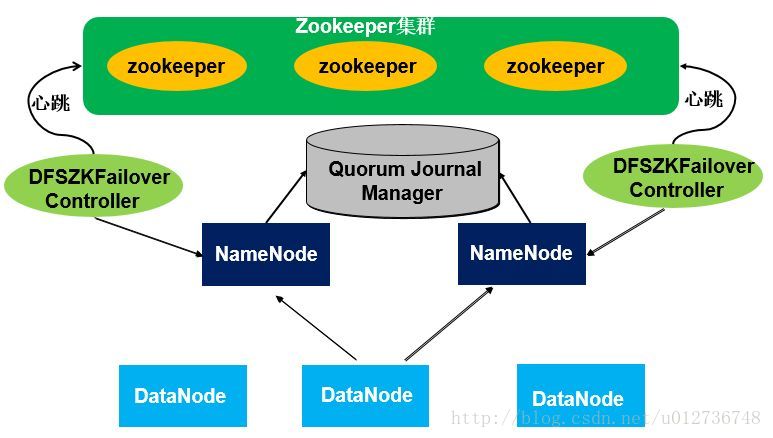
# HDFS HA

参考：<https://blog.csdn.net/u012736748/article/details/79534019>

在Hadoop2.X之前，Namenode是HDFS集群中可能发生单点故障的节点，每个HDFS集群只有一个namenode，一旦这个节点不可用，则整个HDFS集群将处于不可用状态。

**HDFS高可用（HA）方案就是为了解决上述问题而产生的，在HA HDFS集群中会同时运行两个Namenode，一个作为活动的Namenode（Active），一个作为备份的Namenode（Standby）**。备份的Namenode的命名空间与活动的Namenode是实时同步的，所以当活动的Namenode发生故障而停止服务时，备份Namenode可以立即切换为活动状态，而不影响HDFS集群服务。



在一个HA集群中，会配置两个独立的Namenode。在任意时刻，只有一个节点作为活动的节点，另一个节点则处于备份状态。活动的Namenode负责执行所有修改命名空间以及删除备份数据块的操作，而备份的Namenode则执行同步操作，以保持与活动节点命名空间的一致性。

为了使备份节点与活动节点的状态能够同步一致，两个节点都需要**依赖同一组独立运行的节点（JournalNodes，JNS）通信**。当Active Namenode执行了修改命名空间的操作时，它会定期将执行的操作记录在editlog中，并写入JNS的多数节点中。而Standby Namenode会一直监听JNS上editlog的变化，如果发现editlog有改动，Standby Namenode就会读取editlog并与当前的命名空间合并。当发生了错误切换时，Standby节点会保证已经从JNS上读取了所有editlog并与命名空间合并，然后才会从Standby状态切换为Active状态。**通过这种机制，保证了Active Namenode与Standby Namenode之间命名空间状态的一致性，也就是第一关系链的一致性。**

**为了使错误切换能够很快的执行完毕，就要保证Standby节点也保存了实时的数据快的存储信息，也就是第二关系链。**这样发生错误切换时，Standby节点就不需要等待所有的数据节点进行全量数据块汇报，而直接可以切换到Active状态。为了实现这个机制，Datanode会同时向这两个Namenode发送心跳以及块汇报信息。这样就实现了Active Namenode 和standby Namenode 的元数据就完全一致，一旦发生故障，就可以马上切换，也就是热备。

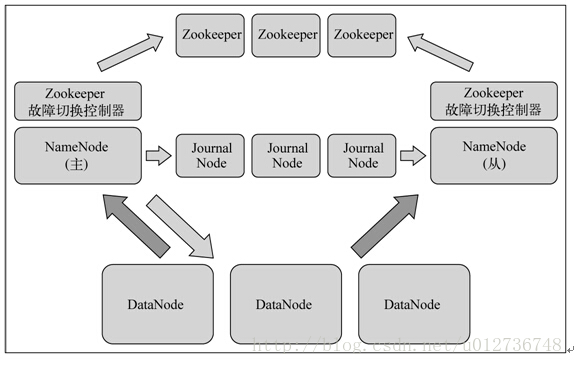
这里需要注意的是 Standby Namenode只会更新数据块的存储信息，并不会向Datanode发送复制或者删除数据块的指令，这些指令只能由Active namenode发送。

**在HA架构中有一个非常重非要的问题，就是需要保证同一时刻只有一个处于Active状态的Namenode，否则机会出现两个Namenode同时修改命名空间的问，也就是脑裂（Split-brain）。**脑裂的HDFS集群很可能造成数据块的丢失，以及向Datanode下发错误的指令等异常情况。为了预防脑裂的情况，**HDFS提供了三个级别的隔离机制（fencing）:**

1. 共享存储隔离：同一时间只允许一个Namenode向JournalNodes写入editlog数据。
2. 客户端隔离：同一时间只允许一个Namenode响应客户端的请求。
3. Datanode隔离：同一时间只允许一个Namenode向Datanode下发名字节点指令，李如删除、复制数据块指令等等。

在HA实现中还有一个非常重要的部分就是Active Namenode和Standby Namenode之间如何共享editlog日志文件。Active Namenode会将日志文件写到共享存储上。Standby Namenode会实时的从共享存储读取edetlog文件，然后合并到Standby Namenode的命名空间中。这样一旦Active Namenode发生错误，Standby Namenode可以立即切换到Active状态。**在Hadoop2.6中，提供了QJM（Quorum Journal Manager）方案来解决HA共享存储问题。**

所有的HA实现方案都依赖于一个保存editlog的共享存储，这个存储必须是高可用的，并且能够被集群中所有的Namenode同时访问。Quorum Journa是一个基于paxos算法的HA设计方案。



Quorum Journal方案中有两个重要的组件：

1. **JournalNoe（JN）：**运行在N台独立的物理机器上，它将editlog文件保存在JournalNode的本地磁盘上，同时JournalNode还对外提供RPC接口QJournalProtocol以执行远程读写editlog文件的功能。
2. **QuorumJournalManager(QJM)：**运行在NmaeNode上，（目前HA集群只有两个Namenode），通过调用RPC接口QJournalProtocol中的方法向JournalNode发送写入、排斥、同步editlog。

**Quorum Journal方案依赖于这样一个概念：HDFS集群中有2N+1个JN存储editlog文件，这些editlog 文件是保存在JN的本地磁盘上的。**每个JN对QJM暴露QJM接口QJournalProtocol，允许Namenode读写editlog文件。当Namenode向共享存储写入editlog文件时，它会通过QJM向集群中所有的JN发送写editlog文件请求，当有一半以上的JN返回写操作成功时，即认为写成功。这个原理是基于Paxos算法的。

使用Quorum Journal实现的HA方案有一下优点：

1. JN进程可以运行在普通的PC上，而无需配置专业的共享存储硬件。
2. 需要单独实现fencing机制，Quorum Journal模式中内置了fencing功能。
3. Quorum Journa不存在单点故障，集群中有2N+1个Journal，可以允许有Ｎ个Journal Node死亡。
4. JN不会因为其中一个机器的延迟而影响整体的延迟，而且也不会因为JN数量的增多而影响性能（因为Namenode向JournalNode发送日志是并行的）

**互斥机制（epoch number机制）**

当HA集群中发生Namenode异常切换时，需要在共享存储上fencing上一个活动的节点以保证该节点不能再向共享存储写入editlog。**基于Quorum Journal模式的HA提供了epoch number来解决互斥问题**，这个概念可以在分布式文件系统中找到。epoch number具有以下几个性质：

1. 当一个Namenode变为活动状态时，会分配给他一个epoch number。
2. 每个epoch number都是唯一的，没有任意两个Namenode有相同的epoch number。
3. epoch number 定义了Namenode写editlog文件的顺序。对于任意两个namenode ,拥有更大epoch number的Namenode被认为是活动节点。

当一个Namenode切换为活动状态时，它的QJM会向所有的JN发送命令，以获取该JN的最后一个promise epoch变量值。当QJM接受到了集群中多于一半的JN回复后，它会将所接收到的最大值加一，并保存到myepoch 中，之后QJM会将该值发送给所有的JN并提出更新请求。每个JN会将该值与自身的epoch值相互比较，如果新的myepoch比较大，则JN更新，并返回更新成功；如果小，则返回更新失败。如果QJM接收到超过一半的JN返回成功,则设置它的epoch number为myepoch；否则它终止尝试为一个活动的Namenode，并抛出异常。

当活动的NameNode成功获取并更新了epoch number后，调用任何修改editlog的RPC请求都必须携带epoch number。当RPC请求到达JN后，JN会将请求者的epoch与自身保存的epoch相互对比，若请求者的epoch更大，JN就会更新自己的epoch，并执行相应的操作，如果请求者的epoch小，就会拒绝相应的请求。当集群中大多数的JN拒绝了请求时，这次操作就失败了。

当HDFS集群发生Namenode错误切换后，原来的standby Namenode将集群的epoch number加一后更新。这样原来的Active namenode的epoch number肯定小于这个值，当这个节点执行写editlog操作时，由于JN节点不接收epoch number小于自身的promise epoch的写请求，所以这次写请求会失败，也就达到了fencing的目的。

**写流程**

1.将editlog输出流中缓存的数据写入JN，对于集群中的每一个JN都存在一个独立的线程调用RPC 接口中的方法向JN写入数据。

2.当JN收到请求之后，JN会执行以下操作：

1）验证epoch number是否正确

2）确认写入数据对应的txid是否连续

3）将数据持久化到JN的本地磁盘

4）向QJM发送正确的响应

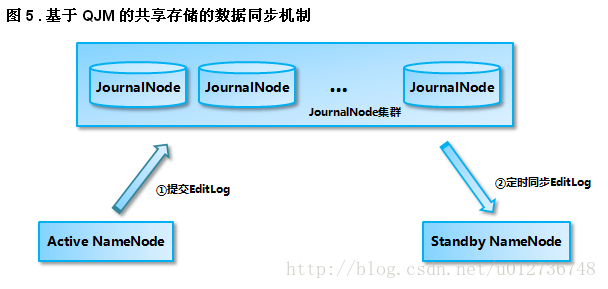
3.QJM等待集群JN的响应，如果多数JN返回成功，则写操作成功；否则写操作失败，QJM会抛出异常。

Namenode会调用FSEditlogLog下面的方法初始化editlog文件的输出流，然后使用输出流对象向editlog文件写入数据。

获取了QuorumOutputStream输出流对象之后，Namenode会调用write方法向editlog文件中写入数据，QuorumOutputStream的底层也调用了EditsDoubleBuffer双缓存区。数据回先写入其中一个缓冲区中，然后调用flush方法时，将缓冲区中的数据发送给JN。

**读流程**

Standby Namenode会从JN读取editlog，然后与Sdtandby Namenode的命名空间合并，以保持和Active Namenode命名空间的同步。当Sdtandby Namenode从JN读取editlog时，它会首先发送RPC请求到集群中所有的JN上。JN接收到这个请求后会将JN本地存储上保存的所有FINALIZED状态的editlog段落文件信息返回，之后QJM会为所有JN返回的editlog段落文件构造输入流对象，并将这些输入流对象合并到一个新的输入流对象中，这样Standby namenode就可以从任一个JN读取每个editlog段落了。如果其中一个ＪＮ失败了输入流对象会自动切换到另一个保存了该edirlog段落的JN上。



**恢复流程**

当Namenode发生主从切换时，原来的Standby namenode会接管共享存储并执行写editlog的操作。在切换之前，对于共享存储会执行以下操作：

1.fencing原来的Active Namenode。这部分在互斥部分已经讲述。

2.恢复正在处理的editlog。由于Namenode发生了主从切换，集群中JN上正在执行写入操作的editlog数据可能不一致。例如，可能出现某些JN上的editlog正在写入，但是当前Active Namenode发生错误，这时该JN上的editlog文件就与已完成写入的JN不一致。在这种情况下，需要对JN上所有状态不一致的editlog文件执行恢复操作，将他们的数据同步一致，并且将editlog文件转化为FINALIZED状态。

3.当不一致的editlog文件完成恢复之后，这时原来的Standby Namenode就可以切换为Active Namenode并执行写editlog的操作。

4.写editlog。在前面已经介绍了。

日志恢复操作可以分为以下几个阶段：

1.确定需要执行恢复操作的editlog段落：在执行恢复操作之前，QJM会执行newEpoch（）调用以产生新的epoch number，JN接收到这个请求后除了执行更新epoch number外，还会将该JN上保存的最新的editlog段落的txid返回。当集群中的大多数JN都发回了这个响应后，QJM就可以确定出集群中最新的一个正在处理editlog段落的txid，然后QJM就会对这个txid对应的editlog段落执行恢复操作了。

2.准备恢复：QJM向集群中的所有JN发送RPC请求，查询执行恢复操作的editlog段落文件在所有JN上的状态，这里的状态包括editlog文件是in-propress还是FINALIZED状态，以及editlog文件的长度。

3.接受恢复：QJM接收到JN发回的JN发回的响应后，会根据恢复算法选择执行恢复操作的源节点。然后QJM会发送RPC请求给每一个JN，这个请求会包含两部分信息：源editlog段落文件信息，以及供JN下载这个源editlog段落的url。

接收到这个RPC请求之后，JN会执行以下操作：

1）同步editlog段落文件，如果JN磁盘上的editlog段落文件与请求中的段落文件状态不同，则JN会从当前请求中的url上下载段落文件，并替换磁盘上的editlog段落文件。

2）持久化恢复元数据，JN会将执行恢复操作的editlog段落文件的状态、触发恢复操作的QJM的epoch number等信息（恢复的元数据信息）持久化到磁盘上。

3）当这些操作都执行成功后，JN会返回成功响应给QJM，如果集群中的大多数JN都返回了成功，则此次恢复操作执行成功。

4.完成editlog段落文件：到这步操作时，QJM 就能确定集群中大多数的JN保存的editlog文件的状态已经一致了，并且JN持久化了恢复信息。QJM就会向JN发送指令，将这个editlog段落文件的状态转化为FINALIZED状态，，并且JN会删除持久化的恢复元数据，因为磁盘上保存的editlog文件信息已经是正确的了，不需要保存恢复的元数据。