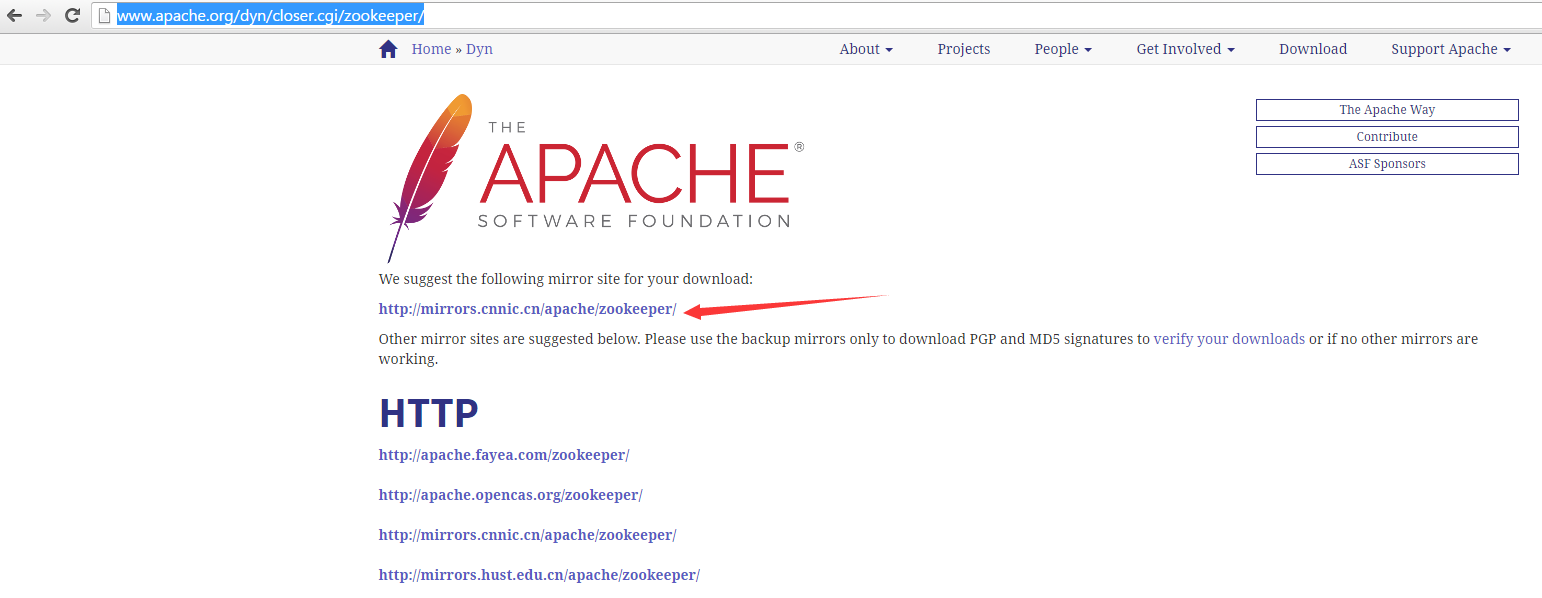
**第12课：HA下的Spark集群工作原理解密**

默认情况下，Standalone的Spark集群是Master-Slaves架构的集群模式，由一台master来调度资源，这就和大部分的Master-Slaves结构集群一样，存在着Master单点故障的问题。如何解决这个单点故障的问题呢？Spark提供了两种方案：基于文件系统的单点恢复(Single-Node Recovery with Local File [system](http://www.07net01.com/tags-system-0.html))和基于zookeeper的Standby Masters(Standby Masters with ZooKeeper)。其中ZooKeeper是生产环境下的最佳选择。

ZooKeeper提供了一个Leader Election机制，利用这个机制你可以在集群中开启多个master并使它们都注册到ZooKeeper实例，ZooKeeper会管理使其中只有一个是Active的，其他的都是Standby的，Active状态的master可以提供服务，standby状态的则不可以。ZooKeeper保存了集群的状态信息，该信息包括所有的Worker，Driver 和Application。当Active的Master出现故障时，ZooKeeper会从其他standby的master中选举出一台，然后该新选举出来的master会恢复挂掉了的master的状态信息，之后该Master就可以正常提供调度服务。整个恢复过程只需要1到2分钟。需要注意的是，在这1到2分钟内，只会影响新程序的提交，那些在master崩溃时已经运行在集群中的程序并不会受影响。

下面我们就实战如何配置ZooKeeper下的spark HA。

1. 下载zookeeper，解压并配置环境变量：
   1. 推荐从官网下载<http://www.apache.org/dyn/closer.cgi/zookeeper/>





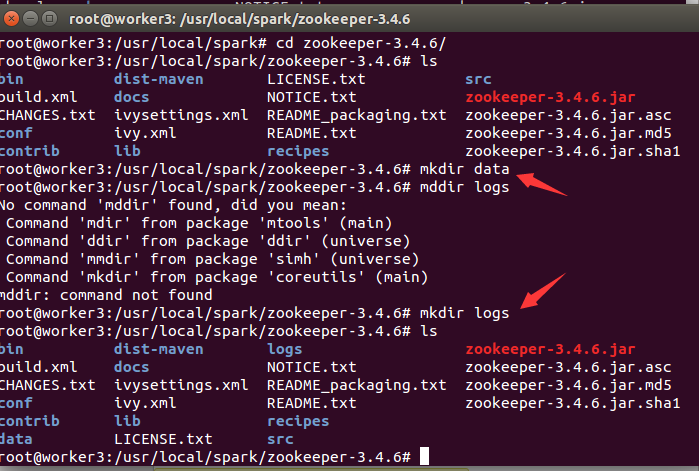
* 1. 解压:tar -xzvf zookeeper-3.4.6.tar.gz
  2. 修改环境变量: vim ~/.bashrc:

export ZOOKEEEPER\_HOME=/usr/local/spark/zookeeper-3.4.6

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

* 1. source ~/.bashrc

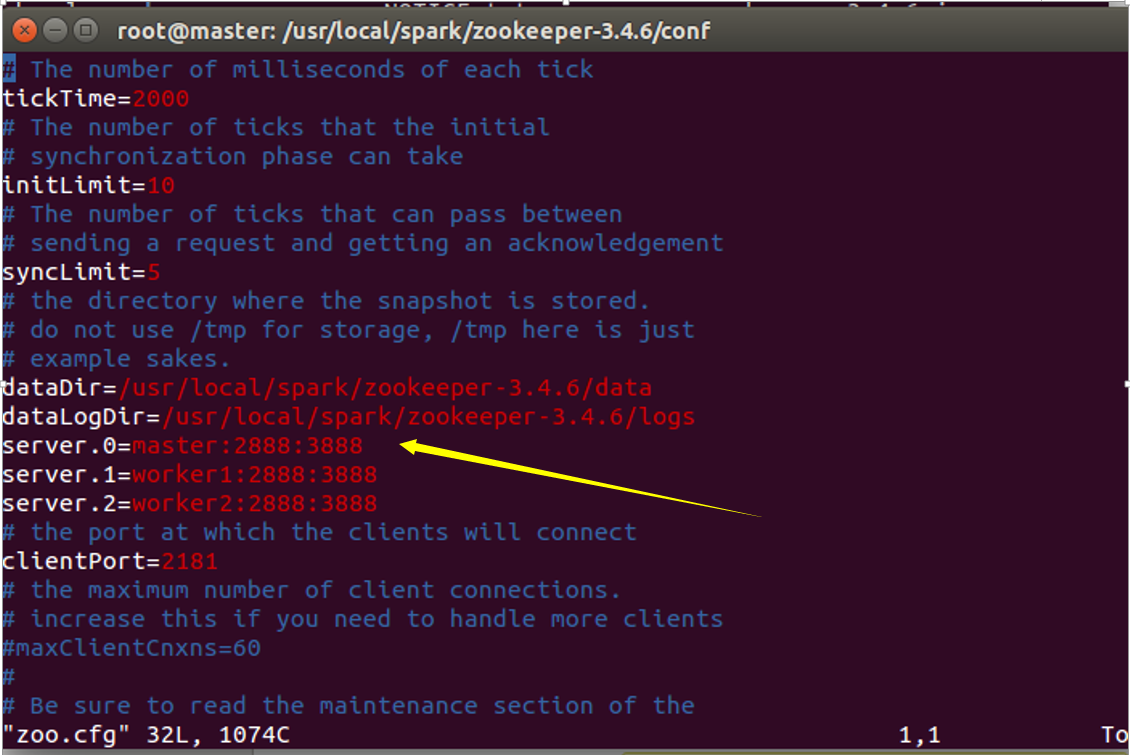
1. 配置ZooKeeper
   1. 首先创建datadir和datalogdir，可以用以下命令创建: mkdir data; mkdir logs:



* 1. 修改配置文件zoo.cfg:

如果$ZOOKEEPER\_HOME/conf文件夹下没有zoo.cfg的话，可以使用命令创建一个：scp zoo\_sample.cfg zoo.cfg

vim zoo.cfg



注：

dataDir指定的是用来存储内存里数据的快照文件的目录；（在不指定dataLogDir时，该目录也是数据库的更改事务日志的存储目录）；

dataLogDir指定的是数据库的事务日志的存储目录；

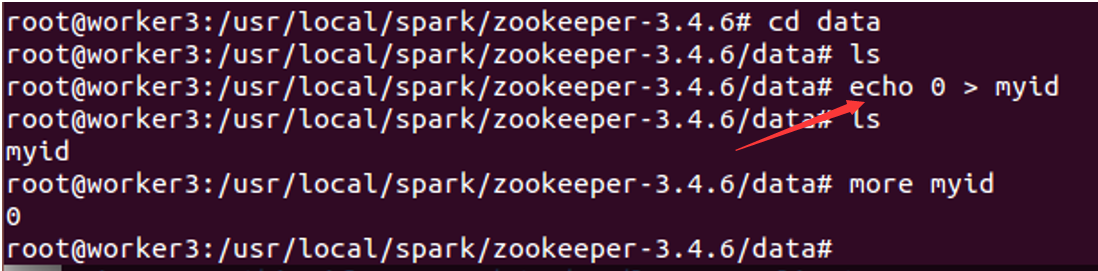
server.0,server.1,server.2指定的是ZooKeeper集群的各个节点，这里的数字012对应该节点datadir下的myid文件的内容(在此强调：必须在这里指定所有ZooKeeper节点！)。

两个端口" 2888" and "3888",前者是ZooKeeper各个节点相互连接通信所用,后者是leader 选举所用。

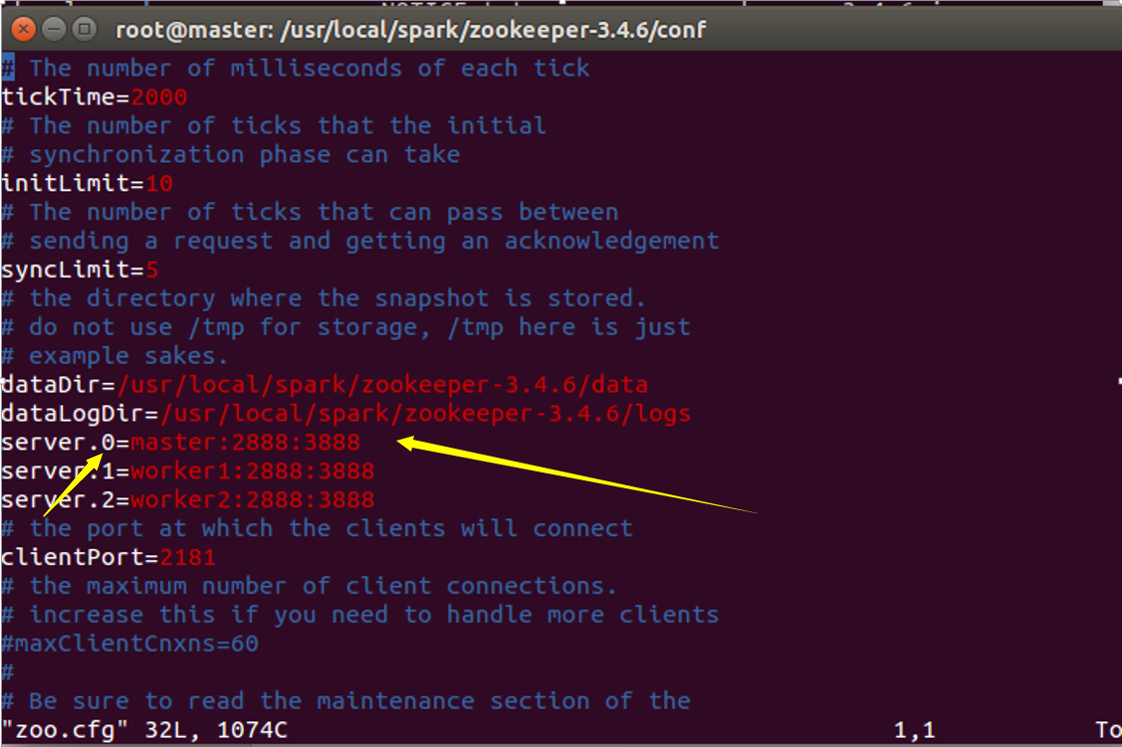
更详细的信息请参阅：

<http://zookeeper.apache.org/doc/trunk/zookeeperStarted.html>

* 1. 在datadir下建立myid文件并修改内容为0:echo 0 > myid



注意：myid文件的内容对应于zoo.cfg中指定的该节点的id.



* 1. 使用scp命令将ZooKeeper拷贝到另外2台机器上（生产环境的最佳实践是用3台机器做HA），配置另外2台机器的环境变量并source使之生效：

scp -r zookeeper-3.4.6/ root@worker1:/usr/local/spark



vim ~/.bashrc:

export ZOOKEEEPER\_HOME=/usr/local/spark/zookeeper-3.4.6

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

source ~/.bashrc

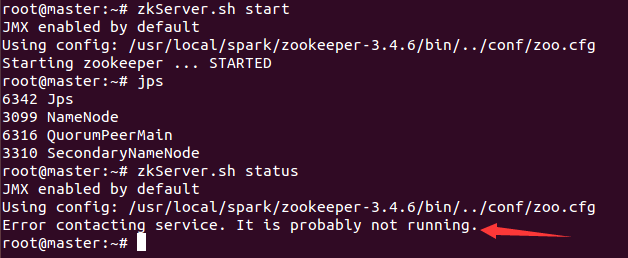
* 1. 在另外2台机器上将data 目录下的文件myid的内容分别修改为1和2。

1. 启动ZooKeeper集群：
   1. 在安装了zk的各个节点上分别执行: zkServer.sh start
   2. jps验证启动成功与否，若见‘QuorumPeerMain’表明进程已启动。

注意：进程启动成功不代表ZooKeeper已经可以开始提供服务。

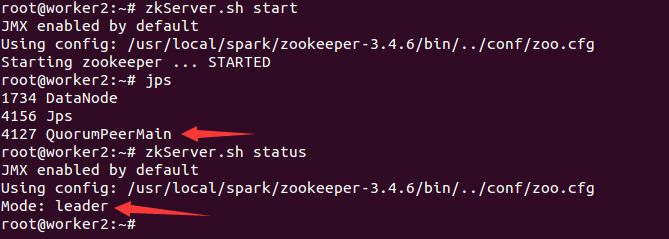
* 1. zkServer.sh status验证启动成功与否，并查看哪台机器被选为leader.

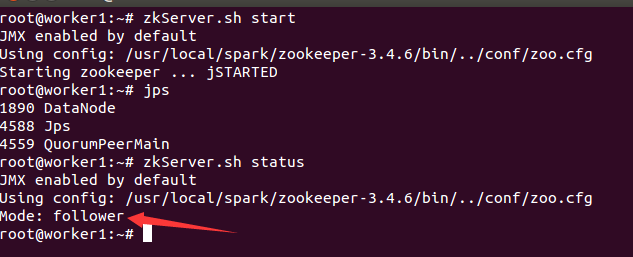
在master节点上启动并验证ZooKeeper状态：



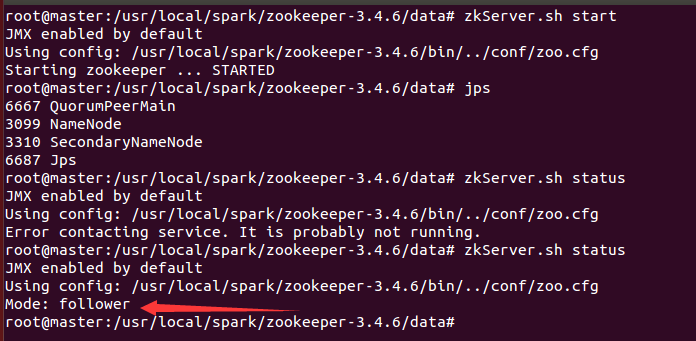
注意: 上图使用命令'zkServer.sh status' 可见如下错误信息，‘Error contacting service. It is probably not running.’， 但这是正常的，一旦你在另外2台节点上启动了ZooKeeper，该错误信息将不会再出现。**一个简单的事实是，只有超过一半的ZooKeeper节点启动后，ZooKeeper才可以正常提供服务**。

在另外2台机器上启动并验证ZooKeeper：





再到master上验证ZooKeeper状态：



可见，现在master节点的错误信息不见了。

现在3台节点的ZooKeeper都已经启动成功，且一台（worker2）是Leader,其它都是follower. ZooKeeper已经启动成功，并正常对外提供服务！

1. 修改spark配置使其支持ZooKeeper下的HA:

cd $SPARK\_HOME/conf;

vim spark-env.sh:

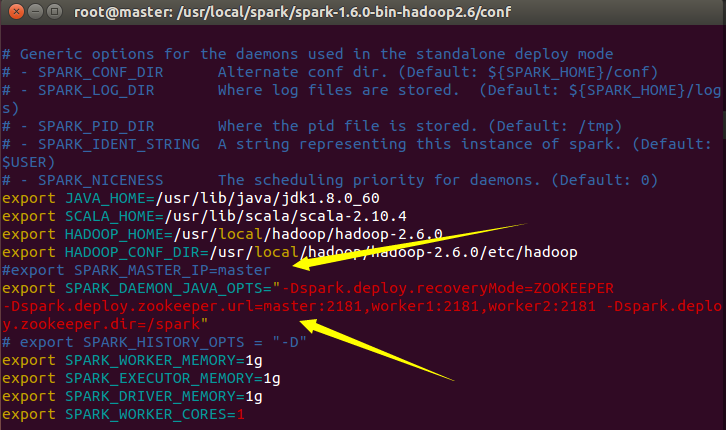
注释掉export SPARK\_MASTER\_IP=MASTER

添加export SPARK\_DAEMON\_JAVA\_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER

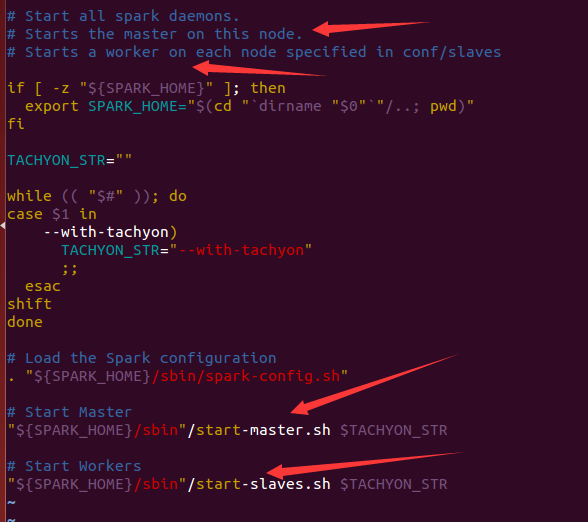
-Dspark.deploy.zookeeper.url=master:2181,worker1:2181,worker2:2181 -Dspark.deploy.zookeeper.dir=/spark"

注意：所有准备用来做master的节点都需要做这个修改。

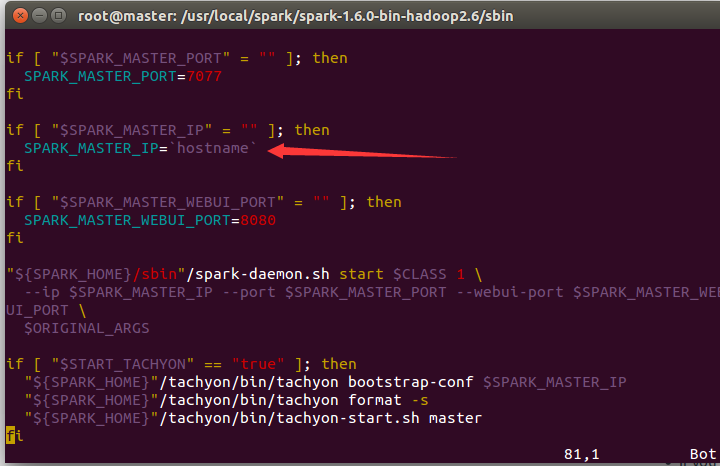




1. 在正式开始启动spark集群之前，我们先来看看启动spark集群的相关命令：
   1. start-all.sh其实质是先后执行了start-master.sh和start-slaves.sh:



* 1. start-master.sh会首先尝试从spark\_env.sh中获取spark\_master\_ip，获取到的话，就在该参数对应的节点上（可以是当前节点也可以不是当期节点）启动master进程；如果没有获取到的话，则会在当前节点上启动master进程：

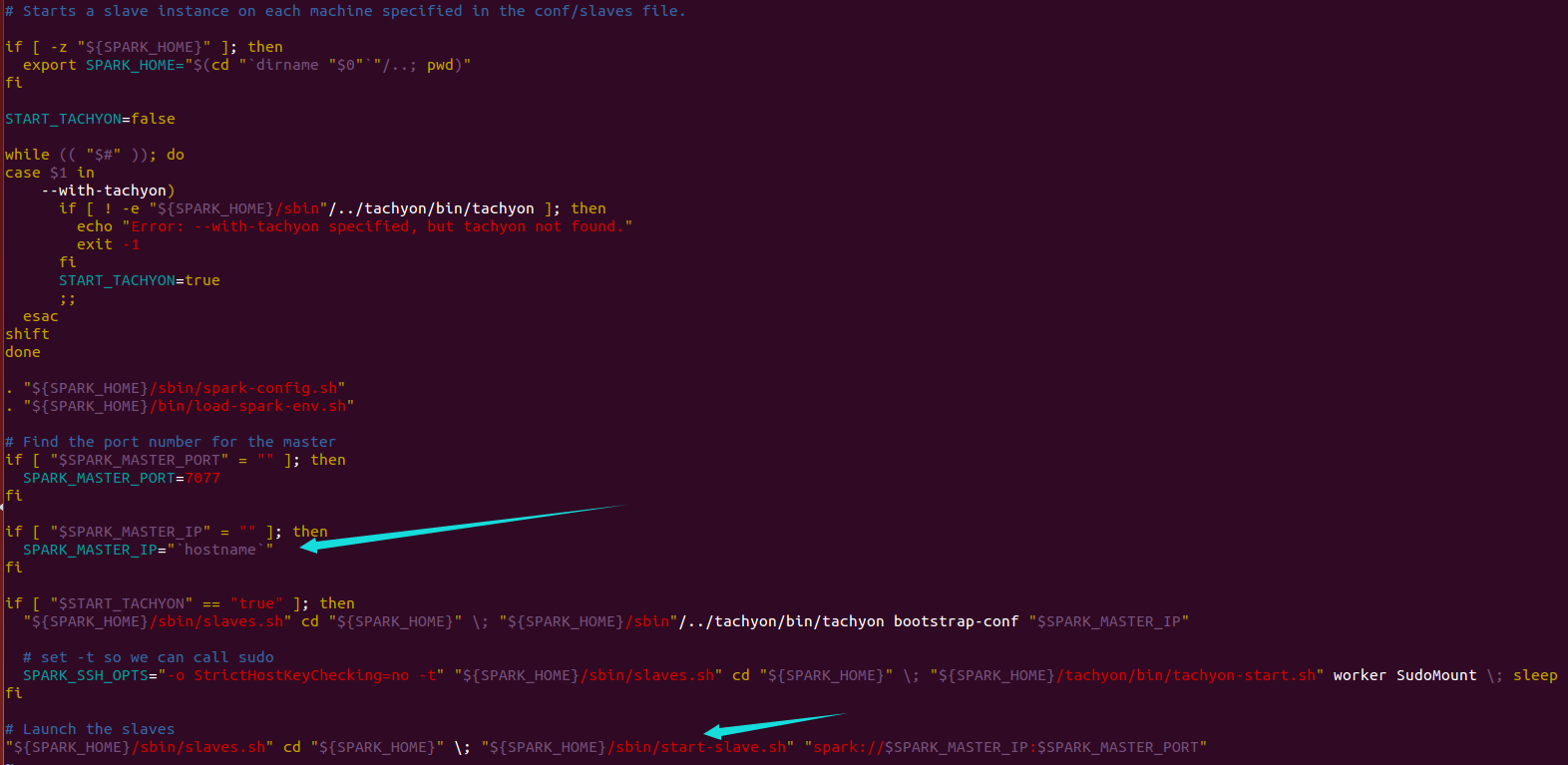


事实上，如果在spark-env.sh中指定了spark\_master\_ip，但它的值不是你当前执行start-master.sh或start-all.sh的节点的hostname的话，会报如下错误：

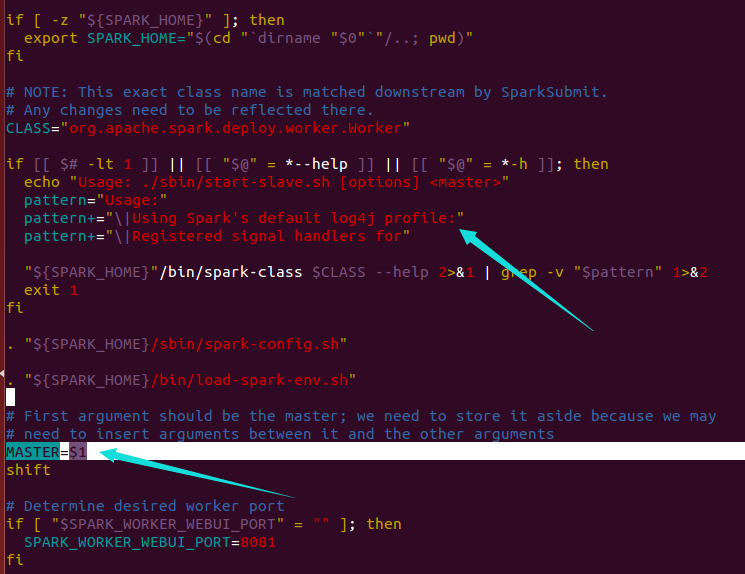
16/02/24 00:21:07 WARN util.Utils: Service 'sparkMaster' could not bind on port 7077. Attempting port 7078.

Exception in thread "main" java.net.BindException: Cannot assign requested address: Service 'sparkMaster' failed after 16 retries!

* 1. start-slaves.sh会在slaves 文件中指定的每个节点上分别调用start-slave.sh来启动worker进程，并尝试注册到特定的master上。这个master通过以下方式获取：首先尝试从spark-env.sh中获取spark\_master\_ip，获取到的话，该参数对应的节点（可以是当前节点也可以不是当期节点）就是master节点；如果没有获取到的话，则会视当前节点为master节点。该命令不能从命令行接受参数：



* 1. start-slave.sh: start-slave.sh必须在命令行指定master(s). 该命令会从命令行的第一个参数中取得master节点。



事实上，该命令常用来在集群中动态添加worker节点，即：整个集群已经启动并正常提供服务，这时有新的节点可用，我们要把它动态添加到集群中，而不必关闭正在运行的集群，就可以使用该命令。

* 1. 不论你用哪种方式启动worker（start-slaves.sh 或start-slave.sh）,当对应的Master没有启动或启动了但不是active状态的话，该worker进程仍然可以启动成功（即Jps可见该进程），但由于没能向master注册成功，它并不能提供服务，它会一直尝试向master注册!

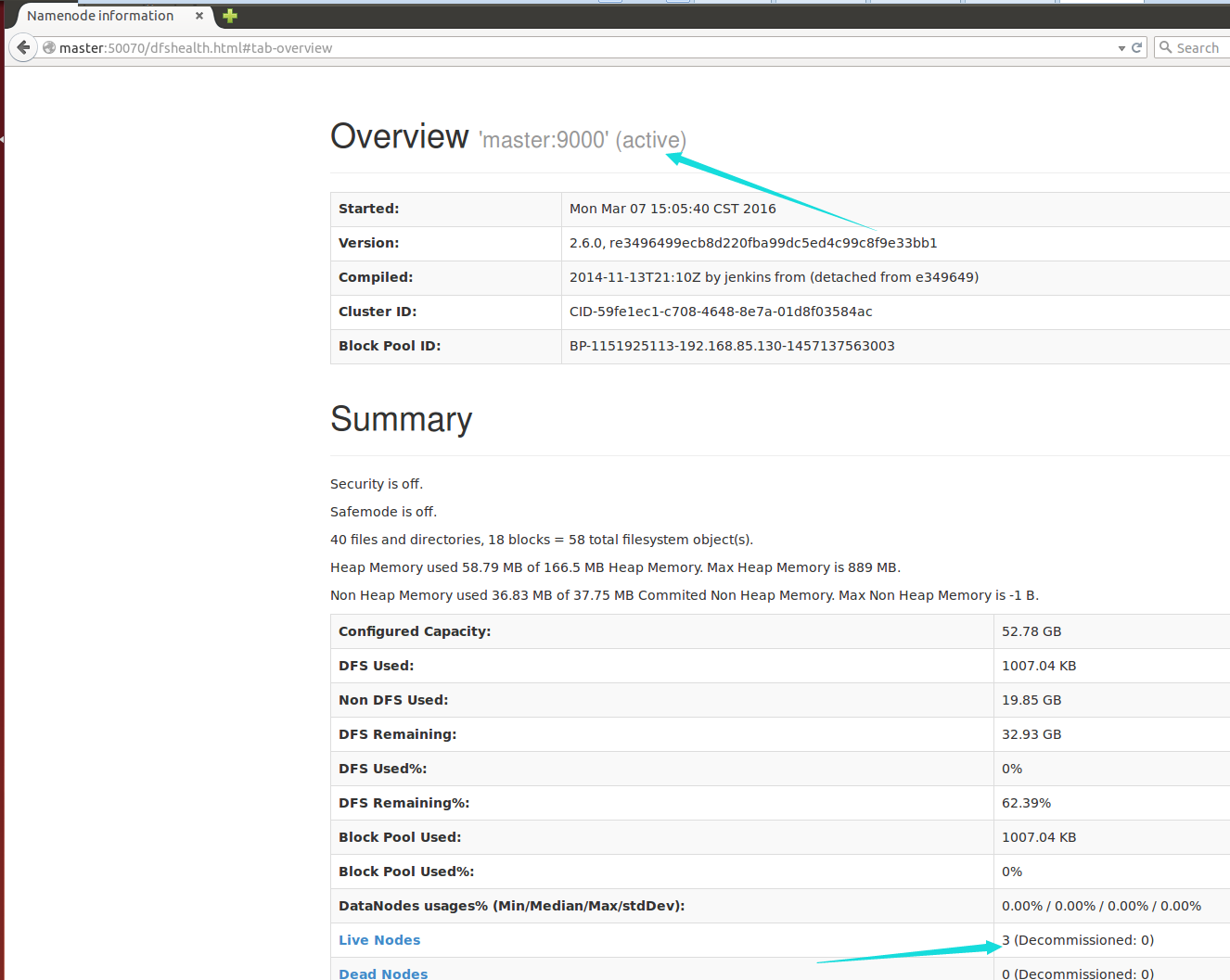
1. 在部署了ZooKeeper的一台host上启动spark：

通过step5中的分析，我们知道当spark\_env.sh中没有配置spark\_master\_ip时，提交命令（start-all.sh, start-master.sh或start-slaves.sh）的host会被视为master节点。在ZooKeeper下，所有的ZooKeeper节点都必须做step4 中的修改，即注释掉SPARK\_MASTER\_IP的配置，并添加SPARK\_DAEMON\_JAVA\_OPTS.

事实上，我们可以在任意一个已经启动了ZooKeeper且ZooKeeper正常提供服务的节点上通过命令start-all.sh来启动spark集群，不管该节点在ZooKeeper中的状态是leader还是follower. (当然你也可以只在该节点上start-master.sh，后续通过start-slaves.sh再启动workers.)

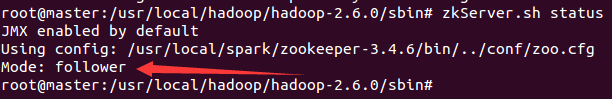
在这里我们在master节点上用start-all.sh来启动spark集群。

因为spark只是一个计算框架，在这里我们还是使用hadoop的hdfs作为文件系统，故需要首先需要确保hdfs已经启动并正常提供服务：

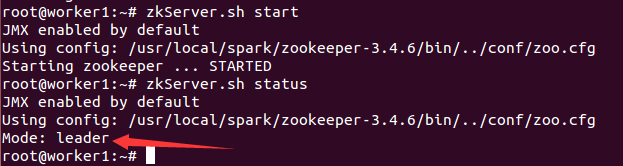


确保ZooKeeper已经启动并正常提供服务：

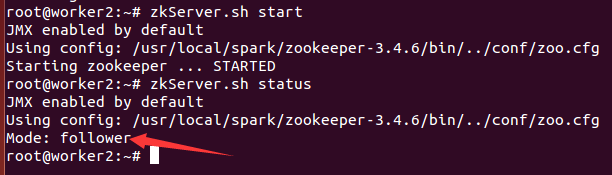
Master节点处于follower状态：



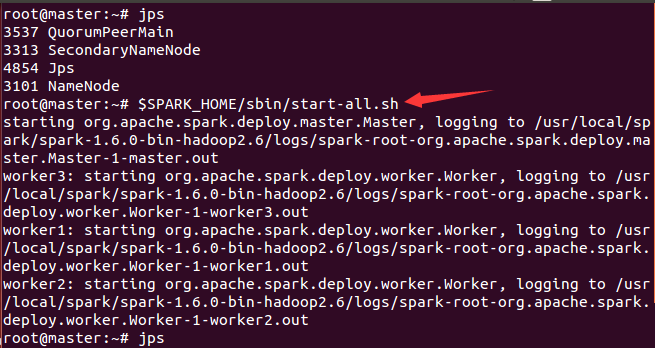
Worker1处于leader状态：



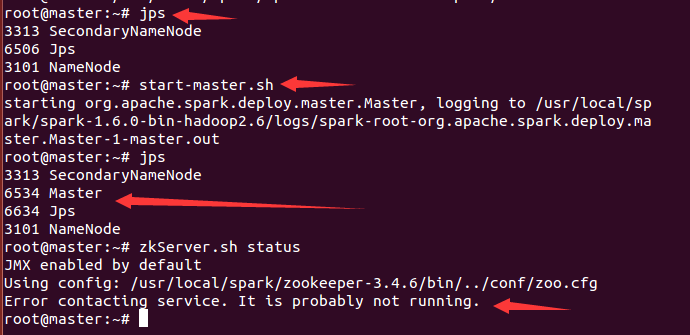
Worker2处于follower状态：

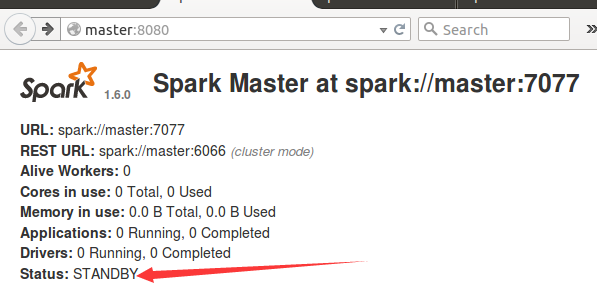


在master节点上用start-all.sh来启动spark集群：（当然你也可以在worker1或worker2上使用start-all.sh来启动整个集群，或只是start-master.sh来启动master进程）：



注：若你正确地配置了某个节点使用ZooKeeper,(即配置了SPARK\_DAEMON\_JAVA\_OPTS且注释掉了SPARK\_MASTER\_IP)，但在没有成功启动ZooKeeper的情况下使用命令start-master.sh来启动master的话，master仍能成功启动，但会处于standby即不能提供服务：

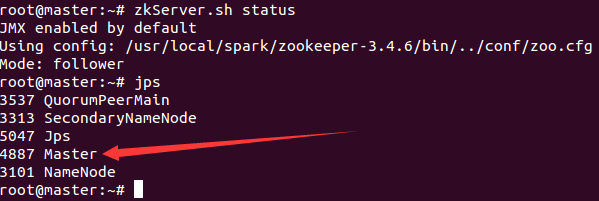




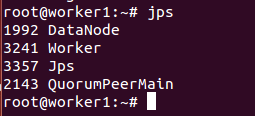
再次强调：需要在成功地启动了ZooKeeper集群的基础上（至少在一半以上的ZooKeeper节点上成功启动了ZooKeeper进程），才能去启动spark集群！

1. jps验证哪些节点有master进程

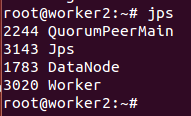
master:



Worker1:



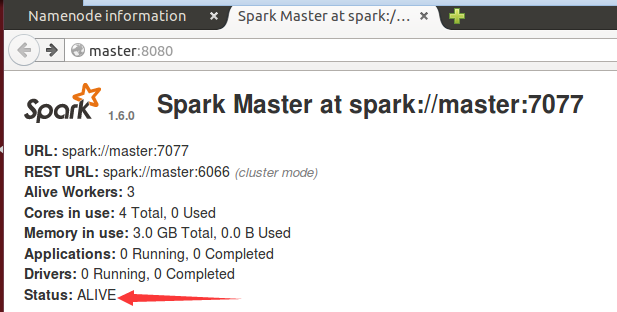
Worker2:



可见只有手动提交了start-all.sh或start-master.sh的master节点有MASTER进程，其他节点则没有**。**

注意：即使有了master进程,也不代表master已经启动并能提供服务,这还要看master的状态是否是ACTIVE.

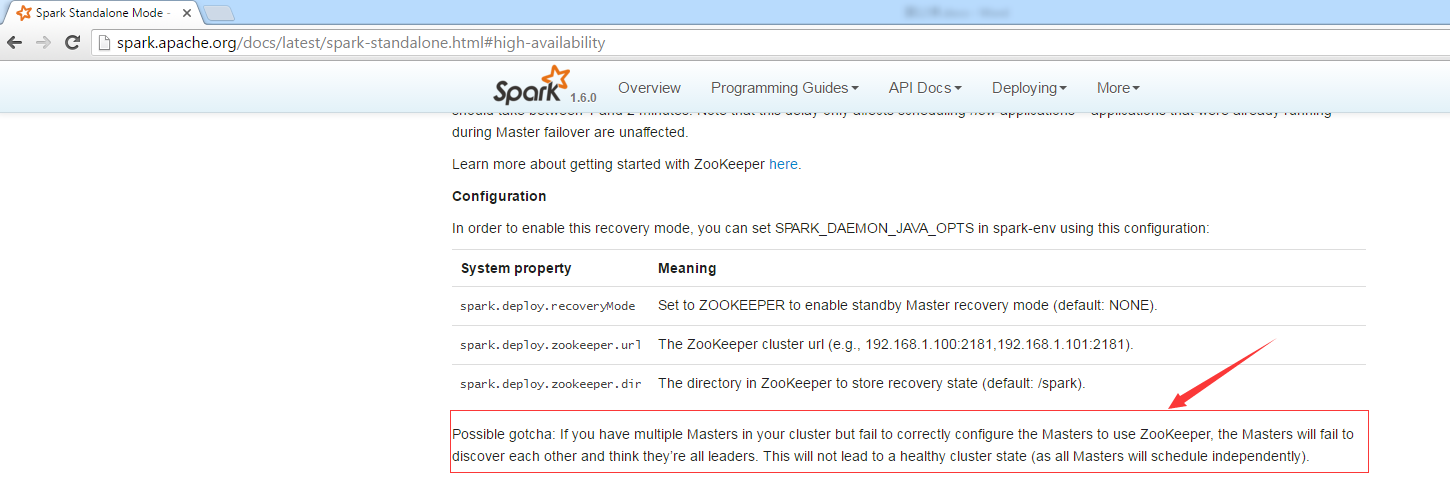
1. WebUI查看master状态.

可见master处于ALIVE状态，在正常提供服务。

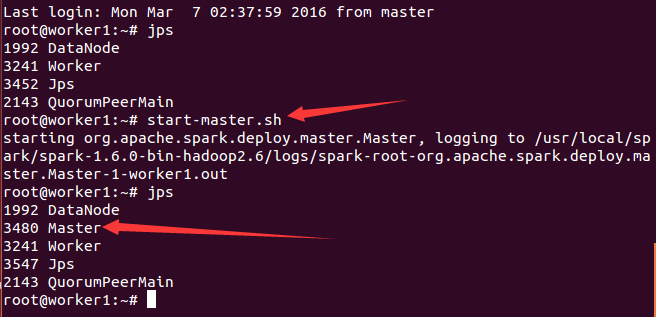
事实上， 第一台手动启动了master进程的节点一定是alive状态的，不管该节点在zk上是leader还是follower!

1. 在其他ZooKeeper节点上手动启动master,并用jps查看Master进程是否启动成功:

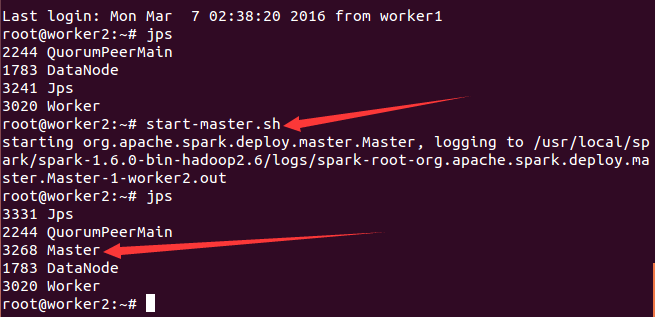
再次强调: 这些节点的spark-env.sh中一定要正确配置为使用ZooKeeper, 即正确配置了SPARK\_DAEMON\_JAVA\_OPTS，否则的话这些启动了的多个masters不能发觉彼此的存在，从而都将自己视为ACTIVE状态，这将置于个集群于不健康状态（因为这些masters都会独立提供资源调度服务）。这个错误如此常见，以至于在官网上也有说明：



在worker1上启动master并jps查看进程:

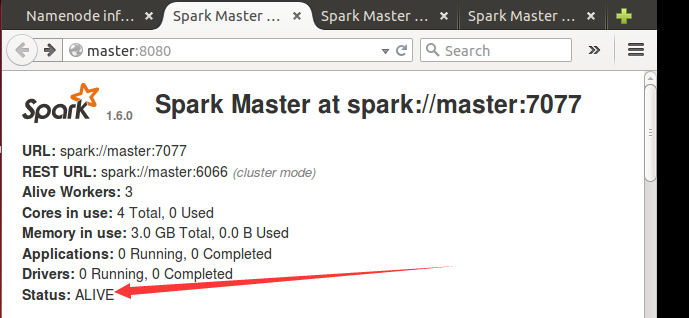


在worker2上启动master并jps查看进程:

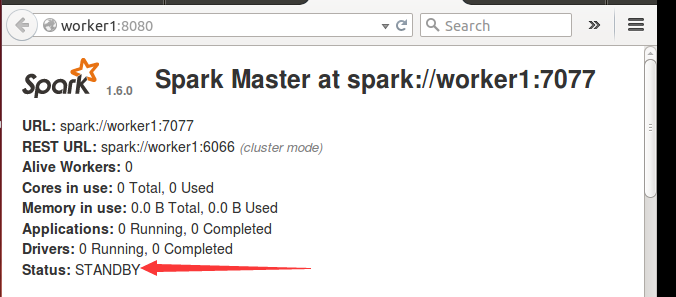


1. WebUI查看所有master的状态。

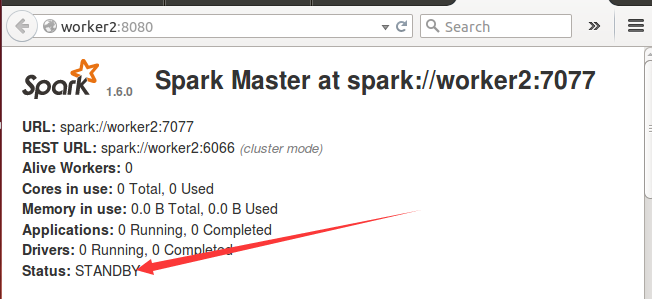
Master:



Worker1:



Worker2:



总结：手动启动的第一个master是ACTIVE状态，后续启动的masters是STANDBY状态, 这与哪个节点是leader,哪个节点是follower没有关系。

注：这些节点有的在zk中可能是leader,但这不影响。

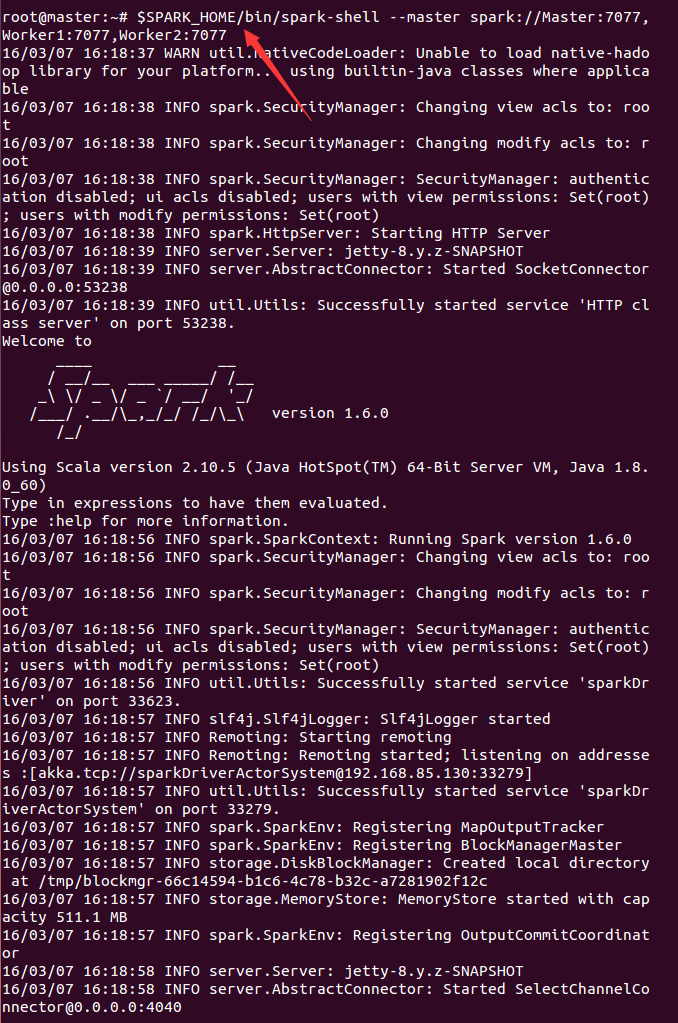
1. 如果你在step6中只是采用start-master.sh启动了master，而不是使用start-all.sh启动了master与conf/slaves文件中指定的workers的话，你需要通过start-slaves.sh启动所有conf/slaves文件中指定的workers.

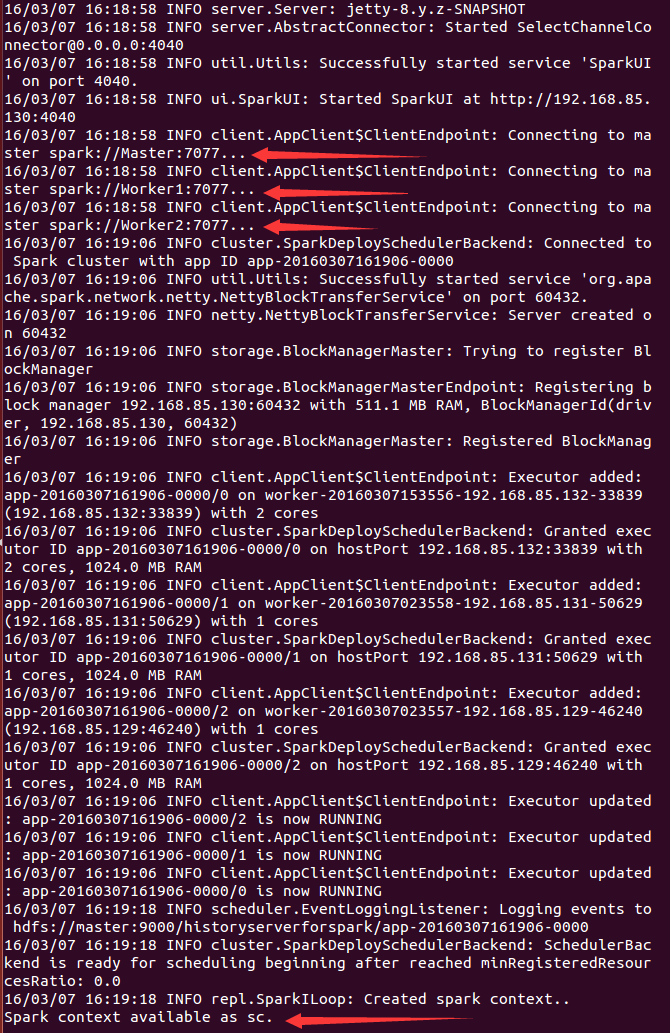
由于我们是使用的start-all.sh启动了整个集群，这里不再需要单独启动workers。

1. 使用ZooKeeper HA下的spark：

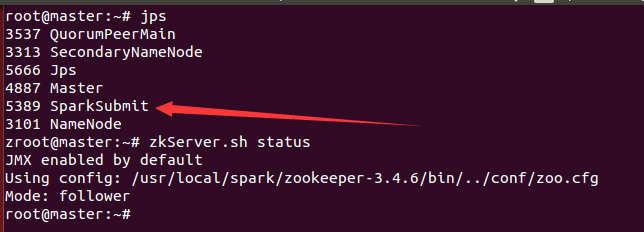
在没有做HA时，当我们提交新程序或动态添加新worker时，需要提供系统唯一的master的ip; 而在ZooKeeper HA状态下的spark集群中,由于有多个master，我们需要提供这些所有的master的ip, Spark程序会依次尝试所有提供的master，知道找到当前正在提供服务的master(即ACTIVE状态的master)。即：Spark程序会跟所有MASTER通信，因为要注册，但最后只跟作为ACTIVE级别的MASTER交互。

验证：$SPARK\_HOME/bin/spark-shell --master spark://Master:7077,Worker1:7077,Worker2:7077

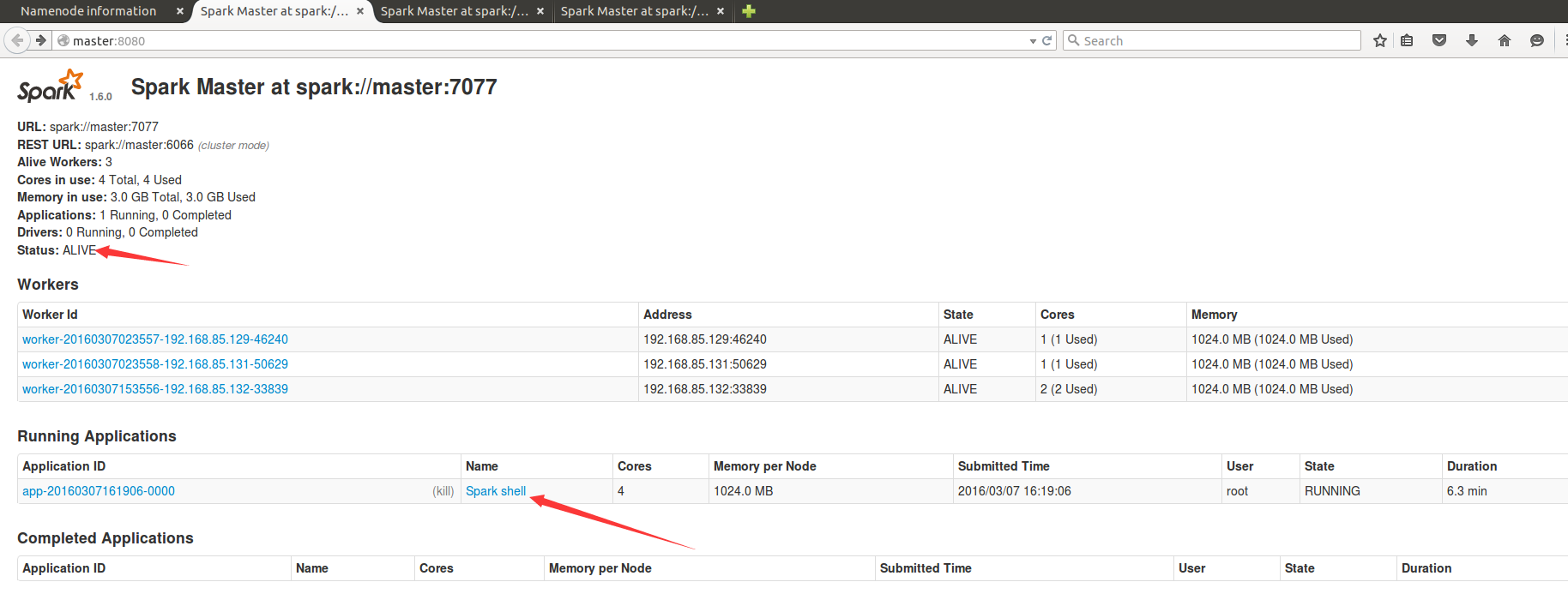


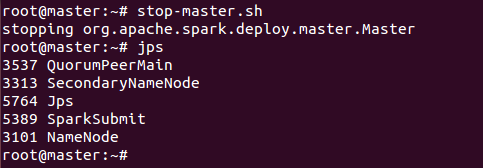


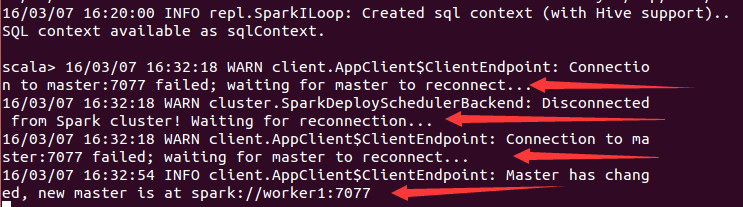
Jps 可见该程序：



Webui 查看该程序：

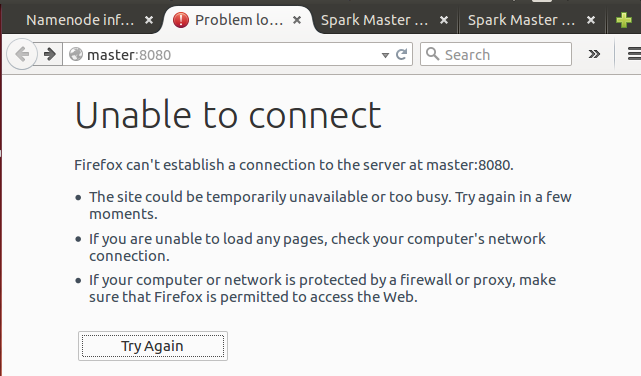


1. 验证HA下的系统故障与恢复:
   1. 在当前ACTIVE状态的master上，用命令stop-master.sh来stop master,模拟实际环境中的机器故障。发布命令后jps,可见master进程确实没有了，spark-shell程序的进程SparkSubmit则一直存在，不受影响：
   2. 此时去spark-shell终端可见如下Log：

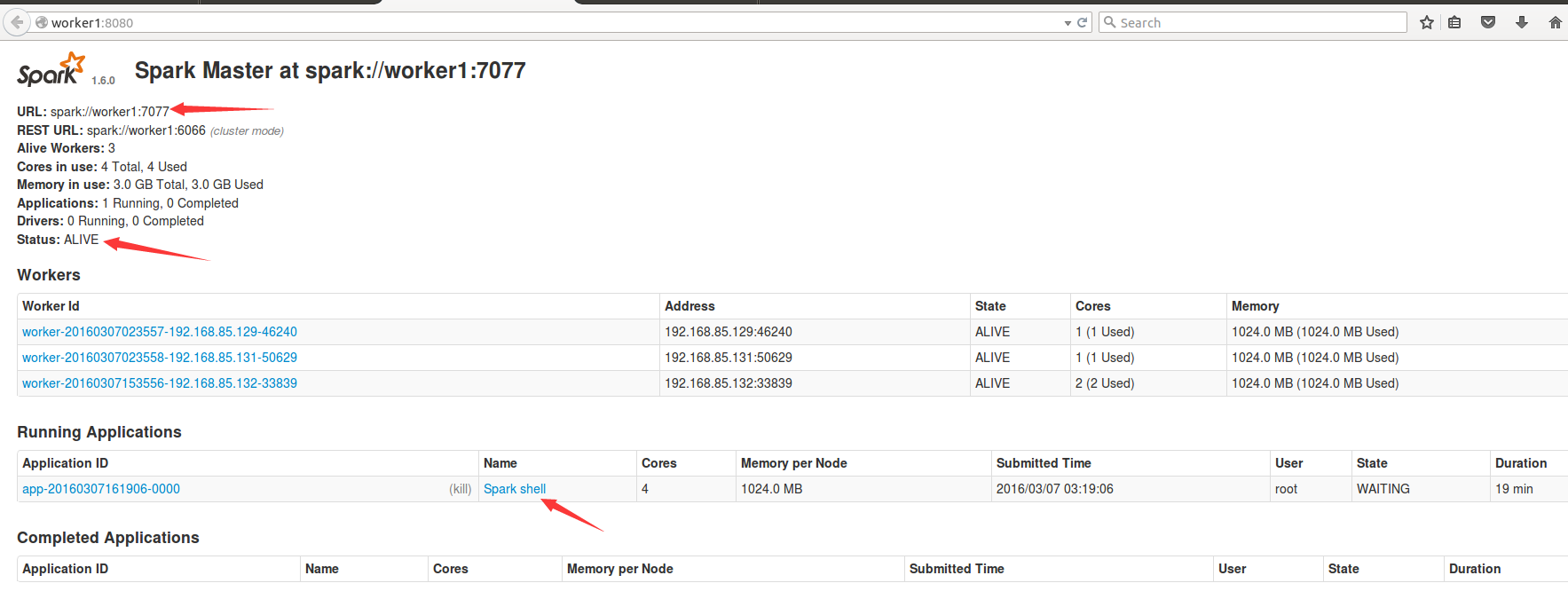


* 1. 去webui查看各个master的状态：

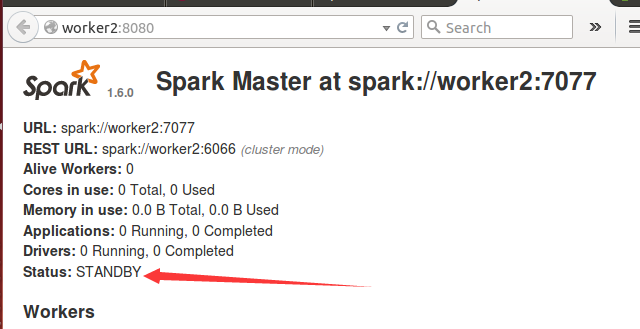
Master节点发生故障无法连接：



Worker1由standby状态变为active状态，且已经开始正常提供服务，在running application下可见提交的spark-shell程序：

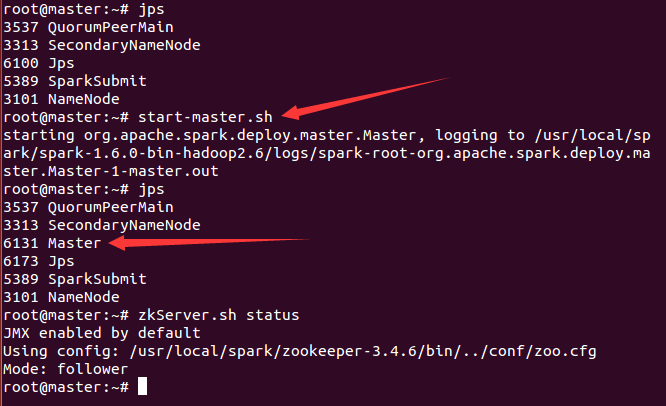


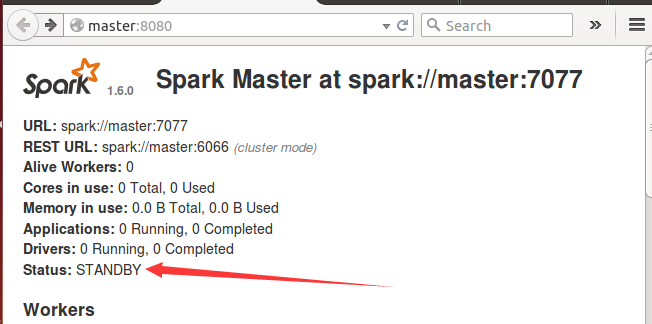
Worker2状态不变，仍是standby状态：



事实上，当提交新程序和动态添加新的worker时，它们需要找到并注册到当前提供服务的处于ACTIVE状态的master,一旦注册成功后，它们就‘在系统中’了（即：相关信息保存在ZooKeeper中了）。后续一旦发生故障，ZooKeeper选举出来的leader会从ZooKeeper保存的信息中恢复发生了故障的master的状态信息，然后开始提供服务，联系所有的已经注册了的application和worker,告诉它们新的ACTIVE的master的ip。这些application和worker在最初启动时甚至不需知道这些后续转为active状态的master的存在！也正因为如此，新的master可以在任何时候被动态创建！

* 1. 在已经发生了故障的master节点上，重新启动master进程，并用Jps和webUI查看其状态：





1. 常见问题与解决方法

if you config as following and start-master on the master host without starting zk:

#export SPARK\_MASTER\_IP=master

export SPARK\_DAEMON\_JAVA\_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.deploy.zookeeper.url=master:2181,worker1:2181,worker2:2181 -Dspark.deploy.zookeeper.dir=/spark"

then below will occur:

1.6.0 Spark Master at spark://master:7077

    URL: spark://master:7077

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

    Alive Workers: 0

    Cores in use: 0 Total, 0 Used

    Memory in use: 0.0 B Total, 0.0 B Used

    Applications: 0 Running, 0 Completed

    Drivers: 0 Running, 0 Completed

    Status: STANDBY