





































































































3616118202

之前给大家讲解Namenode启动流程的时候，非常重要的一个组件，NameNodeRpcServer就是Namenode上接收和处理rpc请求的这么一个关键的入口，按照场景驱动来研究源码，你会发现，你看源码的顺序基本上跟一个集群启动 -> 你使用这个集群，是一样的，所以我们的源码分析，你会发现是一环扣一环的

NameNodeRpcServer他只是一个非常简单的承接rpc接口的组件，其实核心的元数据维护和管理的业务逻辑，都是交给FSNamesystem去处理的

元数据管理的这个流程，我们来通过创建目录这个场景来驱动，画一个大的图，通过场景来驱动，我们其实就可以看到元数据是如何管理的

上一讲已经给大家讲清楚了内存双缓冲的机制，也就是说edits log写入磁盘的时候，是基于内存双缓冲来玩儿的，结合人家的分段加锁的机制，给大家来讲一下，结合内存双缓冲，是如何提升写edits log的并发能力的

synchronized，全部是在纯内存操作，所以虽然说加锁了，但是多线程并发的时候，其实是没问题的，应该是很快的

如果是比如某个比较小的txid在执行flush操作，那么后面来的比较大的txid的线程，就会while true等待，等人家flush完了，他再来flush一次

如果是某个比较大的txid在执行flush操作，那么后面来的比较小的txid的线程，就会直接return掉，因为那个比较大的txid的线程一定会一次性将多个比较小的txid的edits log都一次性flush到磁盘里去，不需要重复flush了

/usr/warehosue/hive/access\_2018\_10\_25.log

/usr/warehosue/hive/access\_2018\_10\_24.log

从根目录触发，/目录，INodeDirectory，children列表就包含了/warehouse目录，/hive -> 两个access日志文件

人家都会通过FSEditlog这个组件去记录一些操作日志，文件目录树的操作、一些参数的设置、机制的开关，吐槽一下，hdfs源码分析的书籍的话，作者绝对是看懂了源码的，作者基本上都是对源码理解的非常透彻了

但是问题在于说，那些书籍协作的顺序，可能就是说FSEditlog以后，人家就会对这个类里每个方法，一个方法一个方法的分析，没有场景驱动的话，你楞看那些源码，真的是很痛苦的，看那个书，压根儿就看不下去

FSEditlog这个组件在本地磁盘上记录日志的时候，transactionId概念，他是基于transactionId将edtis log不是一个大文件，是分成多个文件来进行存储的，包括fsimage也是如此，fsimage文件也会跟transactionId是有关系的

transactionId，比如说你现在创建了一个目录，这是一个更新元数据的操作，这个操作会被分配一个全局唯一的transactionId，是全局递增的，从1开始，每次操作都会绑定一个transactionId

mkdirs()，你这个方法调用一次，更新了一次元数据，就可以认为是一次transaction，会有一个对应的transactionId

我们之前把hdfs集群启动起来以后，我们其实去namenode上的磁盘文件中，看过元数据的文件的存放，实地看过edits log和fsimage是怎么来存放的

我干脆把虚拟机集群跑起来，反正启动一个集群时间花的也不多，我先暂停一下，带着大家再去namenode的元数据目录下，查看一下人家是如何存放editslog和fsimage的

/home/apps/hadoop/tmp/dfs/name/current，可以看到namenode所有的元数据

目前最新的一个fsimage文件，只不过包含了transactionId = 1729的元数据。但是最新的一个edits log的文件，已经到了transactionId = 1761了。很明显可以看到，fsimage是落后于edits log文件的

如果此时你重启namenode，按照我们之前的一个理解的话，应该是将fsimage\_1729和edits\_1761进行合并，将1729以后的transaction都合并到fsimage里去，最后会出来一个新的fsimage文件，fsimage\_1761

然后呢，edits log应该是从edits\_inprogress\_1762开始继续写

在那个EditLogOutputStream里面，其实是可以封装多个流的，主要是在初始化的时候，人家一定会搞一个JournalSet，FileJournalManager（负责写本地磁盘），QuorumJournalManager（负责写journalnodes）

你会基于JournalSet搞一个EditLogOutputStream出来，然后这个东西底层就封装了多个流，你对这个东西调用write()方法的时候，他其实会在底层遍历所有的流，依次调用这些流，而且这些流，他都是先写入内存缓冲的

然后在内存缓冲都写完了之后，会有另外一个单独的方法，来将内存缓冲中的数据刷入磁盘，或者是刷入网络发送到journal node去

standby namenode他从journalnode拉取edits log的核心组件，就是EditLogTailer，这个组件其实就是在standby namenode后台运行的一个线程，不断的从active namenode拉取edits log数据流，应用到自己本地

两个namenode之间的元数据是保持同步的

你看明白了EditLogFileOutputStream以后，双缓冲的机制，你再看他是如何写入JournalNode，其实这个原理几乎是差不多的，journalnode跟namenode、datanode一样，一定是有对应的rpc server

FSDirectory，这个组件其实就是维护内存中的文件目录树的一个核心组件

/usr/warehouse/finance/data

/usr/hbase/data

/usr/warehouse/crm/data

/usr

-> /warehouse

-> finance

-> data

-> crm

-> data

-> /hbase

-> /data

这个任何一个文件系统，都是有根目录的，就比如说/usr/warehouse/finance/data里面的根目录，就是第一个/

ZKFC是如何监控到active namenode死掉的

HA这块的源码，初步的流程来看一下，我们不会对这块源码细节过多的深扣

场景驱动来学习hadoop这种大数据系统的源码

人家最早考虑如何写各个模块的源码的时候，其实肯定也是场景驱动，他首先要考虑自己这个系统要实现个什么功能，然后才会在源码里加入一段一段的代码

分析源码的时候，也得场景驱动，如果你去看一些源码分析的书籍，都是按照一个模块一个模块，一个类一个类的来分析，那个完全不符合场景，很难看懂的。如果你自己去瞎看这个源码，拿到源码之后瞎翻翻

场景驱动

1、场景1：hdfs集群启动

（1）namenode初始化和启动流程

（2）datanode初始化和启动流程

（3）datanode向namenode发起节点注册

（4）datanode定期向namenode发送心跳通知

（5）namenode定时检查datanode是否按时发送心跳

一个空的集群，什么数据都没有，然后他就ready了，然后就可以提供操作，你可以使用hdfs shell命令，或者是用hdfs java api

2、场景2：创建目录

hdfs他其实是一个分布式文件系统，本质是一个文件系统，分布式的而已，你可以把他当做linux文件系统一样来操作都可以，创建目录，创建文件

一定是，创建目录，你需要在hdfs上创建一些目录，然后才能在目录里存放那种超大文件数据集，比如你有一个大文件，达到了1TB，然后你可以把这个文件上传到某个目录里去，人家给你将1TB大文件，拆分成多个block（每个block是一个小文件），分布式存储在多个datanode上

（1）hdfs shell或者java api，如何跟namenode通信

（2）namenode是如何更新和维护内存中的文件目录树的

（3）namenode是如何同时将edits log写入本地磁盘文件以及JournalNode中

（4）JournalNode是如何在本地磁盘上存储edits log

（5）standby namenode是如何从JournalNode拉取edits log流的，以及如何更新到自己的内存文件目录树中，包括写入自己本地磁盘文件中

（6）standby namenode是如何定期发起checkpoint的

（7）namenode每次启动的时候是如何读取fsimage和edits log文件完成内存中的合并的

（8）active namenode如果宕机，是如何由standby namenode完成主备切换的

=> 基于HA双namenode架构，hdfs的元数据是如何管理的

=> HA双namenode架构下，是如何在故障的时候完成主备切换的。前提是搞清楚元数据是如何在双namenode之间管理的，你才知道两个namenode元数据是如何实时保持同步的，才有了说，active namenode故障，standby可以主备切换

======================================================================

hdfs shell命令，或者是java api，来发起一个创建目录的请求，此时首先第一站，就是hdfs shell或者是java api，都会在客户端执行一段DFSClient的代码逻辑，DFSClient就是hdfs客户端代码，他会负责跟namenode、datanode进行通信，完成你的客户端发起的一些请求和操作，所以说，先来看一下DFSClient在干什么

如果你是用java api的话，他其实底层默认是会用反射，创建一个默认的FileSystem的具体的实现类的实例对象出来，供你来使用，你使用的时候，就可以执行各种目录操作，文件操作，等等

默认的实现，就是会创建一个DistributedFileSystem实例，作为FileSystem的实现，来提供hdfs的操作入口，他是FileSystem的子类，一般你在java代码里都是用这个类来操作hdfs的，底层基本都是调用DFSClient的

DFSClient底层是走的rpc代理的请求，ClientProtocol的代理对象，向namenode发起一个网络请求，调用人家的server端接口，完成目录的创建

我们这一讲，已经基本明白了，FSEditlog是如何按照transaction机制，分段存储edits log的，源码里，他到底是如何去存储的呢？

如果active namenode重启，重新启动的话，此时他会将fsimage文件 + 上次checkpoint之后的edits log都加载到内存里，进行合并，再写一份新的fsimage文件，打开一个新的edits log文件供后续写入

读取fsimage和edits log进行合并，琐碎代码。写新的fsimage，清理旧的edits log，打开新的edits log

为什么他要那样来做呢？因为很简单，如果每个namenode自己本地都有edits log的话，那么他们启动的时候，就直接从自己本地加载fsimage和之后的edits log就可以完成合并，以及启动

但是在hadoop有的版本里，hadoop 2.6里面，我们看到的这个源码里面，他其实active namenode在写edits log的时候，是不落本地磁盘的，是直接发给journal nodes的，standby namenode读取edits log仅仅在内存里应用，不落本地磁盘的

在我们看的这个源码的版本里面，他其实是两个namenode都是不落本地磁盘的，他们本地都是没有edtis log文件的，只有journal node上有的

在namenode启动的时候，加载edits文件的时候，就会从journal node通过网络去加载，加载到需要的edis log就在内存中跟fsimage合并。但是大家想一下，为什么说，我们在后来自己部署的hadoop 2.9.1版本中，看到两个namenode上又有edits log文件了呢？

很有可能人家后来发现在namenode启动的时候，最好还是从自己本地磁盘文件来读取edits log比较快，比通过网络请求从journal nodes加载要更快，如果说大家想要印证这一点的话，有兴趣可以自己去查看一下hadoop 2.9.1版本的源码

我们看的这个版本的源码，hadoop 2.6.x的源码，人家就是说直接edits log写入到journal nodes去，namenode启动的时候，是从journal nodes上通过http加载需要的edits log然后直接应用到内存的FSDirectory应用到内存目录树里去

一定要明白，不同的版本，其实集群的表现可能是不一样的，所以说hadoop 2.9.1，稍微有点不一样，但是呢大体是一样的，原理是没变的，active namenode和standby namenode两个，都是有落地edits log文件的

推测：namenode启动不需要通过网络从journal nodes加载edits log文件，journal nodes仅仅是用于在两个namenode之间同步edits log

观察一下active namenode、journal nodes、standby namenode的edits log的变化

很明显的一点就是说，我们如果执行了一个元数据操作以后，hadoop 2.9.1版本，一定是active namenode会写一份到自己本地磁盘，同时也会写一份到journal nodes去，当然核心原理是差不多的

每个版本或多或少都有一些差别的，所以你不用太过于顾虑，你出去面试，跟人家说，我看的是hadoop 2.6版本的源码就可以了，我们这个课程最后会带着大家来看一下hadoop 2.7、2.8、2.9的源码变迁，看看后面源码是如何变化的

全局、唯一、递增的transactionId是如何分配出来的呢？

edits log人家已经绑定了一个全局唯一的transactionId

人家就是要同时将edits log一个是写入自己namenode本地的内存缓冲中，这个是开源的大数据系统，写大量数据到磁盘的时候，都会用内存buffer缓冲的机制，先写内存，写到一定的程度，再一次性的将内存里的数据刷入磁盘中

人家会同时将数据写到journal node中去

人家是如何用一行代码，实现了一个效果：将edits log写到了两个地方去呢？

推测，是在namenode启动的时候，FSImage加载磁盘文件上的fsimage和edtis log进行合并，合并完了以后，重新fsimage写回磁盘，然后开启一个新的edtis log segment，在开启这个信念的edits log segment的时候，应该就是会默认初始化那个EditLogOutputStream

基本上来说，可以把hdfs当做是linux上的文件系统来操作，命令基本上都是差不多的

在hdfs上创建目录，hadoop fs -mkdir -p /user/dir01，创建一个目录层级结构

hadoop fs -rmr，删除目录，mv、cp

linux上是差不多的，hadoop fs -put可以上传本地的大文件到hdfs上去，hadoop fs -get可以将hdfs上的文件给下载下来

形成一系列的文件系统的元数据

目录层级结构，以及目录里有哪些文件，文件 -> block -> 在哪个datanode上面

元数据，是放在namenode上面

每个大文件实际的内容，会被拆分为多个block，放到各个datanode上面去分布式存储

hdfs说白了支持的数据结构，不是mysql那种关系型数据库的库表结构，他支持的是文件系统的层级结构，就是目录-子目录-文件的这种形式。当然其实hadoop生态必然是可以支持类似关系型数据库的库表结构的，但是那是基于hdfs的一个开源项目，hive，干的事儿了，他是针对大数据的数据仓库技术

所以说，基于hdfs就可以创建目录啊，创建文件啊，对文件读写数据啊，然后移动文件啊，删除文件啊，重命名文件啊，balabala的，你能在你win10电脑上的文件的操作，大部分都能在hdfs上干

而且人家hdfs还支持对文件进行quotas就是配额的限制，类似于说限制你某个目录最多只能占用多少磁盘空间了，就这个意思；还支持对文件的权限操作，比如说针对某个用户，对某个目录，可以读，但是不能写，这个文件权限后面讲吧，其实跟linux的文件权限是类似的

所以hdfs支持的文件系统的操作还是蛮丰富的，但是比如linux的一些hard link、soft link之类的概念是不支持的

然后呢，这一整套文件系统相关的元数据，就是目录层级结构，文件，文件和block对应关系，block和datanode对应关系，还有别的文件的一些quotas和权限之类的各种东东，都是存放在namenode里的，这就是所谓的文件系统元数据，英文是：filesystem metadata

你反正对文件的一些比如创建目录拉、删除文件了、重命名文件了，之类的涉及到元数据的操作，都是交给namenode来干的。那么你找人家namenode修改元数据的时候，人家是怎么管理元数据的呢

namenode里有一个东西叫做EditLog，就是编辑操作日志的概念，你比如说创建个目录，那么人家namenode必然往这个edits log里写入一条日志，然后你创建个文件，人家还会再写入一条日志，这个edit log是放在磁盘上的一个日志文件。然后呢，比如整个文件目录组织结构，以及block、datanode的映射关系啊，这些东西存储在FsImage文件里，这个fsimage也是放在磁盘上的文件

然后呢，namenode除了在fsimage文件里存放元数据，还会在内存里保存一份儿，要不然这种元数据，动不动读写文件，不是害死人了，性能绝对是很差的。然后呢，namenode每隔一段时间（这个时间间隔是自己配置的，一个threshold），就会读取磁盘里的edits log出来，全部应用到内存里的fsimage缓存里去，然后将fsimage重新写一份到磁盘里去，接着将edits log给给清空掉

这个操作叫做checkpoint操作，这个checkpoint的时间间隔自己可以配置的。然后在namenode启动的时候，人家也是会从磁盘上读取edits log和fsimage在内存里构造一份缓存数据的

这个checkpoint可以自己配置的，dfs.namenode.checkpoint.period，这个参数配置几秒钟执行一次checkpoint，还有一个，dfs.namenode.checkpoint.txns，这个参数配置当edits log里有多少条数据的时候，就执行一次checkpoint

大数据，一定要给大家从0开始入门的同学，理清楚一些概念，我的讲法，以及课程的编排是跟其他的培训机构，4个月，5个月，从0转型到大数据的一些课程

自己研发类似于hdfs、mapreduce、yarn的一些分布式系统，来实现大数据的分布式存储、分布式计算、分布式资源调度

离线计算，实时计算

几十亿条数据，算出来一张报表，可能就几十条数据而已

用户的每一次操作。。。。

用户的每一次操作。。。。

用户的每一次操作。。。。

。。。3万5千行数据

算出来的报表，iphone 10被点击了35000次

分布式流式计算系统，来一条日志，就计算一条

hdfs，分布式存储

第一个作用，在离线计算里，作为基础的分布式数据存储，供每天凌晨批量计算昨天的数据

第二个作用，在实时计算里，作为基础的分布式数据存储，为hbase分布式NoSQL数据库提供支持，支持实时计算

就是一个分布式存储系统，主要的作用还是在于离线计算里面

所以聊到这里，就有一个问题了，你想啊，比如说一个文件拆分成了10个block，分布在10台机器上，每个机器存储1个block，看起来还挺美的，但是问题在于说，如果某台机器故障了呢？那台机器啥故障都可能有，比如说啊，常见的，磁盘故障，网络故障，内存耗尽，CPU耗尽，或者干脆机器自己宕机了，防不上防啊

所以呢，如果说一个文件少了一个block，就9个block了，那数据不完整了，坑爹，这样人家读取文件都没法弄了，咋整呢？所以为了对集群中的各种机器的故障进行容错，有一个关键的机制，就是副本机制

所以说呢，你可以设置一个关键性的参数，叫做replication factor，号称是叫做复制因子，其实谁知道什么鬼因子的，说白了就是你的每个block要复制几份副本到其他的机器上去，那么如果某台机器挂了，这样好了啊，其他机器上有一模一样的block副本。这个replication factor可以整体设置一下，也可以对每个文件设置一下，然后后续还可以修改

那么在写文件的时候，假如说默认的每个block就是3副本，此时namenode会先根据一个复制算法挑选出来3个datanode，每个datanode放一个block，返回给客户端了。客户端先第一个datanode写入一个block，接着datanode将这个block复制给第二个datanode，然后第二个datanode再将block复制给第三个datanode。

rack aware

namenode在分配block到datanode时候，有一个关键性的机制，叫做机架感知特性，这啥意思呢？就是hdfs集群里的机器与机器之间不是肯定是要进行通信的么，然后一个机架上的机器之间通信速度，要比不同机架上的机器之间的通信速度快的多了。然后namenode比如默认一般一个block是3个副本，你就可以把2个副本放在一个机架上，然后第3个副本放在另外一个机架上。

这样的话，在一个机架里，同步复制2个副本，通信速度很好，仅仅只有一个副本是同步到另外一个机架上去的，确实会影响一点写的速度，但是总比你在三个机架上各方一个副本好吧；然后如果一个机架完全挂了，还有另外一个机架上有一个副本。

这个namenode呢，每隔一段时间就会从各个datanode那里获取一个heartbeat，这是用来确定那个datanode还活着的，所以叫做心跳；还有就是获取一份block report，就是每个datanode报告自己本地可用的block有哪些

这样的话，namenode就可以不断的知道整个集群中的block的情况啦，然后在新创建文件分配block给datanode的时候，不就可以根据各个datanode当前的block数量来均匀的分配了么

在读数据的时候，会优先找离自己最近的那个副本所在的机器，保证读取性能最高

因为hdfs其实本质说白了就是个分布式文件系统，那所以你得考虑一下，比如跟fastdfs之类的java架构领域的一些分布式文件系统相比，他的特点和优势在哪儿呢，下面就是说一下hdfs作者对他的整体架构的一些设计理念

（1）支持超大数据集

hdfs这种分布式存储，其实为啥要分布式存储呢？就是因为数据量太大了，比如说某个表，有30亿数据，你难道存在mysql里？就一台机器里？所以用了hdfs分布式存储，就是可以放在N多台机器上，每台机器放这个大数据集的一部分，比如就放300万条数据。

所以hdfs定位就是针对这种超大数据集的。

（2）绝对能够应对硬件的故障

大数据的理念，是说不要用那种商用的小型机，服务器，商用的存储设备

一遍来说大数据的系统都是部署在普通的机器上面的，一遍来说很有可能就是16核64G的物理机

因为就目前国内的大数据整体情况来看，一般那种外包类的大数据公司，给客户做个项目，大概也就是几十台机器，甚至很多小公司做所谓的大数据，可能就十几台机器；然后如果是中等规模的互联网公司，一般会是上百台机器的规模；如果是那种大型互联网公司，那么上千台机器规模就是很正常的了，甚至是几万台机器

但是一般大数据用的都是普通的机器，比如说那种16核64G的机器，一般不会用那种性能超级高的、稳定性巨好的服务器。所以说这种普通机器其实出故障的概率还是蛮高的，比如说磁盘故障，突然不能读写了，或者是网络故障

那hdfs第一个设计理念，就是可以自动探查到集群中某一台机器故障了，然后可以自动对故障进行恢复，而且速度要比较快一些，比1个小时才检测到故障

（3）流式数据处理

这个啥意思呢，就是说hdfs读写文件系统上的数据的时候，是基于流的一种概念来的，英文名词是：streaming access。你现在先别纠结这个是啥意思了，死记硬背就ok了，然后记住，hdfs用这个所谓的流式数据处理，其实主要就是为了保证高吞吐量的文件读写，而不是低延迟的文件读写。

hdfs是用在离线批处理场景的，尤其是数据仓库，数据分析这块。今天凌晨把昨天所有的数据都给在比如半小时内处理完毕。而不是数据来一条你就算一条。

分布式系统，相当于就是你自己用Java开发出来的系统，jvm进程

（4）简化的数据一致性模型

同时支持对文件的写和读，很麻烦的，大量的并发冲突问题

因为这个hdfs是为了支持超大数据集，分布式存储，离线批量处理的，所以说，他的数据一致性模型是简化的，在他这里的话，一个文件只能一次写入，然后之后就只能追加，不能随便改之前的数据了

他的理念就是，write-once，ready-many-times，一次写，然后多次读，这样就没有数据读写并发冲突，以及数据如何维护一致性的问题了。

（5）尽量移动计算，但是不要移动数据

这个意思就是说，如果你要对分布在多台机器上的数据，进行分布式计算，使用比如mapreduce或者是spark都可以，那么此时尽可能让你的计算任务是靠近这个数据，而不是说在集群里通过网络胡乱传输数据，那样会导致性能极差极差

移动数据

hdfs这样的一个架构下，集群如何保障某个节点故障时候的容错性呢？

（1）集群节点故障

比如说第一种故障情况：网络分区。啥叫做网络分区？英文就是network partition，说白了就是集群里网络故障了，一部分datanode跟namenode无法痛点了，此时网络环境不就相当于是分成了两块儿了，这就是所谓的网络分区。s

因为master-slave架构的分布式系统，一般都会设计心跳机制，就是datanode会定时发送心跳以及block report到namenode去，那如果网络分区了，namenode肯定会感知到的，因为一部分datanode心跳没发送过来了。

这个时候namenode就会将这些无法发送心跳的datanode标记为dead状态，已经死掉了，然后就不会再让hdfs客户端去读写那些datanode了。默认是10min接收不到心跳才会标记datanode死掉了。而且这个时候datanode上的一些block不就不可用了么？这个时候namenode会检测到，然后会发现一些block的replica副本就不够了，那么此时namenode就会让其他的datanode去复制一些replica保证3副本。

除了这种网络分区以外，还有别的一些故障，比如说datanode所在机器宕机了，或者datanode进程就挂了，或者是那个block对应的文件损坏了，都会让namenode感知到，此时namenode会自动在集群里复制block，保证每个block的三副本。

（2）数据破损

此外还有一种机制，就是hdfs的数据完整性校验机制，在一个客户端上传一个文件到hdfs的时候，其实是会基于文件内容算一个校验和出来的，就是checksum，放到一个隐藏文件里去，也是在hdfs里的。

然后在读取文件内容的时候，会对读取到的文件内容重新算一个校验和，与之前上传时的校验和比对一下，如果不一样说明文件破损了，此时他会尝试对某个block读取其他的副本。

（3）元数据文件损坏

fsimage和edits log都是非常关键的元数据，如果这些文件损坏了，那么hdfs可能就无法正常工作了，如果要保护这个namenode的可用性，可以使用namenode HA部署双机进行热备，出现故障自动切换namenode。

为我们后续深入的研究hdfs的源码打下了坚实的基础，如果你脑子里没有这样的一张图的话，你后面根本没法看懂源码的

我在这个基础的原理图里面，有一些阐述和论断，不一定是完全的精准的，

从0入门开始，定位是，只要你有比较不错的java基础，就一定能够听得懂的大数据课程，从0开始3616118202

2个场景：

（1）互联网公司，电商公司，假设类似于京东，在互联网上售卖商品，人家老板，有一个需求，咱们这个公司每天都在不断的运行着，我现在就好奇，我想看看，咱们这个网站售卖的最好的商品是哪些，每天的售卖的数量是多少

（2）传统行业，电信公司，人家的高管，高层，我现在就想看看每年一共会有多少人打出多少分钟的电话，发送多少条短信，消耗多大GB的流量

2014年以前，其实从只要你有电脑开始，有互联网开始，上面这两种需求就都是存在的

只要你有了互联网，人家互联网网站一定会不断的运行，新闻门户网站，社交网站，老板总是要看一些数据的，每个版块每天被访问的次数，最热门的10篇文章分别被访问多少次，社交网站，每天活跃进行聊天的用户有多少个，平均每天耗费在我们网站上的时间是多少分钟

只要从你有电脑开始，软件，90年代开始，各行各业，传统行业，他们都是开始电脑化办公，无纸化办公，软件，OA系统、CRM系统、财务系统、进销存系统、工厂管理系统，皮革厂、电力局、电信公司、银行，就会积累一定的业务数据，传统行业的老板也会要看一些数据的

皮革厂老板，我想看一下财务数据，每天大概有多少订单，每年的旺季是什么时候；电力局，我们要统计一下，全国各地，哪些地方的用电量是最大的，每个省每天的用电量分别是多少，数据

在各行各业里面，只要有互联网、软件，中高层，老板，都要看数据，看数据才了解他的这个公司整体的运行的情况，看数据，最最主要的就是帮助公司里的各个层次的人了解到这个公司运行发展的方方面面的情况

从而让公司的管理者可以更好的去了解自己的公司，管理自己的公司

互联网行业，传统行业的软件系统，他们的数据都存放在哪里？2014年以前，2010年~2012年左右，当时最火的不是什么所谓的大数据，当时DBA，Oracle DBA，当时公认的都是Oracle是数据库行业的老大

无论是互联网，还是传统行业，数据都存放在oracle里面，mysql

SSH框架（Struts2 + Spring 3 + Hibernate + java系统）+ oracle，搞定，java工程师，搞定这套东西，就可以拿个月薪几k了，四五千，技术发展还是比较low的

数据报表从哪儿出？

从oracle里出了，2014年以前，2013年以前，2010年以前，在国内，oracle出报表，mysql出报表，很正常的一个事儿

当时就是直接写一个巨复杂的大SQL，几百行，几千行，直接查库，写一个小系统，定时调度，一下，每天凌晨的时候，系统没人用，跑一个大SQL在oracle上面，耗时几十分钟，甚至一两个小时，跑成功一个大SQL

产出一张报表的数据

第二天，老板过来，就可以看报表

然后这个是指的是数据量还相对大一些，比如库里存放了几十万数据，几百万数据，跑SQL很慢的

如果你的数据量很小，几千条，几万条，随时随地，只要老板要看报表，直接点个按钮，然后直接现场一条几百行的大SQL跑到oracle里去查询出来一张报表，可能稍微慢一点点，大概耗费个几十秒，几分钟

分布式系统其实是有很多架构的，最经典的一种就是master-slave模式的，hdfs作为一种分布式的数据存储系统，其实就是用的master-slave模式的分布式系统架构

在hdfs的架构里，有一个namenode进程（吐槽一点点，很多培训机构，在给入门的同学讲解hdfs课程的时候，namenode，懂java的，namenode也是一个系统，是一个分布式系统的组成结构，启动，jvm进程），这个你可以认为就是个master，就是主人的意思了，在一个普通的hdfs集群架构里，namenode记住只有一个，这个玩意儿你可以认为是整个集群的指挥中心

还有一个进程，叫做datanode进程，每台机器上都有一个datanode进程，负责对这台机器上的数据进行存储的

namenode里面主要放的是所谓的文件系统命名空间，filesystem namespace，很多培训机构就这么干讲，其实根本说不透，说白了，这个东西就是举个例子啊，你在hdfs里的数据都是以目录和文件的形式来组织的

hdfs，分布式文件存储系统，存放的数据的格式都是目录->文件，文件里存放了几百GB的数据

然后呢，你肯定更有很多目录，每个目录下面有子目录，还有很多的文件，对不对啊，就是这个意思了，比如说包含/a，/b，/c三个根目录，/a下面还有/a/hello，/a/hi，之类的子目录，然后每个目录里还有/a/hello/ddd.txt这种文件

所以这套文件系统的目录的层级结构和文件对应关系，就是所谓的filesystem namespace了，然后这套东西就是放在namenode里的

1GB，hdfs分布式存储，把这个文件，拆分成好几个block，每个block是128mb，8个block，尽可能的将8个block放在不同的机器上面

然后还有一个角色叫做datanode，这个玩意儿，说白了，就是每台机器会部署一个datanode节点，然后每个datanode就负责管理自己这台机器上负责存储的数据啊，比如说一个大文件给拆分成了3个部分，每个部分叫做一个block，就是一个文件块，分别在机器01，机器02，机器03上都存了一部分。然后每台机器上都有datanode，就负责管理自己那部分文件啊

所以讲到这里，你还应该知道，那么在namenode里，其实还存储了说，每个文件对应几个block啊，每个block在哪个datanode上啊，所以你可以看到，namenode就是整个hdfs集群的一个大脑啊

比如说你要读取一个文件的数据，那么肯定会找namenode问一下啊，说兄弟我要读取XXX文件，你告诉我这个文件对应哪几个block啊，都在哪几个datanode上存储呢？然后namenode就会告诉你，接着你就要跟各个datanode通信读取他们那上面的block数据出来啊

现在大家都学过了什么block拉，副本之类的东西，可以来看一个机制，叫做安全模式的了。这个安全模式的机制，是在namenode刚启动的时候，就会进入一个模式，叫做安全模式，safe mode，在这个模式下，hdfs集群是不会进行block的复制的

这个时候namenode会等着从各个datanode获取心跳和block report，然后看看集群里的整体的block情况，以及每个block有几个副本，默认是要有3个副本的。如果一个block有3个副本，那么就ok了，安全了

如果一定比例（80%）的block都是有足够的3个副本的，那么namenode就会退出安全模式，namenode一直处于safe mode状态下，就是因为没有达到一定的比例，block是足够的3个副本的，只有50%的block是有3个副本的

此时如果发现有某个block副本数量不够（比如只有2个副本）的，就指示datanode复制足够的副本数量，那么就ok了

这个backup node其实也是在hadoop 1.x提供的一种机制，他的思路其实就是优化和替代掉checkpoint node的那种下载fsimage和edits之后进行合并的思路

backup node的思路，其实就是在内存里维护一份和namenode一模一样的fsimage数据，同时还接收namenode发送过来的edits log数据流，获取到的edits log数据流，每次获取到一条就会往自己本地磁盘的edits log文件里写一条

而且同时还会直接将edits log变更应用到自己内存的fsimage里去，这样就在内存中保存了一份和namenode一模一样的元数据了

然后这个backup node会定时进行checkpoint操作的，说白了就是直接将内存里的fsimage数据直接写一份到磁盘文件里去覆盖之前的那份老的fsimage文件，然后清空edits log不就得了

一个namenode只能挂载一个backupnode，而且一旦使用了backupnode，就不能使用checkpoint node了

其实用backup node，最好的一点，就是不用让namenode自己本地维护一份edits log文件了，自己不用写磁盘了，直接namenode就是内存里一份fsimage数据，然后每次接收到一条元数据修改操作，就应用到自己内存里，然后发送一个edtis log数据流给backup node不就得了

backup node其实就是在负责磁盘上保存fsimage和edits log文件，而且定期进行checkpoint操作，如果要让namenode不要持久化fsimage和edits log文件，只要将dfs.namenode.edits.dir参数设置为空的就可以了

然后每次namenode重启的时候，要用-importCheckpoint指令，从其他地方去加载fsimage数据到自己内存中来，dfs.namenode.name.dir指定一个目录，这个目录必须是空目录，然后指定一个dfs.namenode.checkpoint.dir专门用来加载fsimage文件，然后启动namenode，使用-importCheckpoint就行了

那么这个fsimage从哪儿来呢？其实就从backup node那儿不就可以拿到了，是不是

但是说实话，这都是hadoop 1.x的策略了，现在hadoop 2.x时代，甚至都快进入hadoop 3.x时代了，早就不这么玩儿了

大数据的前身 = 各种公司老板看数据报表的需求 + 几百行大SQL + oracle

2009年的时候，阿里的数据团队，数据仓库，出一些数据报表，其实用的还是oracle，greenplum并行数据库的，还是快速的拥抱了开源的大数据技术

大数据的诞生，存储 + 计算

大量的数据不是都堆积在oracle里面么，互联网公司，就好比说，某个商品被人点击的次数，某个商品被人放入购物车的次数，某个商品被人下单购买的次数，某个商品被人退货的次数，某个商品被人好评的次数

老板，iphone 10这个手机出来了以后，我要看这个手机的一系列的指标，数据报表

如果你要出这么一张数据报表的话，你需要采集哪些数据？？

互联网行业，2010年以前，上网上淘宝的人，有多少？2008年的时候，淘宝的用户有多少

2010年以后，尤其是智能手机普及了以后，大量的人，每天有了大量的时间可以用手机来上互联网，访问互联网公司后台的系统

互联网公司的用户数量直线攀升

在APP/网站的前端，要做一个东西，叫做埋点，在APP上，有人点击了一下iphone 10这个手机的商品，就要记录一条日志，发送到后台存储起来，如果有在APP上将iphone 10放入了购物车里面，APP也要记录一条日志，发送到后台存储起来

就这一天，可能会出现这样的一系列的日志：

张三 android iphone 10 浏览了一次

张三 android iphone 10 放入了购物车

张三 android iphone 10 下了订单

张三 android iphone 10 完成了支付

张三 android iphone 10 事后又退款了

用户在APP手机上胡乱的操作，所有的你的用户行为日志，记录了用户在手机APP上的每一次操作的日志，都会从APP上上传到后台去，后台会将这些日志存储起来

用户行为日志是很多很多的，他只要点击了一次，就会有一条日志出来

或者用户进了iphone 10的页面，然后退出来了，也会有一条日志出来，记录用户所有的行为

电商的手机APP，每天活跃的用户数量是100万，不算多

每个人在手机APP里每天都要做很多的操作，搜索、浏览、下单、收藏、评价、退后、支付、订单，假设每人每天在手机APP里的操作，大概是100次

100万 \* 100次 = 1亿条用户行为日志

1亿日志，存在哪儿？？？2010年以前，网民也很少，PC（电脑），每天如果采集用户行为日志，可能每天就几万，或者几十万的数据量

每天1亿的日志数据量，放在哪儿？放在oracle里？

1亿条日志，根据我们的经验而言，每条日志一般字段是很多的，1G的数据量是有的

oracle的一张表里，一天就1亿条数据，1个月呢？30亿条数据？360亿条数据？放在一张表里？根本不现实

数据库，单表建议一般在1000万数据量以下就可以了

如果oracle有一张表，放了360亿条数据，然后没跑100个几百行的大SQL去统计100张数据报表出来，是多么恐怖的事情

数据越来越大，磁盘空间占用和耗费的越来越大，可能一年下来就把你的磁盘空间给占满了

如果用oracle的一些技术，oralce RAC，多台机器去承载数据，每台机器承载一部分的数据，可以是可以，但是有个问题，基本没人用得起，oracle收费的，搞那么多台oracle的话，就会导致你的费用非常的高昂，到后来没有几个公司用得起了

阿里花了很大的精力，搞去IOE，IBM的小型机，Oracle的数据库，EMC的存储设备

mysql，分库分表，搞多个mysql服务器去存储数据，每个服务器存放一部分的数据

计算也是一个大难题

如果mysql分库分表之后，要跑一个大SQL，不同的机器上的数据要放在一块儿计算，mysql能搞定么？mysql分库分表几乎搞不定，就算搞定成本也很高。

可以勉勉强强的将数据分放在各个机器上存储，比如用mysql分库分表，大报表的SQL根本跑不了，都是要将各个机器上的数据搞出来放在一块计算的

SSH系统 -> oracle -> 几百行大SQL -> 数据报表

互联网公司里，网民太多了，采集的用户行为日志太多了，数据量太大了，单表数据量太大了，数据存储消耗也太大了，根本就不现实

APP、网站上的各种用户行为的日志，指导老板做决策，指导PM改善产品，指导运营更好的做活动

存储，存不下 -> 计算，勉强存下了根本无法计算

mysql分库分表就能搞定这个存储的问题，这个方案很麻烦，大量的要对mysql分库分表进行运维、管理的时候很麻烦

hadoop的诞生以及各种大数据技术的发展、趋势以及应用

你要是比如往一个文件里写数据，那么肯定就是会在namenode那里根据文件的大小，将这个文件拆分为多个block的，每个block存储一部分数据，每个block是有固定大小的，以前老版本的hdfs都是一个block 64mb的，那太小了，现在一般都是一个block是128mb（目前默认的），或者是256mb比较合适

然后人家namenode会规定好的，哪个block放到哪个datanode上去，他会尽量让各个datanode均衡点儿，然后实际上你的文件就会被hdfs的client拆分为多个block写到各个datanode上去了，datanode把每个block就在自己本地磁盘上存为一个文件就ok了

datanode肯定不会傻到说是把所有文件都放在一个目录里的，那会导致linux的文件读写出问题的，他会建立一个合适的子目录层级结构，建立很多的子目录，然后保证每个目录中的文件数量不会过多

datanode每次启动的时候，都会扫描文件系统里的数据然后生成一份自己本地保存了哪些block的list数据，然后报告给namenode，namenode会做一些同步比对、校验啊之类的事情的。

之前简单说了一下hdfs namenode的元数据存储机制，是通过fsimage和edits log来实现的，那么我们这里退回到历史中的hadoop 1.x版本来聊一下，当时的情况是怎么样的，这样让大家知道下hadoop架构的演进

hadoop 1.x的时候，namenode启动，就是将fsimage读到内存里，然后对fsimage应用edits log来推进到元数据的最新状态，将最新fsimage写入磁盘文件，接着清空掉edits log。然后接下来你不是不断的在操作hdfs么，namenode会不断修改元数据，此时都是每次直接在内存里修改fsimage缓存，同时直接将变更日志追加到edits log里去的

那大家想一下，为啥不能每次直接将变更写入fsimage文件啊？因为fsimage太大啦，那是一个完整的hdfs集群元数据，这个文件很大的，还有你怎么随机修改文件里的数据啊，根本不现实么，是不是，磁盘随机读写性能会极差

所以就每次直接元数据修改保存在内存里，然后顺序追加一个edits log

这样有一个很大的问题，那edits log一直被不断的追加，他会变得很大很大很大很大，对不对？直到下一次namenode重启，edits log才能在内存里和fsimage合并后写入一份新的fsimage到磁盘里，然后edits log才能被清空掉

那在下一次namenode重启之前呢？坑爹了吧，edits log变得巨大无比，巨大无比会咋样？下一次你重启namenode的时候速度会超慢，因为要把一个巨大无比的edits log和fsimage进行合并啊

所以有一个思路就是在运行期间，定时将edits log和fsimage进行合并，不断的往私盘里写新的fsimage，同时清空掉edits log，这样就可以保证edits log不会很大了

但是大家有没有考虑过这有个什么问题啊？

就是说如果在namenode上搞这个edits log的不断的读取以及在内存中跟fsimage合并，然后还把fsimage写回去，同时清空edits log，这个操作是非常消耗性能的，因为涉及到很多的文件读写操作，这可能就会导致namenode这块性能下降

所以说，有一个角色，叫做secondary namenode，他一般独立部署在另外一台机器上，他就是专门在后台干这个edits log和fsimage合并的事儿的，他会每隔一段时间会执行一个checkpoint操作，通知namenode别写edits log，暂时写一个new edits log。然后将namenode上的fsimage和edits log拉到自己本地来，然后读到内存里去进行合并，合并后写一份新的fsimage到自己本地磁盘

然后secondary namenode将最新的fsimage推送到namenode上去，同时namenode会将new edits log这个文件变为edits log，o了，这个操作就是所谓的checkpoint操作对吧，那么这样就可以保证edits log定时合并到fsimage中去，不会变得特别大

而且secondary namenode还可以作为是namenode的一个冷备份，因为secondary namenode每次都保存了一个最近一次checkpoint的fsimage的快照，如果namenode上元数据损坏或者丢失，那么可以直接用secondary namenode上的啊，只不过丢失掉上次checkpoint之后发生的元数据变更罢了，因为那还在namenode的edits log中，还没合并到fsimage里去

这个默认的secondary namenode执行checkpoint的周期是3600秒，也就是1小时1次，或者是edits log只要达到了64mb，也会执行一次checkpoint，通过这个来确保edits log不会变的太大了

但是因为所谓的secondary namenode是hadoop 1.x的一个东西，这里咱们就不要讲了

然后这里多提一嘴，因为secondary namenode这个名字比较让人感觉引发了歧义，很多人以为这个secondary namenode是用来做数据备份的，名字起的不太恰当，但是其实这个东东是用来做checkpoint的，就是定期在后台默默的合并fsimage和edits log保证edits log不要太大

所以从hadoop 1.0.4版本之后，就提供了一种checkpoint node，换了个名字，其实就是secondary namenode，一样的

大数据诞生的前夕，国内的趋势

数据量太大，一台机器存不下；存多台机器没法计算

存储大数据的技术，计算大数据的技术，是没有的

平地一声惊雷，2013年开始出现大数据的概念，BAT等大公司当时率先引入了国外的开源的大数据的技术，2010年开始，BAT的一些大公司，团队，已经开始尝试自己研发能够存储和计算大数据的分布式系统了，我也就是当时的一员

2015年开始，大数据就开始变得非常流行了，各种培训机构如雨后春笋般涌现出来，都是号称的是从0基础培训，4个月，6个月培训一下就懂大数据了，就可以干活儿了

hdfs + yarn + mapreduce

hdfs：分布式存储系统，将庞大的数据分布式存储在很多台机器上，hadoop distributed filesystem，hadoop的分布式文件系统

大数据的本质，就是分布式系统，分布式存储系统，分布式资源调度系统，分布式计算系统，分布式流式处理系统，分布式作业调度系统，分布式搜索引擎，分布式NoSQL数据库，分布式数据仓库

各种各样的分布式系统

mapreduce：分布式计算系统，比如说你现在要对100亿条数据进行计算，但是这100亿条数据分布在10台机器上，你该怎么计算呢？难道先针对1台机器上的10亿数据，先算，算好了以后，再算第二台机器上的数据

yarn：分布式资源调度系统，yarn负责将你的计算任务给调度和分发到各个机器上去运行

mapreduce，一个一个的计算任务，你要定义好如何计算和处理每台机器上的数据，基于mapreduce的java api来写java代码的，写java代码还是挺慢的，尤其是出一些数据报表的时候

oracle，几百行的大SQL，很容易就可以写完出一份复杂报表的SQL

大数据技术，hdfs + yarn + mapreduce之后，我还得写java代码，我得用java代码来实现几百行大SQL的计算逻辑，所以现在几乎已经没有人写java代码来进行计算了

做数据分析、统计、出报表，一般都是用SQL的

hive：分布式数据仓库，依赖于mapreduce，干的事情只有一件，将SQL翻译为基于mapreduce的java代码，将你的SQL翻译为基于mapreduce java api的java代码，还是找yarn，提交计算任务，yarn负责将你的计算任务分发到各个机器上去执行，当时有了hive之后，真是解放了全人类，在最开始做大数据开发的时候，还做不了太复杂的报表，只能出一些简单的数据指标和报表，当时开发效率太差了，基于mapreduce来写java代码实现计算的逻辑

flume：分布式日志采集系统，手机APP、网站前端不是会发送大量的日志到后台么？后台接收到了这个日志之后，就需要将这个日志通过flume上传到hdfs上去

最早最早，其实就是用这样的一套东西，就可以支撑最最起码的，针对海量的数据，大量的数据，大数据，用各种各样的分布式系统，实现针对几十亿数据，几百亿数据，甚至是几千亿数据的一个数据分析和统计，产出各种各样的数据报表

大数据技术 = 分布式技术 = 大规模复杂的分布式系统架构

hadoop技术，就是针对大量的数据进行存储、调度和计算的一套分布式系统架构

hbase：分布式NoSQL数据库，他底层是基于hdfs分布式存储来实现的，但是他基于hdfs封装和开发了一套分布式NoSQL数据库，你可以基于hbase实现大量的NoSQL数据库的一些操作，毫秒级~秒级的增删改查

spark生态：分布式计算（替代以前的mapreduce，速度更快，性能更高），spark sql（支持你直接基于spark写大SQL，跑报表统计），spark streaming（针对源源不断过来的数据流，进行分布式的流式计算），spark mllib（基于分布式存储的数据，进行分布式的机器学习）

elasticsearch：分布式数据存储+搜索，基于es分布式存储一些数据，针对数据可以做类似于搜索引擎一样的搜索

kylin：分布式OLAP分析，写一些SQL去跑查询和统计

druid：分布式流式的分析和统计

flink：分布式计算，分布式流式数据的处理

所以如果是hadoop 2.x的双实例HA机制里面，checkpoint是怎么执行的呢？

其实在standy namenode上，会运行一个CheckpointerThread后台线程，他默认是要么1小时一次，要么是有100万条edits log没有合并到fsimage去了，此时就会执行一个checkpoint操作

他执行checkpoint其实非常简单的，因为这个standby namenode有点儿像是hadoop 1.x中的backup node，所以说其实就是将内存中最新的那份fsimage写到磁盘文件上的fsimage文件里去，同时清空掉edits log不就得了

然后会将最新的fsimage文件发送到active namenode上去覆盖之前旧的fsimage，同时将active namenode的edtis logs给清空掉，不就ok了么

反正大家之前也看到了，hadoop 1.x的时代，基本就是一个namenode，然后挂一个secondary namenode，就这意思，namenode就是接收元数据变更操作，比如创建目录，创建文件，给写入文件的数据划分block，给block映射到datanode，类似这些活儿是吧

namenode说白了，就是在自己内存里搞一份元数据，有任何修改就在内存里改改，然后写一份变更日志到edits log文件里去，不停追加，如果重启，那么就读取fsimage文件和editslog，然后editslog与fsimage合并，然后写一份新的fsimage，同时清空edits log，就这原理

所以为了保证edits log不要太大，下次重启时间过长，就会搞一个secondary namenode，一般，不停执行checkpoint操作，在另外一台机器上合并fsimage和edits log，然后更新namenode上的fsimage，清空edits log

同时其实secondary namenode之前也一直相当于是namenode的冷备份，因为有fsimage的快照，如果namenode上的元数据破损了，你可以拷贝secondary namendode上的fsimage过去就ok了

然后这样的不好就在于说，namenode的磁盘要是坏了，还是会丢失部分元数据啊，因为上次checkpoint之后，在edits log里的就丢失了

所以说hadoop 1.x的namenode，首先元数据是可能丢失的，其次是不高可用的，如果namenode宕机了，那么整个hdfs集群就无法使用了

然后进入到了hadoop 2.x的时代，为了保证namenode上的元数据不会丢失，而且是高可用的，搞了一个双实例HA的机制

就是说在集群里启动两个namenode，然后一个是active状态，一个是standby状态，一个是主，一个是备。所有的操作都是发送给active namenode的，然后standby namenode就是一个热备，不停的同步元数据

但是还要在集群里引入一组节点，叫做journal nodes，一般是启动3个journal nodes，这个journal是啥意思？就是日志的意思，叫做journal，顾名思义，肯定是用来保存edits log这种操作日志的了

每次namenode有一个元数据变更，就要将这个edits log发送给journal nodes里的大多数，什么叫做大多数呢？就是quorum，比如3台journal nodes，大多数就是3 / 2 + 1 = 2,2台就是大多数了，只要namenode发送edit log大多数的journal nodes之后，就认为这个元数据变更是安全的了

standby namenode就一直监控着journal nodes里的edits log变更，只要变更了就会读取edits log，同时应用到自己本地的内存里去，形成一个跟active namenode一致的fsiamge数据在内存里

然后如果说active namenode挂掉了，那么此时standby namenode立刻就会感知到的，然后他会确保自己从journal nodes读取了所有的edits log之后，内存的fsimage绝对是最新的之后，就会将自己切换为active namenode，形成主备切换。这个时候，namenode第一数据不会丢失，因为有journal nodes在里面用多台机器保存着，第二，namenode高可用，一台挂了，另外一台立马接管，数据都是一致的

而且所有的datanode都是配置了两台namenode的，那么datanode会将自己的block report汇报给主备两台namenode，确保他们都能感知到集群里的datanode的状态和block的情况啊。。

那么两台namenode是如何在故障的时候自动faillover的呢？靠的是ZKFC两个进程，就是每个namenode机器上都要跑一个ZKFailoverController的进程，简称之ZKFC，他们俩会不断的监控两个namenode，同时在zookeeper集群上（至少3个节点）维护namenode的状态

如果active namenode挂了，那么ZKFC里的也给HealthMonitor就会监控到，然后就会告诉ZKFC里的一个FailoverController通知说namenode挂了，接着FailoverContrller找ActiveStandbyElector组件说要主备重新选举

ActiveStandbyElector就会基于zk集群完成主备选举，这个过程就不说了，总之会选举出来standby namenode作为主的

然后zk会通知standby机器上的ZKFC中的ActiveStandbyElector组件，ActiveStandbyElector通知FailoverController要切换standby为active了，然后FailoverController再通知standby namenode切换为active namenode

而且journal nodes还仅仅只允许一台namenode给他写edits log，就是为了避免脑裂问题，两台namenode的网络环境不通了，他们俩都以为自己是active往journal nodes写数据，此时只能有一台写

下周我们的课程，就是最后研究一下，hdfs文件下载的机制

需要下载一个一个的block，并发的去下载，每个block都是传输一个堆的数据，最后就是拼成一个完整的大文件

下载的过程中，某个block下载过程失败了怎么办，下载到本地之后发现他破损了怎么办，容错的机制，屁股想想，肯定会找namenode通信，说让namenode重新再次分配一个新的datanode来下载那个block副本

下周就把hdfs源码结束，集群启动 -> fsimage & edits log -> 文件上传 -> 数据容错 -> 文件下载，hdfs作为大型复杂的分布式存储系统，核心源码全部结束。源码还有很多其他的，他有很多的机制和功能，代码量特别大

cdh去做大规模的集群部署、运维、管理和监控，这套东西不是什么技术难度，操作性，但是你对底层的源码和原理要非常的熟悉，才能搞明白这个大型集群如何来管理

下个月，进入大数据的第一个大型的项目：hadoop源码二次开发。我们直接对hadoop 2.6.5开源的版本，大量的进行bug fix、性能优化、功能开发，对源码做深度的二次开发和定制，作为我们第一个大型的项目。

我们后面全部以大项目来驱动学习技术和做项目，日志采集平台（埋点平台、离线日志采集、实时数据采集），大型数据仓库，实时数据平台，数据分析平台

下一讲：如果一个datanode发送心跳给namenode的时候，namenode就会给他下发一个复制block的指令

现在有一个block，有3个副本，存放在3个datanode上，这3个datanode都通过5分钟的增量上报机制，都上报了block副本给namenode，namenode现在就知道了，block有 3个副本，在3个datanode上

有一个datanode宕机了，block的副本数量肯定会少了1个

namenode如何感知到datanode宕机导致的block副本数量变少了呢

开动脑筋，如果一个datanode宕机了，就会导致他就不再发送心跳了，HeartbeatManager组件是负责监控各个datanode的心跳的，是否按照一定的固定频率在发送心跳过来，如果一段时间内没有心跳，就认为这个datanode宕机了

猜测一下，如果HeartbeatManager发现datanode宕机了，一定会从内存数据结构中，将这个datanode对应的各种数据全部移除，删掉，清空，认为这个datanode就没有了，我们来回顾一下当年的源码

datanode宕机的时候，就会自动触发BlockManager会去从内存数据结构中移除这个datanode上所有的block副本数据，另外会触发每个block的副本数量的检查，如果低于你期望的副本数量，会放入一个待复制的队列中

namenode可以接受到一个请求，人家datanode通知他说，兄弟，我received block，你来处理一下吧，我们如果要研究一下namenode是如何组织和管理block数据的话，其实我们就是

receiving bock，received block

很明显是说，在刚刚开始准备传输一个block的时候，namenode刚刚申请了一个block的时候，会传递空的block到各个datanode上建立一个数据流管道，在建立管道的时候，各个datanode就会去通知namenode说

准备接收一个block，receiving状态

最后各个datanode接收到了一个空packet之后，就是认为是一个block接收完毕，此时就会上报namenode说，自己刚刚接收到了一个block

每个datanode上报完了自己的block之后，人家都会计算一下，当前block的副本数量，如果副本的数量不充足的话，没有达到3，或者超过了3，都会处理一下，他会去看看当前的block副本数量不足的话，是不是找另外一个datanode给再次复制一份

保证副本数量是3

（1）block上报机制

（2）block副本数量检查机制

（3）副本数量不足的block是如何生成复制指令（选择一个datanode去复制一个新的block副本，针对那个datanode生成一个复制指令）

（4）被选中的datanode后面的话，就会在发送心跳的时候就会接收那个指令

（5）接收到指令的datanode就会复制一个bock给其他的datanode

现在很明年，soruce datanode他需要接受到一个任务，让他把block复制到指定的一个datanode上面去，但是那个soruce datanode如何接受到这个指令呢，通过心跳机制来下发指令

我之前的讲解算错了一个数字，有一个同学来问我

一个packet是127个chunk，一个chunk是512字节，65523个字节，64kb

一个block是128mb，包含2048个packet

不是一个block包含2个packet，说错了，算错数字了，原理和机制是没什么问题的

重新回顾了一下，全量上报block和增量上报block所在的位置

文件上传 -> datanode收到一个block之后是如何上报给namenode知道的

namenode必须得知道各个datanode上有哪些block

namenode可以检查各个block的副本数量，比如说有一个datanode突然宕机了，就会导致他的block副本丢失了，导致一些block只有2个副本了，namenode他需要感知到说，有的block好像副本数量不足3了

他就可以通知某个datanode复制一份block给另外一个好的datanode，保证每个block的副本数量是3

数据容错的机制，主要就是要依赖datanode上报block

hdfs文件的下载，如果你要下载一个1TB的大文件，你要问一问namenode说，兄弟，namenode就要通知hdfs客户端，从哪些datanode去下载block，多个block组成一个大的1TB的文件

datanode如果上传了一个block之后，就会由BlockScanner组件，定时的扫描，检查一下这个block是否有破损

到这一讲为止，我们就搞清楚了，datanode是如何汇报block给namenode的

全量汇报，启动 / 6小时

增量汇报，每隔5分钟汇报一次最近增量变化的

坏块汇报，block scanner定时扫描汇报破损的块的情况

3616118202

dfs/data/current，目录下，就存放了datanode组织的各种数据

BP-497387557-127.0.0.1-1537948984539，BP，BlockPool，hdfs federation架构，是可以有多个namenode的，每个datanode上可以存放多个active namenode的数据，每个active namenode的数据就对应一个BlockPool

一个BlockPool就放了一个active namenode的数据，在这个datanode

没有搭建hdfs deferation架构，HA架构，一个active namenode，在datanode当然就只有一个BlockPool了

rbw目录里的文件，大概是说你正在写的一些磁盘文件，好比说什么呢？你的某些blk的文件，hdfs客户端是通过大量的packet传输来完成一个block的数据的上传的，在上传过程中，刚上传了100多个packet，还有很多没有上传完

finalized目录，放的是最终都传输完毕的block数据文件

subdir0目录，存放的数据越来越多的话，肯定会有subdir1，subdir2，subdir3，一个datanode存放的bock文件是极其多的，一个block文件是128mb，10个block是1g，1000个block文件是1tb，1万个block就是10TB

往多里说，可能一个datanode上有上万个block文件，不能都放在一个目录下，在ilnux里面，如果在一个目录下存放的文件过多的话，就会导致磁盘文件的读写性能会有影响，所以说人家datanode会通过大量的目录的组织，保证有很多个目录，每个目录的子目录的深度不会太深

每个目录里的文件的数量都是固定的，不会让某个目录里的文件数量过多

每个subdir0 -> subdir0，但是他会保证子目录的深度不会太深，每个目录里的文件数量不会太多，在每个目录里存放了一些block文件，每个block文件对应了一个.meta文件，里面存放了一些元数据，checksum校验和

blk\_107374182，在这个block文件里，存放的其实就是你上传的那个大文件的一部分，因为我们上传的那个文件，他的内容量很少，就是一行hello world，所以说他就会直接将那一点点内容就放一个packet，就对应一个block

对应一个block文件

如果你是一个大文件的话，其实对应的就是每个block文件里是包含了那个大文件的一部分

默认的情况下，一个block是3个副本

每个block都有一个唯一分配的这么一个block的ID，一个block ID对应的block，是在集群里有3个副本，一模一样的

NameNode.main() -> 启动入口

NameNode.initialize() -> 初始化的核心逻辑

-> startHttpServer() -> 启动自己的http服务器，后面接收别人发送过来的http请求

-> loadNamesystem() -> 初始化FSNamesystem，磁盘加载fsimage和edits进行内存合并

-> createRpcServer() -> 初始化rpc server，接收别人调用过来的rpc请求

很多搞大数据的同学，可能java基础比较差，所以说可能连http和rpc的区别都没搞清楚，要求是有比较好的java基础的，大数据架构课程之前，是有几个月的java架构的课程，包含了重点的就是三块东西

（1）阶段一：大型java工程的开发，流程，规范，需求，文档

（2）阶段二：分布式系统架构，分布式服务框架（spring cloud为主的http类的服务框架，dubbo为主的rpc类的服务框架），分布式事务，分布式锁，分布式协调（zookeeper，源码剖析，分布式机制和原理）

（3）阶段三：微服务架构，自动化测试、持续集成、自动化部署、监控与运维、分布式日志中心，这块东西本质是涉及到了复杂的大量微服务组成的分布式系统的整体的开发、测试、运维以及管理

深入学习大数据的话，上面几块东西都是必须非常的扎实，否则你根本没这个基础来学习大数据，hdfs源码的时候，rpc是什么都不知道。只要你学过spring cloud和dubbo，你其实肯定是知道http是什么，rpc是什么的

namenode，是分成两个server，一个是http server，监听的是50070，主要是对接你通过浏览器来请求他查看hdfs集群的一些信息的；一个是rpc server，hdfs集群内部，namenode和datanode之间进行通信，或者是dfsclient跟namenode之间进行通信，他其实也是绑定某个端口的，只不过分布式系统各个节点之间互相调用的时候，其实就是跟普通的方法调用是一样的

http去调用，构造一个http地址，附加一些参数，针对这个http地址去发起请求。http://192.168.31.350:50070/getBlockReport?namenodeId=1111

rpc调用，datanodeProtocol.getBockReport()，在这个方法底层，人家去连接datanode的端口，进行底层的网络通信，比如说走底层的tcp协议什么的

1、准备5台linux虚拟机

都是CentOS 7操作系统，64位，每台机器都是1个cpu，1G内存，安装JDK 1.8

（1）使用课程提供的CentOS 7镜像即可，CentOS-7-x86\_64-Minimal-1611.iso。

（2）创建虚拟机：打开Virtual Box，点击“新建”按钮，点击“下一步”，输入虚拟机名称为hadoop01，选择操作系统为Linux，选择版本为Red Hat-64bit，分配1024MB内存，后面的选项全部用默认，在Virtual Disk File location and size中，一定要自己选择一个目录来存放虚拟机文件，最后点击“create”按钮，开始创建虚拟机。

（3）设置虚拟机网卡：选择创建好的虚拟机，点击“设置”按钮，在网络一栏中，连接方式中，选择“Bridged Adapter”。（搭建虚拟机还真不是我专门要讲的一块内容，我一般常用的搭建虚拟机做实验的方式就是桥接网络）

如果你的笔记本电脑在家里用wifi的ip地址配置了一下虚拟机的网络，然后你的电脑拿到外面咖啡馆里去，或者公司里去，ip地址变化了之后，就会导致你的之前搭建的那套虚拟机环境就没法用了

如果你希望无论到哪里，ip地址变化了以后，虚拟机都可以正常工作，不需要重新配置网络，那么你就去百度一下，虚拟机如何配置网络，可以更换环境的时候，不需要重新配置，NAT如何配置

桥接，换了一个网络环境，ip地址都换了，5台虚拟机重新配置一下里面的网卡的一些ip地址什么的，就ok了

（4）安装虚拟机中的CentOS 7操作系统：选择创建好的虚拟机，点击“开始”按钮，选择安装介质（即本地的CentOS 7镜像文件），按照课程选择后自动安装即可

（5）安装完以后，CentOS会提醒你要重启一下，就是reboot，你就reboot就可以了。

（6）配置网络

（如果重启网卡没看到人家给你分配的ip地址，你自己应该是要去连一个路由器就可以了）

vi /etc/sysconfig/network-scripts/ifcfg-enp0s3

先让它动态分配一个ip地址

ONBOOT=yes

service network restart

ip addr

再设置静态ip地址

BOOTPROTO=static

IPADDR=192.168.31.250

NETMASK=255.255.255.0

GATEWAY=192.168.31.1

service network restart

ip addr

配置DNS

检查NetManager的状态：systemctl status NetworkManager.service

检查NetManager管理的网络接口：nmcli dev status

检查NetManager管理的网络连接：nmcli connection show

设置dns：nmcli con mod enp0s3 ipv4.dns "114.114.114.114 8.8.8.8"

让dns配置生效：nmcli con up enp0s3

（7）配置hosts

vi /etc/hosts

配置本机的hostname到ip地址的映射

（8）配置SecureCRT

此时就可以使用SecureCRT从本机连接到虚拟机进行操作了

一般来说，虚拟机管理软件，virtual box，可以用来创建和管理虚拟机，但是一般不会直接在virtualbox里面去操作，因为比较麻烦，没有办法复制粘贴

SecureCRT，在windows宿主机中，去连接virtual box中的虚拟机

收费的，我这里有完美破解版，跟着课程一起给大家，破解

（9）关闭防火墙

systemctl stop firewalld.service

systemctl disable firewalld.service

关闭windows的防火墙

后面要搭建集群，有的大数据技术的集群之间，在本地你给了防火墙的话，可能会没有办法互相连接，会导致搭建失败

（10）配置yum

yum clean all

yum makecache

yum install -y wget

（11）安装JDK

1、将jdk-8u131-linux-x64.rpm通过WinSCP上传到虚拟机中

2、安装JDK：rpm -ivh jdk-8u131-linux-x64.rpm

3、配置jdk相关的环境变量

vi ~/ .bashrc

export JAVA\_HOME=/usr/java/latest

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

source .bashrc

4、测试jdk安装是否成功：java -version

（12）在另外4个虚拟机中安装CentOS集群

按照上述步骤，再安装三台一模一样环境的linux机器

另外三台机器的hostname分别设置为hadoop02，hadoop03，hadoop04，hadoop05

安装好之后，在每台机器的hosts文件里面，配置好所有的机器的ip地址到hostname的映射关系

比如说，在elasticsearch01的hosts里面

192.168.31.250 hadoop0101

192.168.31.xxx hadoop02

192.168.31.xxx hadoop03

192.168.31.xxx hadoop04

（13）配置5台CentOS为ssh免密码互相通信

首先在三台机器上配置对本机的ssh免密码登录

ssh-keygen -t rsa，生成本机的公钥，过程中不断敲回车即可，ssh-keygen命令默认会将公钥放在/root/.ssh目录下

cd /root/.ssh

cp id\_rsa.pub authorized\_keys

将公钥复制为authorized\_keys文件，此时使用ssh连接本机就不需要输入密码了

接着配置三台机器互相之间的ssh免密码登录

使用ssh-copy-id -i hostname命令将本机的公钥拷贝到指定机器的authorized\_keys文件中

2、hdfs HA集群部署规划

namenode，就相当于是在eclipse中新建一个工程，取个名字叫做namenode，在工程里面开发代码，写好代码之后，就可以打包，在linux服务器上，用java -jar给他启动，namenode就是这个意思

namenode就是一个java工程，写完里面的代码，打包，在linux上部署，java -jar启动，出来一个进程，namenode其实就是一个系统，启动的时候就是一个进程

hadoop01：active namenode，ZKFC

hadoop02：standby namenode，ZKFC

hadoop03：datanode，journal node，QuorumPeerMain

hadoop04：datanode，journal node，QuorumPeerMain

hadoop05：datanode，journal node，QuorumPeerMain

3、部署zookeeper集群

在hadoop03、hadoop04、hadoop05三台机器上部署zookeeper

tar -zxvf zookeeper-3.4.9.tar.gz

mv zookeeper-3.4.9 zookeeper

vi ~/.bashrc

export ZOOKEEPER\_HOME=/usr/local/zookeeper ?export PATH=$PATH:$ZOOKEEPER\_HOME/bin

source ~/.bashrc

vi zoo.cfg （$ZOOKEEPER\_HOME/conf）

dataDir=/home/data/zookeeper ?dataLogDir=/home/log/zookeeper?server.1=hadoop03:2888:3888

server.2=hadoop04:2888:3888 ?server.3=hadoop05:2888:3888

mkdir -p /home/data/zookeeper

mkdir -p /home/log/zookeeper

cd /home/data/zookeeper

echo 1 > myid

scp -r /usr/local/zookeeper hadoop04:/usr/local ，修改环境变量，在hadoop04将myid的内容改为2 （echo 2 > myid）

scp -r /usr/local/zookeeper hadoop05:/usr/local，修改环境变量，在hadoop05将myid的内容改为3 （echo 3 > myid）

三台机器上执行：zkServer.sh start

查看集群状态 ?1、jps（查看进程） ?2、zkServer.sh status（查看集群状态，主从信息）

4、部署hadoop集群

先在hadoop01上操作

tar -zxvf hadoop-2.9.1.tar.gz

mv hadoop-2.9.1 hadoop

vi ~/.bashrc

export HADOOP\_HOME=/usr/local/hadoop

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

source ~/.bashrc

vi $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

export JAVA\_HOME=/usr/java/latest

vi core-site.xml

在hadoop01~05都创建目录：mkdir -p /home/apps/hadoop/tmp

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://ns1/</value>

</property>

<property>

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

<value>/home/apps/hadoop/tmp</value>

</property>

<property>

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

<value>hadoop03:2181,hadoop04:2181,hadoop05:2181</value>

</property>

</configuration>

vi hdfs-site.xml

<configuration>

<property>

<name>dfs.nameservices</name>

<value>ns1</value>

</property>

<property>

<name>dfs.ha.namenodes.ns1</name>

<value>nn1,nn2</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn1</name>

<value>hadoop01:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn1</name>

<value>hadoop01:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn2</name>

<value>hadoop02:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn2</name>

<value>hadoop02:50070</value>

</property>

<property>

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

<value>qjournal://hadoop03:8485;hadoop04:8485;hadoop05:8485/ns1</value>

</property>

<property>

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

<value>/home/apps/hadoop/journaldata</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.ns1</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>

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/root/.ssh/id\_rsa</value>

</property>

<property>

<name>dfs.ha.fencing.ssh.connect-timeout</name>

<value>30000</value>

</property>

</configuration>

在hadoop01~05上都创建目录：mkdir -p /home/apps/hadoop/journaldata

vi slaves

hadoop03

hadoop04

hadoop05

上面其实就已经将active namenode给配置好了

然后将这个hadoop拷贝到hadoop02~05台机器上去：scp -r /usr/local/hadoop hadoop02:/usr/local/，修改环境变量

5、启动hadoop集群

（1）在hadoop03、hadoop04、hadoop05上启动journal nodes集群：

sbin/hadoop-daemon.sh start journalnode

（2）格式化namenode（hadoop01上执行）

hdfs namenode -format

hdfs namenode -bootstrapStandby

（3）格式化ZKFC（hadoop01上执行）

hdfs zkfc -formatZK

（4）启动hdfs集群（hadoop01上执行）

sbin/start-dfs.sh，这个脚本会自动在hadoop01和hadoop02上启动一个namenode，同时启动一个ZKFC，然后会自动在hadoop03、hadoop04、hadoop05上分别启动一个datanode

6、查看hadoop集群

在hadoop01和hadoop02访问50070端口即可

namenode他的类注释里都写了，有一个比较关键的组件是FSNamesystem，主要就是用来管理hdfs的元数据，包括内存和磁盘上的元数据，以及相关的流程和机制，都是他在管理的，可以这么来认为

源码课跟一般的源码分析的书籍是完全不一样的，我们一定要从场景出发，说的就是hdfs集群第一次启动，别上来就是一个类一个类，按照顺序每一行代码都分析源码，那样是很不靠谱的，必须从架构原理出发，从各个场景出发，来分析他的源码

NameNode.main() -> 启动入口

NameNode.initialize() -> 初始化的核心逻辑

-> startHttpServer() -> 启动自己的http服务器，后面接收别人发送过来的http请求

-> loadNamesystem() -> 初始化FSNamesystem，磁盘加载fsimage和edits进行内存合并

所有的元数据，目录、文件以及对应的配置，都在namenode的内存里

如果你现在手头有几万台机器，里面存储了几百亿个目录和文件的元数据，元数据大小达到了几百G的话，那你的Namenode一台机器怎么可能放的下呢？

国内真的大数据集群大到，元数据达到几百G，一台namenode内存放不小的情况，也比较少，国内最大的一些大公司才会出现这样的情况

为了解决这个问题，hdfs后来也是推出了一个功能，federation联邦机制

大家发现一个问题没有，现在的话呢，namenode其实哪怕是HA架构，但是也是有问题的，因为hdfs所有的元数据实际上都是放在内存里的，大家已经发现这个问题了吧？所以说怎么办呢？

如果说hdfs集群规模超大，几万台机器，然后大量的目录和文件，元数据的量都达到了几百G，一台机器的内存足够吗？肯定是不够的

所以说hdfs后来出了一个federation联邦架构

这个大概什么意思呢？其实就是说，可以搞多个active namenode，然后每个active namenode可以放一部分元数据，这不就可以横向扩展namenode数量了么？然后每个active namenode再挂一个standby namenode，不是每个namenode同时也是高可用了么？

所以大概就是这么个架构，接下来图解，说一下federation联邦架构的原理

每个namenode都有一个block pool的概念，这里面就是这个namenode管理的所有的block，但是所有的block还是统一存放在datanode集群上的，datanode照样跟所有的namenode进行通信，汇报自己的情况，等等

一个namenode上管理的namespace（命名空间，你可以理解为就是一部分元数据吧），跟对应的block pool结合起来，称之为一个namespace volume。

至于具体的配置就在这里不做了，因为我们的时间有限，不可能每个技术都展开来讲解，这里我们还是专注在hdfs作为一个大规模分布式文件系统的内核源码的角度来讲解这个课程

你可以给namenode机器的配置特别好，物理机，32 core，128G内存，容纳的元数据已经很大了，大部分公司所谓的大数据集群，也不过就是几十台机器，几百台机器，几千台机器的大数据集群的公司不多了

1、准备5台linux虚拟机

都是CentOS 7操作系统，64位，每台机器都是1个cpu，1G内存，安装JDK 1.8

（1）使用课程提供的CentOS 7镜像即可，CentOS-7-x86\_64-Minimal-1611.iso。

（2）创建虚拟机：打开Virtual Box，点击“新建”按钮，点击“下一步”，输入虚拟机名称为hadoop01，选择操作系统为Linux，选择版本为Red Hat-64bit，分配1024MB内存，后面的选项全部用默认，在Virtual Disk File location and size中，一定要自己选择一个目录来存放虚拟机文件，最后点击“create”按钮，开始创建虚拟机。

（3）设置虚拟机网卡：选择创建好的虚拟机，点击“设置”按钮，在网络一栏中，连接方式中，选择“Bridged Adapter”。（搭建虚拟机还真不是我专门要讲的一块内容，我一般常用的搭建虚拟机做实验的方式就是桥接网络）

如果你的笔记本电脑在家里用wifi的ip地址配置了一下虚拟机的网络，然后你的电脑拿到外面咖啡馆里去，或者公司里去，ip地址变化了之后，就会导致你的之前搭建的那套虚拟机环境就没法用了

如果你希望无论到哪里，ip地址变化了以后，虚拟机都可以正常工作，不需要重新配置网络，那么你就去百度一下，虚拟机如何配置网络，可以更换环境的时候，不需要重新配置，NAT如何配置

桥接，换了一个网络环境，ip地址都换了，5台虚拟机重新配置一下里面的网卡的一些ip地址什么的，就ok了

（4）安装虚拟机中的CentOS 7操作系统：选择创建好的虚拟机，点击“开始”按钮，选择安装介质（即本地的CentOS 7镜像文件），按照课程选择后自动安装即可

（5）安装完以后，CentOS会提醒你要重启一下，就是reboot，你就reboot就可以了。

（6）配置网络

（如果重启网卡没看到人家给你分配的ip地址，你自己应该是要去连一个路由器就可以了）

vi /etc/sysconfig/network-scripts/ifcfg-enp0s3

先让它动态分配一个ip地址

ONBOOT=yes

service network restart

ip addr

再设置静态ip地址

BOOTPROTO=static

IPADDR=192.168.31.250

NETMASK=255.255.255.0

GATEWAY=192.168.31.1

service network restart

ip addr

配置DNS

检查NetManager的状态：systemctl status NetworkManager.service

检查NetManager管理的网络接口：nmcli dev status

检查NetManager管理的网络连接：nmcli connection show

设置dns：nmcli con mod enp0s3 ipv4.dns "114.114.114.114 8.8.8.8"

让dns配置生效：nmcli con up enp0s3

（7）配置hosts

vi /etc/hosts

配置本机的hostname到ip地址的映射

（8）配置SecureCRT

此时就可以使用SecureCRT从本机连接到虚拟机进行操作了

一般来说，虚拟机管理软件，virtual box，可以用来创建和管理虚拟机，但是一般不会直接在virtualbox里面去操作，因为比较麻烦，没有办法复制粘贴

SecureCRT，在windows宿主机中，去连接virtual box中的虚拟机

收费的，我这里有完美破解版，跟着课程一起给大家，破解

（9）关闭防火墙

systemctl stop firewalld.service

systemctl disable firewalld.service

关闭windows的防火墙

后面要搭建集群，有的大数据技术的集群之间，在本地你给了防火墙的话，可能会没有办法互相连接，会导致搭建失败

（10）配置yum

yum clean all

yum makecache

yum install -y wget

（11）安装JDK

1、将jdk-8u131-linux-x64.rpm通过WinSCP上传到虚拟机中

2、安装JDK：rpm -ivh jdk-8u131-linux-x64.rpm

3、配置jdk相关的环境变量

vi ~/ .bashrc

export JAVA\_HOME=/usr/java/latest

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

source .bashrc

4、测试jdk安装是否成功：java -version

（12）在另外4个虚拟机中安装CentOS集群

按照上述步骤，再安装三台一模一样环境的linux机器

另外三台机器的hostname分别设置为hadoop02，hadoop03，hadoop04，hadoop05

安装好之后，在每台机器的hosts文件里面，配置好所有的机器的ip地址到hostname的映射关系

比如说，在elasticsearch01的hosts里面

192.168.31.250 hadoop0101

192.168.31.xxx hadoop02

192.168.31.xxx hadoop03

192.168.31.xxx hadoop04

（13）配置5台CentOS为ssh免密码互相通信

首先在三台机器上配置对本机的ssh免密码登录

ssh-keygen -t rsa，生成本机的公钥，过程中不断敲回车即可，ssh-keygen命令默认会将公钥放在/root/.ssh目录下

cd /root/.ssh

cp id\_rsa.pub authorized\_keys

将公钥复制为authorized\_keys文件，此时使用ssh连接本机就不需要输入密码了

接着配置三台机器互相之间的ssh免密码登录

使用ssh-copy-id -i hostname命令将本机的公钥拷贝到指定机器的authorized\_keys文件中

2、hdfs HA集群部署规划

namenode，就相当于是在eclipse中新建一个工程，取个名字叫做namenode，在工程里面开发代码，写好代码之后，就可以打包，在linux服务器上，用java -jar给他启动，namenode就是这个意思

namenode就是一个java工程，写完里面的代码，打包，在linux上部署，java -jar启动，出来一个进程，namenode其实就是一个系统，启动的时候就是一个进程

hadoop01：active namenode，ZKFC

hadoop02：standby namenode，ZKFC

hadoop03：datanode，journal node，QuorumPeerMain

hadoop04：datanode，journal node，QuorumPeerMain

hadoop05：datanode，journal node，QuorumPeerMain

3、部署zookeeper集群

在hadoop03、hadoop04、hadoop05三台机器上部署zookeeper

下载zookeeper 3.4.9：? HYPERLINK "http://archive.apache.org/dist/zookeeper/zookeeper-3.4.9/" \t "https://blog.csdn.net/zz657114506/article/details/\_blank" ?http://archive.apache.org/dist/zookeeper/zookeeper-3.4.9/?，放到/home/apps目录下去

tar -zxf zookeeper-3.4.9.tar.gz

vi /etc/profile

export ZOOKEEPER\_HOME=/home/apps/zookeeper-3.4.9 ?export PATH=$PATH:$ZOOKEEPER\_HOME/bin

source /etc/profile

vi zoo.cfg （$ZOOKEEPER\_HOME/conf）

dataDir=/home/data/zkdata ?dataLogDir=/home/log/zk ?server.1=hadoop03:2888:3888

server.2=hadoop04:2888:3888 ?server.3=hadoop05:2888:3888

mkdir /home/data/zkdata

mkdir /home/log/zk

cd /home/data/zkdata

echo 1 > myid

scp -r /home/apps/zookeeper-3.4.9 hadoop04:/home/apps/ ，然后干一样的事儿?scp -r /home/apps/zookeeper-3.4.9 hadoop05:/home/apps/，然后干一样的事儿

在hadoop04将myid的内容改为2 （echo 2 > myid） ?在hadoop05将myid的内容改为3 （echo 3 > myid）

三台机器上执行：zkServer.sh start

查看集群状态 ?1、jps（查看进程） ?2、zkServer.sh status（查看集群状态，主从信息）

4、部署hadoop集群

先在hadoop01上操作

tar -zxf hadoop-2.9.1.tar.gz -C /home/apps/

vim /etc/profile

export JAVA\_HOME=/home/apps/jdk1.8.0\_111

export HADOOP\_HOME=/home/apps/hadoop-2.9.1

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

source /etc/profile

vi $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

export JAVA\_HOME=/home/apps/jdk1.8.0\_111

vi core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://ns1/</value>

</property>

<property>

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

<value>/home/apps/hadoop-2.9.1/tmp</value>

</property>

<property>

<name>ha.zookeeper.quorum</name> <value>hadoop01:2181,hadoop02:2181,hadoop03:2181</value>

</property>

</configuration>

<configuration>

<property>

<name>dfs.nameservices</name>

<value>ns1</value>

</property>

<property>

<name>dfs.ha.namenodes.ns1</name>

<value>nn1,nn2</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn1</name>

<value>hadoop01:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn1</name>

<value>hadoop01:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn2</name>

<value>hadoop02:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn2</name>

<value>hadoop02:50070</value>

</property>

<property>

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

<value>qjournal://hadoop03:8485;hadoop04:8485;hadoop05:8485/ns1</value>

</property>

<property>

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

<value>/home/apps/hadoop-2.9.1/journaldata</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.ns1</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>

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/root/.ssh/id\_rsa</value>

</property>

<property>

<name>dfs.ha.fencing.ssh.connect-timeout</name>

<value>30000</value>

</property>

</configuration>

vi slaves

hadoop03

hadoop04

hadoop05

上面其实就已经将active namenode给配置好了

然后将这个hadoop拷贝到其他4台机器上去：scp -r /home/apps/ hadoop02:/home/apps/

5、启动hadoop集群

（1）在hadoop03、hadoop04、hadoop05上启动journal nodes集群：

sbin/hadoop-daemon.sh start journalnode

（2）格式化namenode（hadoop01上执行）

hdfs namenode -format

hdfs namenode -bootstrapStandby

（3）格式化ZKFC（hadoop01上执行）

hdfs zkfc -formatZK

（4）启动hdfs集群（hadoop01上执行）

sbin/start-dfs.sh，这个脚本会自动在hadoop01和hadoop02上启动一个namenode，同时启动一个ZKFC，然后会自动在hadoop03、hadoop04、hadoop05上分别启动一个datanode

6、查看hadoop集群

在hadoop01和hadoop02访问50070端口即可

namenode启动流程，http server、rpc server、一堆后台线程、检查safe mode。等待集群里的各个datanode也启动，启动之后来找namenode进行注册，让namenode感知到集群里有哪些datanode。datanode启动流程讲完，datanode往namenode注册讲完，namenode内部如何管理这些datanode信息的

datanode启动的入口

我们如果启动了DataNode的话，那么在几台机器上是不是就是有一个DataNode进程，其实他的意思就是说通过一个DataNode的main()方法启动了一个DataNode进程，所以如果我们要找DataNode启动入口的话，其实说白了，就是找DataNode类的main()方法

类注释会给你解释一些源码里面的概念，说明源码的一些东西，如果你可以先读一些注释对你后面读懂源码会好很多

name是如何启动，NameNode类，main()方法

org.apache.hadoop.hdfs.server.namenode，包，明显这个包下面包含了namenode所有核心的类和源码，找到了NameNode.java

NameNode，进程的名字默认是跟类名是一样的，哪个类的main方法启动进程，进程的名字就是那个类的类名

擅长看注释，我们的课程前面讲解的都是一些基本的原理，粗的，从很大的角度来讲解的，但是当你真正开始看源码的时候，一定要学会看注释，加上自己深厚的技术功底，要学会推测

namenode是用来管理文件系统的namespace，主要管理两类数据（tables，数据表格，数据映射）：

（1）文件 -> block的映射（namespace），上传的文件会被切割为多个block，每个是128m，如果不足128m，就是分配一个block就可以了，这份映射数据，namespace

block -> datanode的映射（inodes），每个block都是会分配给一个datanode的，每个block还有多个副本，默认是3个副本，也就是每个block会对应多个datanode，这个映射数据就是inodes

namespace是存放在磁盘上的，同时也存放在内存里

inodes（那些block是在哪些datanode上），这份数据，是在每次namenode启动之后，由datanode来汇报过来的，而且datanode在运行期间会每隔一段时间都汇报一下自己有哪些block，汇报给namenode

人家说的很清楚了，NameNode主要是三个组件：NameNode、NameNodeServer、FSNamesystem。

NameNode主要是干两件事情，第一件事情就是处理一些配置属性，第二件事情就是启动NameNodeServer，NameNodeHttpServer，NameNodeRpcServer，这个两个server其实就是对外监听了某个端口，datanode、dfsclient都可以通过两个server来请假端口，发起这个请求。干了两件事情

NameNode Server：NameNodeHttpServer、NameNodeRpcServer，http请求，rpc请求，datanode、dfsclient都可以来请求他们，比如说请求这两个server来创建目录、上传文件、下载文件、dfsadmin管理性的操作

FSNamesystem：负责管理元数据，其实是他在管理文件系统的操作，或者内存里元数据的管理，诸如此类的一些操作

是不是直接就可以搜索到main()方法，NameNode.main()，就是namenode启动的一个入口，start-dfs.sh这种shell脚本一定就是执行这个类的这个main()方法，从main()方法开始入口来分析namenode的启动流程

直接在源码里写注释，作为源码的讲解，同时一边写注释，一边分析，一边来画图

在读源码的时候，如果发现他在获取一个参数的值，很简单，直接通过参数的名字，在hdfs-defaults.xml里可以找到参数的默认值，此外，还可以知道，如果你在hdfs-site.xml里覆盖这个配置爹值，可以影响源码运行的效果

NameNode.main() -> 启动入口

NameNode.initialize() -> 初始化的核心逻辑

-> startHttpServer() -> 启动自己的http服务器，后面接收别人发送过来的http请求

namenode启动的流程

NameNode.initialize() -> 初始化的核心逻辑

-> startHttpServer() -> 启动自己的http服务器，后面接收别人发送过来的http请求

-> loadNamesystem() -> 初始化FSNamesystem，磁盘加载fsimage和edits进行内存合并

-> createRpcServer() -> 初始化rpc server，接收别人调用过来的rpc请求

-> startCommonServices()

-> FSNamesystem.startCommonServices()

-> checkAvailableResources() -> 检查磁盘空间是否有100m来写入edits log

-> setBlockTotal() -> 检查是否要进入safemode

-> BlockManager.active() -> 启动一堆后台线程，做一些任务

-> NameNodeRpcServer.start() -> 启动rpc server

NameNode.join() -> 等待rpc server结束，理论上永远不会结束的

（1）namenode + datanode的启动和初始化的过程

（2）datanode向namenode去近进行注册

（3）datanode需要在运行期间不断的发送心跳给namenode

整个这个流程大家一定要清楚几个要点：

（1）namenode和datanode初始化的过程里，各种server以及核心的线程，后面才能去分析集群运行的各种流程，以及各种http、rpc、数据流各种接口是如何调用的，NameNodeRpcServer，BPServiceActor

（2）在注册的过程中，最最核心的要掌握的一点是datanode是如何注册到namenode的内存里的，注意一下那些map数据结构

（3）心跳，最主要掌握hdfs的一个架构特点，datanode跟namenode交互的主要途径，就是两个，一个是发送心跳，另外一个是汇报block，namenode每次都是在接收到心跳的时候去判断是否要给datanode下发什么任务的

下一讲，HeartbeatManager每隔一段时间检查一下当前datanode是否在持续的发送心跳，如果说datanode一段时间没有发送心跳的话，此时就会将datanode从自己内存管理的各种数据结构中给移除掉

在8个月中期规划结束了以后，我们会在长期规划里，对hdfs这块，会主要补充他的大规模的运维、管理、升级、高级功能的使用以及源码的一些东西

1、集群部署

大家觉得大规模的hdfs集群部署，比如几十台，或者几百台，能这么手工的安装么，这不是搞死人了，太麻烦了

其实业内基本上都是用cdh的，也就是cloudera hadoop，他还有一个cm的组件，cloudera manager，这个东西提供了很多企业级的部署、管理以及监控的功能

所以以后要结合cdh来讲解，HA高可用架构、federation联邦架构，都要放在一起上，以及jvm参数，和一些关键性参数的设置

2、监控报警

现在手头有了集群了，然后不就可以使用了吗？其实说白了，用hdfs，一般不可能我们手工乱搞的，弄什么命令行，一般都是flume等系统采集了数据之后放到hdfs上分布式存储，然后hive、spark等技术对上面的数据离线分布式计算

或者是hbase基于hdfs架构一套分布式NoSQL数据库

所以一般hdfs对我们来说，不是以用为主，而是以运维和管理为主，你只要搭建好flume数据采集系统，数据自己就会每天不断的采集进来放到hdfs集群上去的

然后我们要考虑第一件重要的事情，就是集群监控

你要监控几十台几百台机器的cpu、内存、网络、磁盘等资源的使用，设定报警阈值，比如说某台机器磁盘使用率达到了70%了，就要进行扩容了，否则可能就满了

这个事情可以结合cdh，还有一些监控系统来做

3、集群扩容

现在可以日常监控集群的资源使用了，那比较重要的一个运维操作是什么呢？其实说白了，就是扩容，如果一个集群一开始是30台机器，慢慢的很多台机器的磁盘空间都快满了，接着就要扩容，采购加入更多的机器了

那么此时就需要考虑如何进行扩容，包括扩容完了之后，刚加入进去的机器磁盘是空的，你怎么做一个负载均衡？将磁盘快满的机器上的block分配到新加入的机器上来？要用rebalancer机制，hdfs的rebalancer机制

除此之外，还有机器的下线，举个例子，现在是采购了一批新的高配置的机器，替换掉原先低配置的机器，如何先下线机器，再上线机器？

4、集群升级

hadoop版本可是不断的在变更的，比如hadoop 2.9.0，hadoop 2.9.1，hadoop 3.0.2，你如何将一个大规模集群一次性完成整个集群的版本升级？

5、日常运维

数据迁移：如果现在要将一个集群的数据迁移到另外一个集群，应该如何进行？

检查坏块：fsck工具

6、异常排查

资源监控是长期放在那儿的，但是一般都没什么事儿，而集群扩容、集群升级，这些都是偶发性的工作，可能一年都碰不上几次

但是平时日常比较关键的，考验一个大数据平台工程师的技术水平的，就是如何能够在一个集群出技术故障的时候，比如说报错了、jvm oom了，这些异常的时候，对这些异常进行监控，配置集群打印出完整的异常日志，同时有深厚的源码功底，可以结合异常堆栈，以及源码的分析，迅速定位和解决问题

这是最考验一个大数据平台工程师的技术水平的

7、数据权限

对一个大规模的公司而言，比如说，举个例子，一些大的互联网公司，都有很多部分，好比说滴滴，有顺风车团队、专车团队、支付团队、豪华车团队，类似这种，每个业务团队都会有自己的数据

怎么管理不同团队的数据所在目录，以及各个团队的数据资源进行隔离

保证不同的团队之间不能跨团队使用别人的数据？以及如何保证只有授权过的用户才能访问集群里的数据？不是随便哪个阿猫阿狗都可以访问的？那要结合kerbernets来做

还有就是其实应该是可以对每个团队能够使用的资源做一个限制的，如果不限制的话，就会导致可能某个业务团队错误的上传了大量的数据，导致占满了集群的磁盘资源

而且还应该统一收口hdfs集群的访问口径，比如说命令行类的，只能是大数据平台工程师，运维集群的同学来做这个操作，然后Java API类一般是不允许随意访问的，还有RestFUL API类的，可以提供出去，但是必须要授权收口子

8、数据加密

数据落地在存储的时候如何做加密，对数据敏感的公司，对网路传输的过程中的数据还要进行加密

9、新功能支持

hdfs新版本其实支持很多的新功能，举个例子，比如说集中式缓存，数据快照，短路本地读，还有就是基于内存的存储机制，可以加快性能，纠删码的特性，等等吧，很多的需要研究透彻这些新版本新功能，同时考虑如何在业务中有什么使用场景

10、更多的源码剖析

配合着上面那些企业级的运维管理的操作，需要阅读更多的源码

federation架构的源码，集中式缓存的源码，认证授权的源码，ViewFS的源码，WebHDFS的源码，加密的源码，内存/对象等新存储机制的源码，fsck的源码，rebalancer的源码

hdfs，yarn，mapreduce，他们在市面上也有一些源码分析的书籍

普遍的都有一个问题，就是说什么呢？他们其实全部都是对每一块源码，都是一块一块的去讲解和研究

如果大多数的人看那些源码分析的书籍，基本上可能都看不懂

他会拉一个章节，namenode源码，就是在namenode里一块一块的讲解，相当于是针对源码里的某一个包下面的每个类，挑选重要的类，一个一个类的来分析

这样子，除非是那些已经看懂了源码的人，才能够看懂这种源码分析的书

要不然那些没看过源码的人，看你在这里一个一个类的讲解，全部懵，绝对是看不懂的，我看源码，基本上从来不看那些源码分析的书籍，通过看他的书来看懂源码的话，我觉得基本上不太现实

我们的源码分析课，绝对不能这样子，我们必须是按照一个技术在使用时候的一个场景来分析，hdfs源码，并不是说先针对namenode包下面每个类一个一个来分析，场景，hdfs集群启动的场景

hdfs集群，在启动的时候，会干哪几件事情

（1）namenode会启动一个进程，初始化一些发送和接收网络请求的服务

（2）datanode会启动一个进程，初始化一些发送和接收网络请求的服务

（3）datanode一定会请求namenode，跟他进行联系，将自己作为一个数据节点注册到namenode里去

（4）namenode也一定会接收到集群中各个datanode发过来的加入集群的请求，在内存中维护好集群中有哪些datanode

（5）到此为止，hdfs集群完成启动，namenode和datanode就连成一片了

创建目录（管理元数据）

namenode HA

客户端上传一个文件

客户端下载一个文件

各个场景下可能会有一些隐藏的小机制

hdfs核心源码，你就看完了，很多无关紧要的源码，不看也没关系。你就可以跟我说的那样子，反过来去github上研究研究，2.7，2.8，2.9的版本都做了哪些源码的改动。hadoop 3.x现在，官方都没标记为stable

等后面hadoop 3.x稳定了以后，国内大量的公司开始大规模的升级了之后，我们可以再来补充hadoop 3.x里面的架构的稳定性，千万不要跟个愣头青乱用最新版本，要用人家说明的稳定版本

hadoop 2.6.5这个版本，确实比较老了，最新的更新都是2年前，就说明社区基本上现在已经不推荐使用这个版本了，读源码没有关系，核心源码还是一样的，差不多的，修复bug、优化功能

hadoop 2.7和hadoop 2.8这个版本，社区现在还是在不断的维护的，不断的更新版本，在世界、国内都是比较广泛在使用的一些版本

hadoop 2.9，最近比较新的版本，修复bug、优化功能

hadoop 3.x，去年开始发布测试版本，hadoop 3.0.0第一个非测试版本是在去年年底发布的，到今年其实一直在更新，频繁，而且在官网上人家还没有标志hadoop 3.x版本是stable，hadoop 3.x版本还没有在行业里得到大规模的部署和应用

其实hdfs的文件系统权限模型，就是跟普通的linux上的文件权限模型是差不多的

说穿了，对任何一个目录和文件，其实都针对3种角色提供了3种权限，就是owner，group，还有others，3种权限是r（读），w（写），x（执行）

drwxr-xr-x，/user目录的权限

d：代表的是说这是一个directory，目录

rwx：针对owner，针对root这个用户，对于/user这个hdfs上的目录，可以读、写、执行（查看，创建文件）

r-x：针对group，supergroup这个权限组，可以读，可以执行，没有w写的权限

r-x：针对除了owner和group以外的其他的用户，也是有r和x两个权限

默认情况下来说，你创建一个文件或者目录的时候，owner就是执行命令的进程的用户，group就是父目录的group

而且后续你对文件或者目录执行操作的时候，hdfs就会检查你是否拥有这个权限

如果在根目录下创建目录或者文件，默认owner就是linux上的用户（root）；group就是supergroup；如果在某个目录里创建子目录或者文件，那么owner也是linux上的用户；group默认是上级目录的group

来做一些实验试一下吧：

hadoop fs -mkdir /logs

hadoop fs -ls /

看看我们刚刚创建的/logs目录的权限呢？分别展示出了对owner、group、others的不同的权限，以及显示出了所属的owner和group

hadoop fs -chown zhss:data /user

hadoop fs -ls /

再看一下，owner是不是变成zhss了，group变成data了

然后来修改一下权限，比如说让data这个group，是可以对logs目录有写权限的

hadoop fs -chmod -R g+w /user

hadoop fs -ls /

u+w，o+w，w+r，w+x，都可以，这个很形象了，就是给u/g/o，owner，group，others加上某个权限，w、r、x，如果要删除某个权限，那你就u-w，这样子就可以了

试一下给owner减掉w权限，这样就不能在目录里创建文件了，试一下看看

hadoop fs -chown -R u-w /logs

hadoop fs -ls /logs

hadoop fs -put hi.txt /logs/hi.txt

首先到这一步为止，咱们是不是就很ok了，已经明白了hdfs的权限模型，而且可以修改hdfs上的目录和文件的ower和group，还可以对owner、group修改授权了，而且你会发现如果权限不满足，是执行不了对应的操作的

顺带提一嘴，这个文件系统的授权数据，也是元数据，也是放在namenode上的

先简单了解一下，就行，这个课程的重点不是企业级的运维管理，所以主要就是把原生的hdfs支持的一些功能简单说一下

中期规划里，重点专注在源码这块

长期规划中，专门花时间来讲解大规模集群的运维和管理，那得用cdh来做

现在其实咱们已经初步的把hdfs的整体架构原理，包括分布式架构、HA架构、联邦架构、元数据管理机制、分布式存储机制、集群容错机制，都学好了，而且都部署了一个小集群，同时还尝试着操作了一下，试了试这个所谓的分布式文件系统是怎么用的，人家除了命令行以外，还提供了java api

接下来就要开始分析hdfs的核心源码了

大家可以想一下，我们的hdfs核心源码的分析思路是如何的

1、将hadoop 2.9.1源码导入eclipse / intellij idea中，一边看源码， 一边写注释，一边画图，这种感觉是特别好的

2、namenode启动的过程

3、datanode启动的过程

（1）datanode是如何发送心跳的

（2）namenode是如何管理和感知到集群里的那些datanode的

4、hdfs管理元数据的过程

（1）创建一个目录来举例

（2）hdfs client（命令行）是如何跟namenode通信的

（3）hdfs在内存、edits log、fsimage是如何来管理元数据的

5、hdfs HA架构的源码（一）

（1）active namenode是如何将edits log同步给journal node

（2）journal node是如何存储edits log

（3）standby namenode是如何从journal node拉取edits log数据的

（4）standby namenode是如何在内存、edtis log、fsimage里来管理元数据的

6、hdfs checkpoint机制的源码

（1）standby namenode是如何发起checkpoint操作的

（2）standby namenode是如何将合并后的fsimage发送给active namenode来替换的

（3）两个namenode在checkpoint的时候，edits log文件是如何清空的

7、hdfs HA架构的源码（二）

（1）ZKFC如何监控active namenode宕机的

（2）ZKFC是如何基于zookeeper完成一个主备选举的

（3）ZKFC是如何通知standby namenode进行主备切换的

（4）standby namenode是如何将自己切换为active namenode的

8、hdfs client上传文件的过程

（1）hdfs client端是如何与namenode通信，说要上传大文件的

（2）namenode是如何将大文件拆分为多个block的

（3）namenode是如何为block分配3个副本的

（4）hdfs client是如何将文件拆分为多个block，依次上传到各个datanode的

（5）datanode是如何管理block的：读写数据接口、磁盘存储机制、block数据结构

（6）datanode是如何以pipeline管道的模式，依次复制block到下一个datanode的

（7）上传文件全流程的一个容错机制，就比如上传到一半出错，如何自动恢复

9、datanode是如何给namenode定时汇报自己的block report

10、hdfs client下载文件的过程

（1）hdfs client端是如何namenode通信来尝试下载一个大文件的

（2）namenode是如何配合下载的，对大文件的每个block如何挑选一个副本出来

（3）hdfs client是如何与各个datanode通信下载block的

（4）datanode是如何提供block下载的

（5）下载文件全流程的一个容错机制，如果下载到一半出错，如何自动恢复

（6）对于破损块的一个校验和检查的机制，以及自动重新下载的机制

11、namenode集群容错机制

（1）namenode如何发现datanode宕机

（2）datanode宕机之后，在宕机期间，副本数量变少了，是如何使用存活的其他副本来提供高可用的数据读取的

（3）namenode如何通知别的datanode复制新的block副本，让副本数量保持为3

12、二次开发hdfs源码

（1）一个讨巧的办法：查看官方github的pull request以及merge记录

（2）pull request都是为了优化模块代码，或者解决某个bug

（3）看那些merge掉的pull request，就能看到人家是如何优化代码的

（4）然后面试就可以说自己在某个旧的版本也发现了类似的问题，自行优化了代码，配合核心源码的阅读来说，这就非常更有杀伤力了

面向的是两类人：

（1）已经起码实际工作干过大数据，1年以上的人，大数据工程师

（2）一直在做java开发的，java工程师

对hdfs的运维、管理、使用、高级功能，统统没讲，hdfs如何用，我们是不讲的，专注每个大数据技术的核心源码的阅读和剖析上

（1）摸过和玩儿过，接触过真实的大数据集群的，哪怕是小集群，十几台机器的，公司生产集群部署、管理、监控和运维，都做了哪些事情。那么对于你来说，当务之急，其实不是说系统的学习如何搞一些操作性的东西，深入阅读大数据系统的核心源码，甚至掌握二次开发源码的能力和思路，真实干过大数据1~5年的人

（2）更加合适了，java工程师，学大数据，重点是学两块：一块使用性的技能，你可以在java架构里使用一些大数据技术；大数据技术就是世界上最先进，最大规模的大型企业级分布式系统，你学大数据技术，学到的不是怎么用，是各种大型分布式系统的源码、架构以及原理，对java工程师而言，提升自己的分布式系统的架构能力、技术内功修为，特别好的，java架构师而言，也是特别好的，竞争力，熟悉大数据技术者优先，是加分项

中期规划里，将hdfs技术研究到目前的一个水平，基本就可以了，因为大家都是有一定经验的工程师，其实基本都在很多中小公司做过一些简单的大数据项目，摸过生产环境，大致也都知道一些东西怎么弄的

所以现在大家对于出去面试，缺的，其实就是技术的深度，首先把hdfs内核的核心源码搞清楚，同时能找几个源码二次开发的场景，出去直接可以说，你是负责大数据平台的工程师，专门就是研究各个技术的源码，优化代码，解决源码的bug

然后日常的一些运维管理，其实无法都是一些操作性的东西罢了，大家或多或少都知道一些，暂时先说自己知道的一些东西就可以了

这就足够了，已经在增强技术竞争力了

为什么讲hadoop 2.6.5版本的源码？因为其实hadoop 2.x系列的源码，核心流程和机制都是差不多的，各个版本都是修复一些bug，以及加一些新功能罢了，何况hadoop 2.6版本是国内非常广泛应用的版本，2.6.5版本也是2年前推出的版本，所以其实这个版本不能算最新，也不能算最旧

何况hadoop 2.6.5版本源码研究透彻了，其实核心机制以及原理都是差不多的，后面你自己研究其他版本的源码，2.7，2.8，2.9的源码，也基本没什么太大的障碍，而且一般不建议研究所谓最新版本的源码

其实研究hadoop源码，挑选一个近2年的稳定版本即可，未必是最新的版本，最新的版本很多公司里可能都不一定已经落地使用了，没那个必要，尤其是大的互联网公司，集群规模太大，一般版本升级都会滞后社区发展1年是正常的

因为如果你研究了2.6.5的源码，你后面可以自己研究2.7,2.8,2.9每个版本分别都在改良什么东西，代码是怎么变动的，可以自己看一看hadoop代码的变迁，而且实际上大家自己百度一下hadoop 2.7和2.8的新特性，就会发现，其实基本上都是大量的修补和完善，核心机制和流程是差不多的

这个大家后面，我会给大家看，2.7，2.8各个版本都在做哪些代码的优化和修改，然后研究透了可以说是自己在公司里二次开发源码做的修改，这其实才是我最大的一个用意，就是你看下人家2.7，2.8的代码的改动，出去面试，你可以说你用的是2.6版本，但是你们公司内部大量的打patch，修复bug、优化功能，源码级别透彻的理解，以及源码级别二次开发，修复开源版本的bug、优化开源版本的功能的能力

你要是掌握到这个水平的话，出去面试，天，到BAT去面试，也是超级大的杀手锏

1、安装maven以及配置Nexus私服

首先window上要装好maven，别告诉我maven怎么装怎么用都不知道，包括国内阿里云仓库的配置，Nexus私服的搭建，具体参见架构班课程的maven部分

2、安装protobuf

（1）自己找一个目录，解压缩protobuf-2.5.0.tar.gz和protoc-2.5.0-win32.zip两个压缩包

（2）protoc-2.5.0-win32.zip解压缩之后是一个protoc.exe的程序，把这个protoc.exe拷贝到c:\windows\system32目录下，同时拷贝到protobuf-2.5.0.tar.gz解压缩后的protobuf-2.5.0目录的src目录下去

（3）在windows命令行，进入protobuf-2.5.0\java目录下，执行“mvn pakcage”命令，进行打包，会在target目录下生成一个protobuf-java-2.5.0.jar

（4）在windows，我的电脑->右击>属性->高级系统设置->环境变量，在PATH环境变量中，加入protoc.exe所在的目录，比如我电脑上的目录是：F:\development\hadoop-source-code，因为这个目录下有protoc.exe

（5）在window命令行中执行，protoc --version，可以看到安装成功

3、解压缩hadoop 2.6.5的源码

将hadoop-2.6.5-src.tar.gz解压缩，这就是hadoop 2.6.5的源码了

4、hadoop-maven-plugins打包和部署

从windows命令行进入hadoop-2.6.5-src/hadoop-maven-plugins目录，然后执行mvn install，此时会将hadoop-maven-plugins下载依赖，打包以及部署到私服里去

5、为eclipse导入做准备

从windows命令行进入hadoop-2.6.5-src目录下，然后执行mvn eclipse:eclipse -DskipTests，其实就是执行eclipse插件，为源码导入eclipse做准备罢了，此时会下载一大堆的依赖

6、将hadoop源码导入eclipse中

对于看一个开源项目的源码，建议是在eclipse中切换一个workspace，在File->switch workspace那里可以换一个目录，这样这个workspace就是专门用来看hadoop源码就可以了

然后使用import existing project，将hadoop源码导入eclipse即可，此时会做一些处理，稍作等待就行了

7、解决hadoop源码报错

（1）大量的Access Restriction

第一种报错是说大量的讲，XX type不存在，XX method不存在，等等的，这个就是设置一下eclipse不要报error，报成warning就可以了，按照下面的步骤设置就行了

Windows-> Preferences -> Java -> Compiler -> Errors/Warnings ->Deprecatedand trstricted API -> Forbidden reference (access rules): -> change towarning

（2）AvroRecord cannot be resolved to a type

下载java/avro-tools-1.7.7.jar

地址：http://mirrors.hust.edu.cn/apache/avro/avro-1.7.7/java/avro-tools-1.7.7.jar

在windows命令行，进入下面的目录：

hadoop-2.6.5-src\hadoop-common-project\hadoop-common\src\test\avro

执行下面的命令：

java -jar F:\development\hadoop-source-code\avro-tools-1.7.7.jar compile schema avroRecord.avsc ..\java

上面那个命令执行完了之后，就会在：

hadoop-2.6.5-src\hadoop-common-project\hadoop-common\src\test\java目录下的

org\apache\hadoop\io\serializer\avro目录下，生成一个AvroRecord类，你得自己通过windows文件窗口，到上面说的这个目录下，找到这个类，拷贝一下，放到eclipse里对应的包下面去，hadoop-common/src/test/java，找到org.apache.hadoop.io.serializer.avro包，放进去就行了，不然eclipse没法自己刷出来

（3）EchoRequestProto cannot be resolved

在win命令行进入下面的目录：

hadoop-2.6.5-src\hadoop-common-project\hadoop-common\src\test\proto

执行命令：protoc --java\_out=..\java \*.proto

然后你从win文件窗口进入到下面的目录：

hadoop-2.6.5-src\hadoop-common-project\hadoop-common\src\test\java

找子目录：org\apache\hadoop\ipc

会发现这个下面新出来一个protobuf目录，里面有TestProtos.java和TestRpcServiceProtos.java两个类，这都是生成的

然后你就在eclipse里，按照上述对应的目录，在hadoop-common/src/test/java下，自己新建一个org.apache.hadoop.ipc.protobuf包，然后将那两个类拷贝进入，就ok了

（4）Project 'hadoop-streaming' is missing required source folder

这个很简单，右击hadoop-streaming项目，然后在里面的Java Build Path里面，有一个source tab，里面有一个栏目是报错的，把那个报错的栏目remove掉，就ok了

8、完成源码导入

要记住，最终你的源码，应该是绝对不报错的，到这里为止，阅读hadoop源码的环境就ok了，后面我们看hdfs、yarn、mapreduce的源码，都可以在这里看了

1、准备5台linux虚拟机

都是CentOS 7操作系统，64位，每台机器都是1个cpu，1G内存，安装JDK 1.8

（1）使用课程提供的CentOS 7镜像即可，CentOS-7-x86\_64-Minimal-1611.iso。

（2）创建虚拟机：打开Virtual Box，点击“新建”按钮，点击“下一步”，输入虚拟机名称为hadoop01，选择操作系统为Linux，选择版本为Red Hat-64bit，分配1024MB内存，后面的选项全部用默认，在Virtual Disk File location and size中，一定要自己选择一个目录来存放虚拟机文件，最后点击“create”按钮，开始创建虚拟机。

（3）设置虚拟机网卡：选择创建好的虚拟机，点击“设置”按钮，在网络一栏中，连接方式中，选择“Bridged Adapter”。（搭建虚拟机还真不是我专门要讲的一块内容，我一般常用的搭建虚拟机做实验的方式就是桥接网络）

如果你的笔记本电脑在家里用wifi的ip地址配置了一下虚拟机的网络，然后你的电脑拿到外面咖啡馆里去，或者公司里去，ip地址变化了之后，就会导致你的之前搭建的那套虚拟机环境就没法用了

如果你希望无论到哪里，ip地址变化了以后，虚拟机都可以正常工作，不需要重新配置网络，那么你就去百度一下，虚拟机如何配置网络，可以更换环境的时候，不需要重新配置，NAT如何配置

桥接，换了一个网络环境，ip地址都换了，5台虚拟机重新配置一下里面的网卡的一些ip地址什么的，就ok了

（4）安装虚拟机中的CentOS 7操作系统：选择创建好的虚拟机，点击“开始”按钮，选择安装介质（即本地的CentOS 7镜像文件），按照课程选择后自动安装即可

（5）安装完以后，CentOS会提醒你要重启一下，就是reboot，你就reboot就可以了。

（6）配置网络

（如果重启网卡没看到人家给你分配的ip地址，你自己应该是要去连一个路由器就可以了）

vi /etc/sysconfig/network-scripts/ifcfg-enp0s3

先让它动态分配一个ip地址

ONBOOT=yes

service network restart

ip addr

再设置静态ip地址

BOOTPROTO=static

IPADDR=192.168.31.250

NETMASK=255.255.255.0

GATEWAY=192.168.31.1

service network restart

ip addr

配置DNS

检查NetManager的状态：systemctl status NetworkManager.service

检查NetManager管理的网络接口：nmcli dev status

检查NetManager管理的网络连接：nmcli connection show

设置dns：nmcli con mod enp0s3 ipv4.dns "114.114.114.114 8.8.8.8"

让dns配置生效：nmcli con up enp0s3

（7）配置hosts

vi /etc/hosts

配置本机的hostname到ip地址的映射

（8）配置SecureCRT

此时就可以使用SecureCRT从本机连接到虚拟机进行操作了

一般来说，虚拟机管理软件，virtual box，可以用来创建和管理虚拟机，但是一般不会直接在virtualbox里面去操作，因为比较麻烦，没有办法复制粘贴

SecureCRT，在windows宿主机中，去连接virtual box中的虚拟机

收费的，我这里有完美破解版，跟着课程一起给大家，破解

（9）关闭防火墙

systemctl stop firewalld.service

systemctl disable firewalld.service

关闭windows的防火墙

后面要搭建集群，有的大数据技术的集群之间，在本地你给了防火墙的话，可能会没有办法互相连接，会导致搭建失败

（10）配置yum

yum clean all

yum makecache

yum install -y wget

（11）安装JDK

1、将jdk-8u131-linux-x64.rpm通过WinSCP上传到虚拟机中

2、安装JDK：rpm -ivh jdk-8u131-linux-x64.rpm

3、配置jdk相关的环境变量

vi ~/ .bashrc

export JAVA\_HOME=/usr/java/latest

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

source .bashrc

4、测试jdk安装是否成功：java -version

（12）在另外4个虚拟机中安装CentOS集群

按照上述步骤，再安装三台一模一样环境的linux机器

另外三台机器的hostname分别设置为hadoop02，hadoop03，hadoop04，hadoop05

安装好之后，在每台机器的hosts文件里面，配置好所有的机器的ip地址到hostname的映射关系

比如说，在elasticsearch01的hosts里面

192.168.31.250 hadoop0101

192.168.31.xxx hadoop02

192.168.31.xxx hadoop03

192.168.31.xxx hadoop04

（13）配置5台CentOS为ssh免密码互相通信

首先在三台机器上配置对本机的ssh免密码登录

ssh-keygen -t rsa，生成本机的公钥，过程中不断敲回车即可，ssh-keygen命令默认会将公钥放在/root/.ssh目录下

cd /root/.ssh

cp id\_rsa.pub authorized\_keys

将公钥复制为authorized\_keys文件，此时使用ssh连接本机就不需要输入密码了

接着配置三台机器互相之间的ssh免密码登录

使用ssh-copy-id -i hostname命令将本机的公钥拷贝到指定机器的authorized\_keys文件中

2、hdfs HA集群部署规划

namenode，就相当于是在eclipse中新建一个工程，取个名字叫做namenode，在工程里面开发代码，写好代码之后，就可以打包，在linux服务器上，用java -jar给他启动，namenode就是这个意思

namenode就是一个java工程，写完里面的代码，打包，在linux上部署，java -jar启动，出来一个进程，namenode其实就是一个系统，启动的时候就是一个进程

hadoop01：active namenode，ZKFC

hadoop02：standby namenode，ZKFC

hadoop03：datanode，journal node，QuorumPeerMain

hadoop04：datanode，journal node，QuorumPeerMain

hadoop05：datanode，journal node，QuorumPeerMain

3、部署zookeeper集群

在hadoop03、hadoop04、hadoop05三台机器上部署zookeeper

tar -zxvf zookeeper-3.4.9.tar.gz

mv zookeeper-3.4.9 zookeeper

vi ~/.bashrc

export ZOOKEEPER\_HOME=/usr/local/zookeeper ?export PATH=$PATH:$ZOOKEEPER\_HOME/bin

source ~/.bashrc

vi zoo.cfg （$ZOOKEEPER\_HOME/conf）

dataDir=/home/data/zookeeper ?dataLogDir=/home/log/zookeeper?server.1=hadoop03:2888:3888

server.2=hadoop04:2888:3888 ?server.3=hadoop05:2888:3888

mkdir -p /home/data/zookeeper

mkdir -p /home/log/zookeeper

cd /home/data/zookeeper

echo 1 > myid

scp -r /usr/local/zookeeper hadoop04:/usr/local ，修改环境变量，在hadoop04将myid的内容改为2 （echo 2 > myid）

scp -r /usr/local/zookeeper hadoop05:/usr/local，修改环境变量，在hadoop05将myid的内容改为3 （echo 3 > myid）

三台机器上执行：zkServer.sh start

查看集群状态 ?1、jps（查看进程） ?2、zkServer.sh status（查看集群状态，主从信息）

4、部署hadoop集群

先在hadoop01上操作

tar -zxf hadoop-2.9.1.tar.gz -C /home/apps/

vim /etc/profile

export JAVA\_HOME=/home/apps/jdk1.8.0\_111

export HADOOP\_HOME=/home/apps/hadoop-2.9.1

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

source /etc/profile

vi $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

export JAVA\_HOME=/home/apps/jdk1.8.0\_111

vi core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://ns1/</value>

</property>

<property>

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

<value>/home/apps/hadoop-2.9.1/tmp</value>

</property>

<property>

<name>ha.zookeeper.quorum</name> <value>hadoop01:2181,hadoop02:2181,hadoop03:2181</value>

</property>

</configuration>

<configuration>

<property>

<name>dfs.nameservices</name>

<value>ns1</value>

</property>

<property>

<name>dfs.ha.namenodes.ns1</name>

<value>nn1,nn2</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn1</name>

<value>hadoop01:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn1</name>

<value>hadoop01:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn2</name>

<value>hadoop02:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn2</name>

<value>hadoop02:50070</value>

</property>

<property>

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

<value>qjournal://hadoop03:8485;hadoop04:8485;hadoop05:8485/ns1</value>

</property>

<property>

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

<value>/home/apps/hadoop-2.9.1/journaldata</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.ns1</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>

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/root/.ssh/id\_rsa</value>

</property>

<property>

<name>dfs.ha.fencing.ssh.connect-timeout</name>

<value>30000</value>

</property>

</configuration>

vi slaves

hadoop03

hadoop04

hadoop05

上面其实就已经将active namenode给配置好了

然后将这个hadoop拷贝到其他4台机器上去：scp -r /home/apps/ hadoop02:/home/apps/

5、启动hadoop集群

（1）在hadoop03、hadoop04、hadoop05上启动journal nodes集群：

sbin/hadoop-daemon.sh start journalnode

（2）格式化namenode（hadoop01上执行）

hdfs namenode -format

hdfs namenode -bootstrapStandby

（3）格式化ZKFC（hadoop01上执行）

hdfs zkfc -formatZK

（4）启动hdfs集群（hadoop01上执行）

sbin/start-dfs.sh，这个脚本会自动在hadoop01和hadoop02上启动一个namenode，同时启动一个ZKFC，然后会自动在hadoop03、hadoop04、hadoop05上分别启动一个datanode

6、查看hadoop集群

在hadoop01和hadoop02访问50070端口即可

NameNode.initialize() -> 初始化的核心逻辑

-> startHttpServer() -> 启动自己的http服务器，后面接收别人发送过来的http请求

-> loadNamesystem() -> 初始化FSNamesystem，磁盘加载fsimage和edits进行内存合并

-> createRpcServer() -> 初始化rpc server，接收别人调用过来的rpc请求

-> startCommonServices()

-> FSNamesystem.startCommonServices()

-> checkAvailableResources() -> 检查磁盘空间是否有100m来写入edits log

-> setBlockTotal() -> 检查是否要进入safemode

-> BlockManager.active() -> 启动一堆后台线程，做一些任务

BlockManager内部，启动了一些后台线程：

（1）PendingReplicationBlocks.start() -> PendingReplicationMonitor：后台监控那些进行replication复制block被pending住的一些操作

（2）DatanodeManager.activate() -> DecommissionThread + HeartbeatThread：启动datanode manager，这个东东看起来就是用来管理datanode的这么一个组件，docommisson是后台监控datanode状态的一个线程，如果datanode死了就将datanode下线；HeartBeat，心跳，是不是后台监控或者管理跟datanode之间的心跳的线程

（3）ReplicationThread.start()：猜测是不是复制block相关的后台线程

之前给大家反复强调了hdfs的namenode是怎么管理元数据的，说白了就是fsimage和edits两个文件，还有内存里的实时fsimage数据，对吧

现在我们可以通过hdfs提供的工具，来查看一下人家namenode上的fsimage和edits两个文件

hdfs oev -p xml -i edits -o edits.xml

上面的指令就可以查看edits文件，同时输出为一份edits.xml文件，我们可以查看文件里的内容，就可以看到一条一条的元数据变更日志了

hdfs oev -p stats -i edits -o edits.stats

还有这个指令是说对edits日志文件统计下，输出一份统计文件，大概内容就是说，比如有多少个创建目录的操作了，多少个删除文件的操作了，类似这种

http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsEditsViewer.html

同样是建议大家对这种工具类的使用，自己多参考一下官方手册就好了，非常简单的，不涉及到什么技术性的东西

bin/hdfs oiv -i fsimage

这个命令，就会启动一个web服务器，然后我们可以浏览器请求这个临时启动的web服务器，来查看fsimage，也就是整个hdfs集群的元数据，这个还是蛮有用的

bin/hdfs dfs -ls webhdfs://127.0.0.1:5978/，你可以用这个命令直接对本地的那个web服务器使用webhdfs协议来请求，查看一些元数据

hdfs oiv -p XML -i fsimage -o fsimage.xml

这个命令可以将fsimage导出为一份xml格式的文件供我们查看

http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsImageViewer.html

同样是建议大家对这种工具类的使用，自己多参考一下官方手册就好了，非常简单的，不涉及到什么技术性的东西

现在的fsimage里面是没东西的，创建的目录，什么时候才会将edits log中的内容合并到fsimage文件里去啊？是要等待standby namenode每隔1小时执行一次checkpoint操作，他会合并两个文件，然后进行fsimage文件的替换

datanode启动了之后一定会去向namenode完成一个注册

BPServiceActor其实是一个线程，他的第一次注册的行为就是在线程的run()方法里面

（1）namenode + datanode的启动和初始化的过程

（2）datanode向namenode去近进行注册

（3）datanode需要在运行期间不断的发送心跳给namenode

整个这个流程大家一定要清楚几个要点：

（1）namenode和datanode初始化的过程里，各种server以及核心的线程，后面才能去分析集群运行的各种流程，以及各种http、rpc、数据流各种接口是如何调用的，NameNodeRpcServer，BPServiceActor

（2）在注册的过程中，最最核心的要掌握的一点是datanode是如何注册到namenode的内存里的，注意一下那些map数据结构

（3）心跳，最主要掌握hdfs的一个架构特点，datanode跟namenode交互的主要途径，就是两个，一个是发送心跳，另外一个是汇报block，namenode每次都是在接收到心跳的时候去判断是否要给datanode下发什么任务的

（4）心跳检查，每隔30秒检查datanode，如果超过10分钟没发送心跳，就从内存中剔除掉，认为已经宕机了

下一讲，HeartbeatManager每隔一段时间检查一下当前datanode是否在持续的发送心跳，如果说datanode一段时间没有发送心跳的话，此时就会将datanode从自己内存管理的各种数据结构中给移除掉

上面的4点，已经构成了一个最最基础的分布式集群架构的核心要点

（1）master-slave两个角色的所有进程都启动，完成初始化

（2）slave节点必须向master节点进行注册，让master节点感知到集群里有哪些slave节点

（3）在集群运行的过程中，slave节点必须每隔几秒钟就给master发送一次心跳

（4）master必须监控所有的slave节点的心跳，如果一定时间没法心跳，认为slave宕机，从集群中剔除掉

如果是一个空集群刚刚启动的话，其实我们要关注的源码主要就是上面4点，大量的源码，在集群还没有数据的时候，他是不会来执行大量的源码的逻辑的，所以针对那些其他的源码，我们要在后面，构造一个一个的场景来思考，这个是如何处理的

NameNode.initialize() -> 初始化的核心逻辑

-> startHttpServer() -> 启动自己的http服务器，后面接收别人发送过来的http请求

-> loadNamesystem() -> 初始化FSNamesystem，磁盘加载fsimage和edits进行内存合并

-> createRpcServer() -> 初始化rpc server，接收别人调用过来的rpc请求

-> startCommonServices()

-> FSNamesystem.startCommonServices()

-> checkAvailableResources() -> 检查磁盘空间是否有100m来写入edits log

你随便找一台有hadoop目录的机器，但是上面有没有启动namenode、datanode都无所谓

hadoop fs，这个是现在推荐的，也可以用hdfs dfs，都行

然后后面跟着很多的命令，比如说创建一个目录吧，hadoop fs -mkdir -p /user/hadoop/dir1，这个就是说直接创建一个目录层级结构

filesystem namespace

hdfs，底层是一个大规模多台机器组成的分布式存储系统，但是你用起来，就跟好像你在用一个linux机器的文件系统一样

然后就是比如说，创建一个文件，hadoop fs -touchz /user/hadoop/dir1/hello.txt，这就是创建了一个空的文件，里面啥都没有，接着你可以用hadoop fs -appendToFile hi.txt /user/hadoop/dir1/hello.txt，追加本地文件的内容到hdfs文件里去

还可以直接把文件文件上传到hdfs里去，hadoop fs -put hi.txt /user/hadoop/dir1/hi.txt，1GB，那么就会拆分为10个block（128mb），每个block还会有3个副本，3个datanode，每个datanode会有10个block副本，一共有30个block

我其实是忘了去设置每台机器的自己本身的hostname了，这个给大家提一嘴，搭建一个集群简单玩儿一下是没事的，但是在实际的生产集群中，每台机器都是一个hostname的

还可以查看目录下的文件列表，hadoop fs -ls /user/hadoop/dir1

还可以把hdfs的文件下载到本地来，hadoop fs -get /user/hadoop/dir1/hi.txt hi02.txt

如果是一个1GB的大文件，他其实有10个block，此时就会让你选择10个block的一个副本，然后下载下来，将10个bock拼成一个1GB的大文件

一个block对应着磁盘上的一个文件

还有cp、rm、mv、cat等命令，分别可以拷贝、删除、移动、查看文件，其实就跟linux里的一些文件命令是差不多的

我这里就不多讲了，其实推荐大家直接查看人家官网的手册就好了，平时记住几个常用的命令，然后需要的时候查手册就行了

http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.htm

其实大家现在就理解，为啥我先花费了10来讲的时间先说清楚hdfs的整体架构原理，你不了解整体架构原来在这里搞啥子呢，是不是

现在明白了原理，再来搭建hdfs集群，你都知道每个进程部署了是干嘛的，然后现在来简单操作一下hdfs分布式文件存储系统，你也可以知道背后的元数据管理机制，HA高可用机制，checkpoint机制，数据block分布式存储机制，以及数据副本机制，容错机制，等等

上面一堆命令执行的时候，你脑子里大搞都清楚他背后在干什么，你们说是不是

完事儿了，还能通过50070端口查看一下他的元数据，说白了就是filesystem namespace，就是文件层级结构，还有就是每个文件对应的block，block对应的datanode，block的副本，你都大概清楚怎么回事了，对不对

（1）命令行尝试体验了一把hdfs作为一个分布式文件系统是如何操作的，就跟linux文件系统是差不多的

（2）将这些操作跟之前的hdfs架构原理结合起来理解，元数据（目录+文件的层级结构+参数）在namenode管理的，大文件会被切块（block）分布式放到各个datanode上去，每个block有3个副本，直接到各个机器上都直接在磁盘上看到了block副本

（3）我们能否看一下namenode上的元数据呢？

datanode启动之后，完成了注册，他其实就会去第一次汇报自己的全量的block，后面会每隔一段时间都会去汇报一下当前自己的block的情况，以及自己的block的变化，等等吧，汇报各种block的情况

hdfs集群启动的场景

（1）namenode是如何启动的

（2）datanode是如何启动的

（3）datanode是如何向namenode发起注册的

（4）namenode是如何在内存中管理datanode信息的

hdfs集群启动，namenode和多个datanode组成一个集群，这个事情，大家就应该搞定了

namenode是如何启动的，部署hdfs集群的，启动hdfs集群之后，在linux上就会有一个namenode进程，这个进程的名字就是NameNode，很简单，因为这个进程是如何启动的呢？是通过执行一个叫做NameNode这个类的main()方法来启动的

所以说这个进程叫做NameNode

namenode启动，入口，找到一个叫做NameNode的类，里面一定是有一个main()方法的，start-dfs.sh这种shell脚本，里面一定是封装了命令，java -jar命令，就是在执行某个jar包里的某个叫做NameNode类的main()方法

一旦NameNode类的main()方法执行，就意味着会出现一个NameNode进程

hadoop-hdfs，src/main/java一看就是包含了hdfs所有的源码，我们这个课程分析的源码，主要也就是在这个里面，src/main/resources，hdfs-default.xml，在这里看到，这个文件里其实就是默认的hdfs的所有的配置属性

他所有的配置属性，以及属性的默认值，都在hdfs-default.xml文件里

启动hdfs集群的时候，是不是会去配置core-site.xml、hdfs-site.xml、mapred-site.xml，其实我们平时部署hadoop集群的时候，就是在各种xml文件里覆盖掉人家默认的属性的值

<property>

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

<value>file://${hadoop.tmp.dir}/dfs/data</value>

</property>

我们如果在hdfs-site.xml里可以覆盖掉这个默认值

<property>

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

<value>/home/hadoop/user/data/dir</value>

</property>

tools，工具，我们平时用的dfsadmin，hdfs

看完这个包结构，平时你在官网上能找到的可以使用的一些hdfs的功能，那些功能对应的源码都在这个对应的包里

NameNode.initialize() -> 初始化的核心逻辑

-> startHttpServer() -> 启动自己的http服务器，后面接收别人发送过来的http请求

-> loadNamesystem() -> 初始化FSNamesystem，磁盘加载fsimage和edits进行内存合并

-> createRpcServer() -> 初始化rpc server，接收别人调用过来的rpc请求

-> startCommonServices()

-> FSNamesystem.startCommonServices()

-> checkAvailableResources() -> 检查磁盘空间是否有100m来写入edits log

-> setBlockTotal() -> 检查是否要进入safemode

1、准备5台linux虚拟机

都是CentOS 7操作系统，64位，每台机器都是单cpu，1G内存，安装JDK 1.8

启动一个virtual box虚拟机管理软件（vmware，我早些年，发现不太稳定，主要是当时搭建一个hadoop大数据的集群，发现每次休眠以后再重启，集群就挂掉了）

virtual box，发现很稳定，集群从来不会随便乱挂，所以就一直用virtual box了

安装virtual box

用的是什么centos镜像，CentOS比较新的版本是7了，然后服务器上装操作系统的话，内存一般比较大，一般是安装64位的，32位的有一个最大内存4G的限制

（1）使用课程提供的CentOS 7镜像即可，CentOS-7-x86\_64-Minimal-1611.iso。

（2）创建虚拟机：打开Virtual Box，点击“新建”按钮，点击“下一步”，输入虚拟机名称为hadoop01，选择操作系统为Linux，选择版本为Red Hat-64bit，分配1024MB内存，后面的选项全部用默认，在Virtual Disk File location and size中，一定要自己选择一个目录来存放虚拟机文件，最后点击“create”按钮，开始创建虚拟机。

（3）设置虚拟机网卡：选择创建好的虚拟机，点击“设置”按钮，在网络一栏中，连接方式中，选择“Bridged Adapter”。

（4）安装虚拟机中的CentOS 7操作系统：选择创建好的虚拟机，点击“开始”按钮，选择安装介质（即本地的CentOS 7镜像文件），按照课程选择后自动安装即可

（5）安装完以后，CentOS会提醒你要重启一下，就是reboot，你就reboot就可以了。

（6）配置网络

vi /etc/sysconfig/network-scripts/ifcfg-enp0s3

先让它动态分配一个ip地址

ONBOOT=yes

service network restart

ip addr

再设置静态ip地址

BOOTPROTO=static

IPADDR=192.168.31.250

NETMASK=255.255.255.0

GATEWAY=192.168.31.1

service network restart

ip addr

配置DNS

检查NetManager的状态：systemctl status NetworkManager.service

检查NetManager管理的网络接口：nmcli dev status

检查NetManager管理的网络连接：nmcli connection show

设置dns：nmcli con mod enp0s3 ipv4.dns "114.114.114.114 8.8.8.8"

让dns配置生效：nmcli con up enp0s3

（7）配置hosts

vi /etc/hosts

配置本机的hostname到ip地址的映射

（8）配置SecureCRT

此时就可以使用SecureCRT从本机连接到虚拟机进行操作了

一般来说，虚拟机管理软件，virtual box，可以用来创建和管理虚拟机，但是一般不会直接在virtualbox里面去操作，因为比较麻烦，没有办法复制粘贴

SecureCRT，在windows宿主机中，去连接virtual box中的虚拟机

收费的，我这里有完美破解版，跟着课程一起给大家，破解

（9）关闭防火墙

systemctl stop firewalld.service

systemctl disable firewalld.service

关闭windows的防火墙

后面要搭建集群，有的大数据技术的集群之间，在本地你给了防火墙的话，可能会没有办法互相连接，会导致搭建失败

（10）配置yum

yum clean all

yum makecache

yum install wget

（11）安装JDK

1、将jdk-8u131-linux-x64.rpm通过WinSCP上传到虚拟机中

2、安装JDK：rpm -ivh jdk-8u131-linux-x64.rpm

3、配置jdk相关的环境变量

vi .bashrc

export JAVA\_HOME=/usr/java/latest

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

source .bashrc

4、测试jdk安装是否成功：java -version

（12）在另外4个虚拟机中安装CentOS集群

按照上述步骤，再安装三台一模一样环境的linux机器

另外三台机器的hostname分别设置为hadoop02，hadoop03，hadoop04，hadoop05

安装好之后，在每台机器的hosts文件里面，配置好所有的机器的ip地址到hostname的映射关系

比如说，在elasticsearch01的hosts里面

192.168.31.250 hadoop0101

192.168.31.xxx hadoop02

192.168.31.xxx hadoop03

192.168.31.xxx hadoop04

（13）配置4台CentOS为ssh免密码互相通信

首先在三台机器上配置对本机的ssh免密码登录

ssh-keygen -t rsa

生成本机的公钥，过程中不断敲回车即可，ssh-keygen命令默认会将公钥放在/root/.ssh目录下

cd /root/.ssh

cp id\_rsa.pub authorized\_keys

将公钥复制为authorized\_keys文件，此时使用ssh连接本机就不需要输入密码了

接着配置三台机器互相之间的ssh免密码登录

使用ssh-copy-id -i hostname命令将本机的公钥拷贝到指定机器的authorized\_keys文件中

2、hdfs HA集群部署规划

namenode，就相当于是在eclipse中新建一个工程，取个名字叫做namenode，在工程里面开发代码，写好代码之后，就可以打包，在linux服务器上，用java -jar给他启动，namenode就是这个意思

namenode就是一个java工程，写完里面的代码，打包，在linux上部署，java -jar启动，出来一个进程，namenode其实就是一个系统，启动的时候就是一个进程

hadoop01：active namenode，ZKFC

hadoop02：standby namenode，ZKFC

hadoop03：datanode，journal node，QuorumPeerMain

hadoop04：datanode，journal node，QuorumPeerMain

hadoop05：datanode，journal node，QuorumPeerMain

3、部署zookeeper集群

在hadoop03、hadoop04、hadoop05三台机器上部署zookeeper

下载zookeeper 3.4.9：? HYPERLINK "http://archive.apache.org/dist/zookeeper/zookeeper-3.4.9/" \t "https://blog.csdn.net/zz657114506/article/details/\_blank" ?http://archive.apache.org/dist/zookeeper/zookeeper-3.4.9/?，放到/home/apps目录下去

tar -zxf zookeeper-3.4.9.tar.gz

vi /etc/profile

export ZOOKEEPER\_HOME=/home/apps/zookeeper-3.4.9 ?export PATH=$PATH:$ZOOKEEPER\_HOME/bin

source /etc/profile

vi zoo.cfg （$ZOOKEEPER\_HOME/conf）

dataDir=/home/data/zkdata ?dataLogDir=/home/log/zk ?server.1=hadoop03:2888:3888

server.2=hadoop04:2888:3888 ?server.3=hadoop05:2888:3888

mkdir /home/data/zkdata

mkdir /home/log/zk

cd /home/data/zkdata

echo 1 > myid

scp -r /home/apps/zookeeper-3.4.9 hadoop04:/home/apps/ ，然后干一样的事儿?scp -r /home/apps/zookeeper-3.4.9 hadoop05:/home/apps/，然后干一样的事儿

在hadoop04将myid的内容改为2 （echo 2 > myid） ?在hadoop05将myid的内容改为3 （echo 3 > myid）

三台机器上执行：zkServer.sh start

查看集群状态 ?1、jps（查看进程） ?2、zkServer.sh status（查看集群状态，主从信息）

4、部署hadoop集群

先在hadoop01上操作

tar -zxf hadoop-2.9.1.tar.gz -C /home/apps/

vim /etc/profile

export JAVA\_HOME=/home/apps/jdk1.8.0\_111

export HADOOP\_HOME=/home/apps/hadoop-2.9.1

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

source /etc/profile

vi $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

export JAVA\_HOME=/home/apps/jdk1.8.0\_111

vi core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://ns1/</value>

</property>

<property>

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

<value>/home/apps/hadoop-2.9.1/tmp</value>

</property>

<property>

<name>ha.zookeeper.quorum</name> <value>hadoop01:2181,hadoop02:2181,hadoop03:2181</value>

</property>

</configuration>

<configuration>

<property>

<name>dfs.nameservices</name>

<value>ns1</value>

</property>

<property>

<name>dfs.ha.namenodes.ns1</name>

<value>nn1,nn2</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn1</name>

<value>hadoop01:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn1</name>

<value>hadoop01:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn2</name>

<value>hadoop02:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn2</name>

<value>hadoop02:50070</value>

</property>

<property>

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

<value>qjournal://hadoop03:8485;hadoop04:8485;hadoop05:8485/ns1</value>

</property>

<property>

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

<value>/home/apps/hadoop-2.9.1/journaldata</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.ns1</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>

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/root/.ssh/id\_rsa</value>

</property>

<property>

<name>dfs.ha.fencing.ssh.connect-timeout</name>

<value>30000</value>

</property>

</configuration>

vi slaves

hadoop03

hadoop04

hadoop05

上面其实就已经将active namenode给配置好了

然后将这个hadoop拷贝到其他4台机器上去：scp -r /home/apps/ hadoop02:/home/apps/

5、启动hadoop集群

（1）在hadoop03、hadoop04、hadoop05上启动journal nodes集群：

sbin/hadoop-daemon.sh start journalnode

（2）格式化namenode（hadoop01上执行）

hdfs namenode -format

hdfs namenode -bootstrapStandby

（3）格式化ZKFC（hadoop01上执行）

hdfs zkfc -formatZK

（4）启动hdfs集群（hadoop01上执行）

sbin/start-dfs.sh，这个脚本会自动在hadoop01和hadoop02上启动一个namenode，同时启动一个ZKFC，然后会自动在hadoop03、hadoop04、hadoop05上分别启动一个datanode

6、查看hadoop集群

在hadoop01和hadoop02访问50070端口即可

1、准备5台linux虚拟机3616118202

都是CentOS 7操作系统，64位，每台机器都是1个cpu，1G内存，安装JDK 1.8

（1）使用课程提供的CentOS 7镜像即可，CentOS-7-x86\_64-Minimal-1611.iso。

（2）创建虚拟机：打开Virtual Box，点击“新建”按钮，点击“下一步”，输入虚拟机名称为hadoop01，选择操作系统为Linux，选择版本为Red Hat-64bit，分配1024MB内存，后面的选项全部用默认，在Virtual Disk File location and size中，一定要自己选择一个目录来存放虚拟机文件，最后点击“create”按钮，开始创建虚拟机。

（3）设置虚拟机网卡：选择创建好的虚拟机，点击“设置”按钮，在网络一栏中，连接方式中，选择“Bridged Adapter”。（搭建虚拟机还真不是我专门要讲的一块内容，我一般常用的搭建虚拟机做实验的方式就是桥接网络）

如果你的笔记本电脑在家里用wifi的ip地址配置了一下虚拟机的网络，然后你的电脑拿到外面咖啡馆里去，或者公司里去，ip地址变化了之后，就会导致你的之前搭建的那套虚拟机环境就没法用了

如果你希望无论到哪里，ip地址变化了以后，虚拟机都可以正常工作，不需要重新配置网络，那么你就去百度一下，虚拟机如何配置网络，可以更换环境的时候，不需要重新配置，NAT如何配置

桥接，换了一个网络环境，ip地址都换了，5台虚拟机重新配置一下里面的网卡的一些ip地址什么的，就ok了

（4）安装虚拟机中的CentOS 7操作系统：选择创建好的虚拟机，点击“开始”按钮，选择安装介质（即本地的CentOS 7镜像文件），按照课程选择后自动安装即可

（5）安装完以后，CentOS会提醒你要重启一下，就是reboot，你就reboot就可以了。

（6）配置网络

（如果重启网卡没看到人家给你分配的ip地址，你自己应该是要去连一个路由器就可以了）

vi /etc/sysconfig/network-scripts/ifcfg-enp0s3

先让它动态分配一个ip地址

ONBOOT=yes

service network restart

ip addr

再设置静态ip地址

BOOTPROTO=static

IPADDR=192.168.31.250

NETMASK=255.255.255.0

GATEWAY=192.168.31.1

service network restart

ip addr

配置DNS

检查NetManager的状态：systemctl status NetworkManager.service

检查NetManager管理的网络接口：nmcli dev status

检查NetManager管理的网络连接：nmcli connection show

设置dns：nmcli con mod enp0s3 ipv4.dns "114.114.114.114 8.8.8.8"

让dns配置生效：nmcli con up enp0s3

（7）配置hosts

vi /etc/hosts

配置本机的hostname到ip地址的映射

（8）配置SecureCRT

此时就可以使用SecureCRT从本机连接到虚拟机进行操作了

一般来说，虚拟机管理软件，virtual box，可以用来创建和管理虚拟机，但是一般不会直接在virtualbox里面去操作，因为比较麻烦，没有办法复制粘贴

SecureCRT，在windows宿主机中，去连接virtual box中的虚拟机

收费的，我这里有完美破解版，跟着课程一起给大家，破解

（9）关闭防火墙

systemctl stop firewalld.service

systemctl disable firewalld.service

关闭windows的防火墙

后面要搭建集群，有的大数据技术的集群之间，在本地你给了防火墙的话，可能会没有办法互相连接，会导致搭建失败

（10）配置yum

yum clean all

yum makecache

yum install -y wget

（11）安装JDK

1、将jdk-8u131-linux-x64.rpm通过WinSCP上传到虚拟机中

2、安装JDK：rpm -ivh jdk-8u131-linux-x64.rpm

3、配置jdk相关的环境变量

vi ~/ .bashrc

export JAVA\_HOME=/usr/java/latest

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

source .bashrc

4、测试jdk安装是否成功：java -version

（12）在另外4个虚拟机中安装CentOS集群

按照上述步骤，再安装三台一模一样环境的linux机器

另外三台机器的hostname分别设置为hadoop02，hadoop03，hadoop04，hadoop05

安装好之后，在每台机器的hosts文件里面，配置好所有的机器的ip地址到hostname的映射关系

比如说，在elasticsearch01的hosts里面

192.168.31.250 hadoop0101

192.168.31.xxx hadoop02

192.168.31.xxx hadoop03

192.168.31.xxx hadoop04

（13）配置5台CentOS为ssh免密码互相通信

首先在三台机器上配置对本机的ssh免密码登录

ssh-keygen -t rsa，生成本机的公钥，过程中不断敲回车即可，ssh-keygen命令默认会将公钥放在/root/.ssh目录下

cd /root/.ssh

cp id\_rsa.pub authorized\_keys

将公钥复制为authorized\_keys文件，此时使用ssh连接本机就不需要输入密码了

接着配置三台机器互相之间的ssh免密码登录

使用ssh-copy-id -i hostname命令将本机的公钥拷贝到指定机器的authorized\_keys文件中

2、hdfs HA集群部署规划

namenode，就相当于是在eclipse中新建一个工程，取个名字叫做namenode，在工程里面开发代码，写好代码之后，就可以打包，在linux服务器上，用java -jar给他启动，namenode就是这个意思

namenode就是一个java工程，写完里面的代码，打包，在linux上部署，java -jar启动，出来一个进程，namenode其实就是一个系统，启动的时候就是一个进程

hadoop01：active namenode，ZKFC

hadoop02：standby namenode，ZKFC

hadoop03：datanode，journal node，QuorumPeerMain

hadoop04：datanode，journal node，QuorumPeerMain

hadoop05：datanode，journal node，QuorumPeerMain

3、部署zookeeper集群

在hadoop03、hadoop04、hadoop05三台机器上部署zookeeper

tar -zxvf zookeeper-3.4.9.tar.gz

mv zookeeper-3.4.9 zookeeper

vi ~/.bashrc

export ZOOKEEPER\_HOME=/usr/local/zookeeper ?export PATH=$PATH:$ZOOKEEPER\_HOME/bin

source ~/.bashrc

vi zoo.cfg （$ZOOKEEPER\_HOME/conf）

dataDir=/home/data/zookeeper ?dataLogDir=/home/log/zookeeper?server.1=hadoop03:2888:3888

server.2=hadoop04:2888:3888 ?server.3=hadoop05:2888:3888

mkdir -p /home/data/zookeeper

mkdir -p /home/log/zookeeper

cd /home/data/zookeeper

echo 1 > myid

scp -r /usr/local/zookeeper hadoop04:/usr/local ，修改环境变量，在hadoop04将myid的内容改为2 （echo 2 > myid）

scp -r /usr/local/zookeeper hadoop05:/usr/local，修改环境变量，在hadoop05将myid的内容改为3 （echo 3 > myid）

三台机器上执行：zkServer.sh start

查看集群状态 ?1、jps（查看进程） ?2、zkServer.sh status（查看集群状态，主从信息）

4、部署hadoop集群

先在hadoop01上操作

tar -zxvf hadoop-2.9.1.tar.gz

mv hadoop-2.9.1 hadoop

vi ~/.bashrc

export HADOOP\_HOME=/usr/local/hadoop

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

source ~/.bashrc

vi $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

export JAVA\_HOME=/usr/java/latest

vi core-site.xml

在hadoop01~05都创建目录：mkdir -p /home/apps/hadoop/tmp

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://ns1/</value>

</property>

<property>

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

<value>/home/apps/hadoop/tmp</value>

</property>

<property>

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

<value>hadoop03:2181,hadoop04:2181,hadoop05:2181</value>

</property>

</configuration>

vi hdfs-site.xml

<configuration>

<property>

<name>dfs.nameservices</name>

<value>ns1</value>

</property>

<property>

<name>dfs.ha.namenodes.ns1</name>

<value>nn1,nn2</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn1</name>

<value>hadoop01:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn1</name>

<value>hadoop01:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn2</name>

<value>hadoop02:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn2</name>

<value>hadoop02:50070</value>

</property>

<property>

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

<value>qjournal://hadoop03:8485;hadoop04:8485;hadoop05:8485/ns1</value>

</property>

<property>

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

<value>/home/apps/hadoop/journaldata</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.ns1</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>

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/root/.ssh/id\_rsa</value>

</property>

<property>

<name>dfs.ha.fencing.ssh.connect-timeout</name>

<value>30000</value>

</property>

</configuration>

在hadoop01~05上都创建目录：mkdir -p /home/apps/hadoop/journaldata

vi slaves

hadoop03

hadoop04

hadoop05

上面其实就已经将active namenode给配置好了

然后将这个hadoop拷贝到hadoop02~05台机器上去：scp -r /usr/local/hadoop hadoop02:/usr/local/，修改环境变量

5、启动hadoop集群

（1）在hadoop03、hadoop04、hadoop05上启动journal nodes集群：

$HADOOP\_HOME/sbin/hadoop-daemon.sh start journalnode

（2）格式化namenode（hadoop01上执行）

hdfs namenode -format

scp -r /home/apps/hadoop/tmp/ hadoop02:/home/apps/hadoop/

（3）格式化ZKFC（hadoop01上执行）

hdfs zkfc -formatZK

（4）启动hdfs集群（hadoop01上执行）

sbin/start-dfs.sh，这个脚本会自动在hadoop01和hadoop02上启动一个namenode，同时启动一个ZKFC，然后会自动在hadoop03、hadoop04、hadoop05上分别启动一个datanode

6、查看hadoop集群

在hadoop01和hadoop02访问50070端口即可

分析清楚了LeaseManager里面的构造，数据结构，有什么后台线程

下一讲，我们就可以来分析一下，就是说在创建文件的过程中，直接给一个客户端添加了一个文件契约，那个添加文件契约的过程 + 后台线程监控契约的过程，在下一讲，一起来给大家讲一下

csize：校验快的大小，512 bytes

psize：packet，数据包的大小，64mb

chunkSize：packate，一个数据包有多个chunk，516 bytes

chunksPerPacket：每个packet数据包里有多少个chunk，127个chunk

packageSize：每个packet数据包，有65536 bytes，差不多就是64mb

每个文件会被划分为多个block，数据块

默认的话，一个block是128mb，每个block在上传的时候，是由多个packet数据包组成的，每个packet数据包的大小是64mb，每个packet数据包是由多个chunk组成的，一个chunk可以作为一个数据上传的小碎片

chunk是516字节

一个packet数据包里还有多个checksum，校验块，每个是512字节

分下一下，如何为一个客户端添加一个文件契约，后台线程是如何监控各个契约的

LeaseManager，有一个非常经典，值得学习的一个，契约检查的机制，对所有的契约，按照续约的时间在TreeSet里排序，后面检查的时候，每次就挑最老的那个契约来检查，是否超过60分钟

如果超过，就释放契约

再检查第二老的那个契约，如果最老的契约都没过期，那就说明其他的契约肯定都没过期

用这个方法，可以巧妙的避免说，后台线程每隔一定时间，就要把所有的契约都遍历一遍来检查里面的最近一次续约的时间

如果一个客户端申请契约过后，超过1小时，都还没有续约，那么这个契约会被自动释放掉，这是他的一个很重要的机制

之前我们花了很多的时间来研究，通过一个创建目录的一个场景来驱动，我们搞清楚了hdfs是如何管理元数据的，内存元数据，edits log，HA架构下如何同步edits log流，checkpoint，启动时如何合并fsimage和edits log

一般来说会干的下一件事情，就是说上传文件了，这个过程是非常复杂的，而且里面的东西特别的多，我们应该好好来研究一下，一周的课都讲不完，差不多得2周，才能把上传文件的全过程的源码分析完毕

hdfs客户端干很多很多复杂的事儿，namenode也要干不少事儿，datanode那块涉及到复杂的block存储管理，很多事儿，都很复杂，这周的课程，把hdfs客户端和namenode干哪些事情分析清楚，就不错了

举个例子，比如说在java系统中，你需要将一个1GB的大文件，一边读取大文件的输入流，一边获取一个针对hdfs的输出流，流对拷的方式，一边从本地的1GB大文件里读取数据，一边将读取到的数据写入hdfs输出流

输出流就会将数据通过一大套极为复杂的机制，将一个1GB的大文件分散存储在集群中的多台datanode机器上，可能会存储比如说10台机器上，每台机器上存储一个128MB的文件，一部分，每个128MB的block文件还有另外2个副本在其他的机器上

保证说，某台机器如果宕机之后，还有其他的副本是可以用的

在java代码里，会用DistributedFileSystem这个类，来创建一个针对hdfs的输出流，将源源不断的读到的数据，写入hdfs输出流里面去

上传文件的源码核心入口流程

（1）构造一个DFSOutputStream输出流

（2）跟namenode申请当前这个文件的契约，获取一个全局锁的东西

（3）返回一个HDFSDataOutputStream流包裹一下底层的DFSOutputStream流

（4）使用HDFSDataOutputStream流源源不断的对hdfs输出大文件的数据

从下一讲开始，我们就从构造DFSOutputStream开始，一点一点来研究他里面的东西和原理

为下一讲的开头铺垫一些内容

先看一点点DFSOutputStream的构造的过程，决定下一讲来讲解哪些东西

我们其实把创建文件的流程分析清楚了

（1）在内存的文件目录树中，创建一个文件，INodeFile节点

（2）写了一条edits log操作日志

（3）通过LeaseManager添加了一个客户端对这个文件的契约

给下一讲做个预告，干脆下一讲，我们就是来分析一下，在create一个文件的时候，他是如何对应着直接就给客户端添加了一个文件的契约

除了这个以外，我们应该先分析一下LeaseManager这个东西，包括这个东西的初始化的构造以及他是如何进行初始化的，namenode在启动的时候，是如何初始化LeaseManager的，他初始化的过程干了哪些事情？

分析一下，LeaseManager在初始化的时候，是否开启了什么后台线程？

hdfs所有的源码分析的书籍，我都看过，如果没有一个好的视频课程，像我这样手把手带着你来看源码，分析源码，画图，写注释的话，其实如果你看任何一个hdfs源码剖析的书籍，基本都看不懂

书籍，全部都是，一个类一个类的分析，完全不是场景驱动，按照一个正常的集群工作流程来剖析的，所以那种书籍会导致，适合的是，你先自己看懂了源码之后，你再去看那些书籍，就可以看懂了，可以查漏补缺

看看人家对源码的分析，跟你自己理解的源码，是否有什么不同，或者是有某块东西，人家理解到位了，但是可能你没看到，你没注意到

完全跟着我，场景驱动，一点一点的给你分析源码、写注释、画图，一点点的把源码给你讲清楚，我相信大家只要跟着我的源码课程，一定可以把大数据底层的技术原理，都理解清楚的，只不过视频，每个视频都值得看至少2遍

因为我分析的时候，思路是比较快的，你不一定思路可以跟得上我，不一定你看一遍全都理解消化了

看完视频以后，至少你自己要按照这个思路，来调试自己的源码，写注释，加深印象，自己来画图，彻底把这块源码吃透，掌握

分下一下，如何为一个客户端添加一个文件契约，后台线程是如何监控各个契约的

LeaseManager，有一个非常经典，值得学习的一个，契约检查的机制，对所有的契约，按照续约的时间在TreeSet里排序，后面检查的时候，每次就挑最老的那个契约来检查，是否超过60分钟

如果超过，就释放契约

再检查第二老的那个契约，如果最老的契约都没过期，那就说明其他的契约肯定都没过期

用这个方法，可以巧妙的避免说，后台线程每隔一定时间，就要把所有的契约都遍历一遍来检查里面的最近一次续约的时间

如果一个客户端申请契约过后，超过1小时，都还没有续约，那么这个契约会被自动释放掉，这是他的一个很重要的机制

下一讲预告，我们就是要来分析一下，应用程序是如何通过FSDataOutputSTream的write方法，将数据写入dataQueue队列的，而且是以chunk + packet的数据格式，写入那个队列，DataStreamer才能获取数据写给datanode

hdfs客户端就是准备跟第一个block所在的datanode建立socket连接，然后就是获取socket输入流，以及准备读取block的数据

比较重要的一点，就是在BlockReader初始化的过程中，会给datannode发送一个消息，说自己打算要准备read block了

用屁股猜猜，datannode收到那个消息，一定会初始化一个对应的组件，专门负责提供本地的block的数据给hdfs客户端去读，而且会初始化好对应的本地磁盘上的block文件的输入流，准备读取数据

BlockSender不断的发送packet，最后用一个空packet来结尾

一次read就会读出来一块packet的数据，很明显这个packet的数据就可以交给应用系统来写入应用系统本地的磁盘文件里

什么时候会开启一个新的block呢？

每次read读取一个packet放入一个buf数组，返回给应用系统可以写入磁盘文件；不断的读取packet，不断的累加对应的pos位置；当发现这个pos的位置超过了一个block应该有的大小之后，就会开启下一个block的读取

上述步骤依次循环往复，直到所有的block全部读取完毕

如果读取过程中有问题会如何呢？我可以当做交给大家的作业去思考一下，屁股猜测一下都知道，肯定是如果从某个datanode读取block数据失败的话，就会将那个datanode放入一个黑名单中，下次就不读了

会对一个block换一个副本所在的datanode继续来读取

初探，成功了，read()就是马上就是要对第一个block，连接到对应的一个datandoe，获取输入流，准备开始跟datanode读取第一个block的数据

hdfs客户端是怎么跟namenode进行通信加载到一个大文件对应的block列表的

文件下载最后一系列的操作相对比较简单，而且如果你理解了文件上传之后，文件下载算是比较简单的，很好理解，所以我们的图不会画的那么的复杂

下一讲，我们就来分析一下DFSInputStream对应的read()方法，是如何不断的读取一个一个的block的数据的

（1）hdfs集群的运行：master-slave架构，集群启动，注册与心跳

（2）元数据的管理：fsimage & edits log，创建目录等等大量的元数据的修改，如何管理

（3）大文件的上传：分布式存储，各种通信的细节，副本的复制

（4）集群的容错：block上报和管理，副本数量的监控，自动下发复制指令

（5）大文件的下载：分别依次下载block，最终组装成一个大文件

算是初步的对分布式存储系统的架构和原理就了解的比较清楚了，hdfs的源码极为的繁琐和复杂，如果要搞清楚每个细节的话

大数据这块第一个项目实战，源码二次开发，hdfs 2.6.5版本的源码，进行bug fix、优化性能、开发功能，掌握到如何对一个大数据的开源系统进行源码级别的二次开发

文件下载，要比文件上传简单很多，而且很多东西都是非常的类似的，讲的时候，我们会稍微快一些，而且会稍微简化一些，不会讲的那么的细。比如说你之前上传了一个1TB的大文件到HDFS上去，拆分成了很多个block在存储

所以你下载1TB的大文件的时候，其实是从各个datanode所在的机器上，下载这个文件对应的各个block文件，最后组合成一个大文件

文件下载的这个入口，其实是在FileSystem.open()方法，会返回一个输入流，你只要不断的从输入流里读取数据，就可以不断的拿到这个大文件的数据然后写入本地磁盘文件

DFSInputStream -> HDFSDataInputStream -> FSDataInputSream

DFSInputStream在构造和初始化的时候，有一个很关键的操作，其实是去从namenode加载我们需要的那些blocks

datanode如何接收readBlock的请求 -> 初始化的操作 -> 准备提供block的读取

大胆的猜测，BlockSender组件初始化的过程，我们估计就是在创建好针对本地磁盘的block文件的输入流

下一讲：datanode的BlockSender是如何一边读取本地磁盘文件的数据，一边发送给客户端的BlockReader组件的

BlockSender是如何发送数据给客户端来读取的

下一讲：客户端是如何读取一个一个的packet，拼成一个block，最终受到一个空的packet知道一个block传输结束了

block数据肯定是会写入客户端本地的磁盘文件的，如果一个block传输结束了，此时一定会按照一样的流程来读取下一个block，用一个BlockReader连接到下一个block对应的datandoe上去

读取下一个block的数据，依次写入本地的磁盘文件中

如果说建立管道失败的话

（1）抛弃掉这个block，做一个申请block的反向操作

（2）将那个建立连接失败的datanode加入一个hdfs客户端的excludedNodes列表中

（3）如果建立管道失败，最多可以重试3次

（4）后面在重试申请block以及建立管道的时候，就会传递过去excludedNodes列表

（5）namenode再分配新block的时候，就不会绑定有故障的datanode了

我们倒没有分析机架感知的算法，我们剖析了一下namenode分配一个新的block的机制和流程，下一讲我们看看DataStreamer是如何跟datanode建立数据管道的

write\_block操作的处理过程，就会完成一些基本的初始化的事情，以及直接完成数据管道pipeline的建立，包含了多个datanode数据管道

单独开一大块讲datanode本地磁盘的存储，给大家讲解一大堆的概念，FSVolumn、FSDataset，磁盘文件组织机制，学好了这套东西，你就对分布式文件系统的机制了解了很多了

一个FSVolumn，大概相当于是本地磁盘上的一个文件组，包含了一堆的block文件，datanode上有多个FSVolumn

预告：发送空block给第二个datanode，第二个datanode再次发送空block给第三个datanode，其实也是干一样的事情

上一讲，讲到

（1）在namenode内存文件目录树里，创建一个文件

（2）在namenode服务端开启了一个文件契约，在客户端开启了一个续约线程

（3）初始化好DFSOutputStream供客户端写入数据

（4）启动DataStreamer线程监听dataQueue队列中的数据包（packet）

万事俱备，之前东风

DFSOutputSream接收到源源不断写入的数据流之后，就会将数据按照512字节，切割成一个一个的chunk，将每个chunk写入一个packet（数据包），每个数据包默认大小是64mb，如果一个数据包写满了之后

就会将这个数据包发送到dataQueue队列中去

后面的事儿就是DataStreamer在干了，客户端的程序就不用管了，DataStreamer负责将pakcet发送到datanode上去

一个block会有多个packet，一个block写满了之后，就会换一个block

基本上其实也会涉及到block的切割

FSDataOutputStream，里面包裹了一个DFSOutputStream，调用它们的write()方法，往里面远远不断的写入数据，上传一个大文件到hdfs上面去，就是说一边开一个文件输入流，读取数据，一边源源不断的写入DFSOutputStream里面去

block默认都是128mb，如果一个packet是64mb，一般来说一个block就是2个packet

预告下一讲，下一讲应该是讲DataStream，就需要取出来一个一个的packet，他需要先干一件事情，跟namenode申请新的block，知道block的几个副本放在哪些datanode上，他还得跟那个datanode建立一个数据管道

然后你的packet来了，才能往那个datanode上去写

如果一个block已经写入了两个packet了，接收到了一个空的packet，DataStreamer应该就知道，当前block满了，他会跟namenode申请下一个block，接下来的packet就会往新的block里面去写

DataStreamer是如何向namenode申请一个新的block的

必须知道一个新的block，这个block的3个副本放在了哪些机器上面，他需要跟其中的一个datanode建立一个数据管道，把packet往这个数据管道里发送

一个datanode会把block复制给另外一个datanode，以此类推，直到3个副本

关于这个文件上传和文件下载，如果出去面试，一方面是问你底层的原理，各种细节，另外一个，就是问你文件上传的过程中会有哪些错误，出现了错误，人家怎么来处理的？这是分布式系统设计的精华所在

文件下载，他的底层原理和细节，如果文件下载的过程中，遇到了一些错误，该怎么来处理？这种东西，都是一些大厂面试大数据工程师的时候，人家要问的一些东西，都是比较难的一些东西，很考验人的技术功底

机架感知的设计思想，一个机架放2个datanode，另外一个机架放一个datanode

为什么第一个机架要放2个datanode呢？因为hdfs客户端在上传block的时候，是先给第一个datanode上传block，第一个机架里的第一个datanode接收到了block以后，会复制后发送给同一个机架里的另外一个datanode

这个是保证同机架传输，性能较高

为什么要在另外一个机架也放一个datanode呢？如果第一个机架宕机了，那么第二个机架是没问题的，还有一个datanode可以提供一个副本，保证数据的高容错、高可用

我刚才把这块源码又仔细看了一下，我发现我之前规划课程的时候，低谷了这块的复杂度，所以还以为一讲可以讲完

处理那个write\_block操作的时候

（1）通过socket输入流，从hdfs客户端读取那个空的block

（2）将空的block存储到本地，每个block是一个本地的磁盘文件，是不是应该就是要要初始化一个本地的磁盘文件，即使不初始化，也应该做一些事情

（3）第一个datanode需要将block发送给第二个datanode，以此类推，第二个datanode也应该发送block给第三个datanode

（4）如果三个datanode都完成了block的传递和接收，证明这个三个datanode组成的一个pipeline的一个数据管道，就建立成功了

接下来，hdfs客户端才可以开始把属于一个block的packet，都源源不断的写给第一个datanode，第一个datanode接收到一个packet存储在自己本地磁盘文件里，然后发送packet给第二个datanode，以此类推，第二个datanode再发送给第三个datanode

这一讲我们还没讲到建立起来一个数据管道

我们主要是讲到，在这里会尝试跟第一个datanode建立socket连接，尝试初步通信确认连接创建成功

如果连接没有创建成功的话，这个异常处理的机制，下一讲来讲

另外的话呢，就是这一讲，我们可以给大家同时来讲一下，datanode是如何监听到socket连接，发送过来的创建连接的请求的，如何返回响应的

先总结一下，正常情况下，申请了一个空块，这个空块发送出去，各个datanode传递空块，依次初始化好自己的BlockReceiver以及磁盘文件，建立好pipeline数据流管道，可以准备好让DataStreamer来写packet数据了

如果建立pipeline数据流管道处问题了，他会怎么来处理

chunk（512字节 + 4字节checksum） -> packet（127个chunk，64mb）-> block（2个packet + 1个空packet） -> block是一个虚的概念

datanode，BlockReceiver是如何接收一个一个的packet，接收到了以后是如何写入本地的block磁盘文件（blk\_00000001）

hdfs客户端而言，对block的最后一个packet，是空包，发送出去，然后一直卡着，等待这个空包是处理成功

datanode而言，接收到了空包，就会执行finalizeBlock()方法，彻底结束掉接收这个block需要的各种IO资源、线程资源，关闭对磁盘文件的输出流，将block对应的数据加入管理，通知namenode新增了一个block

各个datanode都会针对空包，释放掉block对应的各种资源，通知namenode自己收到了一个block

hdfs客户端收到了通知之后，就会将空包从ackQueue里挪出来，此时hdfs客户端也会关闭IO资源、关闭线程、释放资源、重置状态

我们就要考虑，就好比说，primary datanode已经接收成功了packet了，结果是管道中的第二个datanode他在接收packet的过程中失败了

datanode数量 <= replication / 2，跟namenode申请一个新的datanode补充进来，也就是默认情况下，保证管道里必须至少有2个datanode，如果少了一个副本的话，后面namenode会自动补齐的

后续的操作的步骤其实就是跟上一讲讲的是几乎一样的了

（1）建立管道都失败了，重新申请一个新块和新的管道

（2）如果发送packet给primary datanode故障了，记录故障datanode，从列表中移除，重新用新的管道来传输，记住，可能副本数量 < 3

（3）如果在管道传输过程中失败了，同上

（4）在容错的过程中，如果datanode list中仅仅剩下了1个了，会发现小于等于replication / 2的，此时是会跟namenode申请一个新的datanode，保证这个管道里的datanode的数量不能太少

有一个同学，几次例子了，为什么一定要研究源码，你手上有源码，出去面试大公司还是小公司，才有谈资，才有底气，上来先聊源码

面一家中型的互联网公司，技术不见得说多好，人家就问，hdfs上传文件的过程中，出现了故障是如何恢复的？

预告：下一讲，PacketReceiver如何将数据传输给下游的datanode节点

将获取到packet数据包开始传输给datanode

错了，我忘了，应该先讲一下数据管道传输结果的监听组件

下一讲，datanode数据管道是如何一层一层的上传自己的packet已经写入本地磁盘成功的，一直到hdfs客户端感知到一个packet有3个副本都在各个datanode里存储成功了，才认为是一个packet传输成功

（1）建立数据管道的过程如果失败了应该如何来容错

（2）将packet发送给第一个datanode时候如果失败了如何来容错

1）将ackQueue中的packet全部放入dataQueue，重新再发送一遍

2）关闭PacketResponsder线程

3）关闭所有的针对那个故障的primary datanode的Socket和IO资源

4）从datanode列表中摘除那个故障的datanode

5）如果满足一定的条件，才会重新找namenode来在datanode列表中加入一个新的datanode，但是会传递过去故障的datanode，让namenode在选择的时候不再选择故障的datanode（一般这个条件不太容易满足）

6）接着针对新的datanode列表，建立Socket连接和IO流资源，建立好数据管道

7）找namenode来更新block对应的datanode pipeline、时间戳

此时datanode列表中就只有2个datanode，我们的这个block此时就只会有2个副本，相当于他本来应该有3个副本，结果现在只能传输2个副本了，namenode是会有一个自动检测故障的机制

如果namenode发现某个block的副本数量没有达到要求的数量，他会自动找某一个datanode来复制一份这个block，保证他有3个副本的

下一讲预告：一个block的2个packet都处理成功了，DataStreamer此时就会发送一个空包给下游的pipeline，下游的pipeline如果收到了空包之后，此时就会关闭各个节点的那些block相关的接收线程、本地的IO资源，就认为这个block已经传输结束了

DataStreamer也是一样的，如果空白都传输完毕了，此时就会重置一些资源，标识当前这个block传输完毕，下一次再次接收到一个packet的时候，又会重新去申请一个新的block以及对应的新的3台datanode

下一讲预告：datanode将接收到的packet写入本地磁盘的blk文件里，会去做checksum校验，同时写入本地磁盘的block对应的checksum校验文件里