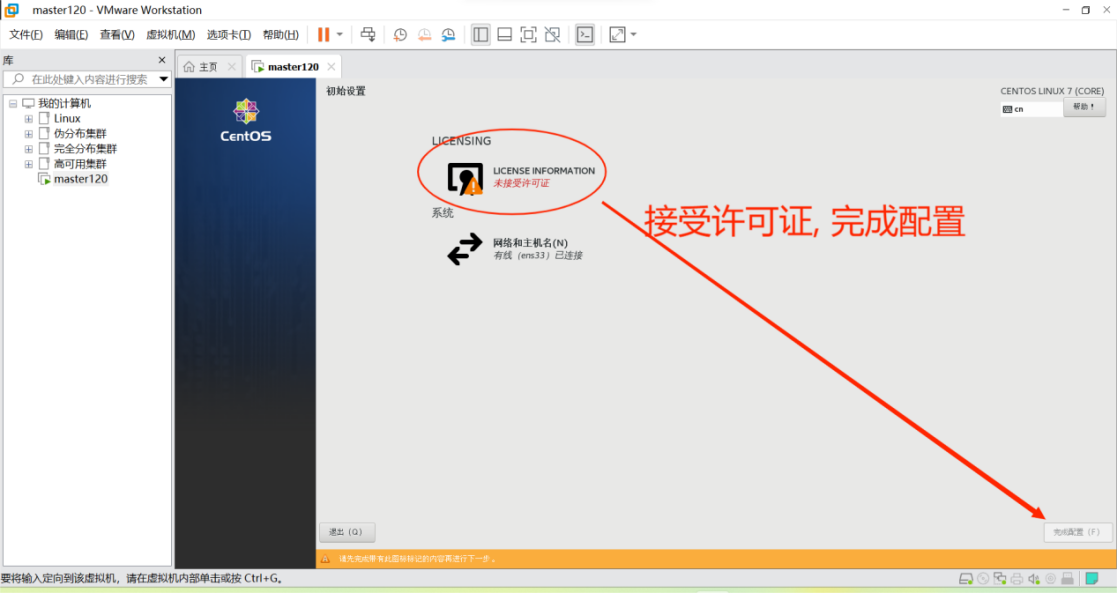


### 2. 打开新建好的虚拟机屏幕截图 2023-06-11 160828屏幕截图 2023-06-11 161432屏幕截图 2023-06-11 161447屏幕截图 2023-06-11 161500屏幕截图 2023-06-11 161516屏幕截图 2023-06-11 161531屏幕截图 2023-06-11 161702屏幕截图 2023-06-11 162829



### 3. 完成配置, 点击开始安装设置用户名和密码如图, 密码123456, 需要点击两遍完成屏幕截图 2023-06-11 161751屏幕截图 2023-06-11 161759

### 4. 等待安装进度条, 单机重启, 虚拟机创建完成屏幕截图 2023-06-11 164424



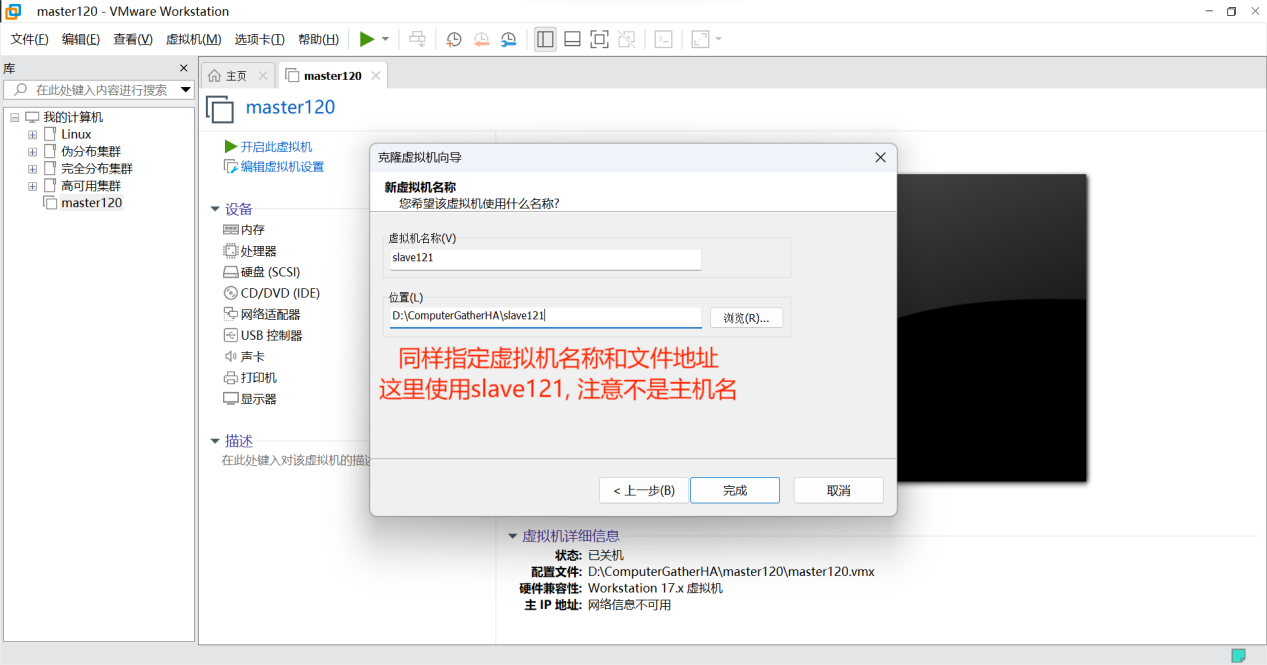
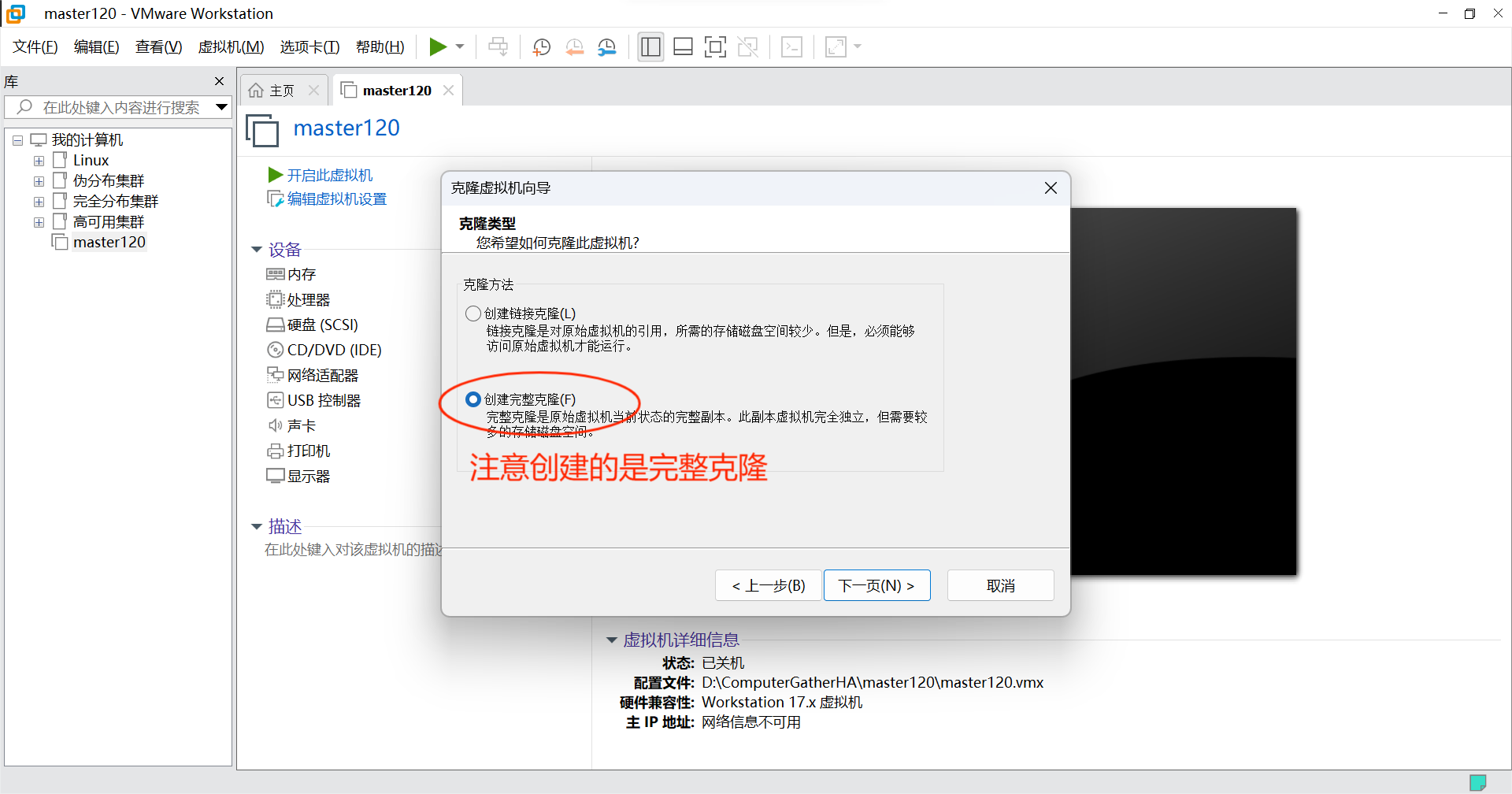
### 5. 为了方便继续选择中文, 成功屏幕截图 2023-06-11 164759

## 第二章. 多节点基础设置

### 1. 克隆

因为Hadoop高可用是多节点, 只有一台虚拟机显然不能满足条件, 又因为zookeeper种种原因(后面会谈到), 我们至少需要三台虚拟机. 为了节省时间, 我们现在需要将创建好的虚拟机克隆两次.

\*补充说明: 若在克隆虚拟机使创建链接克隆, 则需要保证父虚拟机不受损坏. 当父虚拟机删除或损坏时, 采用链接克隆生成的虚拟机也会无法使用! 磁盘空间充足时笔者建议尽量采用完整克隆, 克隆后修改配置方法完全相同.



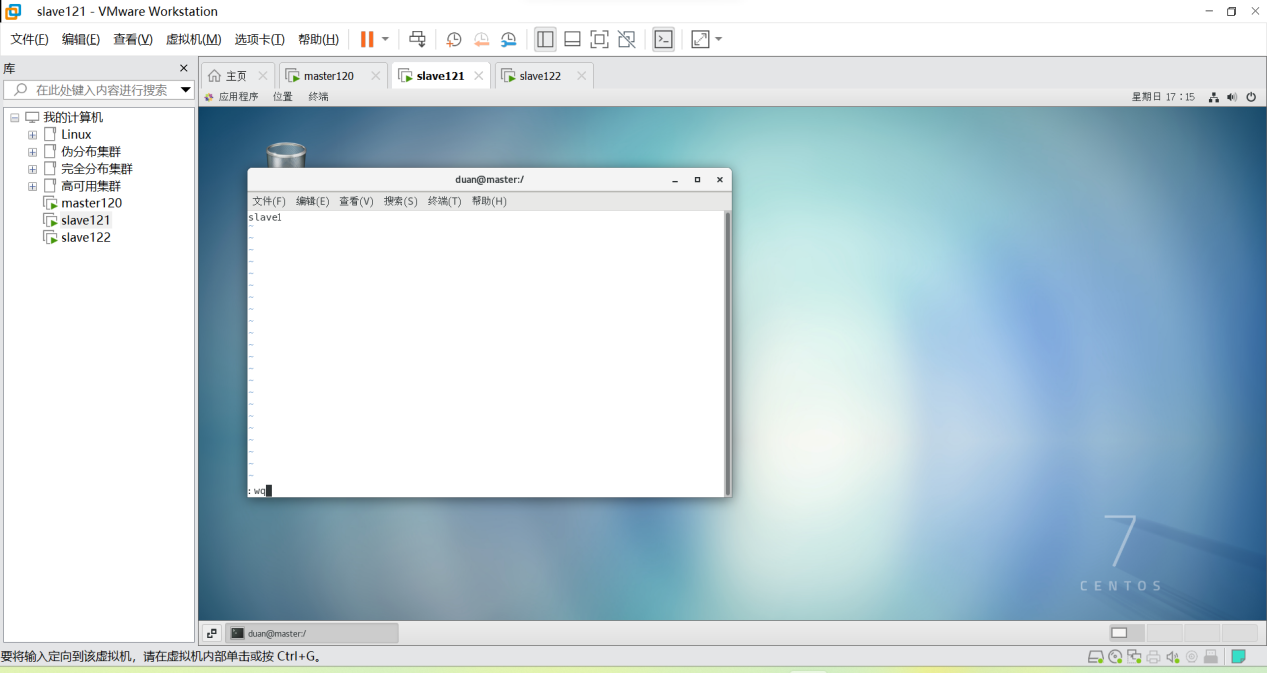
仿照上述方法, 创建了slave122

分别打开master120, slave121, slave122. 这里需要修改后两位的主机名, 打开终端, 使用如下命令

$ su root

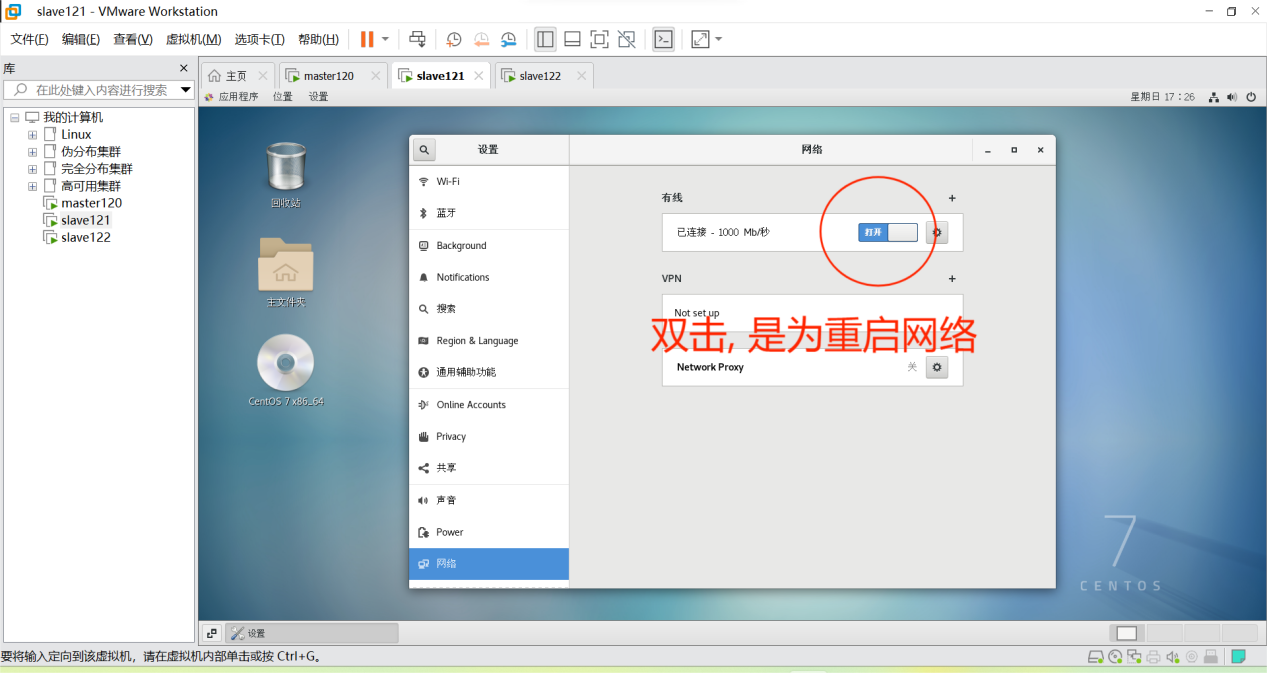
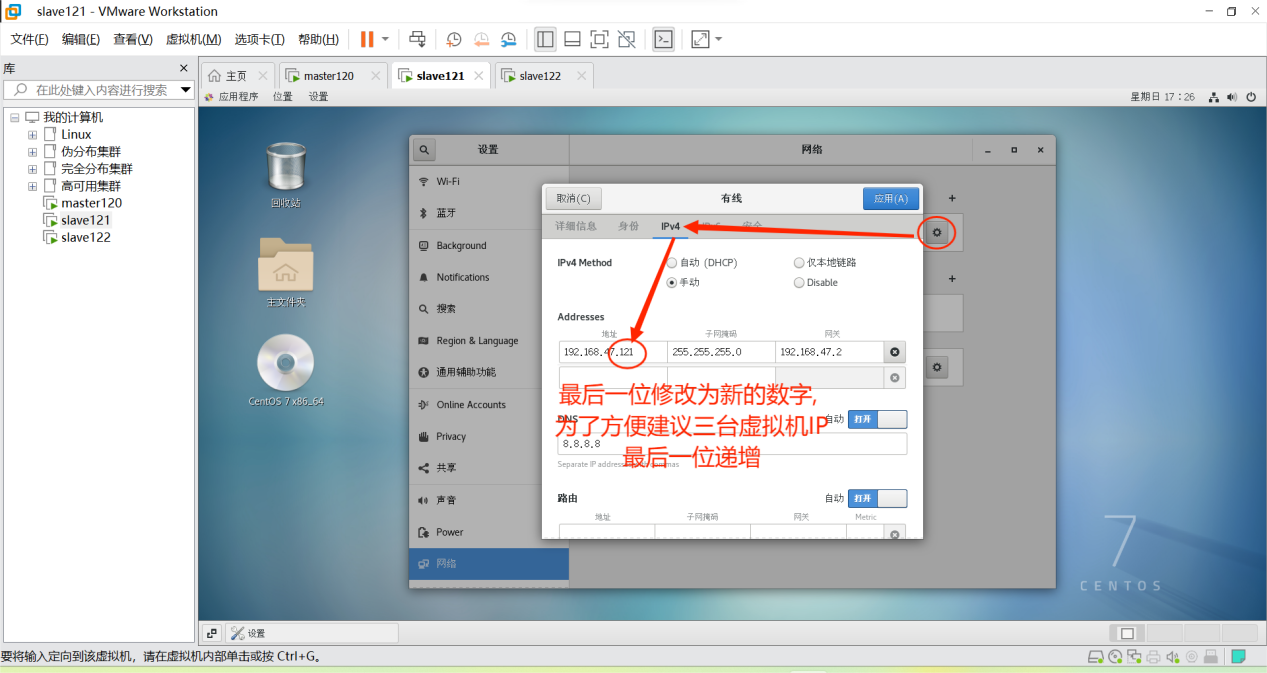
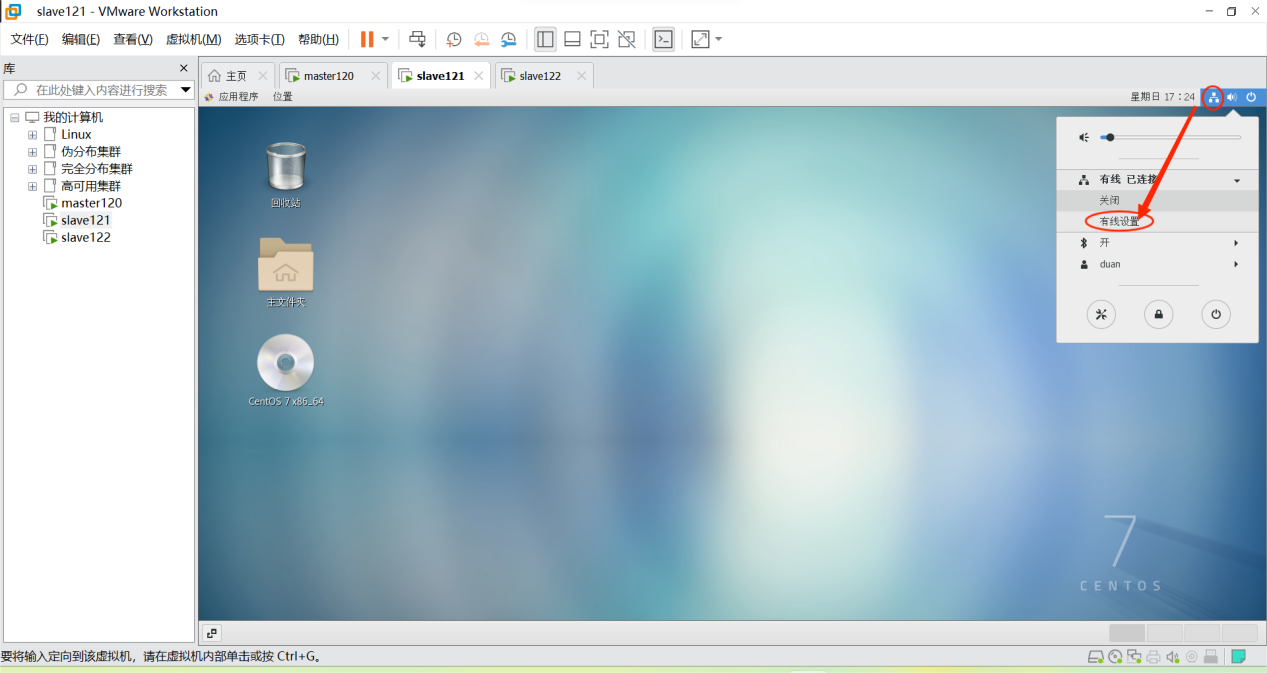
123456

#vim /etc/hostname



按 i 进入编辑模式, 将master改成slave1 , 按 : 回到命令行模式, 输入wq 回车, 是为保存并退出.

继续修改IP地址, 如图所示



再使用reboot命令重启虚拟机. 在slave122虚拟机上将master改成salve2, ip地址最后一位改为 122 , 其余步骤相同, 重启.

屏幕截图 2023-06-11 171534

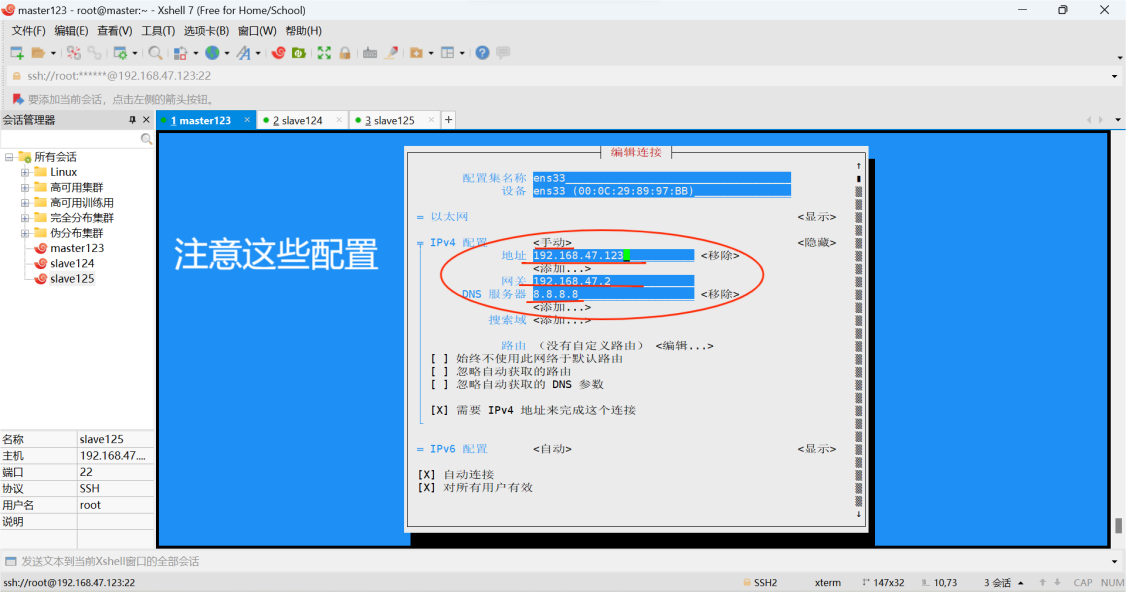
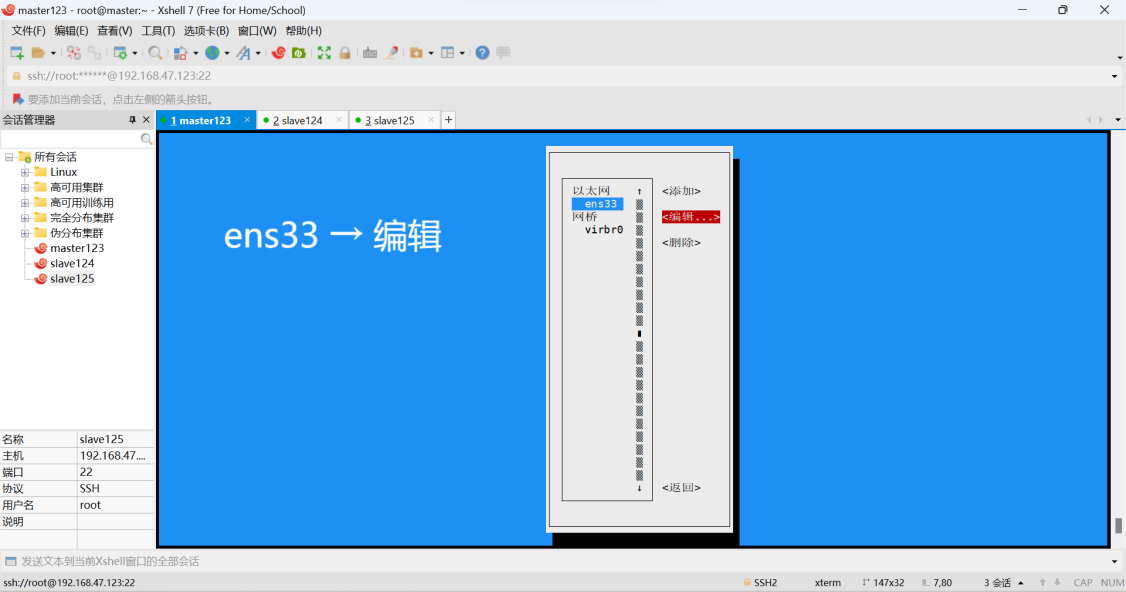
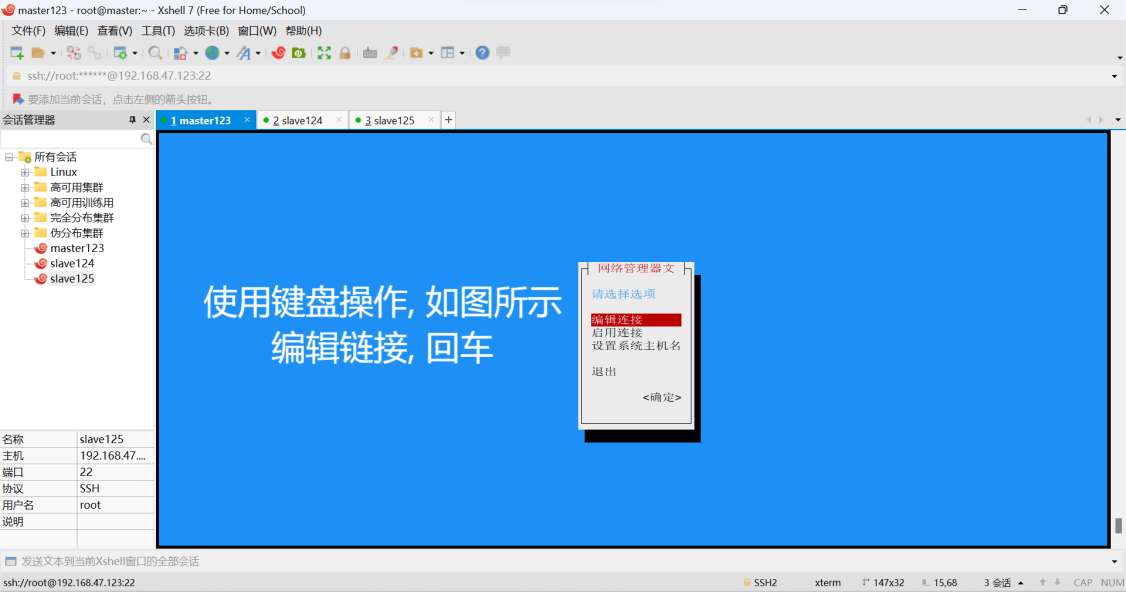
**注意: 下文采用master, slave1, slave2来分别指代三台虚拟机**

### 补充: nmtui界面修改网络

使用如下命令进入Linux自带的图形化网络配置页面:

步骤如图所示:

nmtui

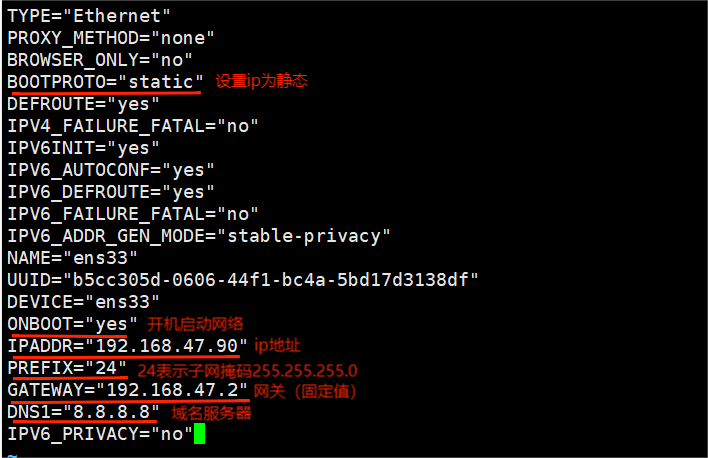


### 补充: 纯命令修改网络

此方法适用于没有图形界面的虚拟机使用，相对麻烦：

vim /etc/sysconfig/network-scripts/ifcfg-ens33

进入以下界面：



注意红色划线部分，第一次配置网络时可能会没有，此时需要手动添加。

备注：ip地址第三位与网关相同，第四位自定义。网关查询在虚拟网络编辑器选VMnet8中的net设置。

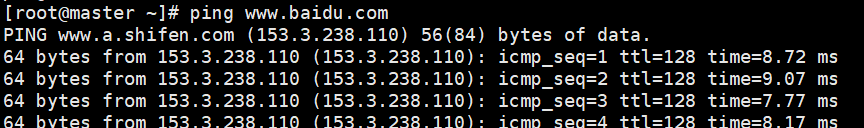
配置完成后Esc + :wq 保存，回到命令行输入

service network restart

出现以下界面重启网络完成：

屏幕截图 2023-09-10 161345

测试网络配置完成/Linux成功连上网：

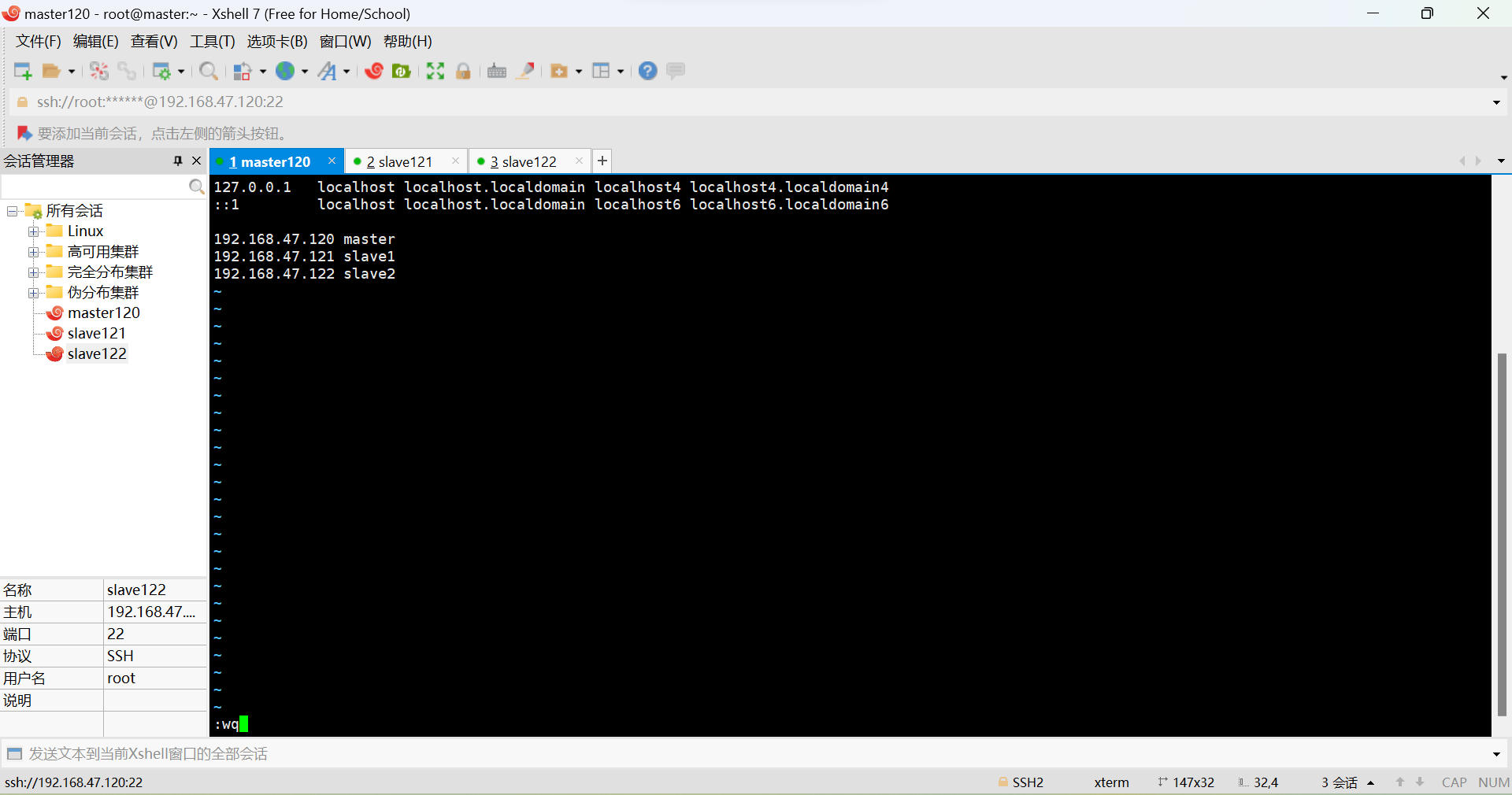
其他节点步骤相同。

### 2. 映射

在后续命令中, 为了方便输入速度和质量, 我们希望系统看到master就能定位到该虚拟机, 一般情况下, 机器靠着ip地址识别主机, 我们需要将主机名和ip地址互相映射, 由于我们已经将ip和网络配置好, 接下来的步骤让我们转到Xshell中实现.

在master中输入:

vim /etc/hosts

添加如下内容

将配置好的hosts文件分发给slave1和slave2, 覆盖它们原有的hosts文件:

系统提示是否继续建立连接, 我们回复 yes 并且输入密码123456, 传输成功.

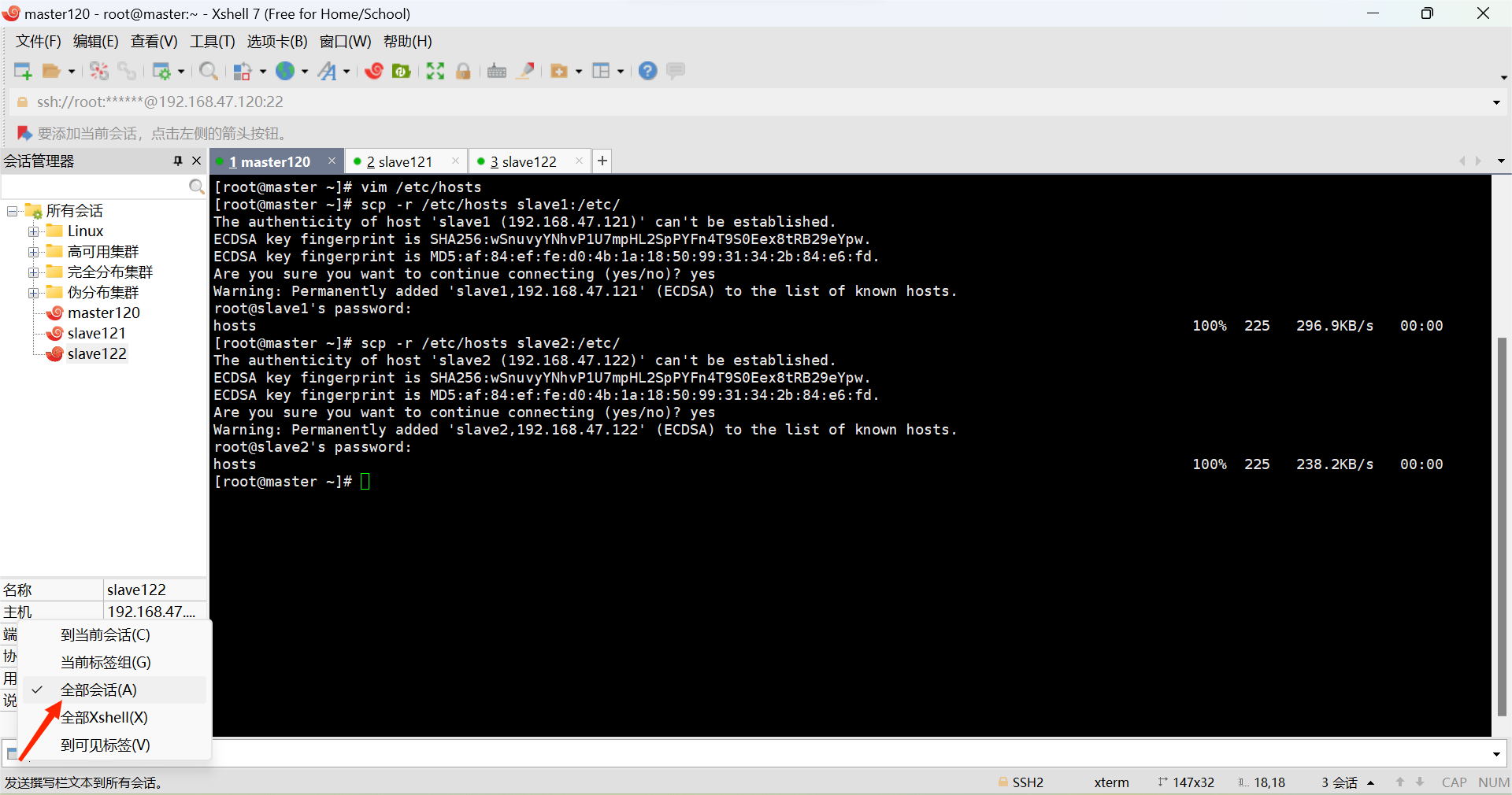
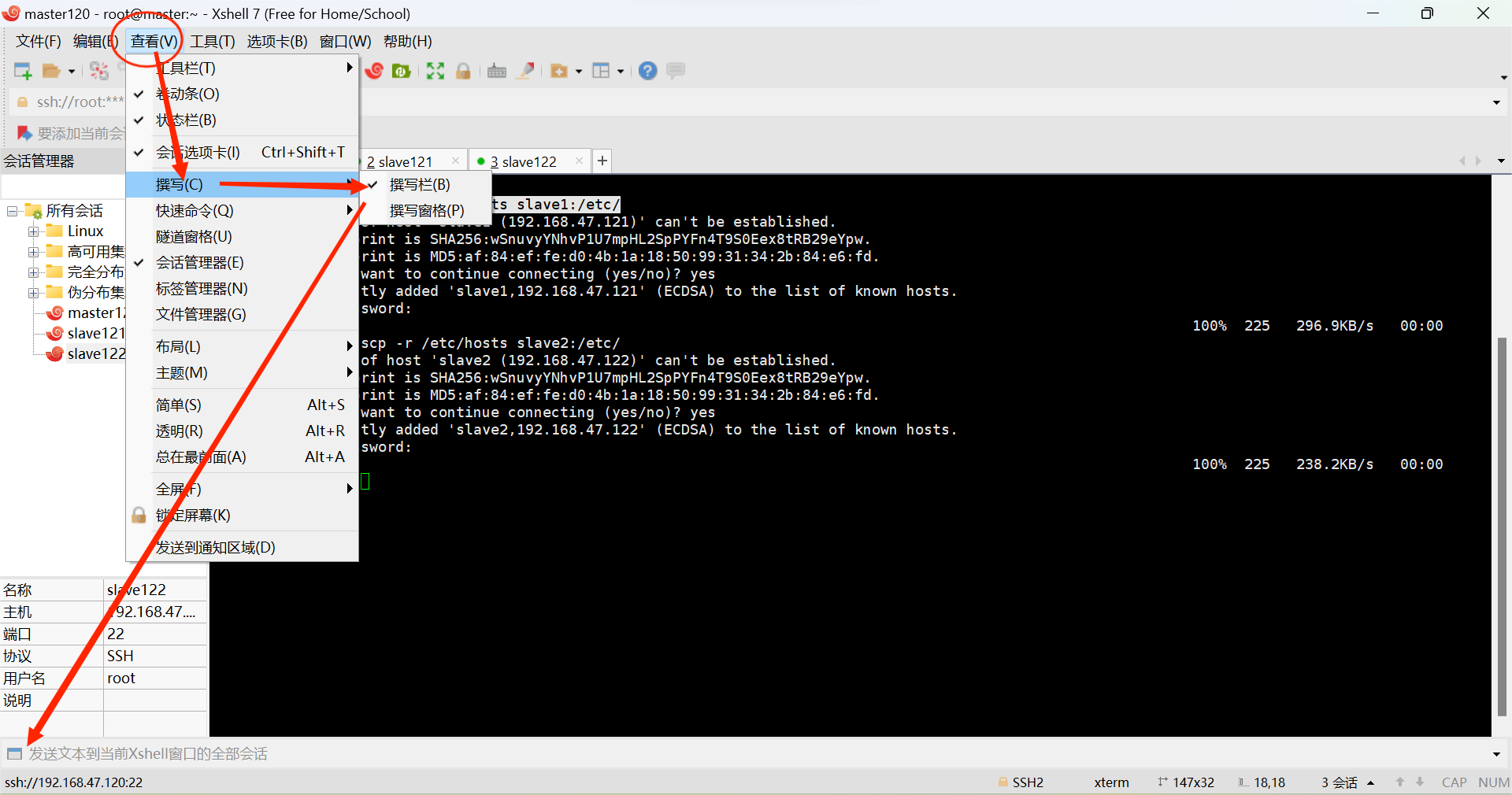


scp -r /etc/hosts slave1:/etc/

scp -r /etc/hosts slave2:/etc/

我们不难发现, 在多节点互相传递文件的时候必须要进行多余的确认和输入密码的步骤, 这是及其繁琐的, 在后续搭建中包括Hadoop的运行会大量传递文件, 所以为了保证高效, 我们需要在三台节点互相建立免密, 方便文件传输.

### 3. 免密

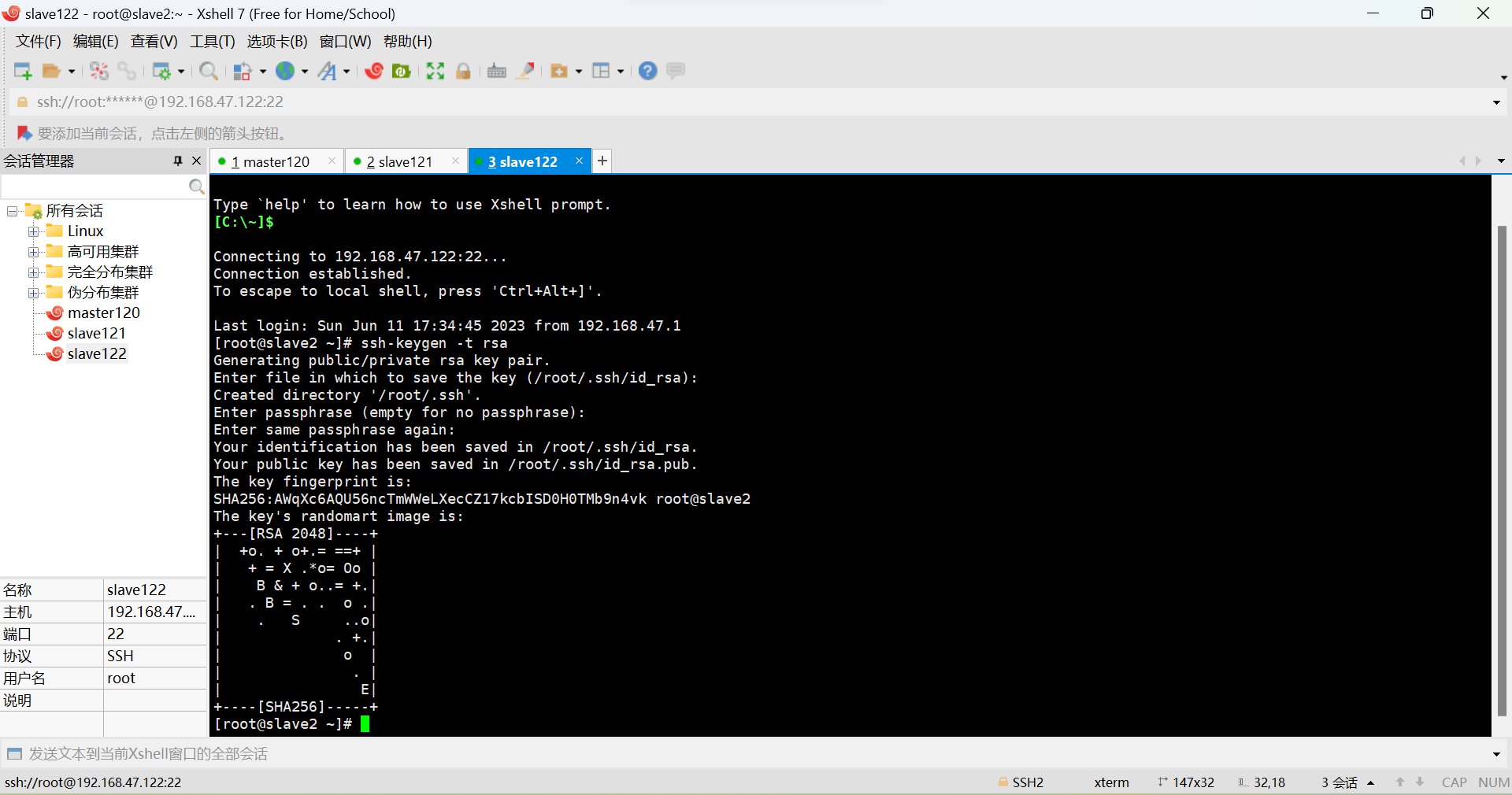
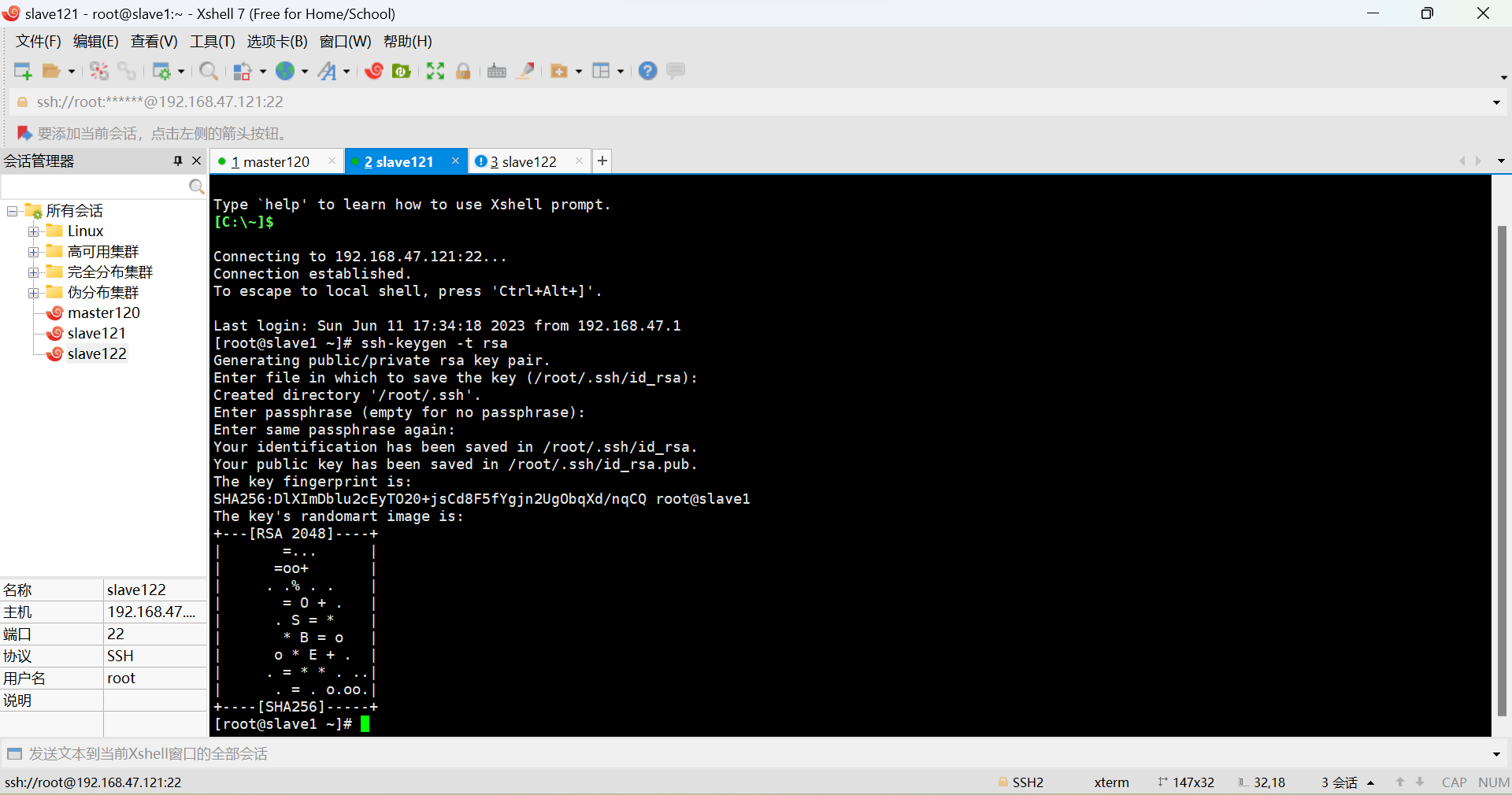
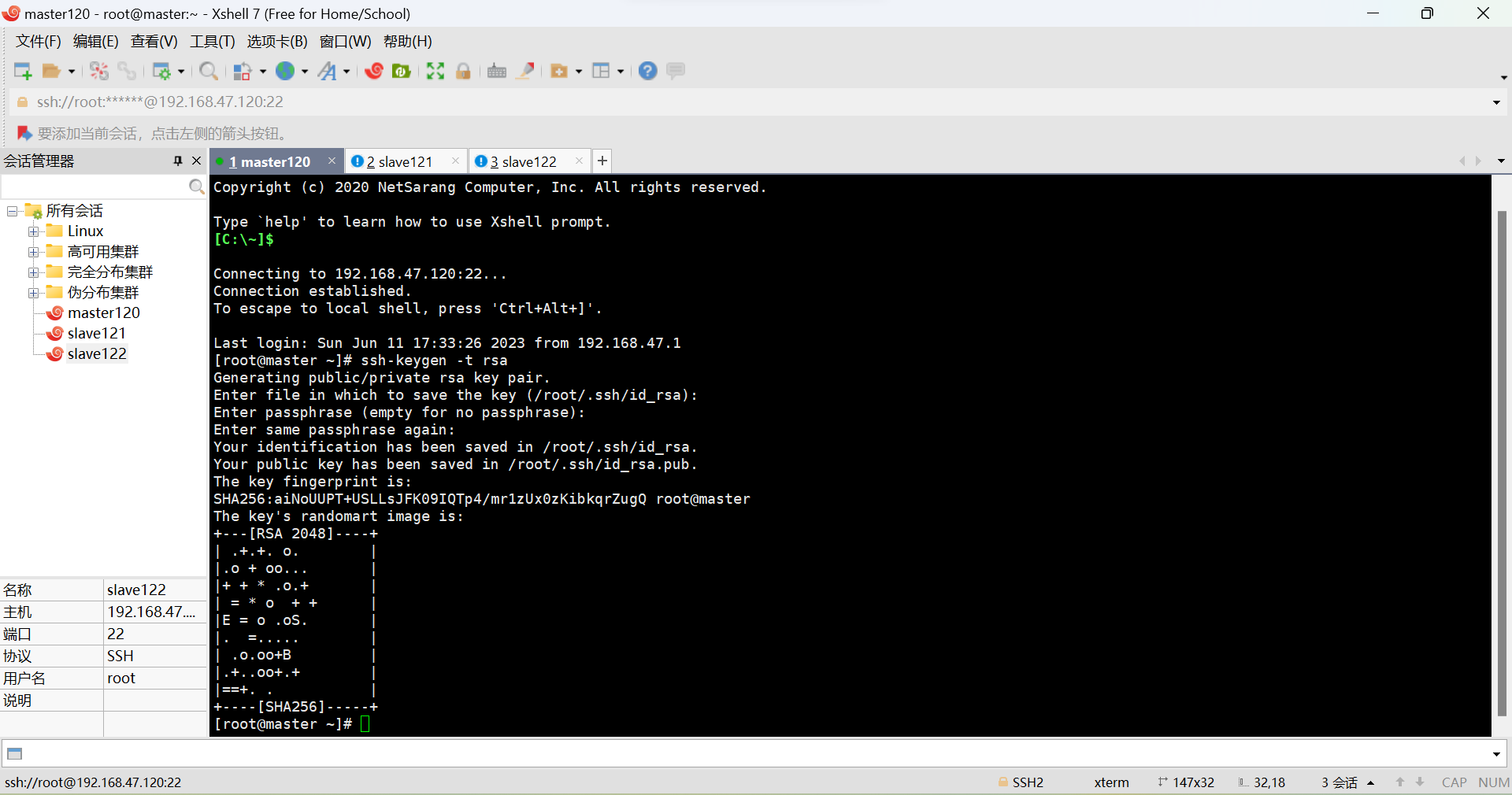
在此步骤我们打开Xshell的撰写工具, 它会极大方便地帮助我们同时操作多台虚拟机

在撰写栏中输入的代码, 会同时传递给三台虚拟机, 让我们试一下.

在**撰写栏**写入中生成密钥的指令:

ssh-keygen -t rsa

输入后在**撰写栏**中四次回车, 就会看到三台虚拟机已经同时创建好了密钥:



在生成密钥后, 将密钥在虚拟机之间互相发送才能得到免密登录这个效果, 所以为了避免所有状况, 我们需要在两两虚拟机之间传递密钥(自己也要给自己传递密钥), 如此一来, 三台虚拟机之间就要传递九次. 撰写工具可以很好地提高效率, 因为我们在撰写框中写的代码会发送给所有运行的虚拟机, 所以传递密钥的步骤我们只需输入三个代码即可.

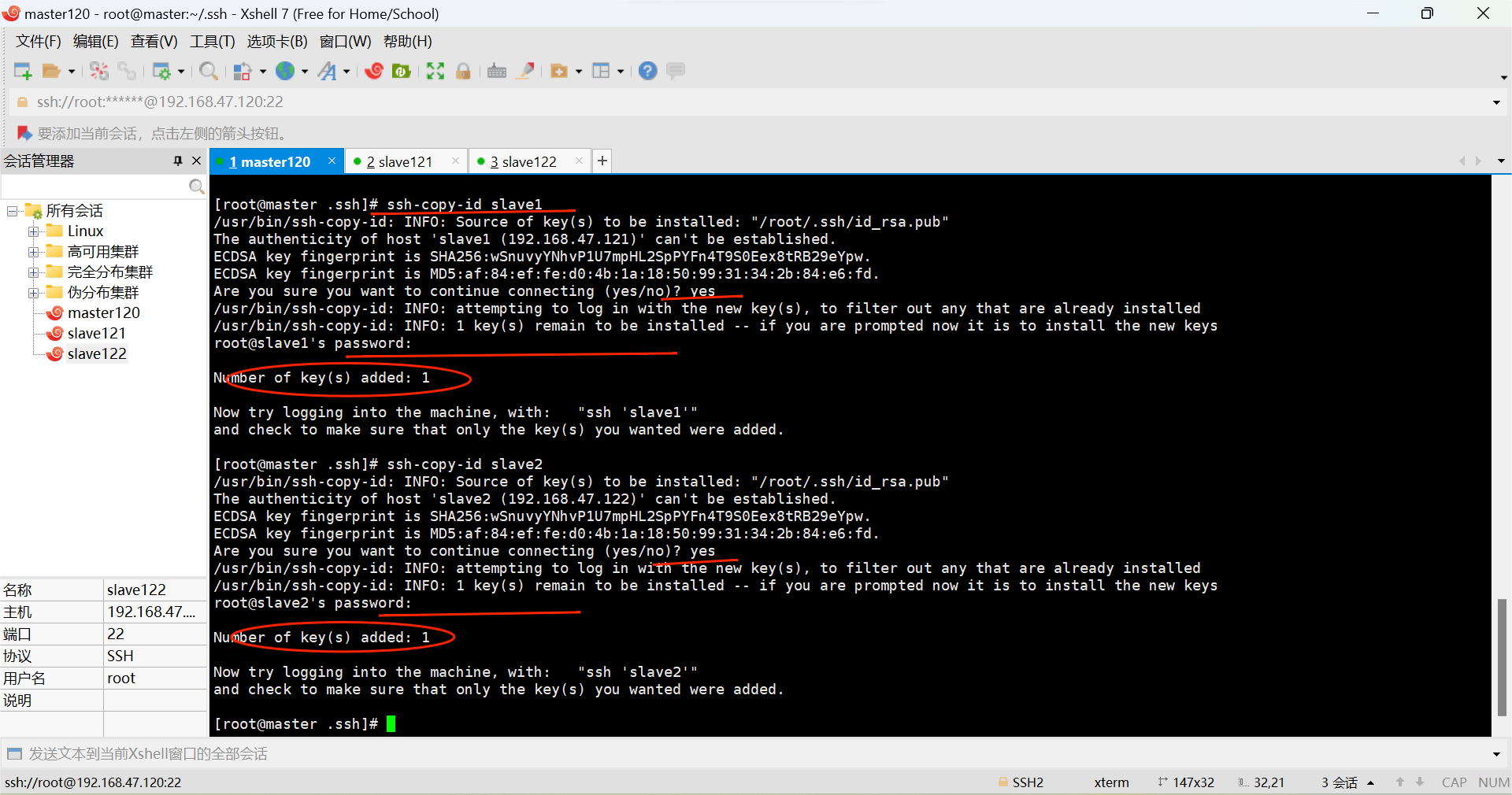
分别在**撰写栏**中输入 yes 和密码(若三台虚拟机密码不相同则分别在三台虚拟机的终端输入)

ssh-copy-id master

ssh-copy-id slave1

ssh-copy-id slave2

效果以master为例:



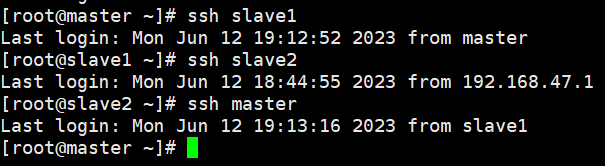
验证免密, 让我们输入:

我们会看到, 无论我们在哪一台虚拟机中输入, 运行后主机名都会被立即改变且不用输入密码, 则成功.

ssh master

ssh slave1

ssh slave2



### 4. 时钟同步

在Windows系统中, 主机时间和网络时间不同会给应用带来很大影响, 甚至导致正常的应用无法运行, 在Linux系统里也是相同. 三台机器的时间不同, 会给Hadoop运行带来难以估计的后果, 因此我们需要进行时钟同步.

这是一条来自阿里云的时钟同步命令, 分别在三台虚拟机中输入, 很明显, 我们可以采用**撰写栏**一步到位:

屏幕截图 2023-06-12 192419效果如下:

ntpdate ntp5.aliyun.com

屏幕截图 2023-06-12 192428屏幕截图 2023-06-12 192437

### 4. 时钟同步的方法二

注意: 两种时钟同步命令请读者择其一使用, 若两者同时使用第一种方法会无效.

安装ntp服务:

yum -y install ntp

修改配置文件:

vim /ect/sysconfig/ntpd

在其中加入:

:wq保存退出

OPTIONS=”-g”

YS\_HWLOCK=yes

设置开启启动和启动ntp服务:

演示略, 可用date命令查看结果

systemctl enabled ntpd

systemctl start ntpd

### 5. 关闭防火墙

三台主机都需要做这一步操作, 在**撰写栏**中键入:

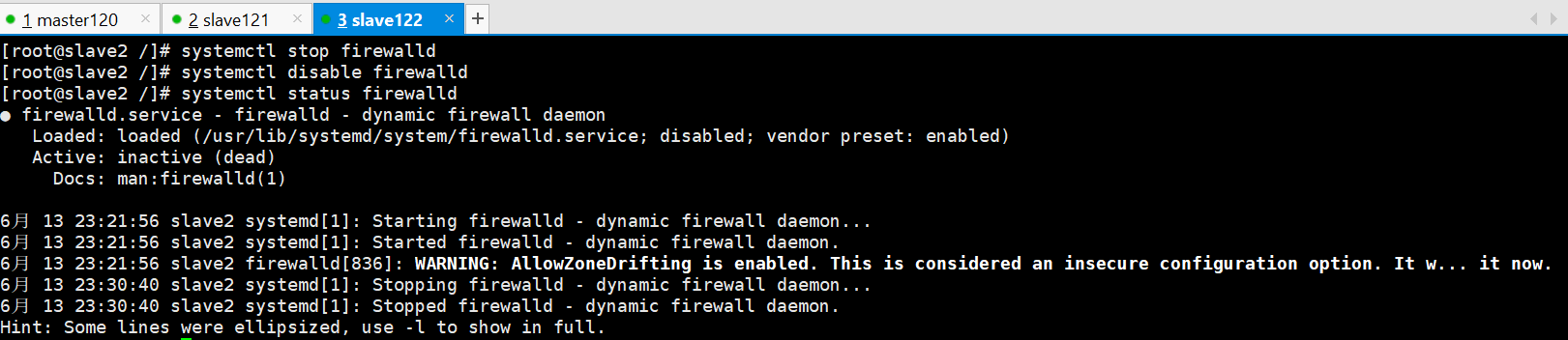
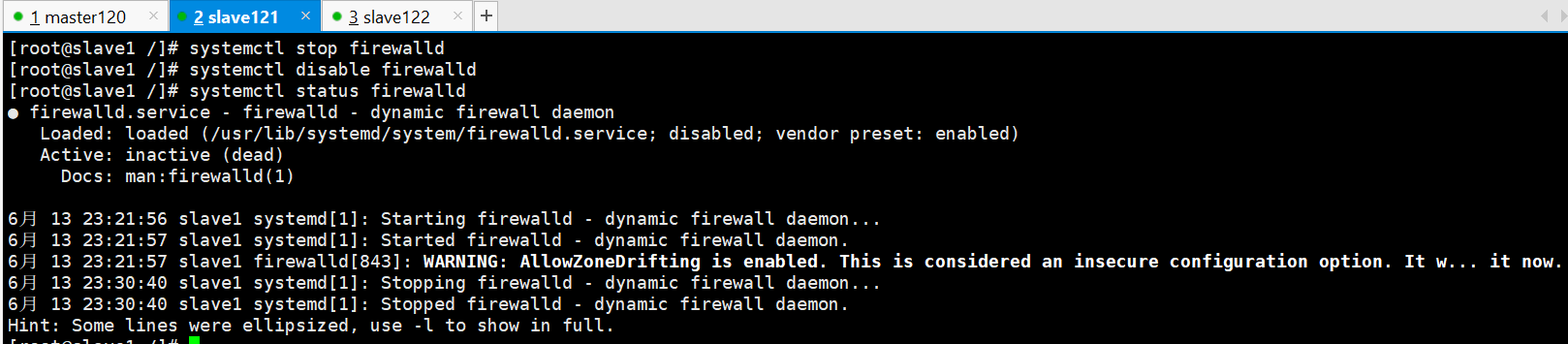
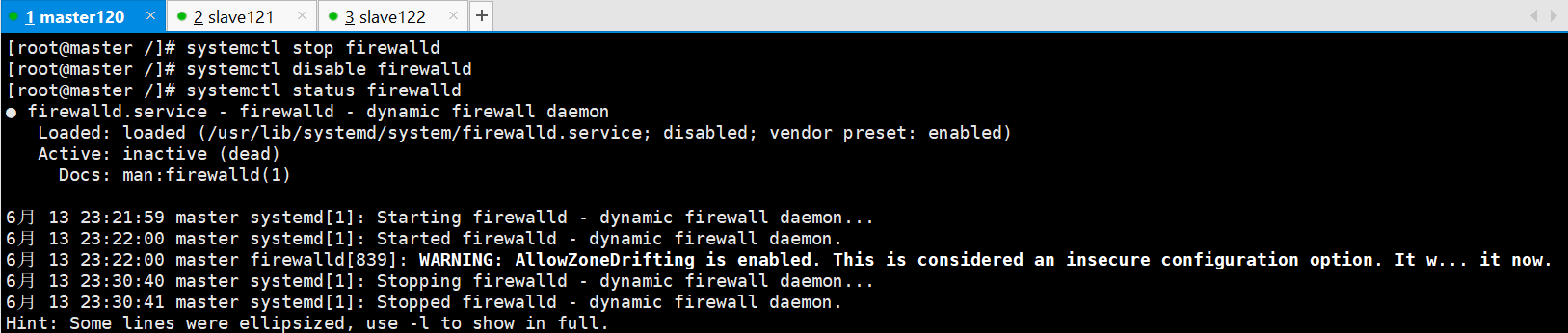
(第三个代码为了检查防火墙是否已经关闭)

systemctl stop firewalld

systemctl disable firewalld

systemctl status firewalld

如图:



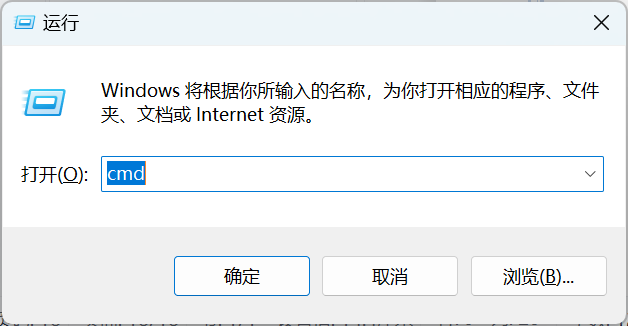
## 第三章. Java安装

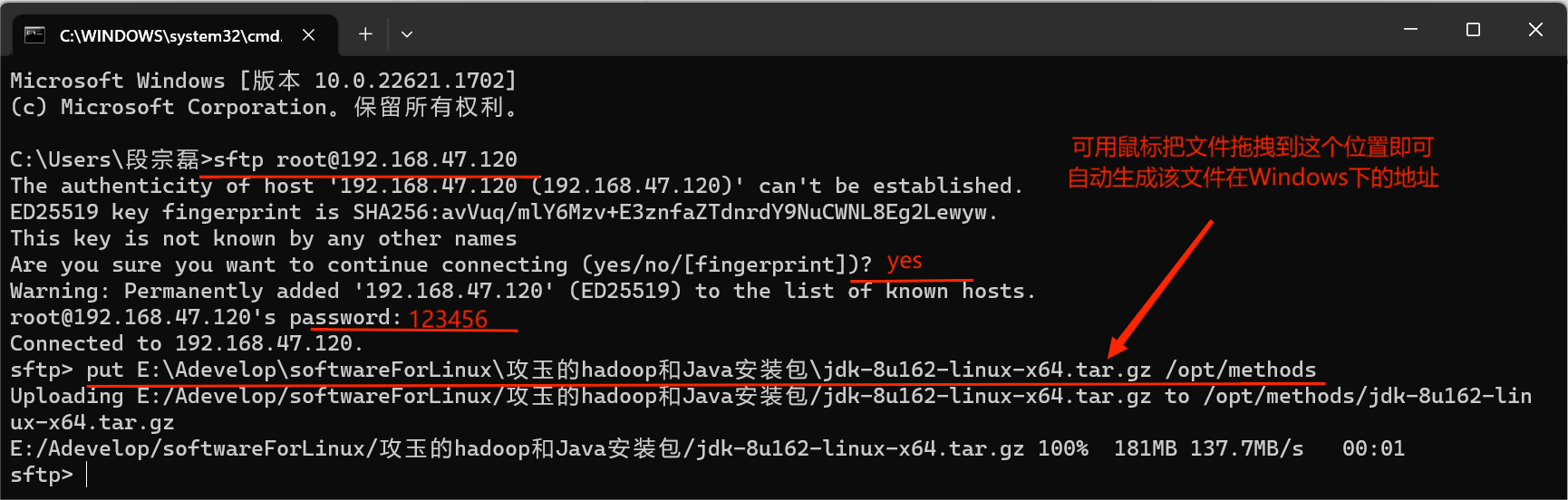
### 1. 导入, 解压, 重命名

在master中创建/opt/methods文件夹用来放置软件压缩包(-p为了防止你连/opt都没有)

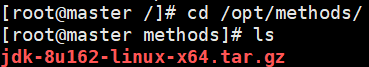
mkdir -p /opt/methods

在Windows中使用 Win + r 输入cmd打开终端

在终端中使用sftp协议连接你的master节点(这里需要使用ip地址), 用put命令将文件传给master中的/opt/methods, 代码和运行如下↓



回到master中检查是否传入成功:

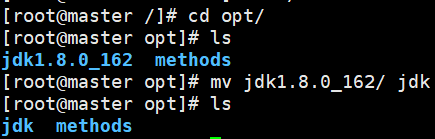


将该安装包解压到/opt下:

回到/opt下, 将解压后的文件名改名为jdk

mv jdk1.8.0\_162/ jdk

tar -zvxf jdk-8u162-linux-x64.tar.gz -C /opt

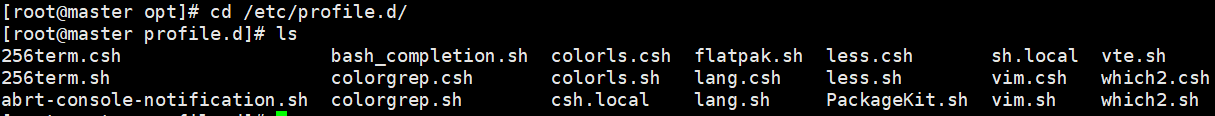


### 2. 设置环境变量

为了全局使用Java的命令, 我们要设置环境变量

为了保护环境变量文件的安全性, 我们进入到/etc/profile.d文件夹下:

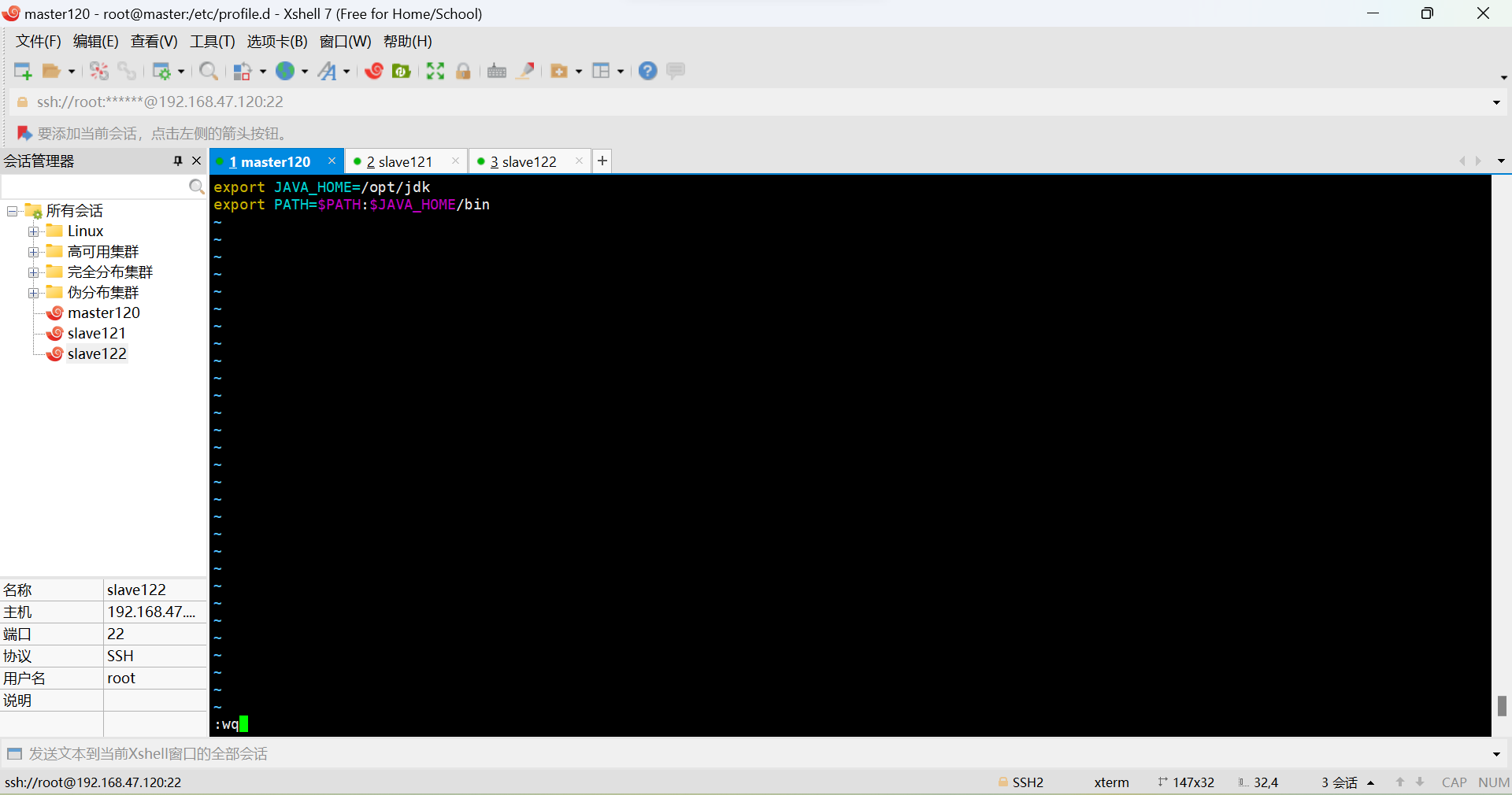
cd /etc/profile.d



在此创建data.sh作为添加环境变量的文件

vim data.sh

并写入以下内容:



:wq保存并退出

export JAVA\_HOME=/opt/jdk

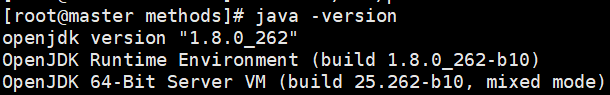
export PATH=$PATH:$JAVA\_HOME/bin

输入:

source /etc/profile

### 3. 测试Java, 分发

java -version



我们已经成功地在master上装好了Java, 那么我们需要将Java文件和环境变量分别传给slave1和slave2:

分别输入:

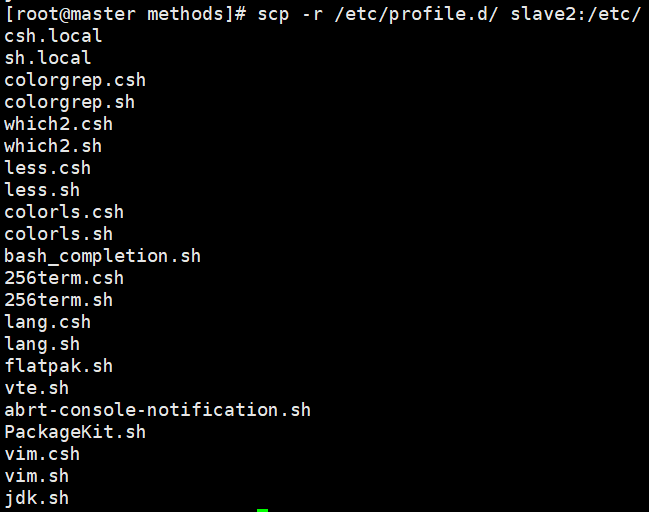
图例(以第四个代码为例):

scp -r /opt/jdk/ slave1:/opt/

scp -r /opt/jdk/ slave2:/opt/

scp -r /etc/profile.d/ slave1:/etc/

scp -r /etc/profile.d/ slave2:/etc/



在**撰写栏**中输入:

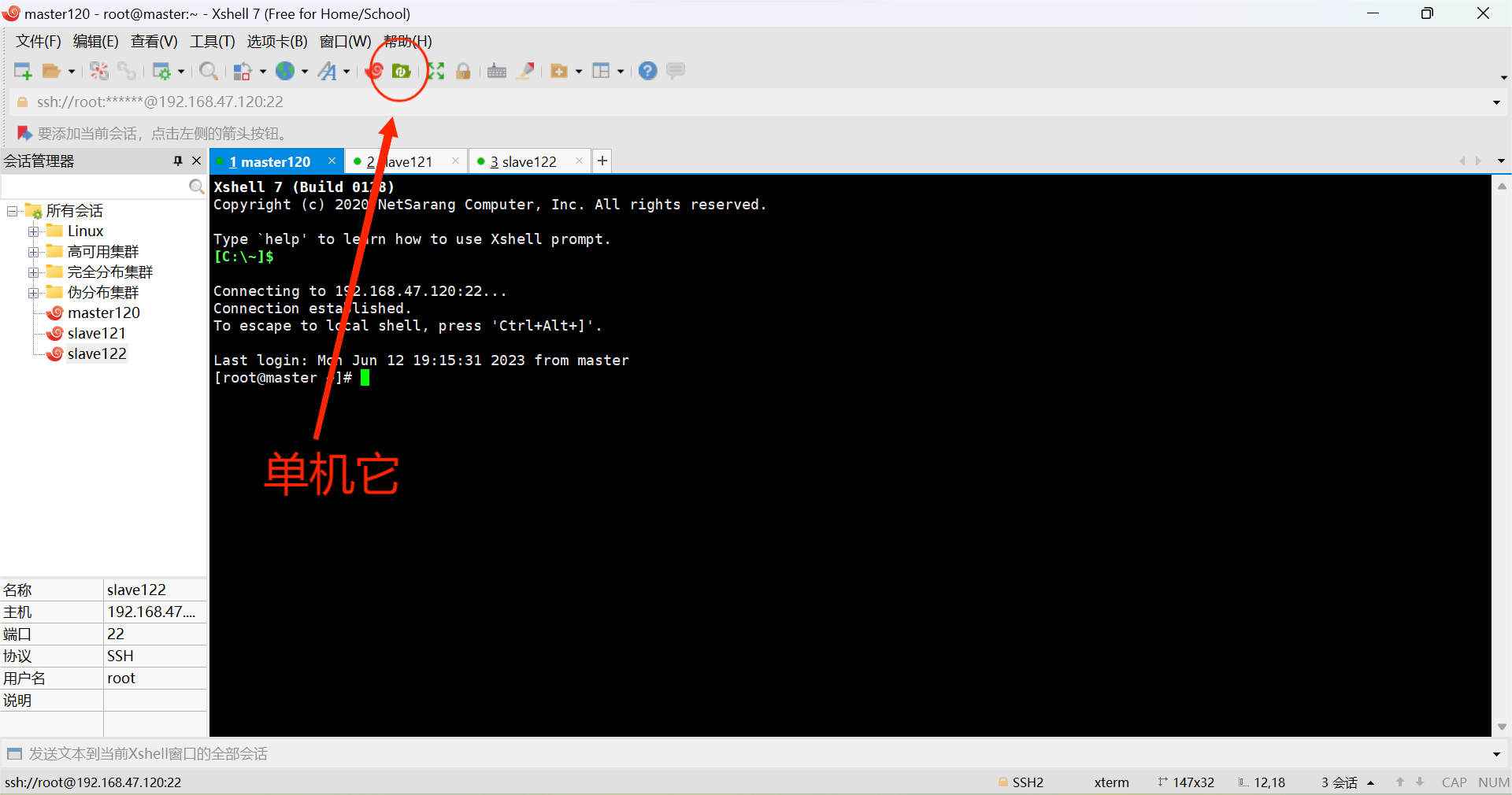
结束Java配置.

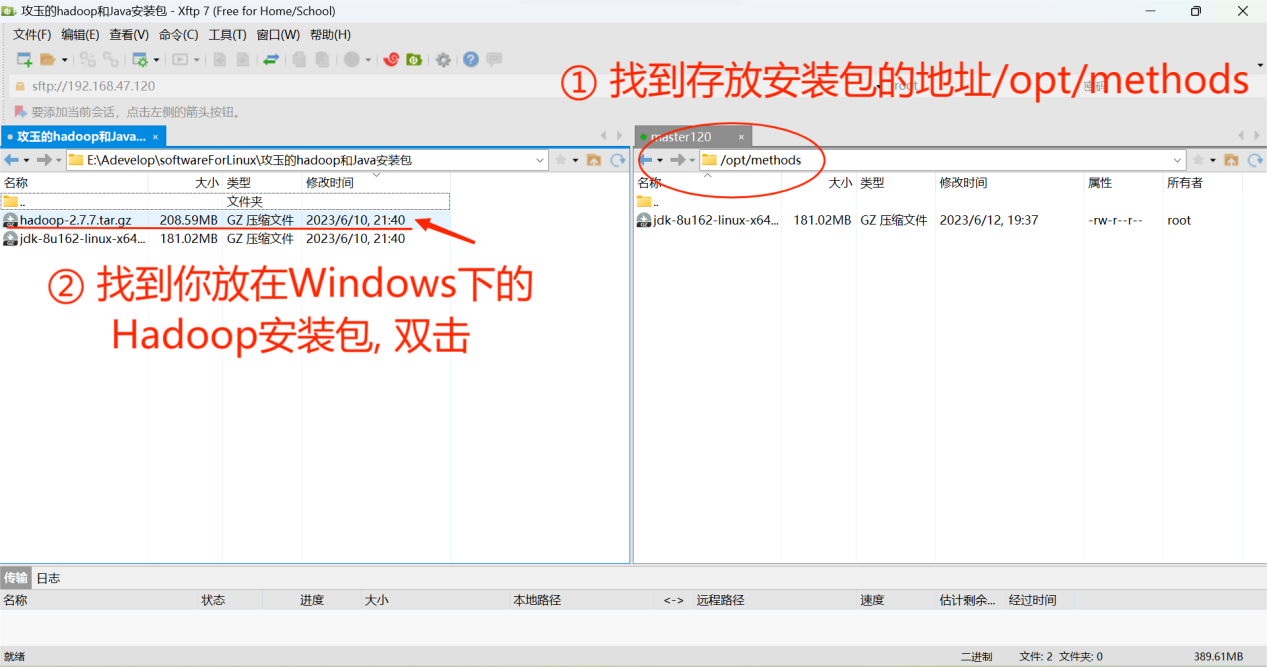
source /etc/profile

## 第四章. Hadoop初步安装

### 1. 导入, 解压, 重命名

如同第三章一样, 导入解压重命名的操作与Java并无二致. 我们当然可以使用上文提到的Windows命令行来进行put导入. 但是这一次我们使用Xftp工具导入

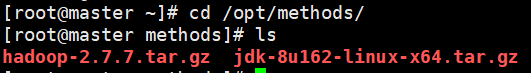
如图:



那么进行到这一步, 我们便可以在/opt/methods中找到你的Hadoop安装包

cd /opt/methods

ls

下一步, 解压它到/opt下

tar -zxvf hadoop-2.7.7.tar.gz -C /opt/

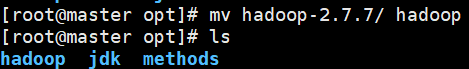
回到/opt下, 改名为hadoop

cd /opt

mv hadoop-2.7.7/ hadoop

ls

如图:



### 2. 设置环境变量

同第三章-2, 设置环境变量

在其中按下 i 输入:

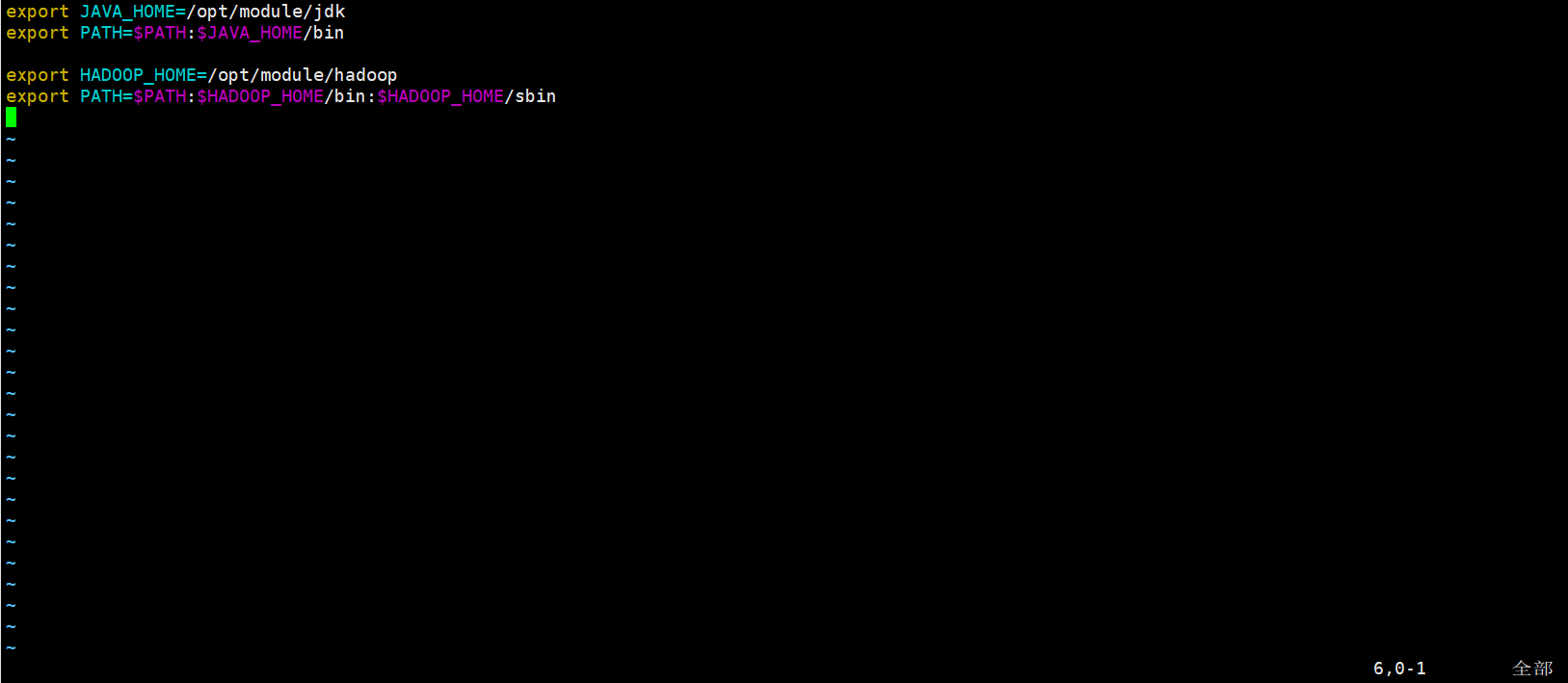
cd /etc/profile.d

vim data.sh

:wq保存退出

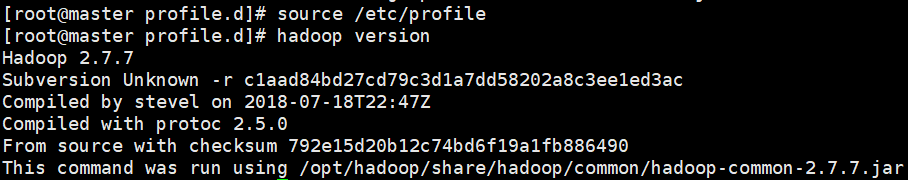
export HADOOP\_HOME=/opt/hadoop

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



### 3. 测试Hadoop是否成功安装

输入命令:



显示了Hadoop版本号, 成功安装!

source /ect/profile

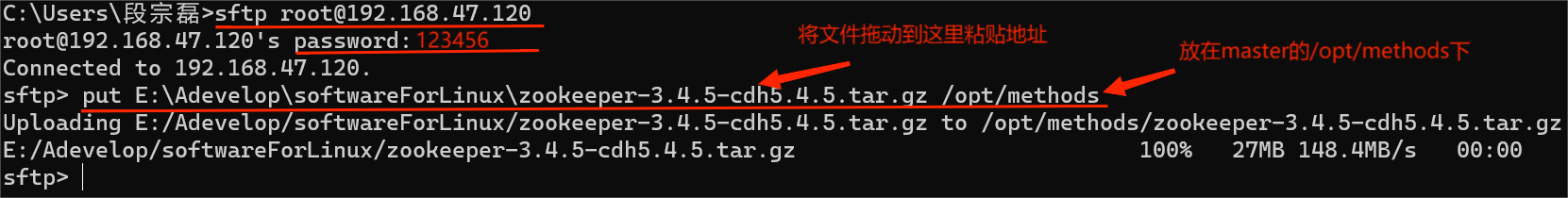
hadoop version

## 第五章. zookeeper安装与配置

### 1. 导入, 解压, 重命名

这一步同上, 不再过度赘述

传入zookeeper安装包:



解压zookeeper安装包:

cd /opt/methods

tar -zvxf zookeeper-3.4.5-cdh5.4.5.tar.gz -C /opt/

重命名为zookeeper:

### 屏幕截图 2023-06-13 2237362. 设置环境变量

这一步基本同上, 在/etc/profile.d下创建zookeeper.sh

cd /etc/profile.d

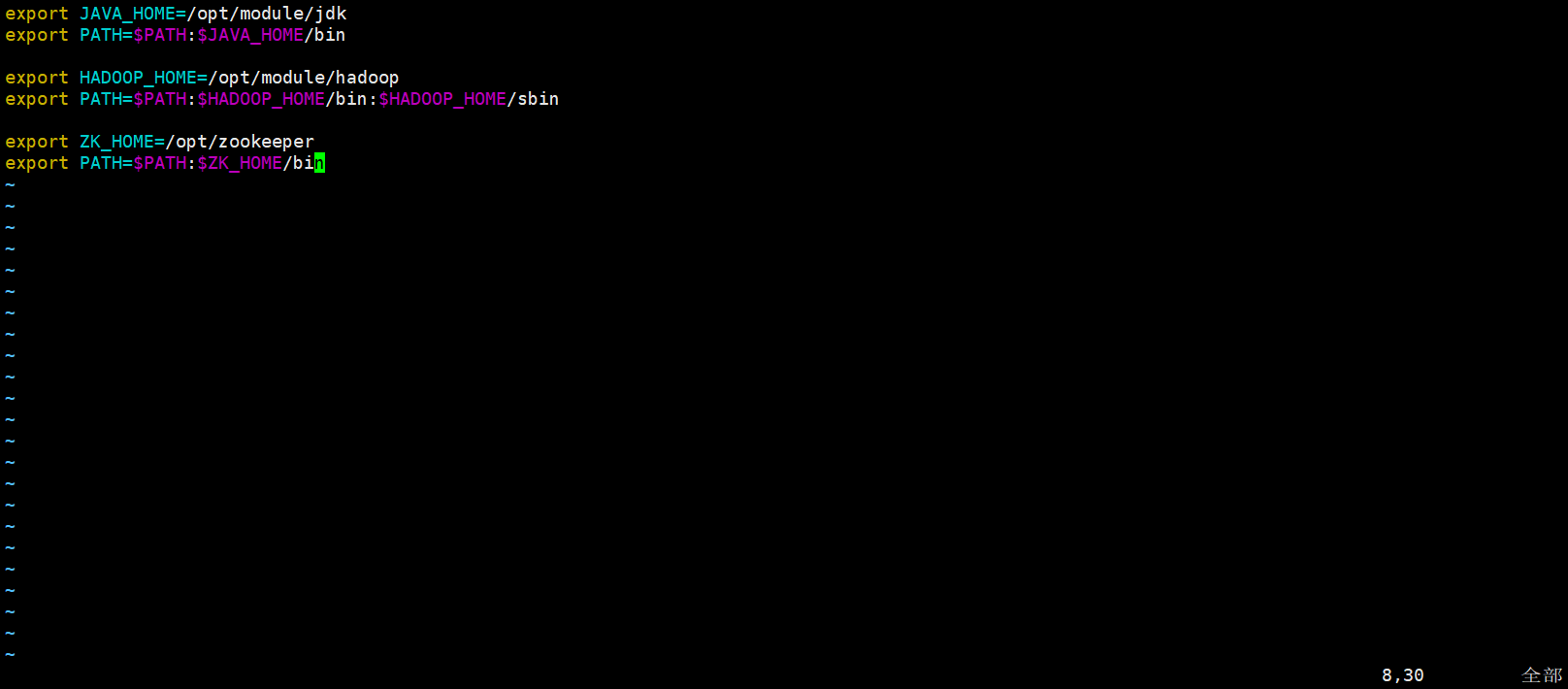
vim data.sh

写入:

:wq保存并退出

export ZK\_HOME=/opt/zookeeper

export PATH=$PATH:$ZK\_HOME/bin



刷新环境变量:

source /etc/profile

### 3. 配置zookeeper文件

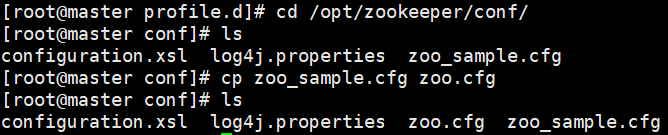
zookeeper需要至少三台机器才能投票选主, 故我们需要配置文件

进入/opt/zookeeper/conf下将zoo\_sample.cfg改为zoo.cfg:

cd /opt/zookeeper/conf

cp zoo\_sample.cfg zoo.cfg

如图:



进入zoo.cfg开始编辑文件:

将其中的文件不同的地方修改成如下这样:

vim zoo.cfg

tickTime=2000

initLimit=10

syncLimit=5

dataDir=/opt/zookeeper/data

dataLogDir=/opt/zookeeper/logs

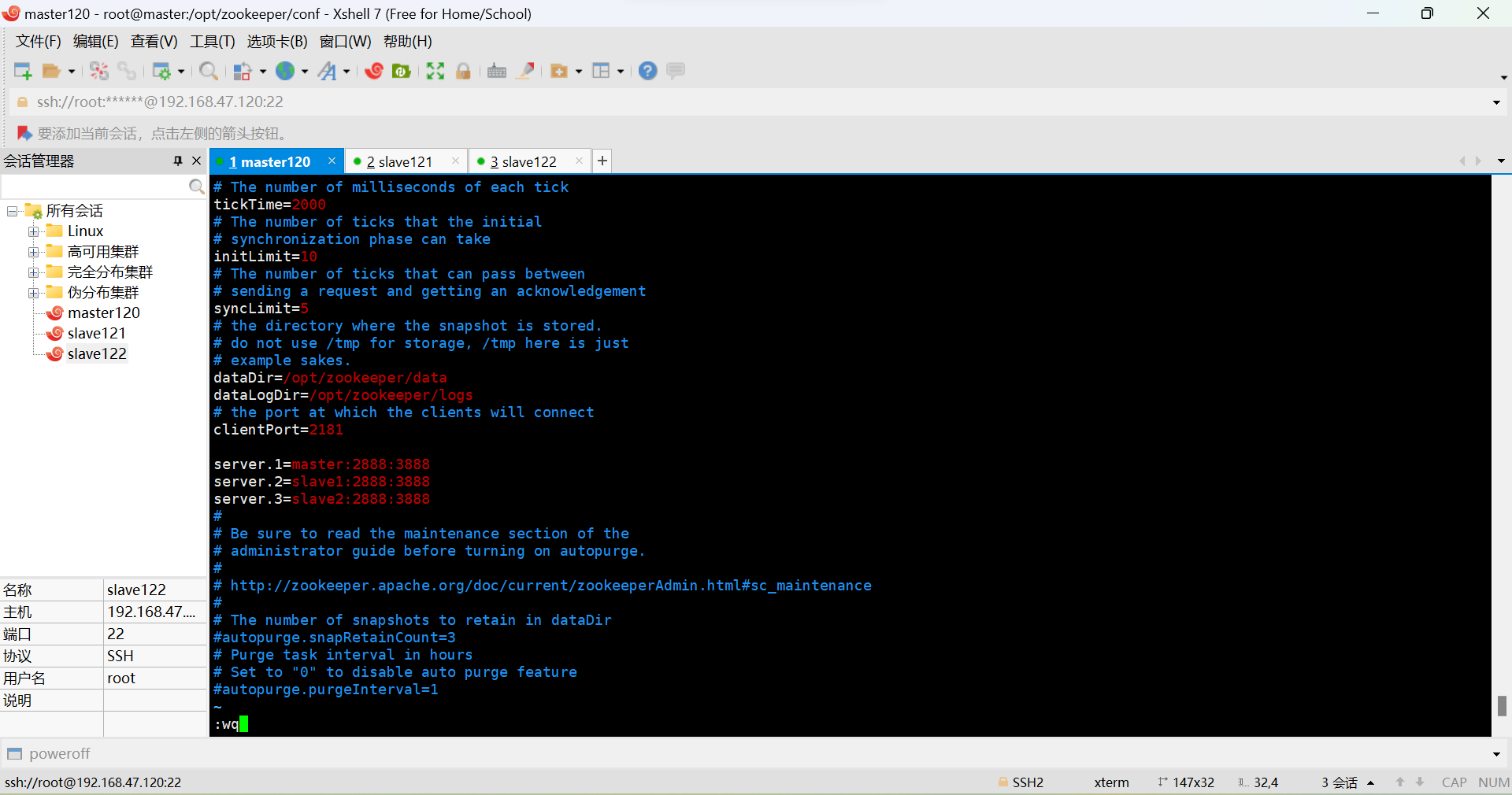
clientPort=2181

server.1=master:2888:3888

server.2=slave1:2888:3888

server.3=slave2:2888:3888

如图(蓝色字体是注释):



:wq保存并退出

创建目录用来存放文件和id号

在/opt/zookeeper/data下创建myid文件夹, 写入1

cd /opt/zookeeper/data

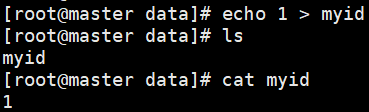
echo 1 > myid

mkdir /opt/zookeeper/data

mkdir /opt/zookeeper/logs

并且检查, 确保里面只有字符1

如图:



### 4. 分发与测试

我们已经配置好一台主机上的zookeeper, 现在需要将配置好的文件分发给其他两台主机

注意: 我们需要转递的是/etc/profile.d下的配置文件和zookeeper文件

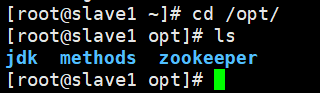
检查(到slave1下查看zookeeper为例):

scp -r /etc/profile.d/ slave1:/etc/

scp -r /etc/profile.d/ slave2:/etc/

scp -r /opt/zookeeper/ slave1:/opt/

scp -r /opt/zookeeper/ slave2:/opt/



更新了环境变量, 我们需要刷新使其生效, 这里我们使用**撰写栏**写入刷新代码:

source /etc/profile

如图:

屏幕截图 2023-06-13 230648屏幕截图 2023-06-13 230657屏幕截图 2023-06-13 230735屏幕截图 2023-06-13 230741将slave1的myid改为**2**, 将salve2的myid改为**3**, 对应zoo.cfg中的配置数字

演示略

vim /opt/zookeeper/data/myid

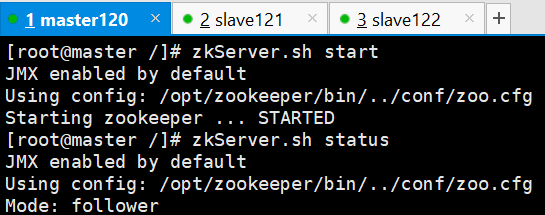
现在我们来测试zookeeper在三台主机上正常运行

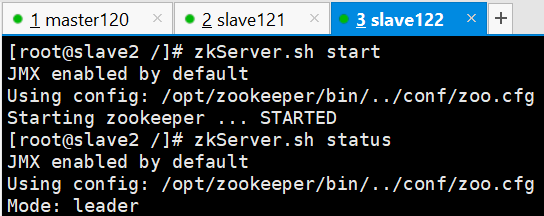
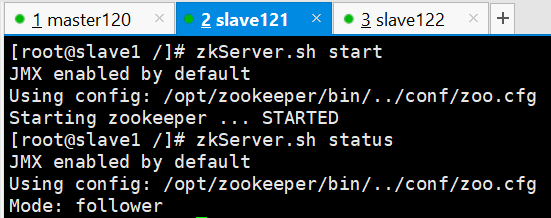
在**撰写栏**中键入:

效果如图所示, 其中有一个节点显示leader:

zkServer.sh start

zkServer.sh status





再输入jps查看所有进程, 此时进程应该是这两个(每台进程编号不同):

屏幕截图 2023-06-13 234327

在**撰写栏**中写入如下命令停止zookeeper的进程:

zkServer.sh stop

zookeeper结束配置.

jps

## 第六章. 高可用部署

### 1. 配置前的准备工作

在此之前, 我们需要创建文件夹用来存放Hadoop运行时产生的数据:

在master中:

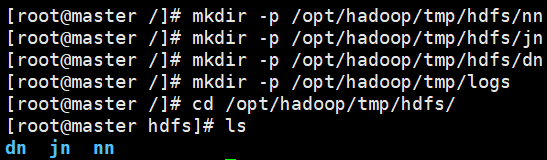
就像这样:

mkdir -p /opt/hadoop/tmp/hdfs/nn

mkdir -p /opt/hadoop/tmp/hdfs/dn

mkdir -p /opt/hadoop/tmp/hdfs/jn

mkdir -p /opt/hadoop/tmp/logs

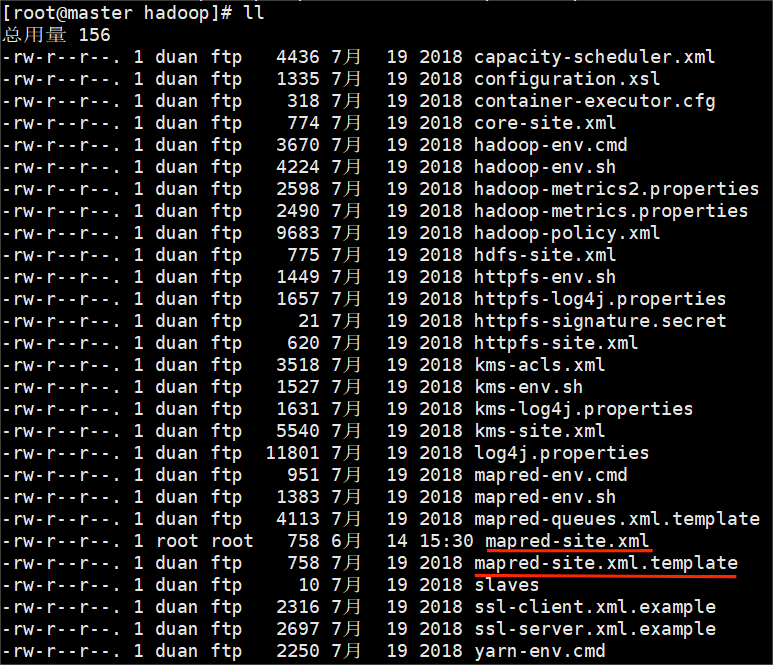


第二步: 将mapred-site.xml.template复制为mapred-site.xml

cd /opt/hadoop/etc/hadoop

cp mapred-site.xml.template mapred-site.xml

ll查看一下:



### 2. 高可用配置文件

至此, 我们所有的准备工作都已经完成, 现在开始进行Hadoop高可用的文件配置

还是在/opt/hadoop/etc/hadoop文件夹下, 我们依次编辑这些文件:

#### ① hadoop-env.sh

在其中将export JAVA\_HOME=${JAVA\_HOME}改成:

vim hadoop-env.sh

如图:

export JAVA\_HOME=/opt/jdk



:wq保存退出

#### ② core-site.xml

vim core-site.xml

在<configuration></configuration>中写入:

<property>

<name>fs.defaultFS</name>

<value>hdfs://mycluster</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/opt/hadoop/tmp</value>

</property>

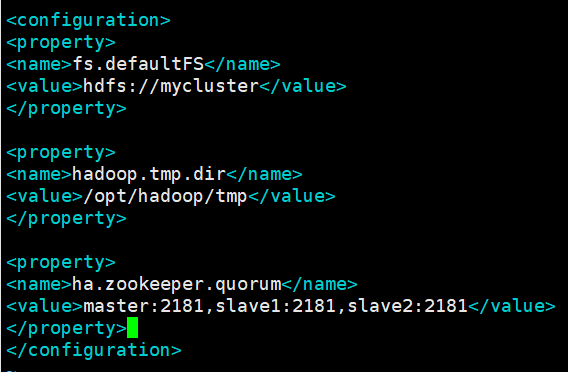
<property>

<name>ha.zookeeper.quorum</name>

<value>master:2181,slave1:2181,slave2:2181</value>

</property>

就像这样:



:wq保存退出

#### ③ mapred-site.xml

在<configuration></configuration>中写入:

vim mapred-site.xml

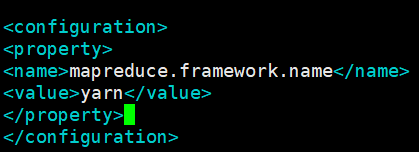
就像这样:

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>



#### ④ hdfs-site.xml

vim hdfs-site.xml

在<configuration></configuration>中写入:

<property>

<name>dfs.nameservices</name>

<value>mycluster</value>

</property>

<property>

<name>dfs.ha.namenodes.mycluster</name>

<value>master,slave1</value>

</property>

<property>

<name>dfs.namenode.rpc-address.mycluster.master</name>

<value>master:8020</value>

</property>

<property>

<name>dfs.namenode.rpc-address.mycluster.slave1</name>

<value>slave1:8020</value>

</property>

<property>

<name>dfs.namenode.http-address.mycluster.master</name>

<value>master:50070</value>

</property>

<property>

<name>dfs.namenode.http-address.mycluster.slave1</name>

<value>slave1:50070</value>

</property>

<property>

<name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://master:8485;slave1:8485;slave2:8485/mycluster</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.mycluster</name> <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

</property>

<property>

<name>dfs.ha.fencing.methods</name>

<value>

sshfence

shell(/bin/true)

</value>

</property>

如图(部分):

<property>

<name>dfs.namenode.name.dir</name>

<value>/opt/hadoop/tmp/hdfs/nn</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/opt/hadoop/tmp/hdfs/dn</value>

</property>

<property>

<name>dfs.journalnode.edits.dir</name>

<value>/opt/hadoop/tmp/hdfs/jn</value>

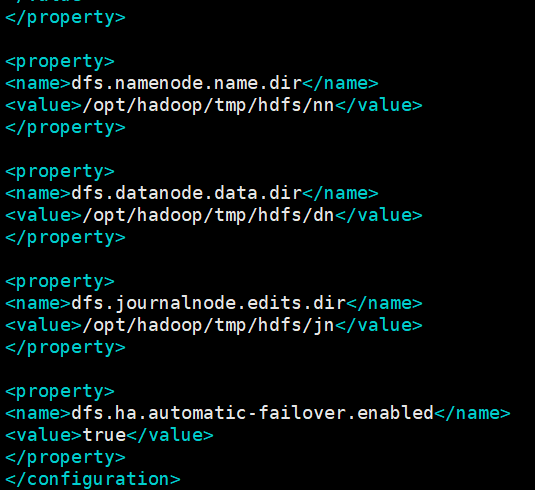
</property>

<property>

<name>dfs.ha.automatic-failover.enabled</name>

<value>true</value>

</property>

:wq保存退出

#### ⑤ yarn-site.xml

vim yarn-site.xml

在<configuration></configuration>中写入:

<property>

<name>yarn.resourcemanager.ha.enabled</name>

<value>true</value>

</property>

<property>

<name>yarn.resourcemanager.cluster-id</name>

<value>yrc</value>

</property>

<property>

<name>yarn.resourcemanager.ha.rm-ids</name>

<value>master,slave1</value>

</property>

<property>

<name>yarn.resourcemanager.hostname.master</name>

<value>master</value>

</property>

<property>

<name>yarn.resourcemanager.hostname.slave1</name>

<value>slave1</value>

</property>

<property>

<name>yarn.resourcemanager.zk-address</name>

<value>master:2181,slave1:2181,slave2:2181</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.resourcemanager.recovery.enabled</name>

<value>true</value>

</property>

<property>

<name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>

</property>

如图(部分):



:wq保存退出

#### ⑥ slaves

vim slaves

将其中文字清空, 换成如下内容:

master

slave1

slave2

如图:



:wq保存退出

至此在master上Hadoop高可用配置完毕

### 3. 集群启动

将在master上配置好的文件分发给slave1和slave2:

#### **①. 启动zookeeper**

scp -r /opt/hadoop slave1:/opt

scp -r /opt/hadoop slave2:/opt

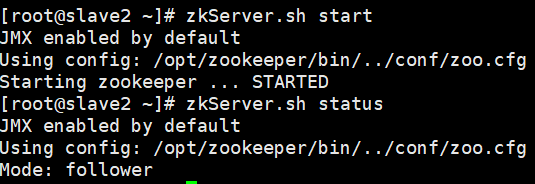
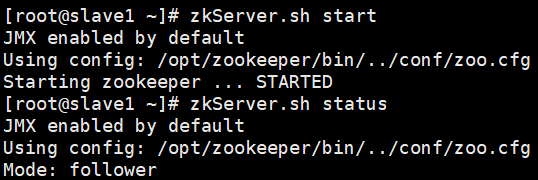
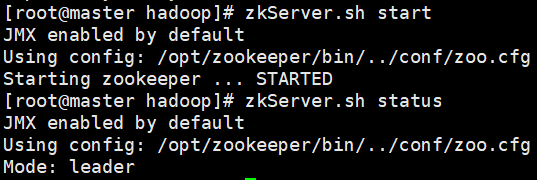
在**撰写栏**中键入:

出现两个follower和一个leader则为正常启动

zkServer.sh start

zkServer.sh status

如图:

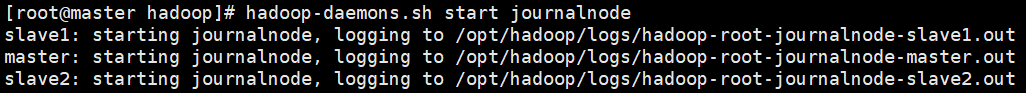


#### ② master节点Hadoop初始化

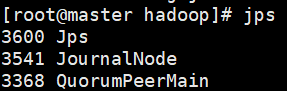
在master节点上启动journalnode服务，namenode互相通信进行数据同步

效果如图所示:

hadoop-daemons.sh start journalnode



为了确保正常开启, 我们使用jps 查看进程, 每个节点进程应该都是



请确保三台节点进程与上图相同再进行下一步.

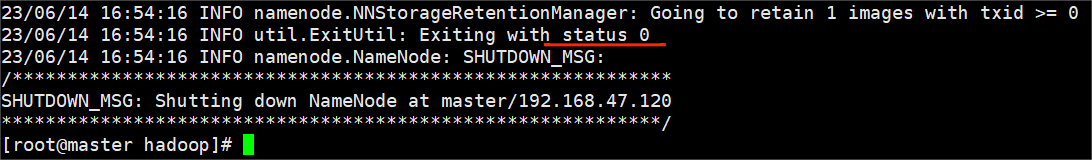
在master上分别输入以下内容:

注意:这一步只能在master节点上做

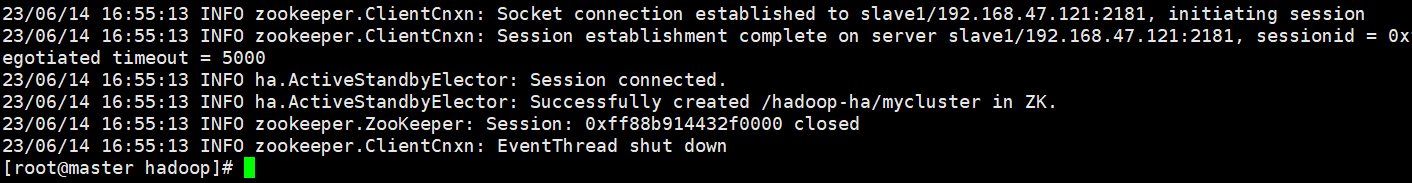
hadoop namenode -format

hdfs zkfc -formatZK

第一个代码成功图例如下:



第二个代码成功图例如下:



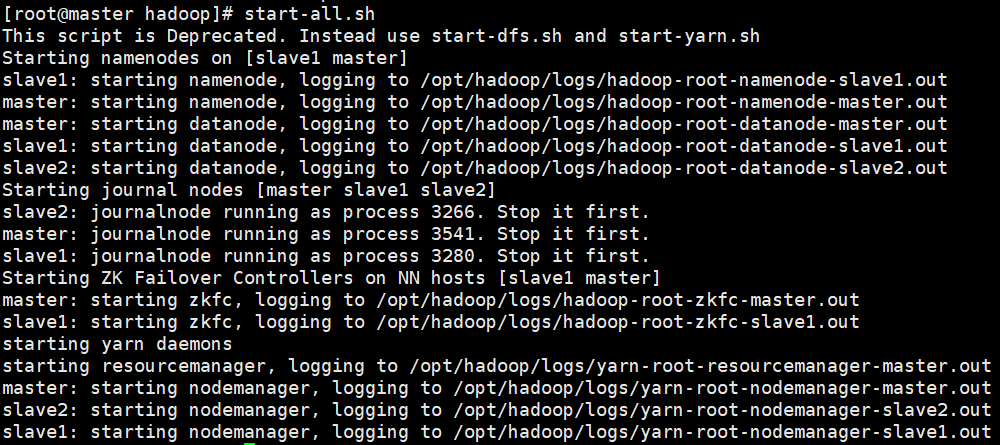
#### ③ 运行Hadoop

在master节点上输入:

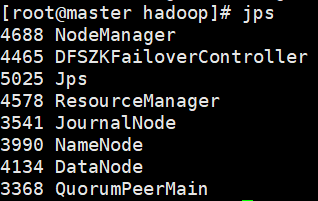
start-all.sh

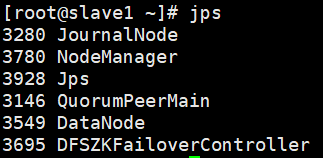
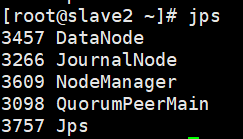
关注运行结果(可以依照它寻找报错)

正常运行结果如图:



下面我们分别用jps查看三个节点的进程

正常运行如图所示:



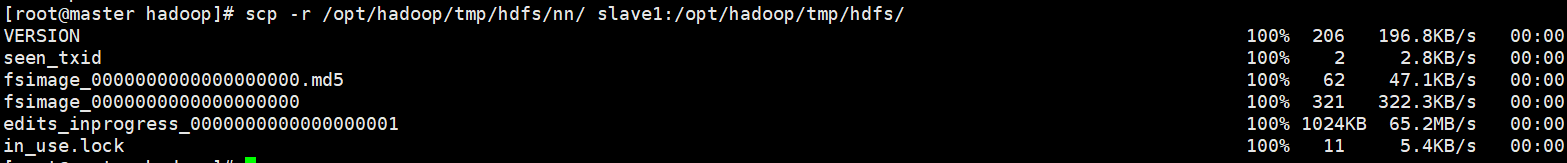
因为高可用的特性之一是namenode主备切换, 所以我们要将namenode数据给我们的备用节点slave1也发一份.

在master上输入:

scp -r /opt/hadoop/tmp/hdfs/nn slave1:/opt/hadoop/tmp/hdfs/

或者在slave1使用hdfs namenode -bootstrapStandby（只需要执行一条命令）

如图:

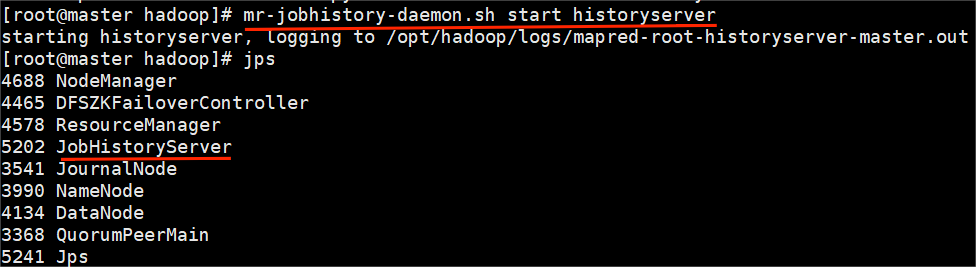


下面我们开启剩余的节点:

在master上开启历史服务:

mr-jobhistory-daemon.sh start historyserver

运行结果和进程如图所示:



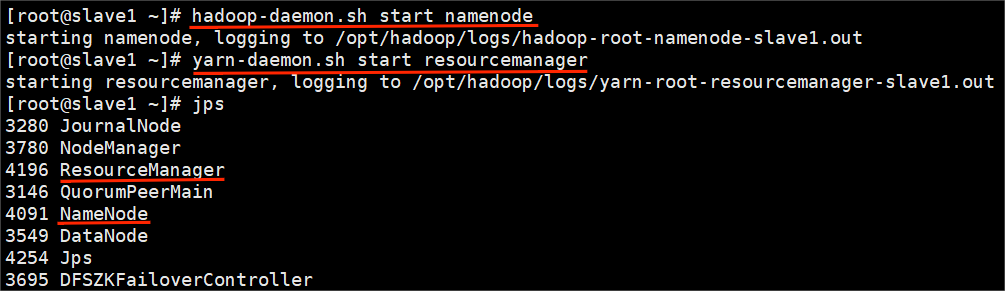
在slave1上开启namenode和resourcemanager:

分别输入:

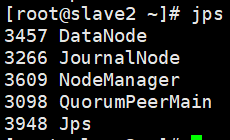
hadoop-daemon.sh start namenode

yarn-daemon.sh start resourcemanager

运行结果和进程如图所示:



顺便看看slave2的进程:



三台进程数分别是9, 8, 5 高可用配置完成.

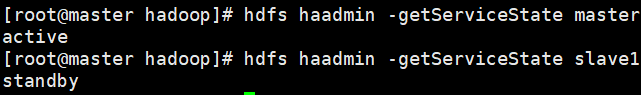
### 4. 测试高可用特性

代码介绍(括号内可以替换):

#### ① 查看namenode状态:

hdfs haadmin -getServiceState master

效果图:



#### ② 设置active 和standby 节点状态, 第一个是备用节点, 第二个是主节点:

hdfs haadmin -failover master slave1

效果图略

### 后续启动

1. 启动zookeeper服务
2. 可以在master上start-all.sh
3. 启动历史服务mr-jobhistory-daemon.sh start historyserver（查看mapreduce历史任务）

## 第七章. 附录

hadoop高可用用来解决什么问题：

——单点故障，namenode或者resourcemanager节点出现故障，非高可用会导致集群无法运行。

journalnode进程有什么作用

——namenode进行数据同步

zkfc进程有什么作用

——监视namenode的存活状态

查看active节点状态：

——get -s /hadoop-ha/mycluster/ActiveStandbyElectorLock

单独启动namenode：

——hdfs --daemon start namenode

——或者hadoop-daemon.sh start namenode

调用API指定hdfs地址：

——hdfs://mycluster/