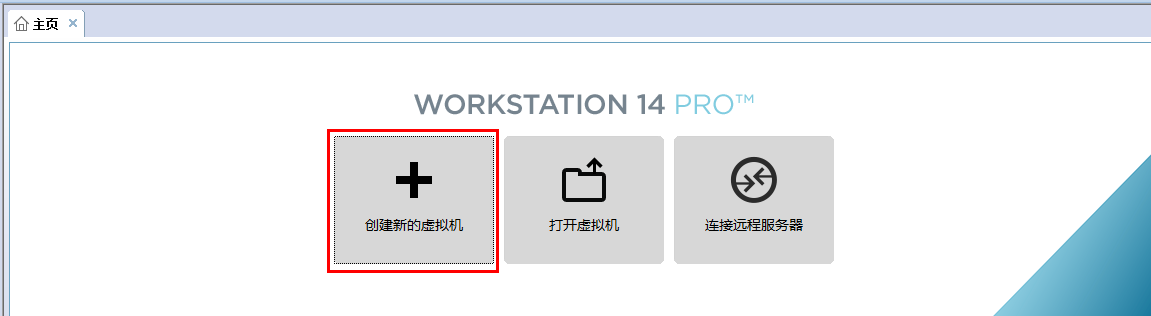
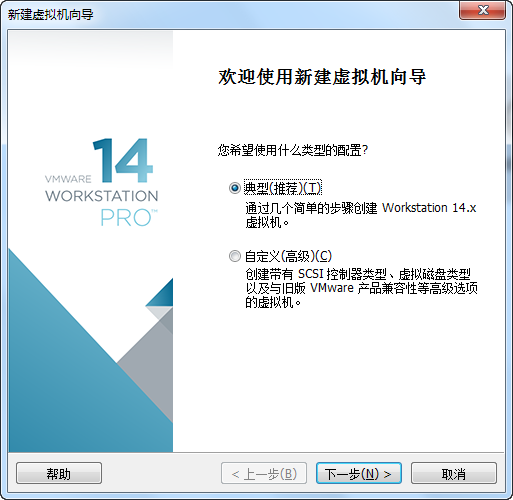
1. 安装VMware
2. 安装CentOS-7
3. 创建新的虚拟机

创建新的虚拟机：

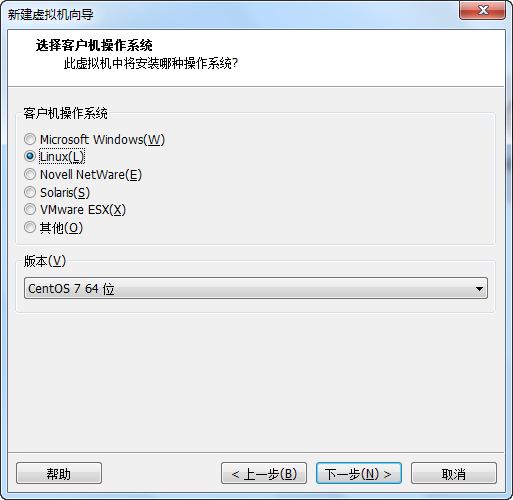
选择典型：



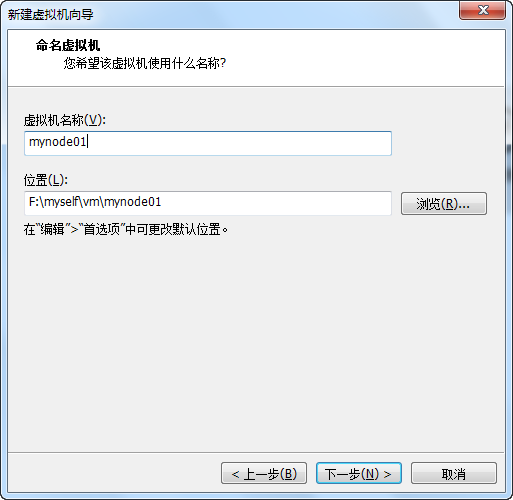
选择稍后安装操作系统【或者傻瓜式安装选择安装程序光盘映像文件(iso)，选择镜像，直接安装成功】：



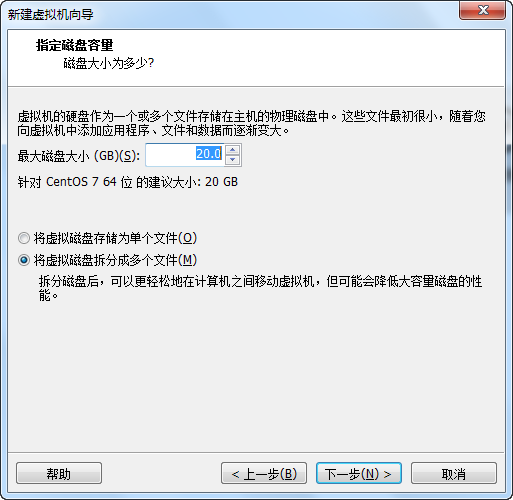
选择Linux，版本选择CentOS 7 64位：



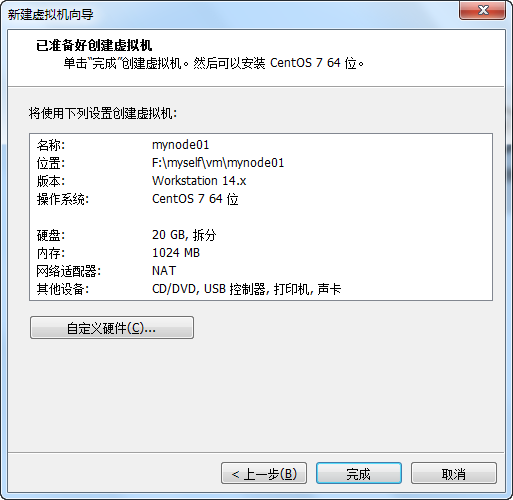
输入虚拟机名称和位置：



磁盘容量可以按照默认不选，直接下一步：

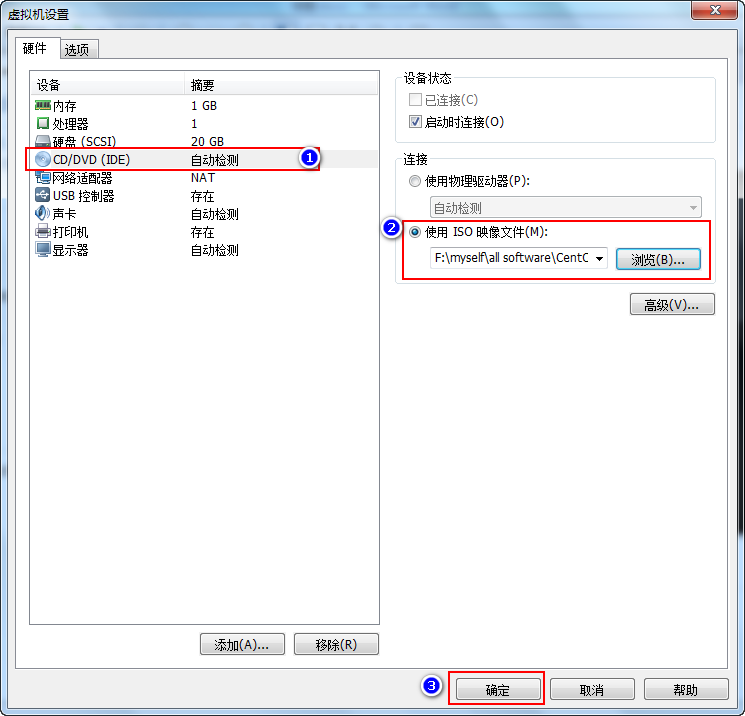


点击完成：



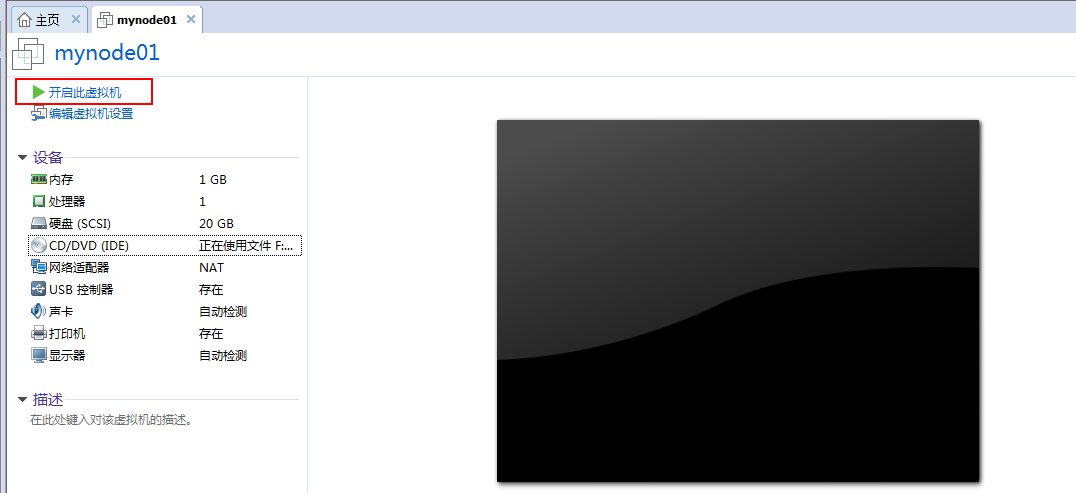
配置虚拟机镜像路径，点击编辑虚拟机设置：  


选择CD/DVD(IDE)，右侧连接中选择使用ISO映像文件(M)，选择CentOS7的镜像位置，点击确定。

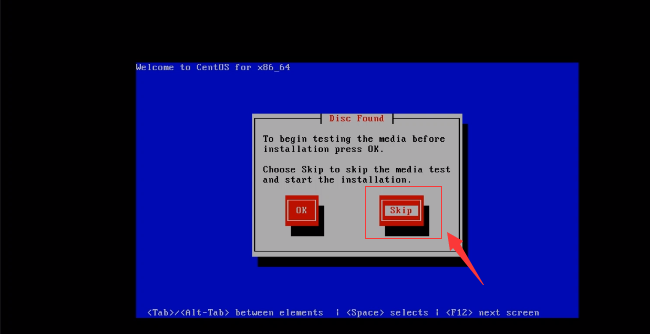


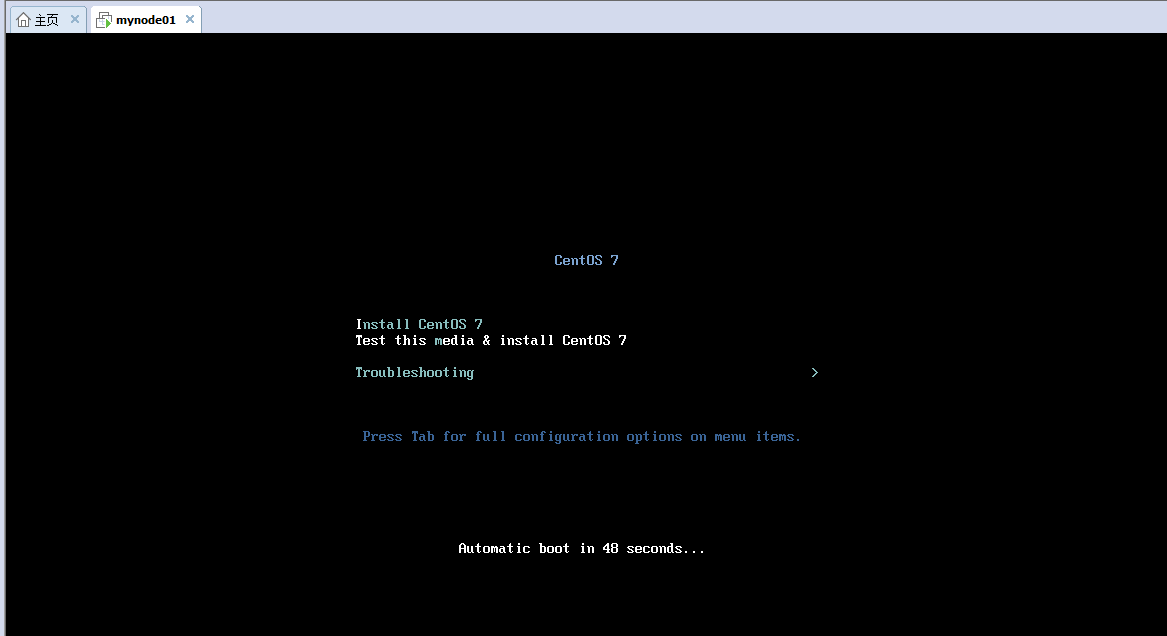
1. 安装配置虚拟机

点击开启此虚拟机：

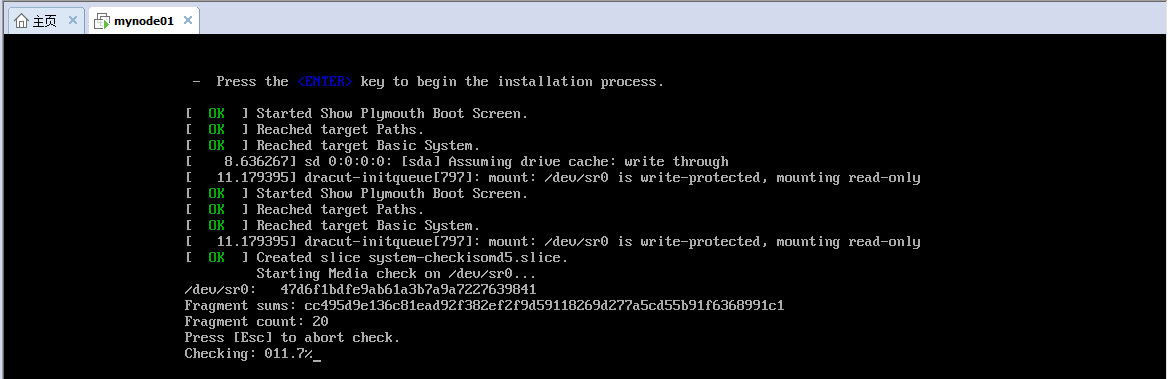


等待1分钟，也可以直接按enter键继续：





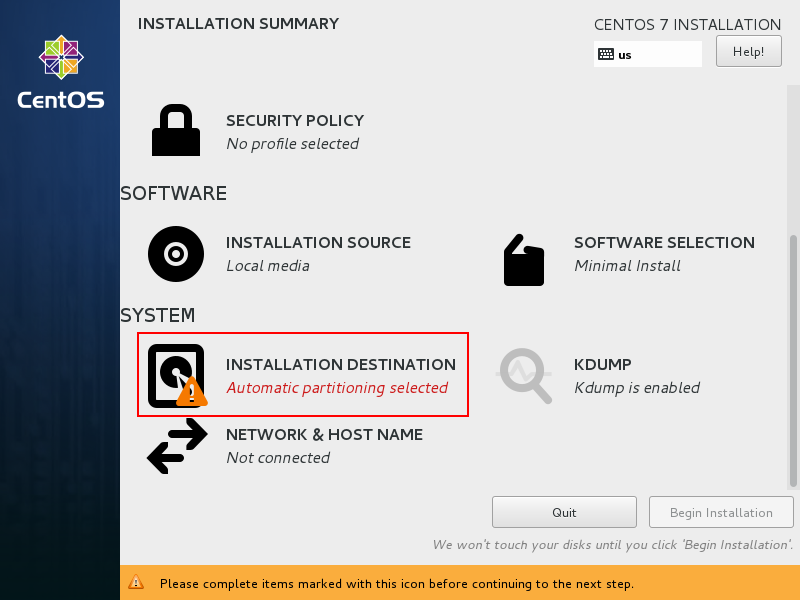
等待检查镜像文件完整，达到100%后自动安装：



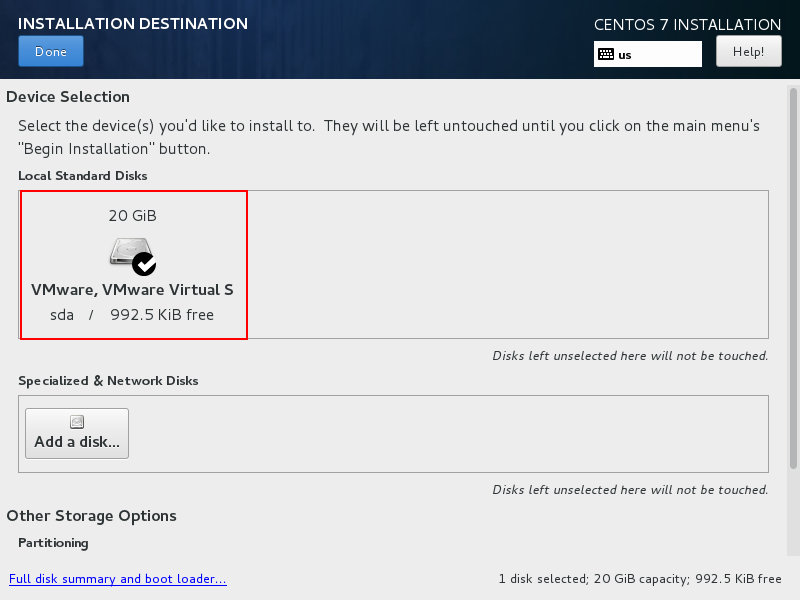
点击Continue:



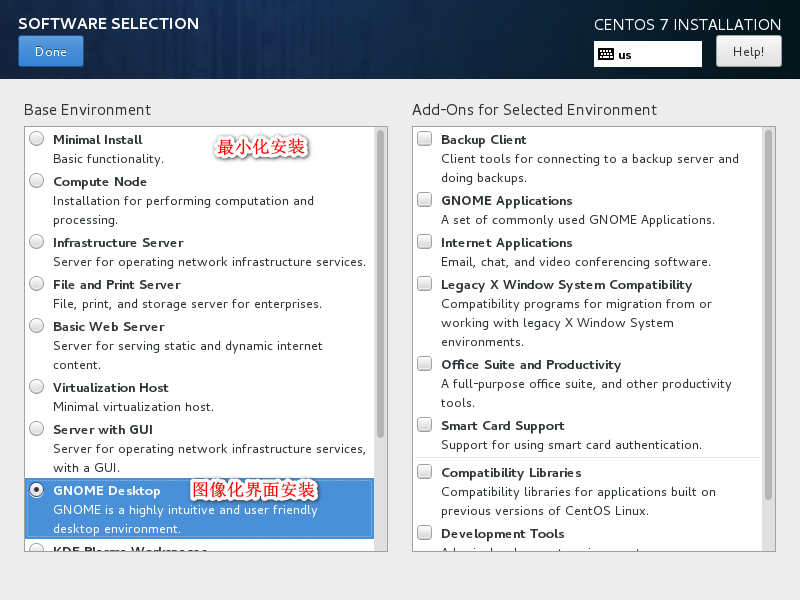
黄色感叹号的选项必须配置，如下：SYSTEM中INSTALLATION DESTINATION 配置磁盘分区规划。



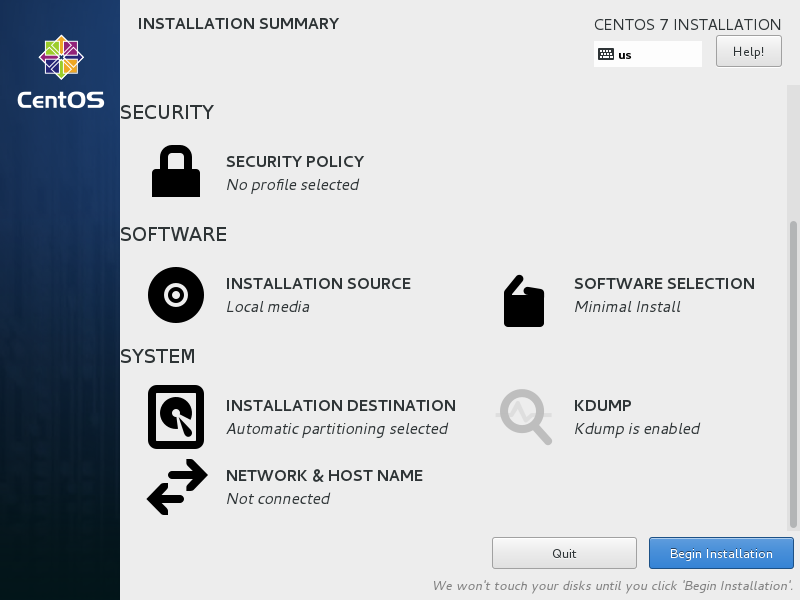
选择默认磁盘分区即可，点击Done。



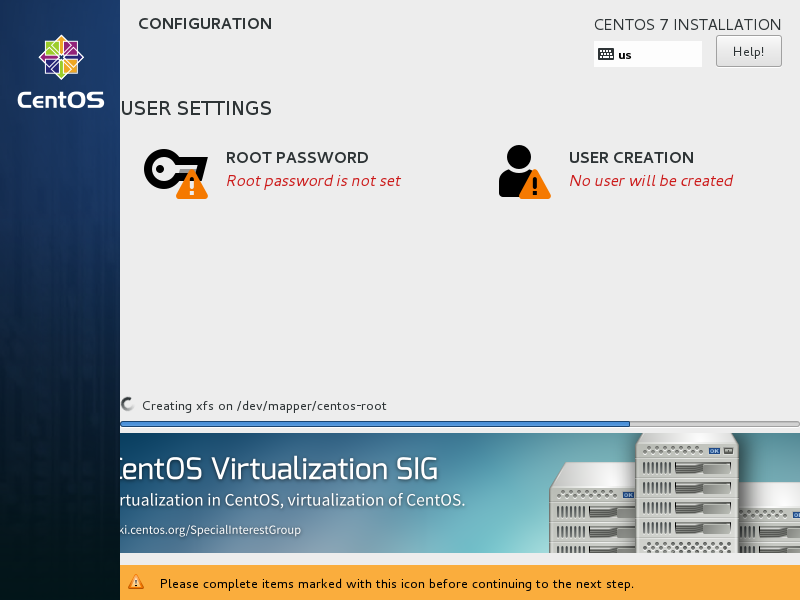
选择 SOFTWARE下的SOFTWARE SELECTION ，选择图形化界面安装，点击Done：



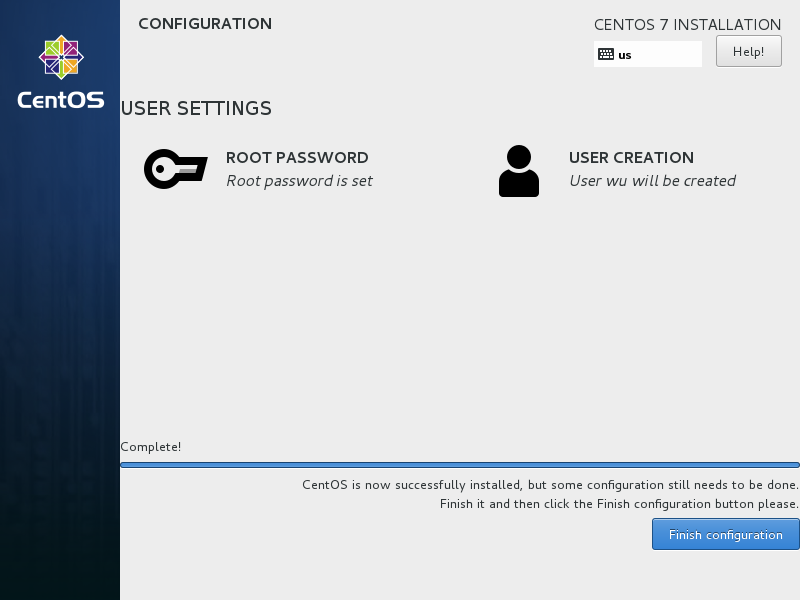
点击Begin Installation安装：



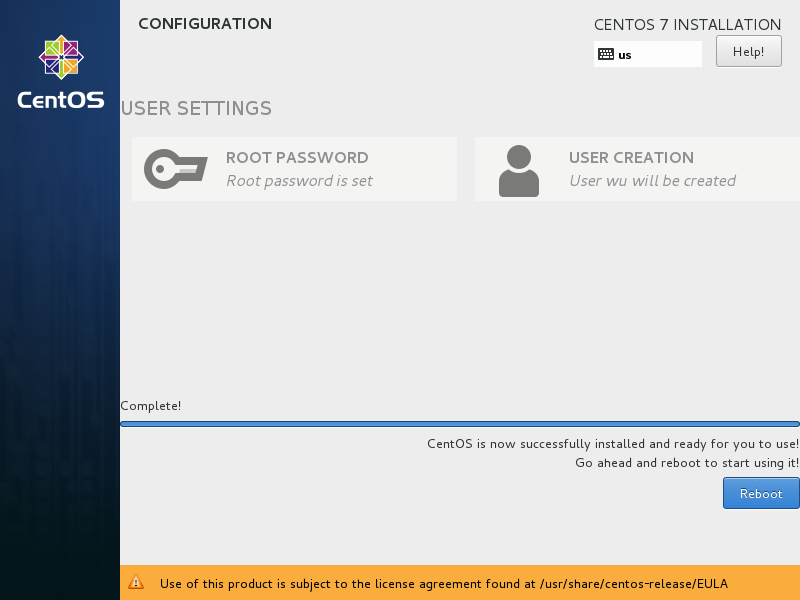
配置root用户密码和添加新的用户：



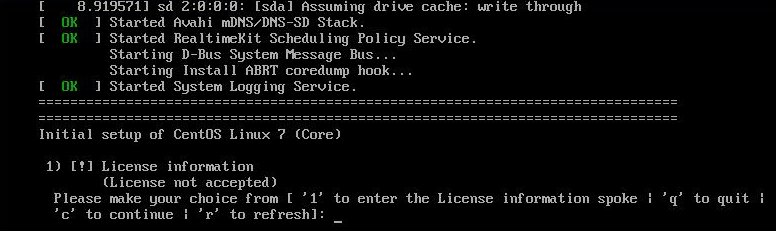
配置完成后，点击Finish configuration。



点击Reboot重启机器。



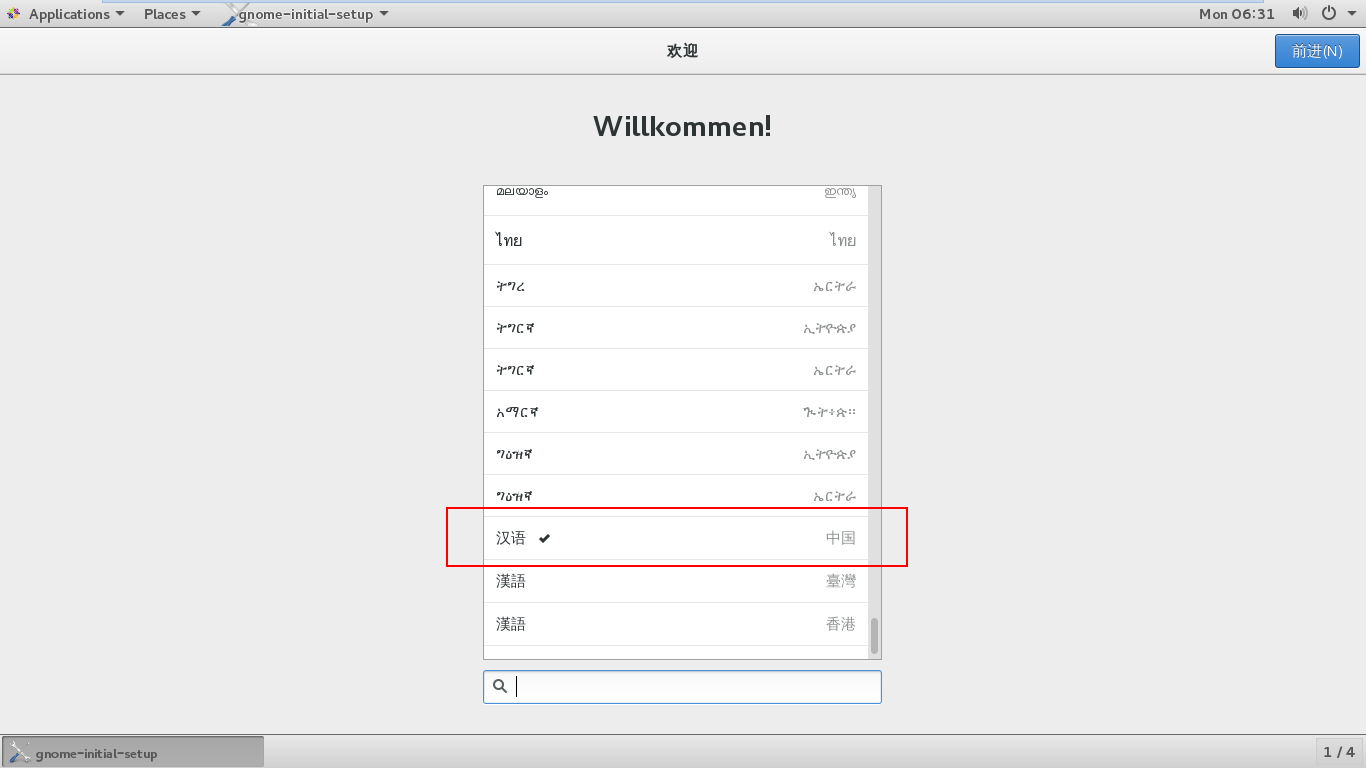
重启过程中会遇到没有接受许可证的状况：



按1->2->c->c选择接受许可证，继续启动即可。

1. 配置启动向导

启动之后，可以设置向导为汉语，点击前进：



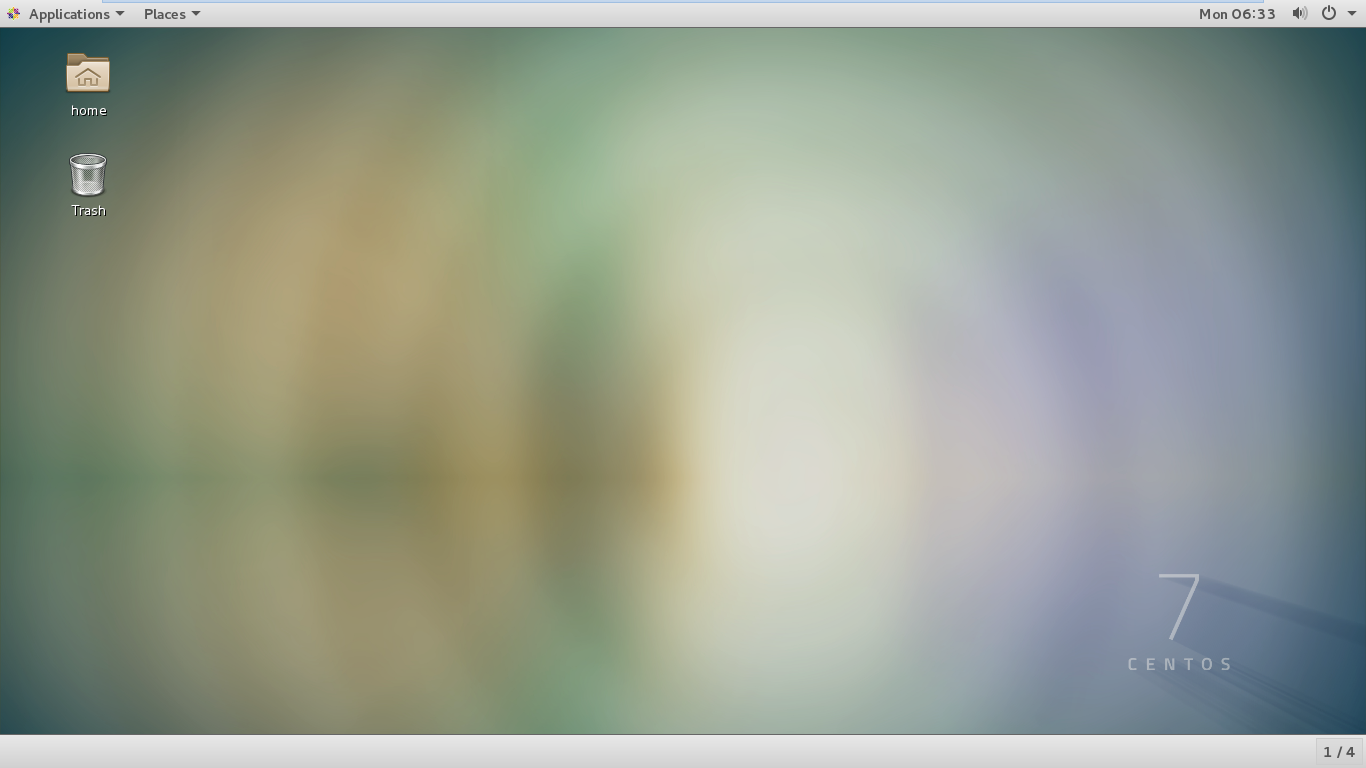
键盘输入就选择默认即可。



点击开始使用。



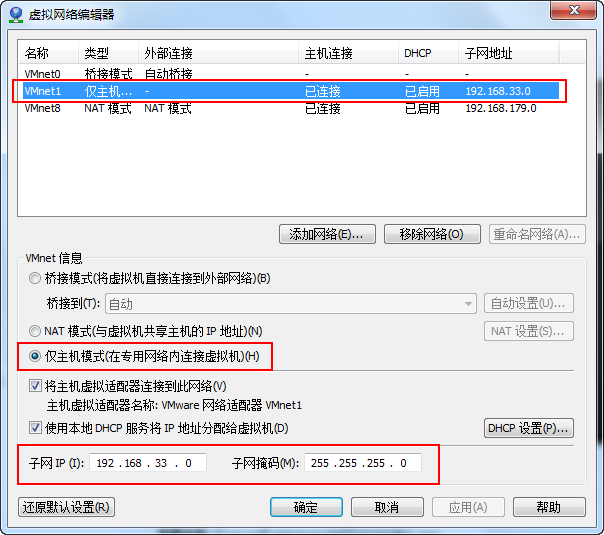
安装完成。



1. 配置网络环境
2. 配置VMware网络环境

在VMware中，打开编辑->虚拟网络编辑器：

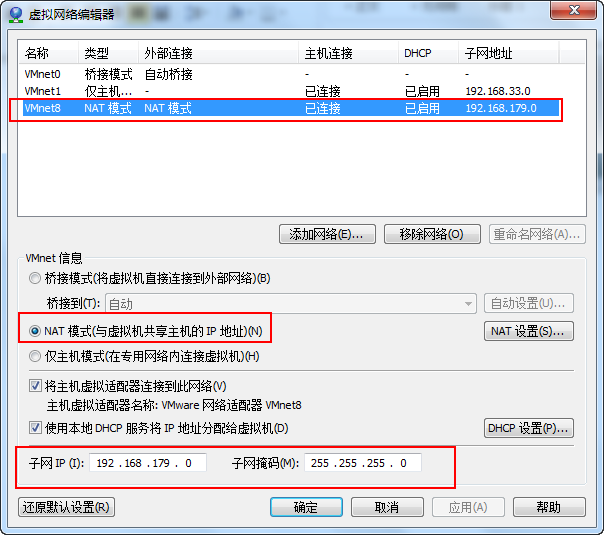
配置VMnet1 仅主机模式:



点击DHCP设置，采用默认即可：



配置VMnet8 NAT模式：

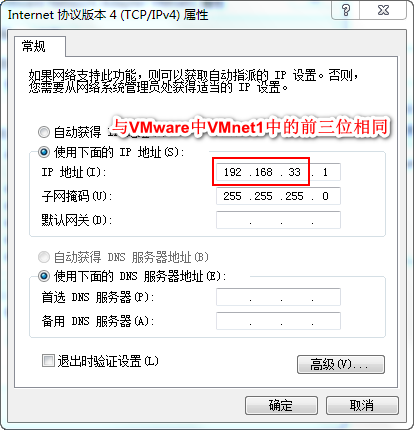


点击DHCP设置，采用默认即可：

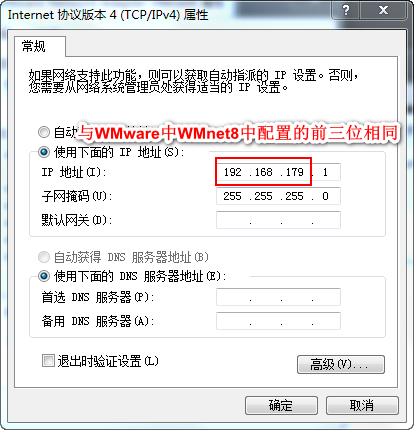


1. 配置本地网卡环境

在网络连接中打开VMnet1,右键->属性->IPv4:



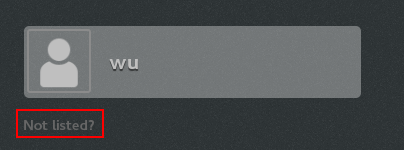
在网络连接中打开VMnet8,右键->属性->IPv4:

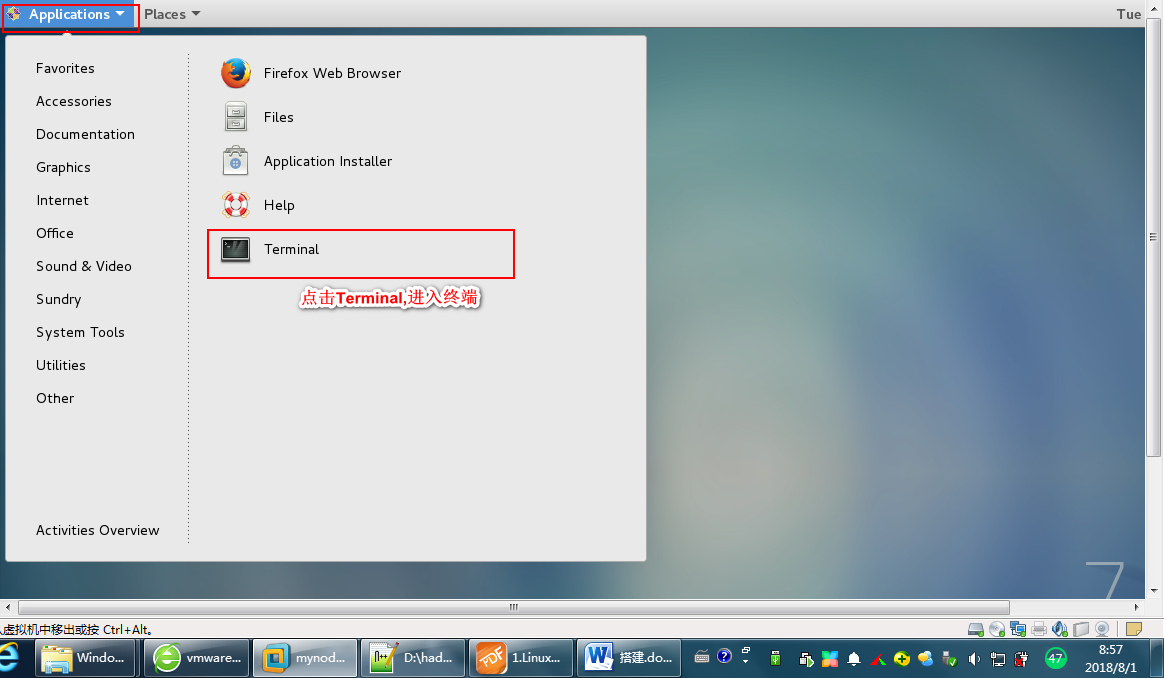


1. 启动虚拟机配置虚拟机IP

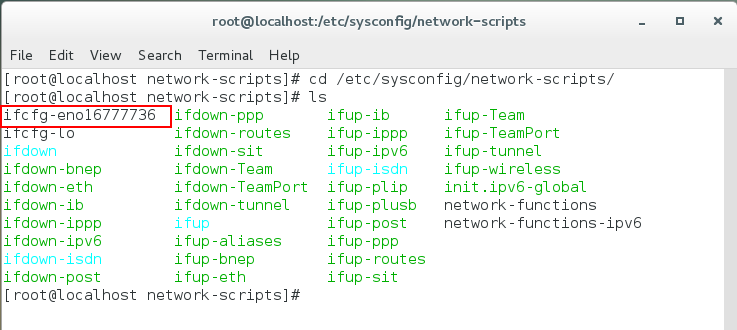
使用root用户登录系统：

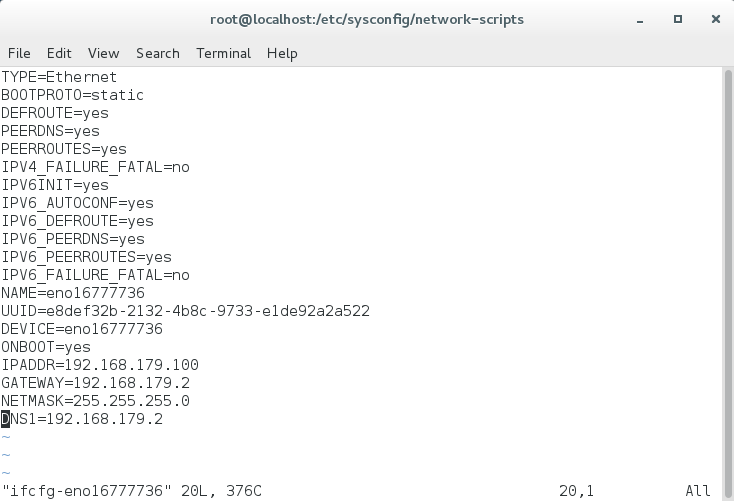
点击Not listed? 输入用户名root和密码登录系统。





进入/etc/sysconfig/network-scripts中，修改文件ifcfg-eno16777736，如下配置好之后，wq保存：





|  |
| --- |
| TYPE=Ethernet  BOOTPROTO=static #使用static配置  DEFROUTE=yes  PEERDNS=yes  PEERROUTES=yes  IPV4\_FAILURE\_FATAL=no  IPV6INIT=yes  IPV6\_AUTOCONF=yes  IPV6\_DEFROUTE=yes  IPV6\_PEERDNS=yes  IPV6\_PEERROUTES=yes  IPV6\_FAILURE\_FATAL=no  NAME=eno16777736  UUID=e8def32b-2132-4b8c-9733-e1de92a2a522  DEVICE=eno16777736  ONBOOT=yes #开机启用本配置  IPADDR=192.168.179.100 #静态IP  GATEWAY=192.168.179.2 #默认网关  NETMASK=255.255.255.0 #子网掩码  DNS1=192.168.179.2 #DNS配置 可以与默认网关相同 |

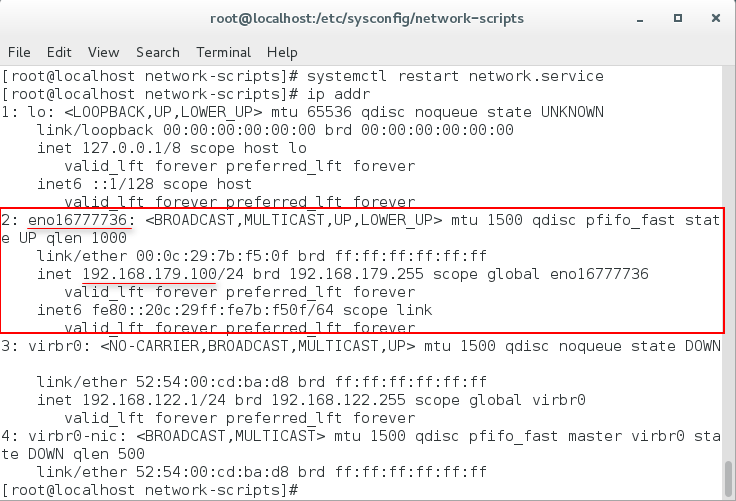
重启网络服务：



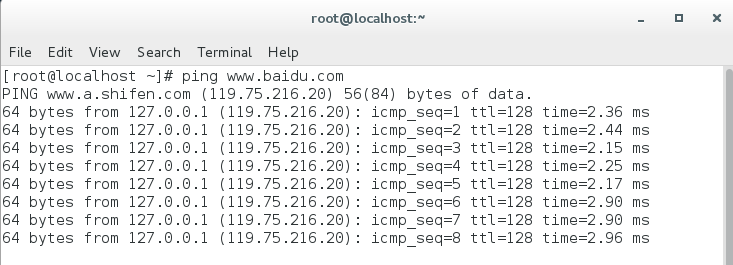
|  |
| --- |
| systemctl restart network.service |

检查ip是否修改，ip addr 查看静态ip,也可以使用ifconfig查看：

|  |
| --- |
| Ip addr |

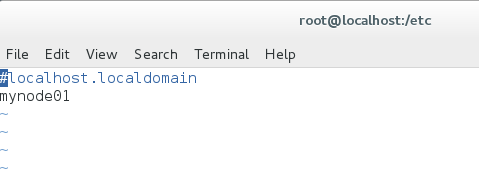


测试ping外网：



1. Centos7节点配置
2. 配置主机名

修改/etc/hostname文件中的主机名称，vim /etc/hostname：



1. 关闭防火墙

检查防火墙的状态 firewall-cmd --state：

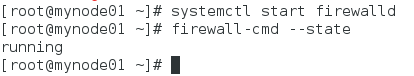


临时关闭防火墙，开机又会启动。

systemctl stop firewalld 或者systemctl stop firewalld.service：



开启防火墙 systemctl start firewalld 或者 systemctl start firewalld.service：



设置开机不启动防火墙 systemctl disable firewalld:



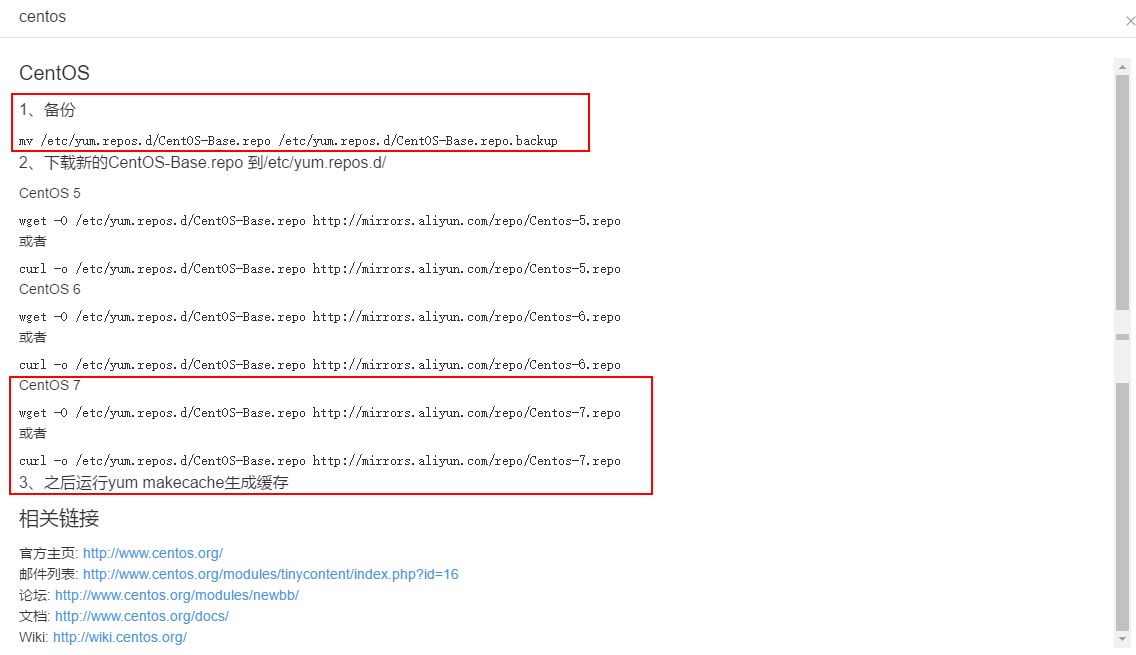
1. 配置阿里云yum源

阿里云开源镜像网站：<https://mirrors.aliyun.com/>

现在改成这个网站：<https://opsx.alibaba.com/mirror?lang=zh-cn>

登录网站，找到对应的linux系统，点击帮助找到配置yum源的步骤：





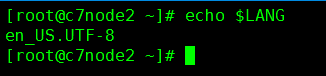
下面给出详细的步骤：

|  |
| --- |
| #安装wget，wget是linux最常用的下载命令(有些系统默认安装，可忽略)  yum -y install wget  #备份当前的yum源  mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup  #下载阿里云的yum源配置  wget -O /etc/yum.repos.d/CentOS-Base.repo <http://mirrors.aliyun.com/repo/Centos-7.repo>  #清除原来文件缓存，构建新加入的repo结尾文件的缓存  yum clean all  yum makecache |

1. 设置Linux系统显示中文/英文

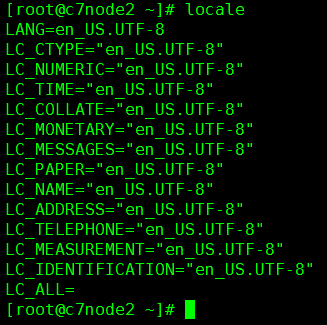
查看当前系统语言：

|  |
| --- |
| echo $LANG |



查看安装的语言包：

|  |
| --- |
| locale |



如有zh cn 表示已经安装了中文语言，如果没有，可以通过命令安装中文语言包：

|  |
| --- |
| yum groupinstall chinese-support |

如果安装不上，可以直接忽略跳过。

修改系统语言为中文【如果想修改成英文，设置LANG="en\_US.UTF-8"】：

如果想临时修改Linux系统的语言环境，重启之后消失，可以直接设置：

|  |
| --- |
| LANG="zh\_CN.UTF-8" |

如果想永久修改系统默认语言，需要创建/修改文件：

|  |
| --- |
| # vim /etc/locale.conf 修改完成之后，记得重启机器  LANG="zh\_CN.UTF-8" |

1. 设置自动更新时间

修改本地时区及ntp服务：

|  |
| --- |
| yum -y install ntp  rm -rf /etc/localtime  ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime  /usr/sbin/ntpdate -u pool.ntp.org |

自动同步时间：

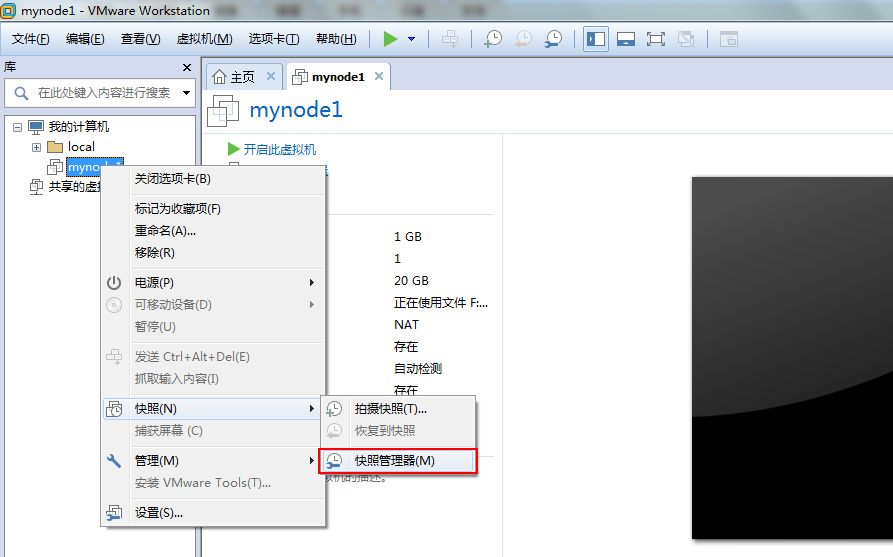
|  |
| --- |
| #设置定时任务，每10分钟同步一次，配置/etc/crontab文件，实现自动执行任务  #建议直接crontab -e 来写入定时任务即可。  \*/10 \* \* \* \* /usr/sbin/ntpdate -u pool.ntp.org >/dev/null 2>&1  #重启定时任务  service crond restart  #查看日期  date |

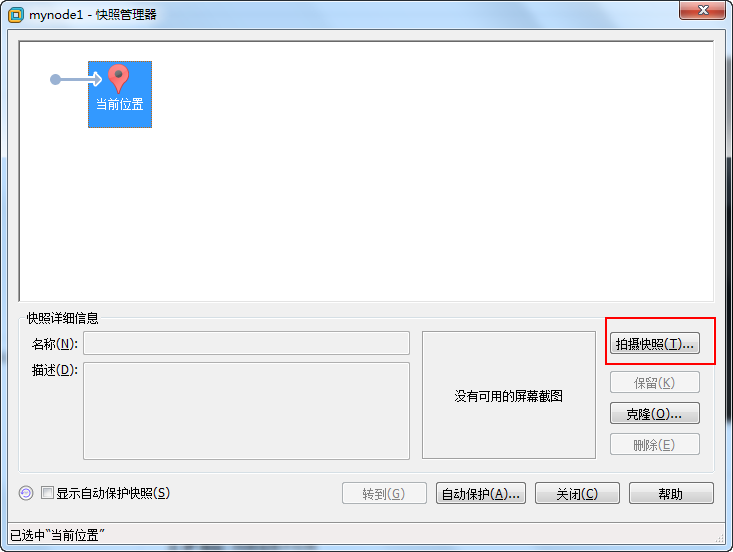
1. 清除节点MAC地址

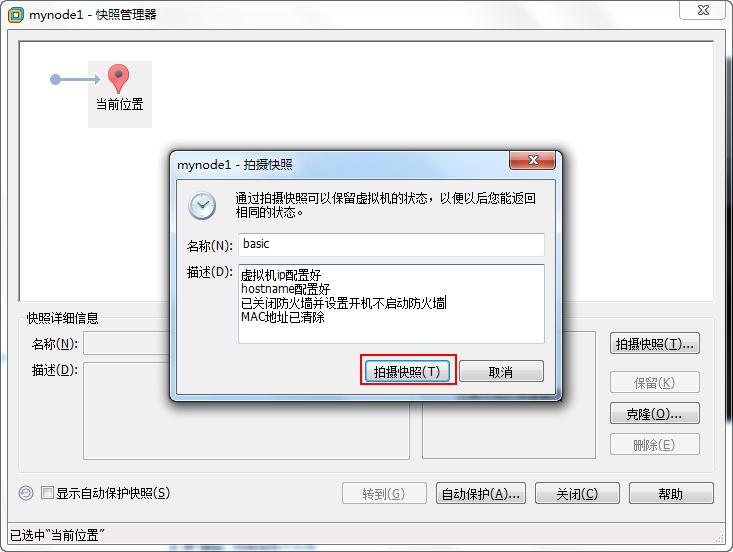
清除/etc/udev/rules.d/ 70-persistent-ipoib.rules文件，这个文件记录了这台机器的MAC地址，虚拟机在第一次启动时候会在这个文件中自动生成MAC地址，下面我们要克隆虚拟机，需要将这个文件删除，如果不删除，克隆出来的虚拟机也是这个MAC地址，那么就会有冲突，导致新克隆的机器ip不可使用。



1. 保存快照

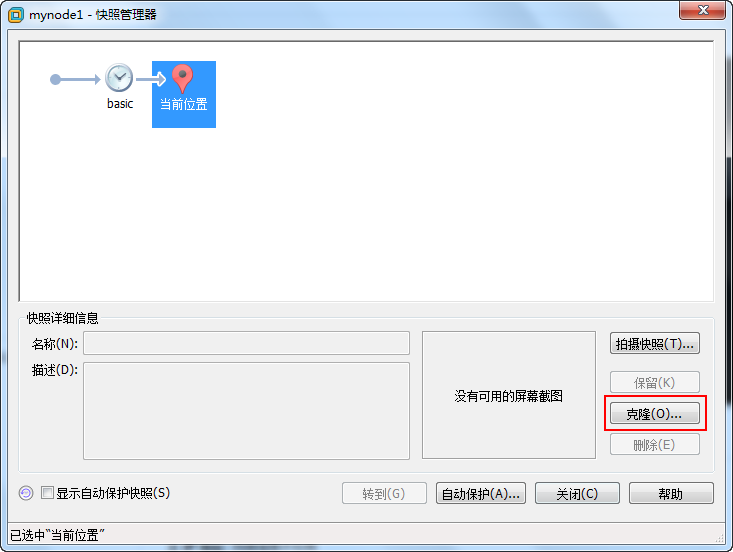


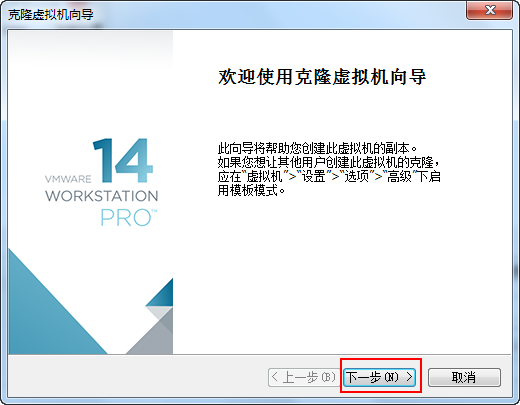
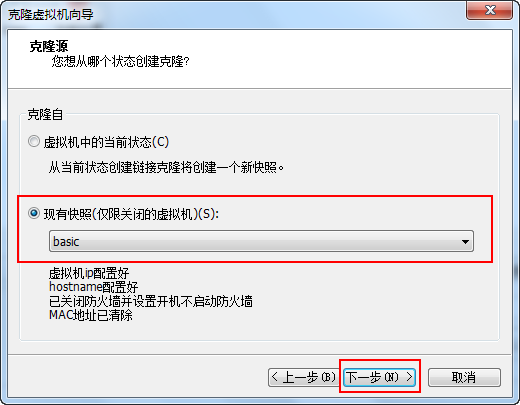


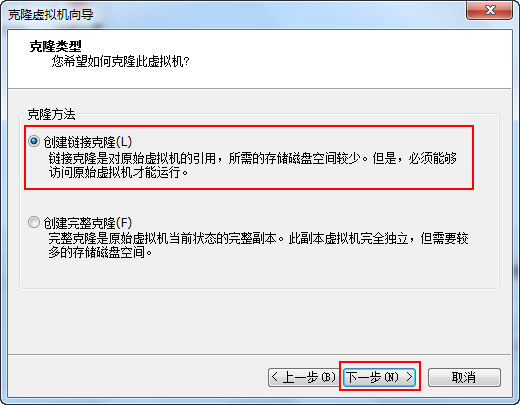
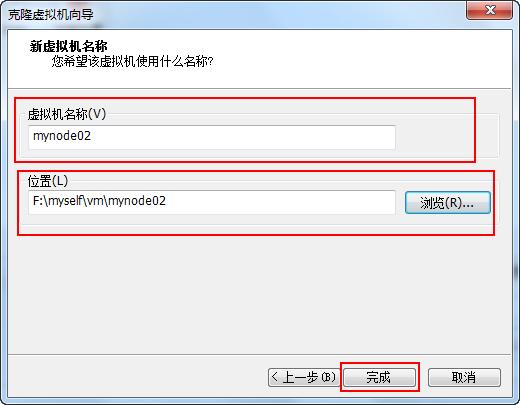


1. 克隆虚拟机节点

点击克隆：

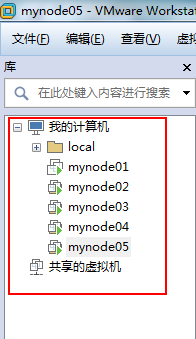




按照以上步骤，依赖这个基础的快照克隆出来其他4台虚拟机，每台机器克隆好之后，都保存一个basic快照即可。如下图：

注意，最好每个克隆的虚拟机都拍个basic快照。

1. 给每台克隆的节点配置ip

在每台节点的/etc/sysconfig/network-scripts/ ifcfg-eno16777736中修改ip:

192.168.179.100

192.168.179.101

192.168.179.102

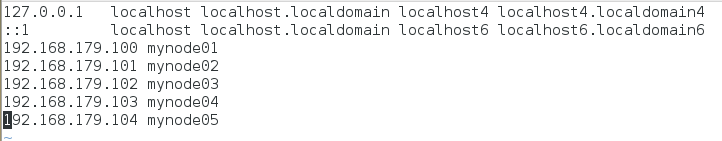
192.168.179.103

192.168.179.104

1. 设置每台节点ip映射

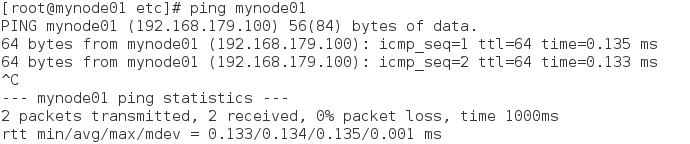
在集群中，每台机器都有自己的一个主机名，便于节点与节点之间的区分，因此可以为每台节点设置主机名：

进入每台节点的/etc/hosts下，修改hosts文件，vim /etc/hosts:



|  |
| --- |
| 127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4  ::1 localhost localhost.localdomain localhost6 localhost6.localdomain6  192.168.179.100 mynode01  192.168.179.101 mynode02  192.168.179.102 mynode03  192.168.179.103 mynode04  192.168.179.104 mynode05 |

测试 ping mynode01:



1. 配置节点之间免密访问

在每个节点上执行如下命令生成密钥对，一直回车就可以：

|  |
| --- |
| ssh-keygen -t rsa -P '' |

在每个节点上执行如下命令将公钥文件写入授权文件中，并赋予权限：

|  |
| --- |
| cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys  chmod 600 ~/.ssh/authorized\_keys |

节点到节点之间的无密码访问【想登录哪台机器，就把当前机器的公钥copy追加写入到登录哪台机器的授权文件中即可】：

以c7node01无密码登录所有的节点为例，如果要实现所有节点之间免密登录，就每个节点都执行如下命令一次即可。：

|  |
| --- |
| #在c7node1节点中执行命令，需要输入密码：  scp ~/.ssh/id\_rsa.pub root@c7node2:~  scp ~/.ssh/id\_rsa.pub root@c7node3:~  scp ~/.ssh/id\_rsa.pub root@c7node4:~  scp ~/.ssh/id\_rsa.pub root@c7node5:~  #在每个从节点中执行  cat ~/id\_rsa.pub >> ~/.ssh/authorized\_keys  chmod 600 ~/.ssh/authorized\_keys  rm -rf ~/id\_rsa.pub |

1. 其他
2. 关闭Centos7的图形化界面

Centos7如果选择图形化界面安装，那么未来启动每个虚拟机时除了使用分配外的内存，图形化界面还会单独使用内存，这部分内存也会很大。这样如果有多台虚拟机的话，图形化界面总体使用的内存是一部分很大的开销。

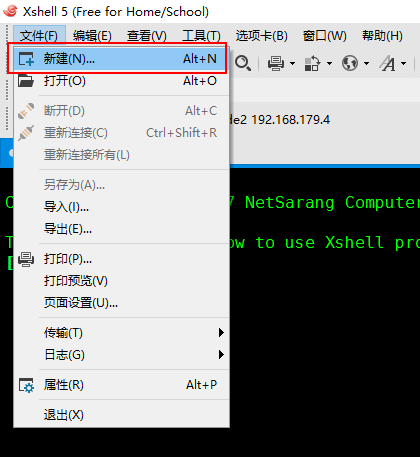
可以安装好虚拟机Centos7后使用命令关闭图形化界面，新版本的CentOS系统里使用’targets’ 取代了运行级别的概念。系统有两种默认的’targets’: 多用户.target 对应之前版本的3 运行级别;而图形.target 对应之前的5运行级别

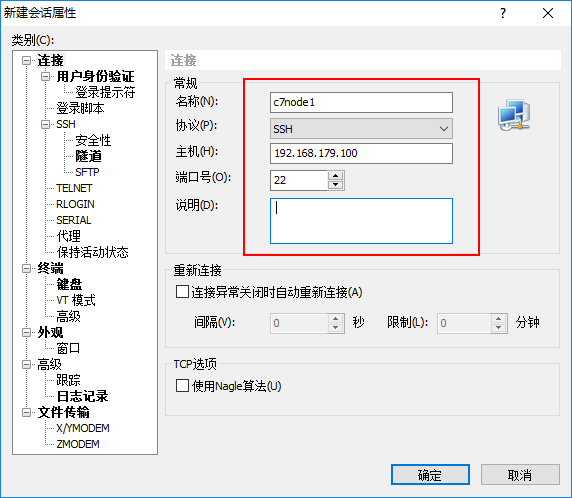
|  |
| --- |
| #查看默认的target,执行如下命令：  systemctl get-default  #开机以命令模式启动，执行：  systemctl set-default multi-user.target  #开机以图形界面启动，执行：  systemctl set-default graphical.target |

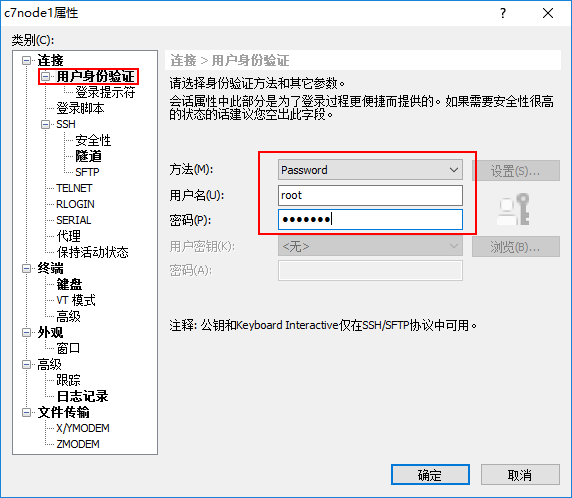
1. 安装XShell操作虚拟机
2. 下载安装Xshell

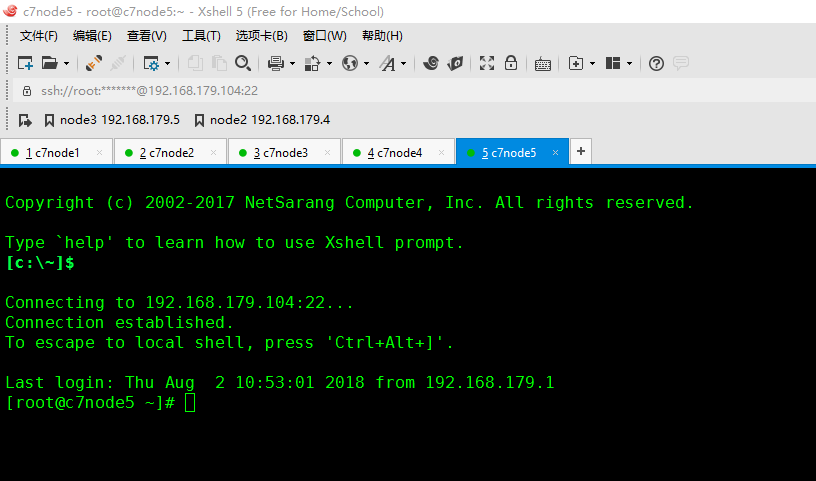
最好下载Xshell5，因为最新版本Xshell6只支持同时在一个Xshell窗口内打开4个连接窗口。

1. 配置连接









1. 安装Hadoop集群

注意：所有软件都会安装到/software/中，需要在每个节点中创建software目录。

1. 节点划分

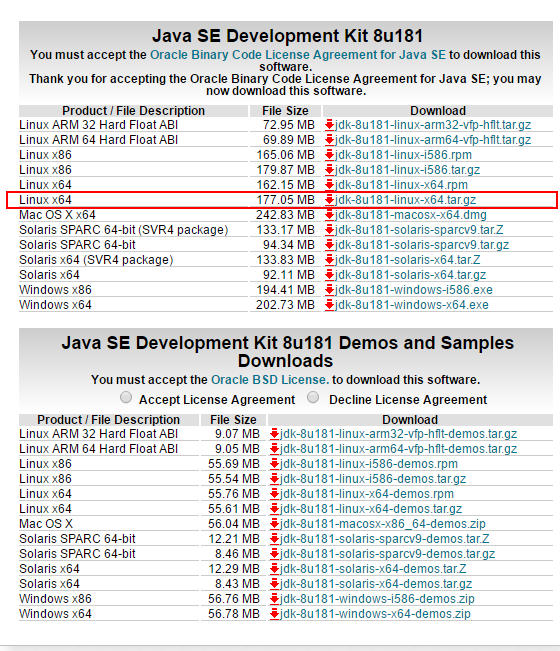
HDFS集群划分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 节点 | NN | DN | ZK | ZKFC | JN | RM | NM |
| c7node1 | ★ |  |  | ★ |  | ★ |  |
| c7node2 | ★ |  |  | ★ |  | ★ |  |
| c7node3 |  | ★ | ★ |  | ★ |  | ★ |
| c7node4 |  | ★ | ★ |  | ★ |  | ★ |
| c7node5 |  | ★ | ★ |  | ★ |  | ★ |

1. 安装jdk8

下载jdk8:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>



安装jdk：

将下载好的jdk上传到c7node1中的/software/下，解压：



|  |
| --- |
| tar -zxvf ./jdk-8u181-linux-x64.tar.gz -C /software/ |

将jdk添加到环境变量：

|  |
| --- |
| vim /etc/profile  #在文件最后加入如下,保存：  #如果安装过jdk,这么写：export PATH=$JAVA\_HOME/bin:$PATH  export JAVA\_HOME=/software/jdk1.8.0\_181  export PATH=$PATH:$JAVA\_HOME/bin  #使修改生效  source /etc/profile |





将解压的jdk和/etc/profile发送到其他节点：

|  |
| --- |
| scp -r ./jdk1.8.0\_181/ c7node2:`pwd`  scp -r ./jdk1.8.0\_181/ c7node3:`pwd`  scp -r ./jdk1.8.0\_181/ c7node4:`pwd`  scp -r ./jdk1.8.0\_181/ c7node5:`pwd`  scp /etc/profile c7node2:/etc/  scp /etc/profile c7node3:/etc/  scp /etc/profile c7node4:/etc/  scp /etc/profile c7node5:/etc/  #最后在每台节点执行以下命令，使配置生效  source /etc/profile |

1. 安装zookeeper

下载zookeeper: <http://archive.apache.org/dist/zookeeper/>

将下载好的zookeeper上传到c7node3节点的/software/下，解压：

|  |
| --- |
| #使用 --no-same-owner 会指定操作用户作为解压后文件的所有者  tar -zxvf zookeeper-3.4.13.tar.gz -C /software/ --no-same-owner  #创建zookeeper软链接  ln -sf zookeeper-3.4.13 zookeeper |

配置zookeeper的环境变量：

|  |
| --- |
| #进入vim /etc/profile，在最后加入：  export ZOOKEEPER\_HOME=/software/zookeeper  export PATH=$PATH:$ZOOKEEPER\_HOME/bin  #使配置生效  source /etc/profile |

配置zookeeper：

|  |
| --- |
| #修改zoo\_sample.cfg zoo.cfg  cp /software/zookeeper/conf/zoo\_sample.cfg /software/zookeeper/conf/zoo.cfg  #清空 zoo.cfg文件内容，写入如下内容：  tickTime=2000  dataDir=/opt/data/zookeeper  clientPort=2181  initLimit=5  syncLimit=2  server.1=c7node3:2888:3888  server.2=c7node4:2888:3888  server.3=c7node5:2888:3888  #将zookeeper的目录发送到其它两个节点，注意：环境变量其它节点也配置上  scp -r ./zookeeper-3.4.13/ c7node4:`pwd`  scp -r ./zookeeper-3.4.13/ c7node5:`pwd`  #在c7node3,c7node4,c7node5节点分别创建/opt/data/zookeeper  mkdir -p /opt/data/zookeeper  #创建节点ID,在每台节点配置的dataDir路径/opt/data/zookeeper中添加myid文件  #在c7node3的/opt/data/zookeeper中创建myid文件写入1  #在c7node4的/opt/data/zookeeper中创建myid文件写入2  #在c7node5的/opt/data/zookeeper中创建myid文件写入3 |

每台节点启动zookeeper:

|  |
| --- |
| #每台节点中启动zookeeper  zkServer.sh start  #检查每台节点zookeeper的状态  zkServer.sh status |

1. 配置Hadoop环境变量

上传解压hadoop，上传到c7node1节点的/software/下，解压：

|  |
| --- |
| tar -zxvf hadoop-3.1.0.tar.gz -C /software/ --no-same-owner  #创建hadoop的软连接  ln -sf hadoop-3.1.0 hadoop |

配置hadoop的环境变量：

|  |
| --- |
| #编辑 vim /etc/profile,添加下面内容：  export HADOOP\_HOME=/software/hadoop  export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin:  #使配置生效  source /etc/profile |

1. 配置高可用的HDFS和Yarn文件

配置hadoop-env.sh文件，加入JAVA\_HOME:

|  |
| --- |
| #在第19行加入JAVA\_HOME  export JAVA\_HOME=/software/jdk1.8.0\_181 |

配置hdfs-site.xml 如下：

|  |
| --- |
| <configuration>  <property>  <!--这里配置逻辑名称，可以随意写 -->  <name>dfs.nameservices</name>  <value>mycluster</value>  </property>  <property>  <!-- 禁用权限 -->  <name>dfs.permissions.enabled</name>  <value>false</value>  </property>  <property>  <!-- 配置namenode 的名称，多个用逗号分割 -->  <name>dfs.ha.namenodes.mycluster</name>  <value>nn1,nn2</value>  </property>  <property>  <!-- dfs.namenode.rpc-address.[nameservice ID].[name node ID] namenode 所在服务器名称和RPC监听端口号 -->  <name>dfs.namenode.rpc-address.mycluster.nn1</name>  <value>c7node1:8020</value>  </property>  <property>  <!-- dfs.namenode.rpc-address.[nameservice ID].[name node ID] namenode 所在服务器名称和RPC监听端口号 -->  <name>dfs.namenode.rpc-address.mycluster.nn2</name>  <value>c7node2:8020</value>  </property>  <property>  <!-- dfs.namenode.http-address.[nameservice ID].[name node ID] namenode 监听的HTTP协议端口 -->  <name>dfs.namenode.http-address.mycluster.nn1</name>  <value>c7node1:50070</value>  </property>  <property>  <!-- dfs.namenode.http-address.[nameservice ID].[name node ID] namenode 监听的HTTP协议端口 -->  <name>dfs.namenode.http-address.mycluster.nn2</name>  <value>c7node2:50070</value>  </property>  <property>  <!-- namenode 共享的编辑目录， journalnode 所在服务器名称和监听的端口 -->  <name>dfs.namenode.shared.edits.dir</name>  <value>qjournal://c7node3:8485;c7node4:8485;c7node5:8485/mycluster</value>  </property>  <property>  <!-- namenode高可用代理类 -->  <name>dfs.client.failover.proxy.provider.mycluster</name>  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>  </property>  <property>  <!-- 使用ssh 免密码自动登录 -->  <name>dfs.ha.fencing.methods</name>  <value>sshfence</value>  </property>  <property>  <name>dfs.ha.fencing.ssh.private-key-files</name>  <value>/root/.ssh/id\_rsa</value>  </property>  <property>  <!-- journalnode 存储数据的地方 -->  <name>dfs.journalnode.edits.dir</name>  <value>/opt/data/journal/node/local/data</value>  </property>  <property>  <!-- 配置namenode自动切换 -->  <name>dfs.ha.automatic-failover.enabled</name>  <value>true</value>  </property>  </configuration> |

配置core-site.xml

|  |
| --- |
| <configuration>  <property>  <!-- 为Hadoop 客户端配置默认的高可用路径 -->  <name>fs.defaultFS</name>  <value>hdfs://mycluster</value>  </property>  <property>  <!-- Hadoop 数据存放的路径，namenode,datanode 数据存放路径都依赖本路径，不要使用 file:/ 开头，使用绝对路径即可  namenode 默认存放路径 ：file://${hadoop.tmp.dir}/dfs/name  datanode 默认存放路径 ：file://${hadoop.tmp.dir}/dfs/data  -->  <name>hadoop.tmp.dir</name>  <value>/opt/data/hadoop/</value>  </property>  <property>  <!-- 指定zookeeper所在的节点 -->  <name>ha.zookeeper.quorum</name>  <value>c7node3:2181,c7node4:2181,c7node5:2181</value>  </property>  </configuration> |

配置yarn-site.xml:

|  |
| --- |
| <configuration>  <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  <property>  <name>yarn.nodemanager.env-whitelist</name>  <value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_CONF\_DIR,CLASSPATH\_PREPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_MAPRED\_HOME</value>  </property>  <property>  <!-- 配置yarn为高可用 -->  <name>yarn.resourcemanager.ha.enabled</name>  <value>true</value>  </property>  <property>  <!-- 集群的唯一标识 -->  <name>yarn.resourcemanager.cluster-id</name>  <value>mycluster</value>  </property>  <property>  <!-- ResourceManager ID -->  <name>yarn.resourcemanager.ha.rm-ids</name>  <value>rm1,rm2</value>  </property>  <property>  <!-- 指定ResourceManager 所在的节点 -->  <name>yarn.resourcemanager.hostname.rm1</name>  <value>c7node1</value>  </property>  <property>  <!-- 指定ResourceManager 所在的节点 -->  <name>yarn.resourcemanager.hostname.rm2</name>  <value>c7node2</value>  </property>  <property>  <!-- 指定ResourceManager Http监听的节点 -->  <name>yarn.resourcemanager.webapp.address.rm1</name>  <value>c7node1:8088</value>  </property>  <property>  <!-- 指定ResourceManager Http监听的节点 -->  <name>yarn.resourcemanager.webapp.address.rm2</name>  <value>c7node2:8088</value>  </property>  <property>  <!-- 指定zookeeper所在的节点 -->  <name>yarn.resourcemanager.zk-address</name>  <value>c7node3:2181,c7node4:2181,c7node5:2181</value>  </property>  <!-- 启用节点的内容和CPU自动检测，最小内存为1G -->  <!--<property>  <name>yarn.nodemanager.resource.detect-hardware-capabilities</name>  <value>true</value>  </property>-->  </configuration> |

配置mapred-site.xml:

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

配置datanode节点：

|  |
| --- |
| #在vim /software/hadoop/etc/hadoop/workers中配置  c7node3  c7node4  c7node5 |

将配置好的hadoop目录发送到每一个节点中：

|  |
| --- |
| scp -r ./hadoop c7node2:`pwd`  scp -r ./hadoop c7node3:`pwd`  scp -r ./hadoop c7node4:`pwd`  scp -r ./hadoop c7node5:`pwd`  #分别在c7node2,c7node3,c7node4,c7node5中配置hadoop的环境变量 |

1. 启动HDFS和Yarn

先启动zookeeper:

|  |
| --- |
| zkServer.sh start |

在其中一个namenode中格式化zookeeper:

|  |
| --- |
| hdfs zkfc -formatZK |

在每台journalnode中启动所有的journalnode:

|  |
| --- |
| hdfs --daemon start journalnode |

格式化namenode:

|  |
| --- |
| #在c7node1中格式化namenode  hdfs namenode -format |

启动namenode，以便同步其他namenode:

|  |
| --- |
| #在c7node1中启动namenode  hdfs --daemon start namenode |

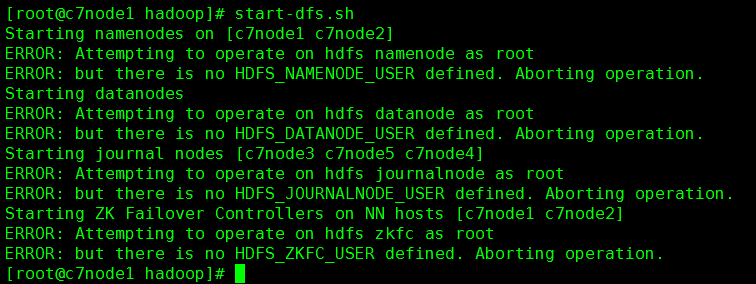
同步其他namenode:

|  |
| --- |
| #高可用模式配置namenode,使用下列命令来同步namenode(在需要同步的namenode中执行):  hdfs namenode -bootstrapStandby |

启动HDFS,启动yarn:

|  |
| --- |
| start-dfs.sh  start-yarn.sh |

注意：如果启动报错：



解决：

在../sbin/start-dfs.sh ../sbin/stop-dfs.sh两个文件中顶部添加如下参数：

|  |
| --- |
| HDFS\_DATANODE\_USER=root  HDFS\_DATANODE\_SECURE\_USER=hdfs  HDFS\_NAMENODE\_USER=root  HDFS\_JOURNALNODE\_USER=root  HDFS\_ZKFC\_USER=root |

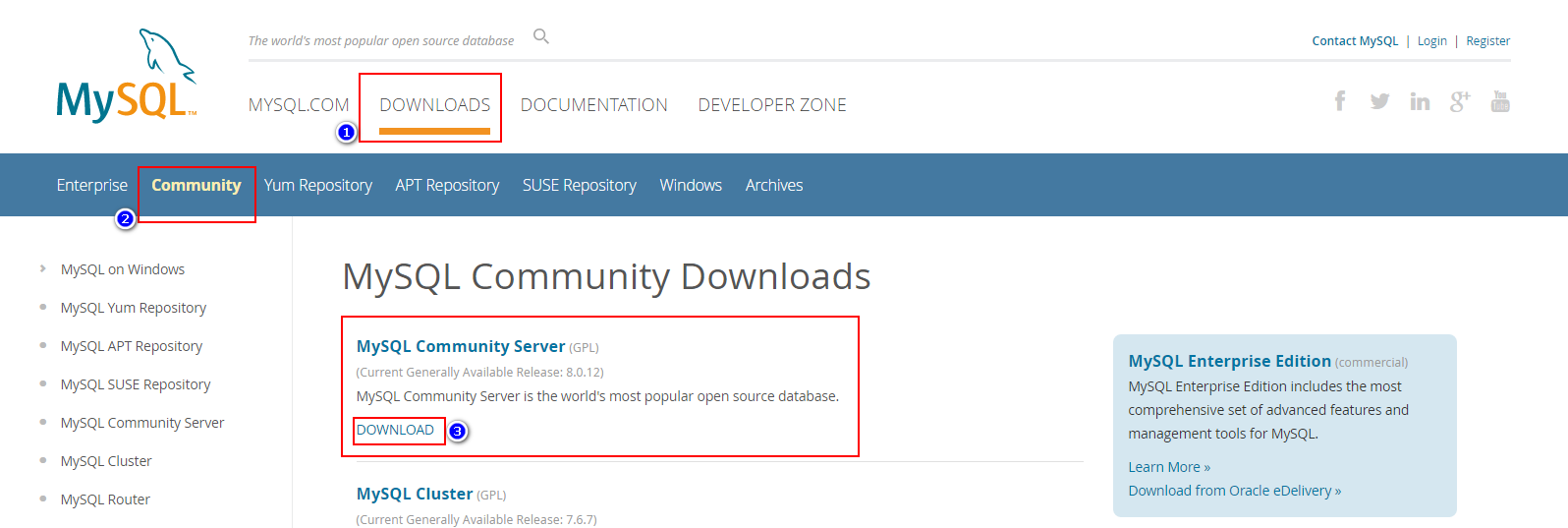
在../sbin/start-yarn.sh ../sbin/stop-yarn.sh两个文件中的顶部添加如下参数：

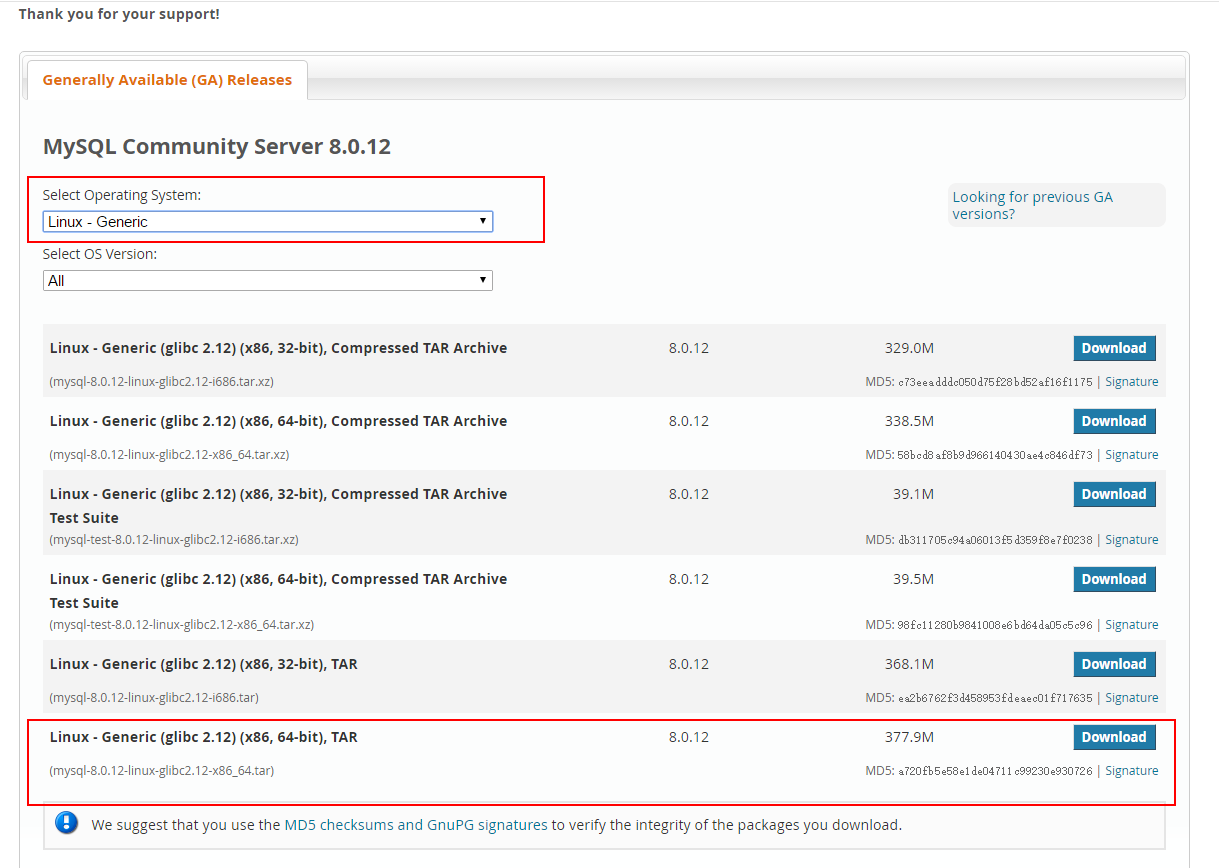
|  |
| --- |
| YARN\_RESOURCEMANAGER\_USER=root  YARN\_NODEMANAGER\_USER=root |

1. 安装Mysql

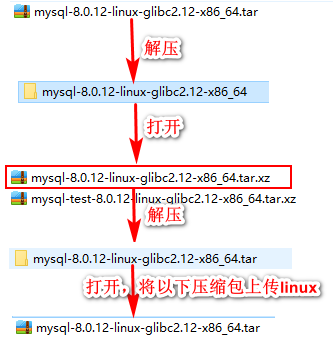
由于之后的要安装的Hive使用MySQL管理原数据。这里需要安装MySQL。MySQL安装在c7node2节点中，下面采用二进制包方式安装mysql。

进入MySQL官网下载最新版本MySQL, <https://www.mysql.com>:





将下载好的文件在window中解压：



一般默认将mysql安装到/usr/local/mysql中，也可以指定某些特定的路径，但是mysql在liunx中默认会找/usr/local/路径，如果更改路径比较麻烦。这里就选择安装在这个路径/usr/local/mysql下。

将以上mysql-8.0.12-linux-glibc2.12-x86\_64.tar包上传到/usr/local/mysql中目录中：

|  |
| --- |
| #创建路径/usr/local/mysql  mkdir /usr/local/mysql  #创建mysql数据存放目录  mkdir /usr/local/mysql/mysqldb  #将mysql-8.0.12-linux-glibc2.12-x86\_64.tar解压到/usr/local/mysql中：  tar -xvf /usr/local/mysql/mysql-8.0.12-linux-glibc2.12-x86\_64.tar -C /usr/local/mysql  rm -rf /usr/local/mysql/mysql-8.0.12-linux-glibc2.12-x86\_64.tar  #将解压的目录下的所有目录放在/usr/local/mysql下：  cp -rf /usr/local/mysql/mysql-8.0.12-linux-glibc2.12-x86\_64/\* /usr/local/mysql/  rm -rf /usr/local/mysql/mysql-8.0.12-linux-glibc2.12-x86\_64/  #创建mysql用户组  groupadd mysql  #创建MySQL用户但该用户不能登陆(-s /bin/false参数指定mysql用户仅拥有所有权，而没有登录权限)  useradd -r -g mysql -s /bin/false mysql  #将创建的mysql用户加入到mysql组下,更改/usr/local/mysql下文件的拥有者为mysql  chown -R mysql:mysql /usr/local/mysql/ |

创建MySQL初始化配置文件，vim /etc/my.cnf，把[mysql-safe去掉]：

|  |
| --- |
| [mysqld]  # 设置3306端口  port=3306  # 设置mysql的安装目录  basedir=/usr/local/mysql  # 设置mysql数据库的数据的存放目录  datadir=/usr/local/mysql/mysqldb  # 允许最大连接数  max\_connections=10000  # 允许连接失败的次数。这是为了防止有人从该主机试图攻击数据库系统  max\_connect\_errors=10  # 服务端使用的字符集默认为UTF8  character-set-server=utf8  # 创建新表时将使用的默认存储引擎  default-storage-engine=INNODB  # 默认使用“mysql\_native\_password”插件认证  default\_authentication\_plugin=mysql\_native\_password  [mysql]  # 设置mysql客户端默认字符集  default-character-set=utf8  [client]  # 设置mysql客户端连接服务端时默认使用的端口  port=3306  default-character-set=utf8 |

初始化MySQL，启动MySQL:

|  |
| --- |
| #进入/usr/local/mysql/bin/目录  cd /usr/local/mysql/bin/  #初始化MySQL,记住初始密码！！！  ./mysqld --initialize --console    #启动MySQL服务：  cd /usr/local/mysql/support-files/  #更改/usr/local/mysql下所有文件的执行权限  chmod -R 777 /usr/local/mysql  #启动MySQL  ./mysql.server start  #修改随机登录密码，设置允许远程登录  /usr/local/mysql/bin/mysql -u root -p    alter user 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY '123456';    use mysql;  update user set host = '%' where user = 'root';  flush privileges; |

设置MySQL开机启动:

|  |
| --- |
| #将MySQL加入系统进程中：  cp /usr/local/mysql/support-files/mysql.server /etc/init.d/mysqld  #重启MySQL  service mysqld restart  #赋予可执行权限  chmod +x /etc/init.d/mysqld  #添加服务  chkconfig --add mysqld  #显示服务列表，如果看到mysql的服务，并且3,4,5都是on的话则成功，如果是off，则键入：chkconfig --level 345 mysqld on  chkconfig --list  #重启电脑，检测：  init 6  netstat -na | grep 3306 |

配置MySQL环境变量：

|  |
| --- |
| #配置环境变量 vim /etc/profile 最后加入：  export PATH=$PATH:/usr/local/mysql/bin  #使配置生效  source /etc/profile |

1. 安装Hive

Hive的安装方式采用Hive多用户模式，有服务端和客户端：

|  |  |  |  |
| --- | --- | --- | --- |
| 节点 | Hive服务端 | Hive客户端 | MySQL管理原数据 |
| c7node1 | ★ |  |  |
| c7node2 |  |  | ★ |
| c7node3 |  | ★ |  |

将Hive的安装包上传到c7node1节点的/software下：

|  |
| --- |
| #上传/software/下，解压  tar -zxvf ./apache-hive-3.0.0-bin.tar.gz -C /software/  rm -rf ./apache-hive-3.0.0-bin.tar.gz  #创建Hive软连接  ln -sf ./apache-hive-3.0.0-bin/ hive  #发送到c7node3节点，在c7node3中也创建软连接  scp -r ./apache-hive-3.0.0-bin/ c7node3:`pwd`  【在c7node3中/software/下】ln -sf ./apache-hive-3.0.0-bin/ hive |

配置Hive的环境变量：

|  |
| --- |
| #分别在两台节点中配置Hive的环境变量  vim /etc/profile  export HIVE\_HOME=/software/hive  export PATH=$PATH:$HIVE\_HOME/bin  #每台节点中使配置生效：  source /etc/profile |

在c7node1节点中创建/software/hive/conf/hive-site.xml,写入：

|  |
| --- |
| #创建hive-site.xml  cp ./hive-default.xml.template hive-site.xml  #配置  <configuration>  <property>  <name>hive.metastore.warehouse.dir</name>  <value>/user/hive/warehouse</value>  </property>  <property>  <name>javax.jdo.option.ConnectionURL</name>  <value>jdbc:mysql://c7node2:3306/hive?createDatabaseIfNotExist=true</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>123456</value>  </property>  </configuration> |

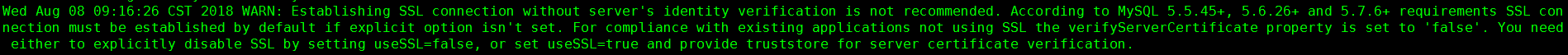
在c7node3节点中创建/software/hive/conf/hive-site.xml,写入：

|  |
| --- |
| #在c7node3中创建hive-site.xml  cp ./hive-default.xml.template hive-site.xml  #配置  <configuration>  <property>  <name>hive.metastore.warehouse.dir</name>  <value>/user/hive/warehouse</value>  </property>  <property>  <name>hive.metastore.local</name>  <value>false</value>  </property>  <property>  <name>hive.metastore.uris</name>  <value>thrift://c7node1:9083</value>  </property>  </configuration> |

在c7node1节点中初始化hive

|  |
| --- |
| #需要将mysql-connector-java-8.0.12.jar 包上传到c7node1节点的/software/hive/lib下  #初始化hive,hive2.x版本后都需要初始化  schematool -dbType mysql -initSchema  #在c7node1节点中启动hive 测试  hive  #创建表test  create table if not exists test (name string comment 'String Column',age int comment 'Integer Column') row format delimited fields terminated by '\t';  #插入一条数据  insert into test values ("zhangsan",18);  #查询  select \* from test; |

问题：启动hive之后，一大串的警告：Establishing SSL connection without server's identity verification is not recommended… ...

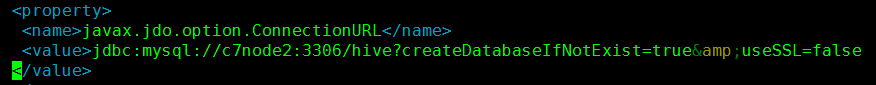


这是由于MySQL库的原因，解决：使用JDBC连接MySQL服务器时设置useSSL参数：设置useSSL=false即可。

这里注意，一般在连接数据库路径后面加上***&useSSL=false***即可：

jdbc:mysql://c7node2:3306/hive?createDatabaseIfNotExist=true***&useSSL=false***

但是在hive中 ***&*** 符号使用 ***&amp;*** 来表示，即在hive-site.xml中修改配置即可：



在客户端使用Hive:

|  |
| --- |
| #需要在服务端启动Metastore 服务,在c7node1节点中：  hive --service metastore &  #在c7node3节点中使用hive，查询到的表和c7node1中一样:  hive |

注意：在Hive服务端安装路径下的lib下要放有连接Mysql的包：mysql-connector-java-8.0.12.jar

1. 安装HBase集群
2. 安装Redis

在c7node4中安装redis，这里安装redis单节点。

访问网站：<http://www.redis.cn/download.html>中有详细的安装步骤。

|  |
| --- |
| #将下载好的redis-2.8.18.tar.gz上传解压  tar -zxvf ./redis-2.8.18.tar.gz -C /software/  rm -rf ./redis-2.8.18.tar.gz  #安装需要的C插件  yum install gcc tcl –y  #进入/software/redis-2.8.18中进行编译  make  #创建安装目录，安装  mkdir –p /software/redis  make PREFIX=/software/redis install |

现在就可以使用redis了，进入/software/redis/bin下，就可以执行命令。下面将redis加入环境变量，加入系统服务，设置开机启动：

|  |
| --- |
| #将redis做成后台daemon  cd /software/redis-2.8.18/utils/  ./install\_server.sh  注意：一直回车，下一步，可能出现错误：    #以上错误解决，执行以下命令：  ln -sf /software/redis-2.8.18/src/redis-server /usr/local/bin/  cd /software/redis-2.8.18/utils  ./install\_server.sh  #执行完以上安装，在/etc/init.d下会创建redis\_6379文件，修改名称，加入系统服务  cd /etc/init.d  mv redis\_6379 redisd  chkconfig --add redisd  #检查加入系统状态,3,4,5为开，就是开机自动启动  chkconfig --list  #启动redis 可以使用下列命令  service redisd start  #将redis配置环境变量  vim /etc/profile  加入：  export REDIS\_HOME=/software/redis  export PATH=$PATH:$REDIS\_HOME/bin  #使配置生效  source /etc/profile |

redis命令使用：

|  |
| --- |
| #启动redis  [root@node01 ~]# cd /etc/init.d  [root@node01 init.d]# ll  上面的可以查看网络端口  ls | grep redis可以查看你的Redis脚本文件名  [root@node01 bin]# ./redis-server –help可以查看文件的使用方法  查看端口方法   1. [root@node01 init.d]# netstat -ntpl 2. [root@node01 init.d]# ss -nal   service redisd start 或者 ./redis-server ../redis.conf  #查看redis 进程，由于redis不是java进程，不能jps查看。执行命令，有6379端口即可。  ps -aux|grep redis  #关闭redis服务  ./redis-cli shutdown  #开启redis客户端  /software/redis-2.8.18/src/redis-cli 配置了环境变量可以直接 redis-cli  #查看所有的库,进入redis客户端  config get databases    #切换不同的库  select 1  #查看所有的key  keys \*  #插入某个key  hset rediskey zhangsan 100  #获取刚设置的“rediskey”中所有的key和value的值  hgetall "rediskey"  #获取某个key 对应的value值  hget "rediskey" "zhangsan"  #删除某个key  del key    #查看当前数据库下key的个数,统计所有未被销毁的key  dbsize  #清空所有的数据库数据,16个库的  flushall  #清空当前库下的数据，当前所在库的  flushdb |

1. 安装kafka
2. Kafka节点划分

|  |  |  |  |
| --- | --- | --- | --- |
|  | c7node1 | c7node2 | c7node3 |
| brokers | ★ | ★ | ★ |

1. 上传解压

|  |
| --- |
| #将kafka安装包上传到c7node1中/software/下，解压  tar -zxvf /software/kafka\_2.11-0.11.0.3.tgz -C /software/  rm -rf ./kafka\_2.11-0.11.0.3.tgz  #创建软连接  ln -sf kafka\_2.11-0.11.0.3/ kafka |

1. 配置Kafka

|  |
| --- |
| #进入/software/kafka/config/中修改server.properties  vim /software/kafka/config/server.properties  修改如下内容：  broker.id=0 注意：这里要唯一的Integer类型  port=9092 kafka写入数据的端口  log.dirs=/opt/data/kafka-logs 真实数据存储的位置  zookeeper.connect=c7node3:2181,c7node4:2181,c7node5:2181 zookeeper集群  #将以上配置发送到其他节点 c7node3,c7node4,同时各个节点创建软连接  scp -r /software/kafka\_2.11-0.11.0.3/ c7node2:`pwd`  scp -r /software/kafka\_2.11-0.11.0.3/ c7node3:`pwd`  ln -sf kafka\_2.11-0.11.0.3/ kafka  #修改c7node2,c7node3节点中/software/kafka/config/server.properties的broker.id  顺序分别为1,2  #在每台节点/software/kafka/路径下创建startKafka.sh脚本，写入：  vim /software/kafka/startKafka.sh  nohup bin/kafka-server-start.sh config/server.properties > kafkalog.txt 2>&1 &  注意：以上路径可以写全路径，可以在不同路径中，写全路径中执行startKafak.sh  chmod +x ./startKafka.sh 修改权限  #在每台节点中启动Kafka  /software/kafka/startKafka.sh  #jps查看kafka是否启动,jps有kafka进程就可以  jps |

1. kakfa命令测试

|  |
| --- |
| #创建topic  ./kafka-topics.sh --zookeeper c7node3:2181,c7node4:2181,c7node5:2181 --create --topic testtopic --partitions 3 --replication-factor 3  #查看kafka集群所有topic  ./kafka-topics.sh --zookeeper c7node3:2181,c7node4:2181,c7node5:2181 –list  #console控制台向topic 中生产数据  ./kafka-console-producer.sh --broker-list c7node1:9092,c7node2:9092,c7node3:9092 --topic testtopic  #console控制台消费topic中的数据  ./kafka-console-consumer.sh --bootstrap-server c7node1:9092,c7node2:9092,c7node3:9092 --topic testtopic |

1. 安装Spark集群
2. Spark安装划分

|  |  |  |  |
| --- | --- | --- | --- |
| 节点 | Master | Worker | Worker |
| c7node1 | ★ |  |  |
| c7node2 |  | ★ |  |
| c7node3 |  |  | ★ |

1. 上传解压包

|  |
| --- |
| #将安装包spark-2.3.1-bin-hadoop2.7.tgz上传，解压  tar -zxvf ./spark-2.3.1-bin-hadoop2.7.tgz -C /software/ --no-same-owner  rm -rf ./spark-2.3.1-bin-hadoop2.7.tgz  #创建Spark 软连接  ln -sf ./spark-2.3.1-bin-hadoop2.7/ spark |

1. 配置Spark

|  |
| --- |
| #配置Worker节点  cp /software/spark/conf/slaves.template slaves  vim /software/spark/conf/slaves  添加worker节点：  c7node2  c7node3  #配置Master节点  cp /software/spark/conf/spark-env.sh.template spark-env.sh  vim /software/spark/conf/spark-env.sh  添加配置：  export SPARK\_MASTER\_HOST=c7node1  export SPARK\_MASTER\_PORT=7077  export SPARK\_WORKER\_CORES=2  export SPARK\_WORKER\_MEMORY=3g  #将配置好的spark包发送到其他节点，其他两个节点也配置软连接  c7node1: scp -r /software/spark-2.3.1-bin-hadoop2.7/ c7node2:/software/  scp -r /software/spark-2.3.1-bin-hadoop2.7/ c7node3:/software/  c7node2: ln -sf ./spark-2.3.1-bin-hadoop2.7/ spark  c7node3: ln -sf ./spark-2.3.1-bin-hadoop2.7/ spark |

1. 启动Spark

|  |
| --- |
| #进入c7node1节点的/software/spark/sbin/start-all.sh 启动Spark集群  cd /software/spark/sbin/  ./start-all.sh |

1. 搭建Spark 提交任务的客户端

原封不动的将Spark安装包复制到一台新的节点就可以，这里是c7node4

|  |
| --- |
| scp -r /software/spark-2.3.1-bin-hadoop2.7/ c7node4:/software/  ln -sf ./spark-2.3.1-bin-hadoop2.7/ spark |

1. 配置Spark运行在Yarn上

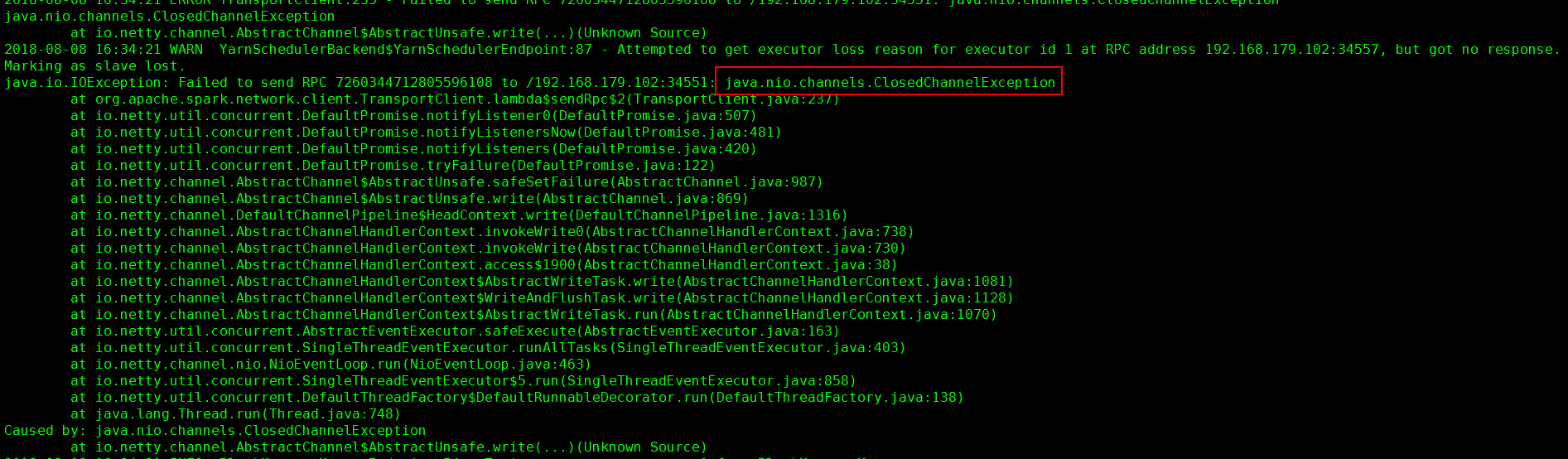
|  |
| --- |
| #进入c7node4中 /software/spark/conf中  vim /software/spark/conf/spark-env.sh  添加：  export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop |

1. 提交SparkPi任务测试

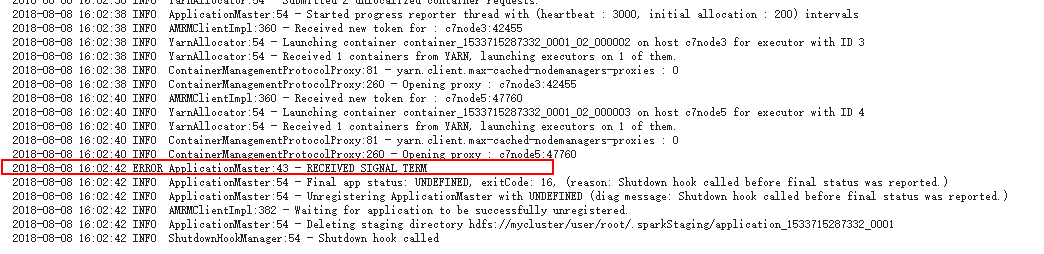
|  |
| --- |
| #基于Standalone提交SparkPi任务  ./spark-submit \  --master spark://c7node1:7077 \  --class org.apache.spark.examples.SparkPi \  ../examples/jars/spark-examples\_2.11-2.3.1.jar 100  #基于Yarn提交SparkPi任务,需要启动Yarn  ./spark-submit \  --master yarn \  --class org.apache.spark.examples.SparkPi \  ../examples/jars/spark-examples\_2.11-2.3.1.jar 10000 |

1. 问题解决

问题：在基于Yarn运行时，提交Spark 任务一直报错：

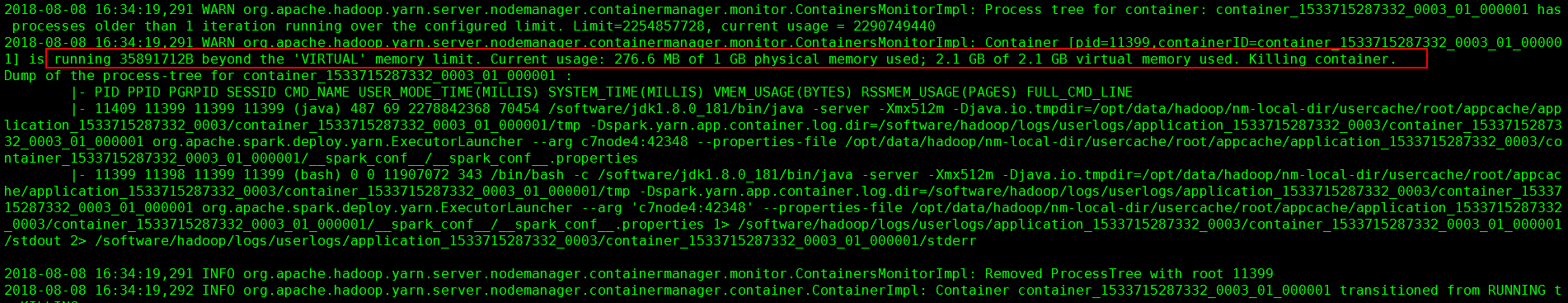


报错：java.io.IOException: Failed to send RPC 7260344712805596108 to /192.168.179.102:34551: java.nio.channels.ClosedChannelException。这个错误去Yarn 中查看日志，也只能看到错误：



其他错误看不到，应该是启动Executor出现了问题，可以去NodeManager节点查看NodeManager的日志，发现如下错误：

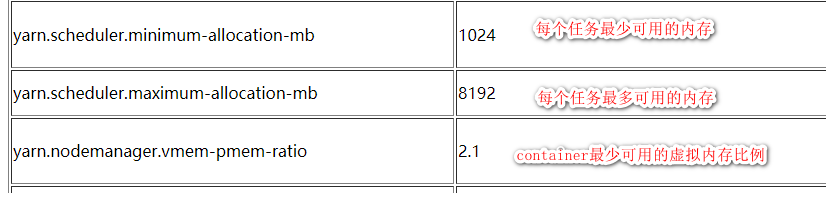
tail -1000 hadoop-root-nodemanager-c7node3.log



错误：container is running 35891712B beyond the 'VIRTUAL' memory limit. Current usage: 276.6 MB of 1 GB physical memory used; 2.1 GB of 2.1 GB virtual memory used. Killing container.

以上错误是说container的虚拟内存不足，导致container启动失败。

以上虚拟内存的设置是在yarn-site.xml中配置来的。



可以在yarn-site.xml中增加配置：

|  |
| --- |
| <property>  <name>yarn.scheduler.maximum-allocation-mb</name>  <value>9000</value>  <discription>每个任务最多可用内存,单位MB,默认8192MB</discription>  </property>  <property>  <name>yarn.scheduler.minimum-allocation-mb</name>  <value>200</value>  <discription>每个任务最少可用内存，单位MB</discription>  </property>  <property>  <name>yarn.nodemanager.vmem-pmem-ratio</name>  <value>4.1</value>  <discription>container最少使用的虚拟内存比例</discription>  </property> |

也可以关闭虚拟内存检查：

|  |
| --- |
| #在yarn-site.xml中设置：  <property>  <name>yarn.nodemanager.vmem-check-enabled</name>  <value>false</value>  </property> |

1. Spark on Hive的配置
2. 配置连接Hive的原数据信息

在Spark的客户端目录/software/spark/conf下创建hive-site.xml文件，配置连接Hive的原数据路径端口，写入如下内容：

|  |
| --- |
| <configuration>  <property>  <name>hive.metastore.uris</name>  <value>thrift://c7node1:9083</value>  </property>  </configuration> |

1. 在Hive的服务端启动Hive 的metastore服务。

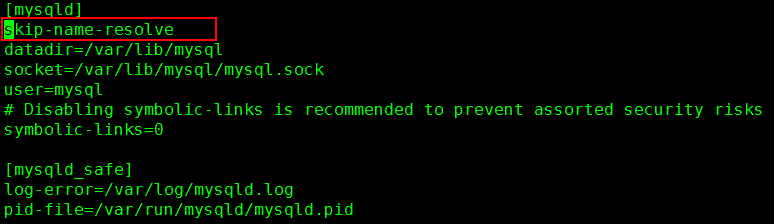
|  |
| --- |
| hive --service metastore & |

1. 在spark客户端使用spark-shell测试

进入spark目录 ../bin/下，启动spark-shell 。

|  |
| --- |
| spark-shell –master spark://c7node1:7077    在spark-shell中Spark session available as 'spark'. 直接使用spark就可以。 |

1. 问题解决
2. window连接linux中的mysql 非常慢，查询语句执行也是非常慢？
   1. .修改/etc/my.cnf文件，在[mysqld]下加入：skip-name-resolve，然后保存。
   2. .重启mysql服务

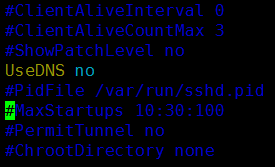


添加skip-name-resolve就跳过反向解析过程: mysql接收到连接请求后，获得的是客户端的ip，为了更好的匹配mysql.user里的权限记录（某些是用hostname定义的）。  
如果mysql服务器设置了dns服务器，并且客户端ip在dns上并没有相应的hostname，那么这个过程很慢，导致连接等待。

1. window本地xshell连接linux虚拟机慢？

在使用shell连接虚拟机时连接等待时长太长，ssh的服务端在连接时会自动检测dns环境是否一致导致的，修改为不检测即可。

* 1. .打开sshd服务的配置文件/etc/ssh/sshd\_config，把UseDNS yes,改为UseDNS no。



* 1. .重启SSH服务

Centos6: service sshd restart

Centos7: systemctl restart sshd.service

1. Spark提交任务的jar包管理问题？
2. Spark基于Standalone模式提交任务

基于Standalone每次提交任务时，都会在Worker节点Spark安装目录的/work目录下生成一个命名为app-xxx-xxx的目录，这个目录下存放程序运行时所需的依赖的jar包。每次提交任务都会在这个work目录下生成一个application目录且不会自动清理。如果时间长了就有可能占用大量的磁盘空间。

清理：可以在worker节点的Spark-env.sh中配置如下参数，定期清理work目录。

|  |
| --- |
| export SPARK\_WORKER\_OPTS="  -Dspark.worker.cleanup.enabled=true #是否开启自动清理  -Dspark.worker.cleanup.interval=1800 #每隔多长时间清理一次，单位s  -Dspark.worker.cleanup.appDataTtl=604800" # 保留最近多长时间的数据,单位s |

以上参数中:

spark.worker.cleanupenabled=true 只有运行完成的application才会被清理。

spark.worker.cleanup.interval 清理周期，单位s,默认值为30分钟。

spark.worker.clearnup.appDataTtl 保存多长时间的数据，单位s,默认是一周。

standalone-client模式：

这种模式提交任务时，以下方式可以解决jar包依赖问题：

1. 可以将依赖的jar包全部打入一个jar包中，直接提交任务，jar包相对大。这个jar包会被复制到每个work节点的work目录app-xx-xx目录下。

|  |
| --- |
| ./spark-submit  --master spark://node1:7077  --class com.bjsxt.spark.areaRoadFlow.AreaTop3RoadFlowAnalyze  /root/test/TrafficProject-1.0-SNAPSHOT-jar-with-dependencies.jar |

1. 可以将依赖的jar包在提交任务时使用--jars 来指定，这种方式每次都会给每个work节点的worker目录中application-xx-xx路径复制一份依赖的jar包(提交的任务jar包也会被复制到这个路径下)。提交任务时，使用不含有依赖的jar包。

|  |
| --- |
| ./spark-submit  --master spark://node1:7077  --jars /root/test/mysql-connector-java-5.1.47.jar,/root/test/fastjson-1.2.11.jar  --class com.bjsxt.spark.areaRoadFlow.AreaTop3RoadFlowAnalyze  /root/test/TrafficProject-1.0-SNAPSHOT.jar 4 |

1. 提交任务时使用不含有依赖的jar包，将依赖的jar包放入每台worker节点的spark安装目录下jars目录下（如果客户端时单独一台，客户端这个目录下也要放jar包），这种模式不会将依赖的jar包复制到worker节点worke下application-xxx-xxx目录，class所在的提交jar包会在这个目录下：

|  |
| --- |
| ./spark-submit  --master spark://node1:7077  --class com.bjsxt.spark.areaRoadFlow.AreaTop3RoadFlowAnalyze  /root/test/TrafficProject-1.0-SNAPSHOT.jar |

standalone-cluster模式：

cluster模式提交任务时，由于Driver是在worker节点中启动，会去worker节点路径中寻找class所在的jar包，这样需要将class所在的jar包在每个worker点中有一份，比较麻烦，最好将class所在的jar包上传到hdfs中的某个路径中，提交任务时指定hdfs路径即可。

1. 提交任务时将所有依赖包打入一个jar包，使用含有依赖的jar包，这个jar包会被复制到每台worker节点的worke目录app-xx-xx下。

|  |
| --- |
| ./spark-submit  --master spark://node1:7077  --deploy-mode cluster  --class com.bjsxt.spark.areaRoadFlow.AreaTop3RoadFlowAnalyze  hdfs://node1:9000/spark/TrafficProject-1.0-SNAPSHOT-jar-with-dependencies.jar |

1. 提交任务时使用不含有依赖包的jar包，使用- -jars 来指定依赖的jar包。由于Driver会在worker节点启动，所以每台worker节点上都要含有所有- - jars指定的路径和包。-- jars 也可以指定hdfs中的路径，这样就不需要每台worker节点要含有 - -jars的路径和包，但是依赖的hdfs中的jar包会被复制到每台worker节点的work目录app-xx-xx下。

|  |
| --- |
| ./spark-submit  --master spark://node1:7077  --deploy-mode cluster  --jars hdfs://node1:9000/spark/fastjson-1.2.11.jar,hdfs://node1:9000/spark/mysql-connector-java-5.1.47.jar  --class com.bjsxt.spark.areaRoadFlow.AreaTop3RoadFlowAnalyze  hdfs://node1:9000/spark/TrafficProject-1.0-SNAPSHOT.jar |

1. 提交任务时使用不含有依赖包的jar包，将所有依赖的jar包在每台worker节点的spark安装目录下的jas目录中，class所在的jar包会被复制到每台worker节点的work目录app-xx-xx下，依赖的jar包不会被放在这个路径下。

|  |
| --- |
| ./spark-submit  --master spark://node1:7077  --deploy-mode cluster  --class com.bjsxt.spark.areaRoadFlow.AreaTop3RoadFlowAnalyze  hdfs://node1:9000/spark/TrafficProject-1.0-SNAPSHOT.jar |

1. Spark基于Yarn模式提交任务

无论spark基于Yarn提交任务的client模式还是cluster模式，都会在HDFS中/user/${username}/.sparkStaging目录下产生一个application的目录，这个目录存放class所在的jar包以及- -jars 指定的依赖的jar包，客户端提交任务后，spark任务节点的spark安装目录jars目录下的所有jar包以\_\_spark\_libs\_\_5619457283046496725.zip的形式上传到这个路径下。如果spark\_home/jars目录下的jar包很多，会上传很久，导致任务执行很慢，可以通过在客户端spark\_home/conf/spark-defaults.conf中配置spark.yarn.jars hdfs://node1:9000/sparkjars/\* （sparkjars 目录需要在hdfs中创建，要配置访问权限755），将spark\_home/jars下的所有jar包都上传到hdfs中sparkjars目录下，这样每次提交任务时，就不会从客户端的spark\_home/jars下上传所有jar包，只是从hdfs中sparkjars下读取，速度会很快，省略了上传的过程。

每运行一个application都会在sparkStaging路径下产生一个目录，这个路径默认是没有读取权限的，如果读取查看，可以使用命令：hdfs dfs -chmod -R 755 /user/root/.sparkStaging 来更改权限查看。当application运行完成，这个路径会自动删除，可以通过参数spark.yarn.preserve.staging.files false来配置，这个参数配置在Spark提交任务节点的spark安装目录下的conf/spark-defaults.conf中，默认为false,每次运行完成任务之后会自动清除，如果改成true,每次运行完成任务之后不会清除目录。

提交任务有依赖jar包时，有以下三种方式选择：

1. 提交任务时，可以将所有依赖的jar包打入一个jar包，这样运行任务时，这个jar包会被上传到HDFS中/user/${username}/.sparkStaging/application-xxx下，如果jar包很大，上传速度慢。
2. 提交任务时，不将所有的依赖jar包打入一个jar包，可以使用—jars来指定，但是每次指定的jar包会上传到hdfs中/user/${username}/.sparkStaging/application-xxx路径下，如果依赖jar包很多，上传会很慢，导致任务执行慢。
3. 提交任务时，不将所有的依赖jar包打入一个jar包，将所有的依赖放入hdfs路径sparkjars中 ，这样提交任务时不需要指定- - jars，直接运行即可，默认在执行任务时，会将hdfs中sparkjars包下的所有包复制到/user/${username}/.sparkStaging/application-xxx下，省去上传的过程。

以上所有提交任务的命令，类似standalone 命令，只是master 为yarn 。

1. xxx