HDFS初步笔记1



今天我们来讲HDFS初步，今天的课比较轻松，因为这个暂时还目前不会涉及到关于mapreduce的开发，因为之前mapreduce开发其实它读的数据也就是存在hdfs的，所以相当于也就是在之前的实践里面已经把两者进行一个融合。

HDFS就是一个分布式文件系统，文件系统就是不管是win的系统下还是在linux系统下，那存储的文件它所承载的这个文件的空间它是需要一定的磁盘格式化。比如在win下面有一些文件系统叫一些什么htfs这些文件，你在linux下常见的就是ext3 ext4等，也就是说不同类别的文件系统它相当于是你存储数据的格式是不一样的，你可以任务hdfs也是一种文件系统，只不过是hdfs这个文件系统它是一个分布式的文件系统，就是在多个不同的机器上然后成立了一个集群，在这个集群的规模上做了一个文件系统，相当于这个集群是对你来说是透明的，但是你往上存储的数据细节来讲它其实遍布了不同的机器节点上。

所以有这样的文件系统的话，它也可以存储一些海量的数据并且你在这hdfs上存的这个数据类型也是多种多样的，很丰富，比如说大数据里面有我们上节课主要接触文本数据，那其实比如说做一些图像识别，那可能有一些数据都是图像数据，还有一些视频数据，还有一些文档数据，这些数据类型都非常的丰富，那就看你的具体的业务需要对接什么样的数据，所以之前因为也提到一些hdfs，所以我们讲过的内容就做一个巩固。

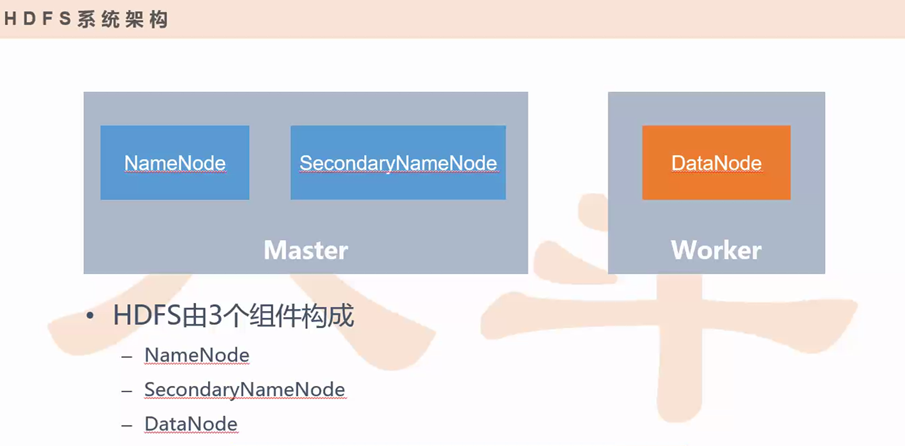
图1：



其实整体的来说比较清晰然后主要分成两个部分，一个是基础理论，还是有一个是实践(如上图1所示)

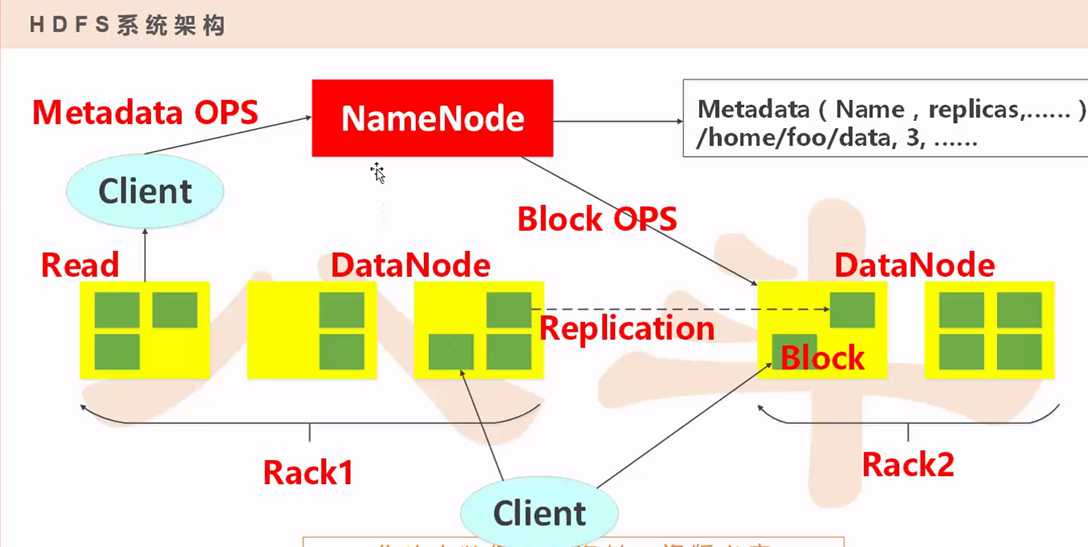
那如果是按照集群角色的划分，因为集群里面我们之前讲mapreduce里面也说了，对这个集群来说那肯定是有一个主和很多的从，一般集群都是这种主从结构，不管你是文件系统HDFS的文件系统的集群还是mapreduce集成框架的集群，那都是有一个主或者是多个从。那在这个HDFS里面的主就是NameNode，从就是DataNode（如下图2所示）

图2：



那在那个主（NameNode）和从（DataNode）其实它是两个完全不同的进程，然后这主(NameNode)和从(DataNode)分别部署到了不同的机器上，那么主机器我们通常就叫Master，然后从机器我们就叫Worker，那当然我们在搭建集群环境里，给从取名为slave，其实也是一样的，就是在主机器上搭这个NameNode，从节点机器上搭这个DataNode对于HDFS来说，所以它的角色就已经很清晰了，但是在主机器上除非一个NameNode以外还有一个叫SecondaryNameNode，那从这个名字上看像是NameNode的一个备份。其实这SecondaryNameNode并不是一个NameNode，它仅仅是一个你可以认为的镜像文件，一些个数据的备份，所以对hadoop1.0来说主要HDFS就有三个组件：1.NameNode，2.SecondaryNameNode，3.DataNode，主要分成这三个内容，那像hadoop2.0里面就在这上面进行了升级和丰富，这个后面会说。

图3：



刚才也说了主是NameNode，那么NameNode下面就是从(dataNode)，dataNode有很多个，相当于我们之前搭集群也是master就只有一台机器，其中从机器就可以搭建很多台。

那么在图3里面红色图就是主（NameNode），黄色图的部分就都是从(dataNode)，那除了这两个部分还有一个Client，Client是我们提交任务的那个客户端。

通常我们现在的实践都是在这master上做一个客户端的一个文件的提交，那么也可以在从（dataNode）上进行提交文件，当然实际上工作的过程中公司一定会让你去登录一个主一个从节点吗？这肯定不能的，这连着的机器权限非常保密的一个状态，所以不可能让你登到一个主一个从上让你提交任务的，那这只能让你登录另外一台机器，那这台机器肯定是不属于hadoop整个集群生态里面的一个份子，但是这台机器可以把你的任务能够指定与这个主（NameNode）进行联系，然后把这个任务进行提交，那这个机器可以认为是你的客户机。

那这个客户机需要把配置文件配好就可以了，然后配置文件里面配置好了整个HDFS的它的那个主节点的一个地址，它就自然的会去与这台主机器建立关系，所以这整个HDFS集群生态里面包含三个角色：1.主(NameNode)，2.从（dataNode），3.客户机(Client)。

这个NameNode目前来说在HDFS集群里面就有一个，并且在Mater节点上也只能启动一个。

然后在Worker这个机器上启动一个DataNode，虽然DataNode有很多，在整个HDFS环境里有很多，但是每一台机器只能启动一个DataNode。

然后client可以提交任务，可以往集群上去写，当然也可以在集群上读，但是在读的时候这个客户端并不是从NameNode中去读数据，而是通过DataNode上去读数据。Client作为客户端来说它怎么知道在那么多个DataNode中它所要的数据在哪个具体的DataNode中呢？

那怎么确定这个目标呢？这个是由NameNode去告诉client客户端的。

另外一个，你看上面的图3黄色的部分是DataNode，可以是认为一台机器，那在这个DataNode中可以从这个名字上可以看出，在这个机器的进程里面它主要管理当前这个机器主要是存数据的，也就是存的一些我们之前所说的那些block，那之前说我们HDFS它的数据的最小单元是一个block，那一个block默认是64m，当然这64m是可以改的，可以通过配置文件去改，另外一个就是因为block它可以直接影响你这个mapreduce中你的map并发的个数，所以在去mapreduce做配置文件的时候，你也可以按照block多大去做一个分割，这是可以配置的。

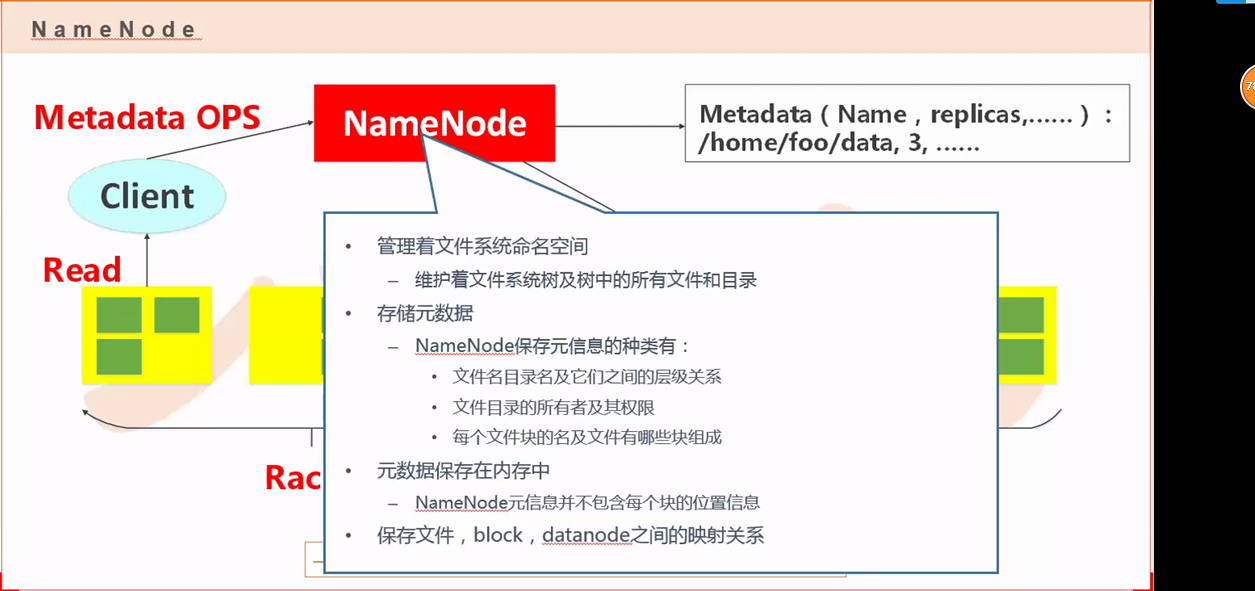
图3中黄色图中里面有很多的一些相当于是绿色的正方形，每一个绿色的正方形你可以认为是一个block，假设每一个block都是64m，相当于这绿色正方形里面有很多64m的文件。

有一个文件或者是有一个dataNode这个机器挂了或者是这个进程出现了一些问题被杀死了或者是挂掉了，那相当于这个节点就丢失了也就是说这个节点上的所有数据也就随着丢失了，

那既然丢失的话就会导致个别的数据它是不能够满足你3副本的一个平衡，这么一个打破了这个平衡，那么也就是这个副本就剩了原来的三个，因为你丢了一个机器，一般来说这个机器要存一个副本，它只存某一个数据一个副本，不可能是存同样一份数据的多个副本，所以通常来说挂一台机器那也就是个别的一些副本由3变成了2，那这个时候就还是存在一定隐患的，因为在配置文件里面之所以配三个是为了保证你的数据的高可用，既然你的高可用受到了一定的挑战了，应该从图3里面的Replication进行复制，比如说有一台机器它挂掉了，那么它副本全局打破平衡了，那应该从某些其他的机器上然后把这个副本从2再复制一份出来，从而由2变成3，这样就又保证了整个集群的一个副本数的一个平衡。

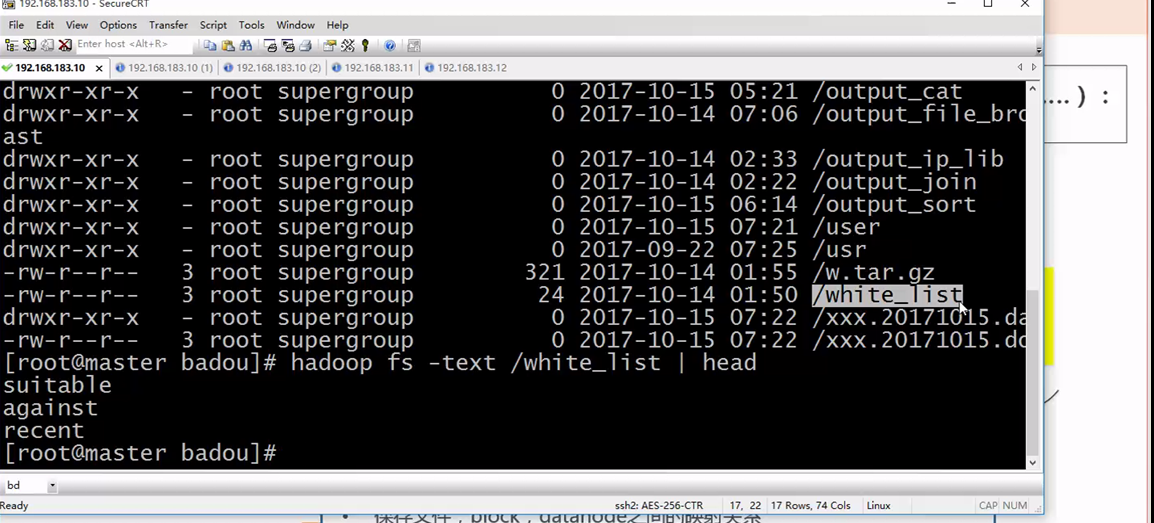
另外要说的是DataNode和NameNode之间的关系是非常紧密的，不是说客户机Client来任务来请求了，NameNode和DataNode才进行交互，这不是的。比如说这client提交了一个任务，那首先这个NameNode看一下哪些DataNode比较空闲的，然后把那些数据往哪个空闲的DataNode节点上进行写，那么有一个疑问就来了，NameNode怎么知道那些DataNode空闲，是发报告发心跳，发心跳就是相当于DataNode不断的告诉你的NameNode当前的一个状态，由DataNode主动申报。就相当于这个老师上课问哪些学生没有写作业，那不可能老师挨个去检查，那没有写作业的直接举手就这个意思。所以有这么一个情况的话就有助于整个的NameNode来管理DataNode，因为DataNode它是存储着具体的block，那也就是说它顺便的话就是NameNode就管理着整个集群的block。

图4：



我们逐个节点去看去深入，先看NameNode，这个NameNode是一个非常重要的，它管理着整个文件系统的一个命名空间（如上图4所示），然后他维护着整个文件系统里面的所有的文件和目录，也就是说举个例子：我们之前往hdfs上随便去上传一个文件，那这个文件（/white\_list）（如下图5所示），这个文件/white\_list是我们从本地上传上去的，那也就是说上传了以后，执行了hadoop fs -text /white\_list | head这条命令相当于把根目录下所有文件都列表列了出来，然后这个时候这个文件相当于前面是有个斜杠(/)，其实就是一个路径，然后路径后面就是文件名。那这个路径就是在NameNode要存的，就如同上图4红色图NameNode右手边的那个路径，然后这个路径会存储哪个信息呢？会存储这个路径的访问权限，就哪些人是允许来去操作这个路径（读，写），那么这些信息就是NameNode的元信息

图5：

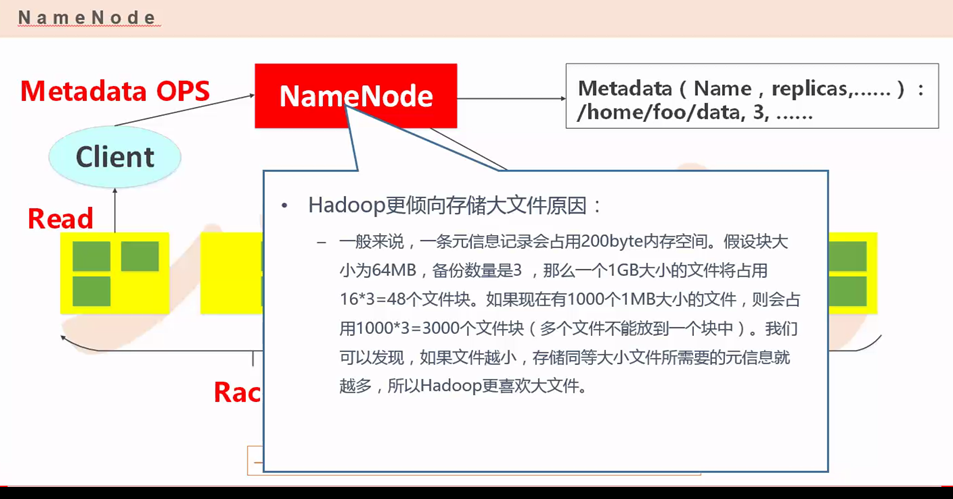


NameNode的元信息都是存在内存里面，当然NameNode的元数据它也会往磁盘上去存，但是真正对外提供服务的这些数据都是在内存进行实时的处理，那磁盘上的数据永远赶不上内存的变化，也就是磁盘上的数据就是某一时刻的状态你是很难能够浮现出当前内存的那种状态。

另外一个很重要的点就是元信息里面有两类很重要的数据，可以认为是数据，也可以认为是映射数据，第一个映射数据是文件名到block，第二个就是block到DataNode，就是我们作为一个客户端来说，操控的是直接是你的(图5中的/white\_list)文件，而不关心你的文件的背后是什么block，这个block叫什么名，这个是没有办法去关心的，仅仅只是这个文件名才对我们有意义的，但是我们要对某一个文件做一个处理，比如说是读取，首先得知道这个文件的文件名背后存了哪些block，虽然我们看不到这个block，但是它本质是把这个大的文件都是按照block去切分好并且散落在不同的机器上散落在不同的DataNode上，但是NameNode会帮你根据文件名能够映射出来它背后有哪些block，NameNode它仅仅能够告诉你有哪些block，但是block它不在NameNode存，它仅仅告诉你block的名字是什么，但是对于你作为客户端来说仅仅告诉你block是没有意义的，那什么有意义呢？那就是这个block在什么地址在什么位置这是有意义的，所以有了这个block之后，你还要知道block到底存在哪个机器上，那个机器肯定是DataNode中的一台，所以有了这两个映射以后相当于来了一个文件名，我就可以知道这个文件被哪些机器存储了，那这个时候这个client客户机就应该知道哪个DataNode上去读数据了。

那既然从哪读数据了也是不够的，因为client已经知道具体的DataNode了，那DataNode上其实就可以帮你能够找到这些block的存储的这个path。所以一个client进来了，它能够访问到具体的这个文件的数据就是通过以上所说的映射关系来拿到，所以NameNode记录了哪些文件包含哪些block，这个DataNode在这个NameNode里面只是作为一个value的形式存在。

图6：

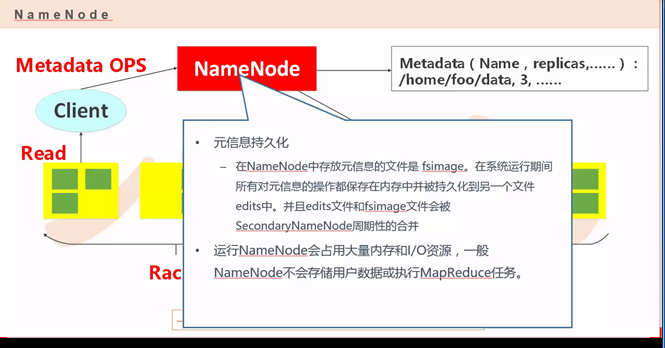


接下来看一下有哪些DataNode的注意事项，那我们之前说的这个NameNode在整个Hdfs上它是一个非常大的隐含，因为NameNode在全局里面它就有一个进程，如果是这个进程挂了的话那就相当于集群不提供服务了，那先不说挂了的事情，先说它挂之前的事情，那刚才说整个元数据都是存在内存当中，那你可以想象的出来如果是这个集群规模特别大的话，它挂的话，那么很多个DataNode节点上有很多个文件，那这文件的背后又有很多的block，你会发现这个数据是非常庞大的，那完全存在于NameNode内存里面肯定是吃不消的。

那一旦这个数据变大了之后，那一方面是它不能够持久化这一方面，另外一方面那他不可能永远庞大下去，所以它就制约着整个集群的规模的扩大，具体扩大这一块在hadoop1.0肯定是解决不了这样的问题，那只有在hadoop2.0才能去解决。

然后我们再看一下，如果仍然去用hadoop1.0的话，那就有一个问题，就是说如果一个hdfs里面存了一个文件和存了100个文件效果肯定是不一样的，对于NameNode内存的存储的数据量肯定是不一样的，那也就是说对于hdfs来说那肯定是存的数据量文件数据越少越好，但是文件数据越少越好并不代表你的数据你的信息变少了，有可能你的文件数据非常非常多，但是很多的文件都是非常非常小的文件，可能有几k几m，那对于这种几K几m的来说，虽然它远远与小于你的标准的64m这个单位，但是每一个文件它仍然独占一个block，也就是说你每一个数据可能占了几k，但是它仍然按照64m这个为单位这么一个容器在你的载体去存储，所以大部分的空间都是被浪费掉了，所以hadoop是更倾向于大文件这个一个原因。

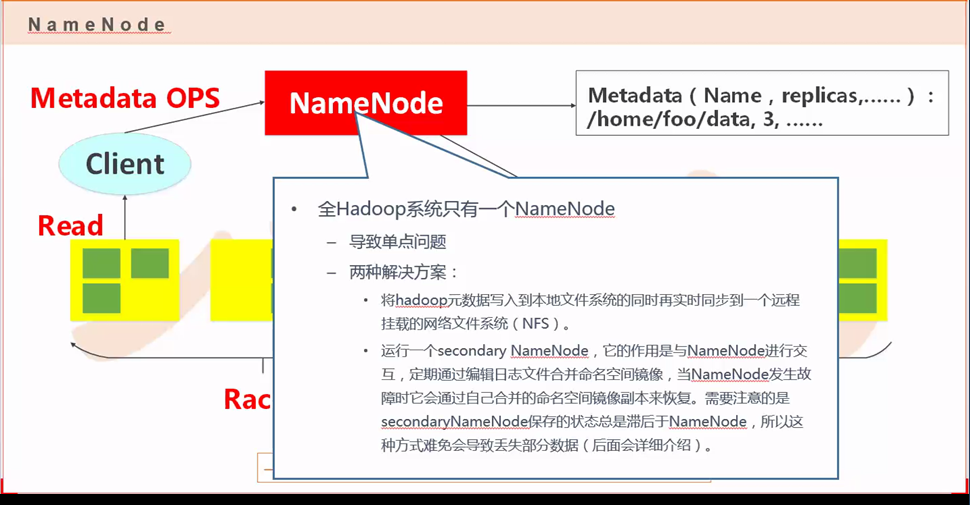
图7：



这个时候刚才我们说过了，这个NameNode它的信息是存在两个地方，一个是存在了内存里，这个内存里的数据可以直接对外进行服务的，还有一个是一模一样的数据它存在了本地磁盘里存了一份，但是这存在本地磁盘上的数据不会对外提供服务，它仅仅是做了一个数据备份，那这个数据备份这个名字叫做fsimage（如上图7所示），也就是说会在hdfs系统运行期间会对这个NameNode内存里的元信息会定期的保存到一个文件里面去，这个文件就叫做fsimage，那当然这个fsimage光靠这个文件也是不行的，那比如说fsimage这个文件已经比较大了，你再往里面去追加的话不太合适，所以它真正的实现是这样的它有一个fsimage这个文件是在某一个时刻的一个镜像，比如一天之前的镜像，那么我今天跑来好几个小时了，那这每一个小时的一些数据可能会被你持久化到edits logs一个文件里面去。

它只持久化那些已经修改的数据，如果你这样集群搭建完之后你可能一两天都不用，它也不会出来，因为你没有修改，然后这个时候你会发现时间一长的话，这个edits logs这个文件也是越来越多，这也是个问题，这时候就需要把edits logs里面的信息就要定期的往fsimage里面去做一个默值。就是说fsimage的默值可能比较久了，可能一年之前两年之前的了，但是edits里面的信息可以认为是实时更新的，其实它还是不是那么实时，可能他也就是会有一点的延迟，这个是无法避免的，那只能让edits logs尽可能的保证你这个内存当前的状态，那它就会慢慢的会往这个fsimage去合并，那这合并是由依赖于SecondaryNameNode去做的，所以它周期性的进行合并，它就是一个进程，这个进程就是不断的把这个edits去往fsimage去合并，然后这个NameNode我们都知道，它是做一个调度然后一个元数据一个存储，它不会存储真实的数据或者执行mapreduce任务，像fsimage这个文件的话你现在看是有一个，其实是有两个。

图8：

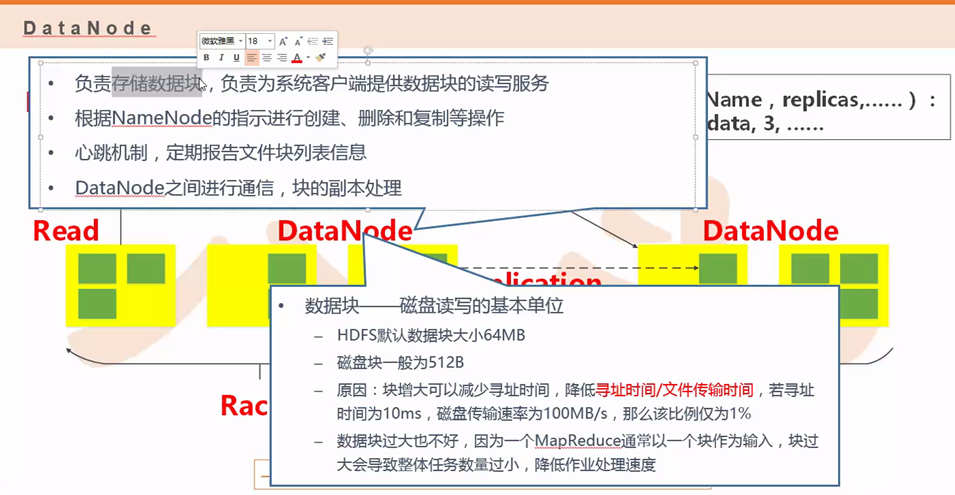


那刚刚也说过了NameNode有致命的缺点，缺点主要是来自于单点问题，在整个集群就这么一个NameNode，那对于这种问题的话也有一些解决的方案，那第一种解决的方案是举个例子：

比如说网络上的一个机器或者是多个节点上的机器然后组成这么一个文件系统一个空间，被你通过远程的网络的方式挂载到了你的电脑上的D盘上，那这个时候你以为这个盘是你本地的，其实你往D盘上去读写数据的时候根本这个数据是不存在你本地，而是存在了远程的某一个机器上或者是远程的某几个机器上面，就相当于这远程挂载的有一种文件系统，这个文件系统叫做NFS（如上图8所示），既然这个元数据它不是会定期的会往本地去存，那如果万一这个NameNode所在的这个机器突然挂掉了，那么挂掉以后这个机器的磁盘就被烧了，然后这个机器起不来了，那这个时候损失就大了，那么从这个点考虑的话看看能不能把这个数据尽可能的降低风险能够存到其他的机器上面去，即使本机挂掉了没关系，但你数据是存在了你的网络里，也就是在网络上有某一台机器或者是某几个机器的一个文件系统这么个空间，所以它数据不会丢，这是一个解决方案，但是这个解决方案你会发现还不行，因为NameNode毕竟是内存最真实的，而再怎么存你的离线数据还是有延迟的。

还有一个解决方案，虽然也不彻底，但是从hadoop1.0以上看，还是不错，就是secondary NameNode，刚才也说了这个secondary NameNode会把NameNode里面的这个镜像会做一个定期的合并，那这个编辑日志就是刚刚的讲的那个edits文件，然后合并命名镜像，这个镜像就是fsimage，那么目的就是恢复不起来的时候尽可能保留更多的信息。

图9：

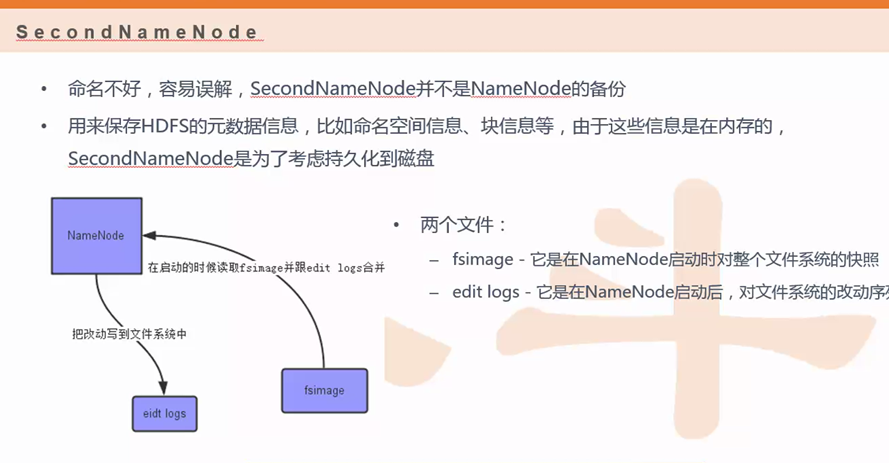


那么我们再来看看DataNode，DataNode也比较好理解，它主要负责数据的存储（如上图9所示）

那所有的数据块都存在于你DataNode这个节点上。然后并且为系统客户端（client）提供一些数据的读写服务，那么也就是说数据读写直接跟DataNode进行交互，那另外一个毕竟DataNode在一家公司里面是一个小员工，那么这个NameNode是老板，那你毕竟要听老板的话，所以DataNode要听从NameNode的一些指示。并且DataNode要及时的上报给NameNode告诉NameNode当前自己的一个状态。DataNode假如马上要挂掉，那有可能网络对外传输心跳的时候就不会那么保证每三秒发送一次自己的状态给NameNode了，这个时候可能心跳就有些抖，那这个时候NameNode就可能会根据这个心跳不稳定就可能会判断这台DataNode挂掉了，从整个集群里面把DataNode这个节点就移除掉了，另外一个如果你的DataNode上的数据的空间已经达到了一定状态的话，那NameNode也不会给你去分更多新的数据去做存储，所以要心跳机制去保证的。

那么还有一个我们刚说了如果有一个DataNode挂掉了，导致了整个这个集群上这个块的副本就不平衡了，那你就需要在DataNode之间建立一个通信，然后进行一些个副本的拷贝，当然除了你这个机器挂了导致你这个副本拷贝失败，当然你这个数据洗的时候肯定也要做一个副本拷贝。

图10：



SecondNameNode命名空间是非常不好的，由于这个命名很容易遭到误解。

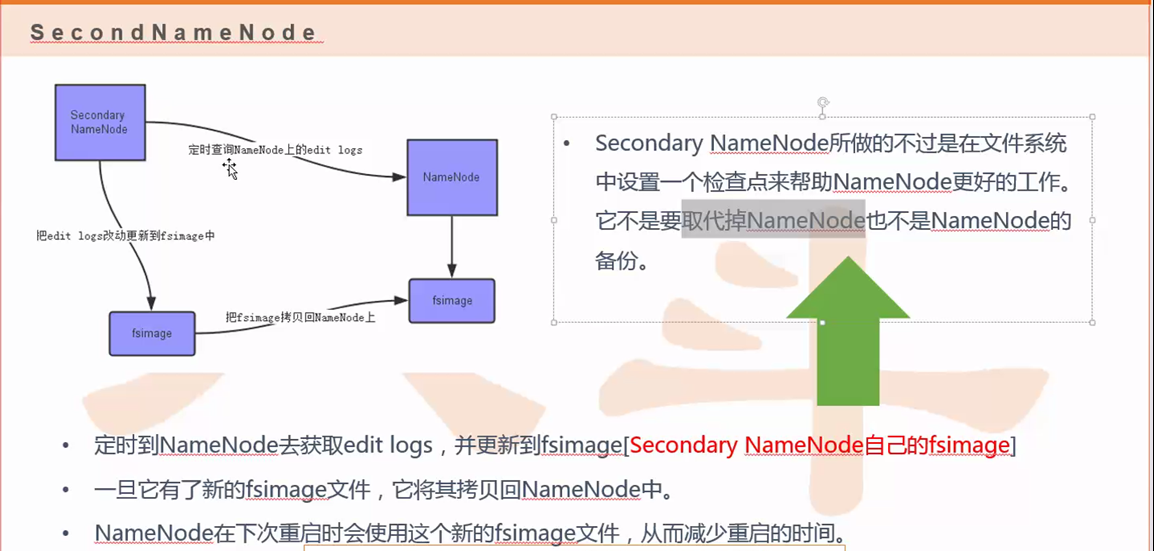
NameNode主要存储与HDFS的元数据，当运行的时候这些信息是存在内存里面的，但是这些信息也可以到磁盘，那可以看到如上图10里面的那张图。

这张图里面有一个NameNode，然后有一个eidt logs和fsimage，当这个NameNode重启的时候内存是空的，那么NameNode在重启的时候需要把内存里面要还原出那些元数据的那些内存的状态，那这个状态这个数据是来自fsimage的，当这个NameNode重启的时候先读fsimage，并且跟eidt logs做一个合并，相当于NameNode内存里面既包含了fsimage一部分信息，因为它这个数据fsimage你可以认为是一个标量数据也可以认为是一个基准数据，然后eidt logs相当于是在fsimage之后的不断的增量，那也就是说它需要把fsimage和eidt logs这两个数据全部加载之后，NameNode里面的这个内存空间它才是一个最完整的一个状态，然后相当于NameNode重启之后在eidt logs里面的数据才会与这个fsimage进行合并，从而通过这样的合并的fsimage，就会得到最新的快照，但是有一个问题，这个NameNode它是很少重启，意味着你的NameNode运行了很长的时间之后你会发现eidt的文件个数特别特别多，并且总体来说这个量比较大，因为你时间一长，中间有很多的改动都会通过这样的你可以认为是溢出来的数据它就会记录这段时间变化的数据，那随着时间变成，肯定是数据越来越大。所以这里面会引发了这么几个问题，那第一个问题就是说这个eidt文件变大了，那怎么去管理这些文件也是一种挑战。另外一个就是文件时间又多，文件个数又大那构建时间又重启，导致NameNode重启的时间会花费很多，主要是因为加载数据量较大，另外一个是刚刚说过了你要重启的时候呢你需要fsimage去做一个合并，那你这个eidt有很多并且这个文件比较大，那合并的时候也是比较耗时间的，那么另外一个就是说如果你这个NameNode挂了的话，那么我们肯定会丢失很多的一些改动，所以这点来说fsimage是永远你也赶不上NameNode的最新状态，因为fsimage这个文件是比较旧的。

所以为了避免这样的一种情况，我们就需要一个机制，来尽可能想办法怎样减小你的eidt，并且让

fsimage这个文件能够尽可能的保持最新的数据状态，这样的话NameNode重启的话就不需要有合并这样的影响了，他直接就可以从fsimage直接读数据然后启动。所以呢这个事情就有SecondNameNode来干的，具体怎么干的呢？请看下图11

图11：



其实也比较容易，SecondNameNode它叫一个助手节点也可以叫它检查点。

那也就是说在关于NameNode什么时候改动，就是写到你的eidt logs里面，如果这个NameNode它一旦有这个信息的改动的话，那最理想的情况下我应该把该数据eidt logs就应该实时的查出来，然后放到edit里面，那NameNode什么时候改动会放到edit里面，就是谁触发了这个改动，肯定是DataNode来触发这个改动，如果要是说DataNode变化的话肯定NameNode里面的信息也会随着变化，因为你DataNode会发心跳，相当于所有的这个改变的背后的主谋都是DataNode来搞的，是由DataNode写操作来触发NameNode里面的内存中的变化，所以最终的根本原因是DataNode，那也就是说当往这个DataNode上去写数据的时候，那DataNode会自动的与这个NameNode进行通信，会告诉NameNode什么文件第几个block放在哪里，那这时候NameNode会把这些信息更新到内存里面的元数据，然后它一旦更新完毕之后会往它的磁盘上会写一个eidt logs。

然后这时候SecondNameNode也不是闲着的，他一旦发现了NameNode的eidt logs发生改动了发生变化了，那么此时SecondNameNode作为一个进程来说就可以把这个发生变化的信息直接吸收过来，NameNode这边出了哪些eidt logs信息，然后SecondNameNode就读NameNode出的那些eidt logs信息，读过来之后刚刚也说了fsimage有两个，但是真正让NameNode加载的只有一个fsimage，只不过另外一个fsimage是一个可以认为的一个备份的fsimage，那么也就是说SecondNameNode看看NameNode的eidt logs有没有更新，如果更新了的话，就会把更新的eidt logs同步到备份的fsimage，这个备份的fsimage是由SecondNameNode来写的，所以你会发现由SecondNameNode来写的fsimage会永远比NameNode加载出来的fsimage要新，如果要是说在NameNode还有挂掉之前，能够及时的把备份的fsimage这个文件能够拷贝到NameNode这个节点上加载原先的fsimage文件并且能够替代它，相当于再重启的时候就不需要从特别特别久远的某一个时间点再做一些个数据恢复，这样会加快元数据的一个空间效率，所以SecondNameNode就是这么个思想。