# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/159\_再测试基于已经分布式多副本存储的文件是否可以读取/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/147\_客户端在传输大数据包时如果出现拆包问题应该如何处理？/笔记.doc**

进行数据传输的时候必须要考虑的一个底层的细节问题

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/149\_再来仔细看看NIO到底是如何实现客户端连接请求的监听的？/笔记.doc**

我们已经讲完了请求的发送，现在就是进行到NIO服务端，我们为什么要把这块代码仔仔细细的讲解一般，再进行测试，是因为现在我们基于NIO写的代码已经完全实现了工业级了，代码是很复杂的

所以说我们需要把这块代码仔细的给讲解清楚才可以

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/154\_基于测试用例走读一下文件下载的正常流程中的代码/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/150\_当接收到客户端发送过来的请求时是如何基于多线程进行处理的？/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/156\_在文件下载的代码流程中加入测试需要观察的一些日志/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/145\_基于JDK注释深入分析NIO客户端发起连接的过程以及原理/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/148\_动手完成解决客户端大数据包类拆包问题的代码实现/笔记.doc**

把代码来写一下

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/143\_探探我对于JDK中的NIO API设计的一些缺陷的看法以及改进的思考/笔记.doc**

我们继续来梳理咱们的NIO代码，这个过程急不得，我一定要把这块东西给大家讲深讲透，理解，掌握，这块东西真的是一个程序员最最底层的核心技术

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/157\_清理一下本地磁盘中的文件准备一个干净的环境用来测试/笔记.doc**

就是 可以实现，数据节点是可以部署多台机器的，Master节点会负责整个集群的管控，客户端上传文件的时候走的是一个分布式存储，就是这个文件仅仅会在某个机器上存储一份，但是同时为了高可用，会做一个双副本的冗余

每个文件在其他机器上都会有一个副本存在

Master可以收集和管理所有的集群里的元数据，文件目录树，各个机器上的存储的情况

文件下载，会从一个文件的多副本中随机选择一个出来，从某台机器上下载文件到本地

工业级的分布式系统的元数据管理机制、分布式存储架构、多副本冗余的机制、分布式集群中的数据读取的机制、工业级的网络通信/磁盘读写/内存管理/并发控制，跟我反馈可能有一些少数的小bug

在公司里做项目，如此复杂的系统，有一些小bug是很正常的，实际上那些bug需要依靠大量的测试，在测试环境真实部署一套集群出来，然后模拟生产环境的各种情况来进行测试，才能知道具体是怎么回事

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/158\_先测试文件上传后的分布式存储以及多副本冗余等机制是否正常/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/152\_接着分析一下服务端接收文件的拆包粘包解决方案的代码/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/151\_先分析一下服务端接收文件的正常情况下的核心流程/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/144\_NIO编程时如何避免对同一个网络事件进行多次重复处理？/笔记.doc**

很多同学都问过我的一个问题

本质上也是人家NIO API设计的一个不足之处，如果你获取了一个SelectionKey出来，此时就应该自动把他给删除掉，不应该继续留在自己内部的selectedKeys集合里了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/146\_工业级的网络数据包封装：如何设计数据包的包头？/笔记.doc**

在直接基于NIO进行网络编程的时候，必须是得自己设计数据包的格式，你一次调用SocketChannel.write写出去的是你自己封装好的一个完整的数据包，你必须考虑好这个包里有哪些数据

而且包里的数据是按照什么样的格式和顺序来放置的

kafka的数据包的设计，也就是类似这样而已，其实也没什么特别的东西

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/160\_对分布式文件系统的架构总结：元数据管理、分布式存储、多副本冗余/笔记.doc**

也就是说，来看看目前为止整体的架构

（1）元数据管理机制：纯内存来进行管理，editslog机制，backup备份机制，fsimage合并机制，实现了元数据的初步的高可用的架构（冷备份实现的）

（2）分布式存储架构：海量文件分布式的存储在很多机器上，每台机器存储一部分的文件，后续就可以实现可伸缩性，可以进行集群扩容，就可以无限的扩容让更多的文件可以存储在集群里

（3）多副本冗余架构：因为是这样子的，文件上传的时候多副本冗余存储，如果说某台机器宕机，不要紧，这些文件有其他的副本在别的机器上，这样的话就可以保证了数据容错性，高可用性

（4）内存管理机制、并发控制机制、磁盘读写、网络通信

（5）FastDFS（分布式文件系统，c语言写的，不太适合Java技术栈来使用，分析一下他的架构上的优点），TFS（淘宝内部的分布式文件系统，架构设计），吸取人家架构中的精华和优秀的架构设计思想

（6）高可用、高并发、高性能、可伸缩

非常非常具有工业级的价值和意义的，完全可以说你在自己公司里在基础架构的团队，负责带了几个小弟从0开始研发了这么一套分布式存储系统，支撑了公司上亿图片的存储，里面的架构设计的思想，都是极为有价值的

FastDFS太烂了，他是很难说是真的可以去读源码，分析他的架构，如果出了问题，你很难说去维护和改动他的源码

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/153\_在客户端动手编写一个文件下载的测试用例代码/笔记.doc**

在客户端动手来写一个文件下载的测试用例的代码

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/155\_在文件上传的代码流程中加入测试需要观察的一些日志/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/142~160资料/142\_结合代码再谈NIO设计思想：到底什么是同步非阻塞？/笔记.doc**

这个步骤是必须得做的，之前大量的重构了我们的文件上传和下载的细节上的通信的代码，NIO相关的，代码重构量实在是太大了，所以必须得在这里详细的把文件的上传和下载，分别梳理一下代码流程

基于原生的NIO来编程，其实主要要处理的就是说类拆包和类粘包的两种，底层的网络通信的模式是一种数据流的方式，各种各样的数据在底层会转换为电路信号不停的发送给你，所以你要知道如何切分开来不同的数据包

拆包是很典型的一个问题，就是一个文件，11.2MB，结果一次OP\_READ就读取到了5.6MB，另外一部分在下一次OP\_READ才可以读取到，这个就是所谓的典型的类拆包的问题，一个完整的数据需要多次读到

其实在真正的工业级的网络通信的程序里，主要要处理的就是这块东西，其他的一些，调参优化，参数调优，很多参数都还好，在kafka的源码分析的课程里，就是重点讲解了kafka源码中是如何进行工业级的NIO编程的

关于阻塞和非阻塞的关系，之前已经给大家讲解过了

经历 过了一连串的编程之后，大量的代码实战之后，在这里必须从头对里面的NIO这块的代码做一下梳理和总结，反复加深大家的印象，彻底打通网络通信这块的任督二脉

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/122\_如果想要从分布式系统读取一个图片文件应该怎么做？/笔记.doc**

文件的上传都给搞定了，假设一个图片，先从NameNode获取两台机器作为多副本的存储机器，得根据机器的存储资源排序，优先选择那些存储资源数据量最少的机器，可以保持机器之间的存储量的平衡

接下来就可以依次通过NIO跟两台机器来通信，把文件给上传上去

两台机器需要主动去通知NameNode说自己收到了这个文件，每个文件有几个副本，每个副本是在哪台机器上。DataNode突然重启，快速重启，此时不需要管；如果NameNode自己突然重启，会要求各个DataNode重新上报存储；如果是集群从0开始重启，NameNode优雅关闭，接下来自然重启之后，就会重建所有的元数据

NameNode会fsimage和editlog重建自己的内存文件目录树，各个DataNode还会上报自己的存储信息上去，NameNode就知道这个分布式文件系统的完整文件目录树，有哪些目录和哪些文件，每个文件有几个副本，每个副本在哪些机器上

NameNode都是知道的

创建目录、上传文件 -> 元数据管理机制 + 图片多副本分布式存储机制

文件读取

（1）首先当然是应该找NameNode去获取这个文件某个副本所在的机器，其实比如说有两台机器有两个副本，此时NameNode只要告诉人家客户端一个机器就可以了

（2）对NameNode而言，每次对于一个文件给人家返回某个副本所在机器的时候要采取随机的策略，这次给客户端返回的是机器01，下次最好就是机器02，很多客户端都要读取一个图片，此时会把流量均匀打散在机器01和机器02上

（3）就是应该跟那个机器进行NIO网络通信，完全可以在之前的那个代码基础之上来改一改就可以了，之前客户端从本地磁盘读取一个图片成为字节数组的数据格式，然后通过NIO把这个图片给服务器发过去，服务器读取这个图片写入自己本地磁盘；客户端先通过NIO发送一个请求过去，意思就是说，我想要读取某个图片，服务器就是从本地磁盘读取这个图片，然后给客户端写回去，客户端就读取这个图片就可以了

目录的管理（元数据机制），文件上传，文件读取，高可用架构

NameNode突然宕机了，怎么进行故障的恢复，可能会导致丢失多少数据；DataNode突然宕机了，对文件上传会有影响吗，对文件读取有影响吗，图片的副本变少了；宕机的DataNode后来恢复了，图片副本过多

分布式文件系统核心的功能就全部都做好了，抽一周的时间，给大家来讲一下FastDFS、TFS两个类似的分布式文件系统他们的架构原理，高并发、高性能、高可用、可伸缩的架构设计的理念，优缺点，吸取人家的优点，站在架构设计的角度，来考虑一下我们的DFS，SS-DFS，架构是否还有一些改进的 空间在里面

除了一些功能上的缺失，删除目录，文件重命名，shell脚本/python脚本封装一些脚本命令，留一些作业，你自己完全可以参考人家去做一下。log4j，logback，把完善的日志打印出来，Metric统计，提供一些HTTP接口出去

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/129\_初步搭建起来客户端读取图片的NIO网络通信程序的架子/笔记.doc**

先梳理出来一个网络通信的架子，Kafka源码剖析的课，Kafka客户端就是基于NIO进行网络通信的，也是类似这样的思路，但是他有一些地方做了工业级的封装，生产的一些参数配置，解决了生产问题的拆包和粘包的问题

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/126\_为客户端定义一个对外提供的读取文件的接口/笔记.doc**

给客户端来定义一个接口

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/132\_实现数据节点根据请求定位以及读取本地磁盘文件的逻辑/笔记.doc**

我们已经接收到了读取文件的请求，此时就应该解析出来对应的文件名，然后就是根据相对路径的文件名转化为本次磁盘的绝对路径，接着针对本地磁盘文件构造输入流，准备从本地磁盘文件读取数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/135\_在NIO中处理完一次读写请求之后应该如何处理事件的监听？/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/123\_为Master节点设计一个用于获取图片副本所在机器的RPC接口/笔记.doc**

第一个件事情，必然是想要去NameNode调用一个RPC接口，读取他里面某个副本所在的机器，意思是我们传入过去一个文件名，相对路径，/image/product/iphone.jpg，NameNode就需要去找到这个文件名的副本所在的机器

我们之前做了一个Map数据结构，专门用来存放这个东西，对两台机器采用随机的方式，选择一台出来返回给客户端即可

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/128\_客户端使用短连接与数据节点进行通信以及图片读写是否合适？/笔记.doc**

采取短连接，大家可以思考一下，有没有必要，写文件，读文件，可能会相对频繁，但是不也不是太频繁的跟DataNode进行通信，你用来读写图片的客户端可能是很多的，比如说客户端所在的机器都有20台

每分钟一共也就是有一两百个图片会进行上传，每台客户端每分钟可能也就发起5~10个图片的读写的操作，对应的可能是部署在也是同样小规模的这么一个30台机器的分布式存储系统上

客户端跟一个DataNode可能他通信的频率可能是很低的，可能几分钟才会对某个DataNode发起一次NIO读写请求，你一开始可能就是采取短连接的方式也没什么问题，短连接的好处，就在于客户端不需要维护那么多的网络资源

每次对一个DataNode读写请求，他都要进行连接、通信、断开连接，但是如果是每隔5~10分钟才会发起对DataNode读写操作，这个过程还算是可以接受的。Kafka，客户端其实是会维护对各个Broker的长连接，但是，他默认情况下就是如果9分钟都没有对一个Broker服务器发起读写请求

此时就会收回那个Broker的连接，断开连接，销毁掉

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/125\_使用读写锁优化对图片副本数据结构的并发读写性能/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/127\_实现客户端与Master节点通信获取图片副本所在机器/笔记.doc**

一点一点来实现图片下载的过程

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/122~141资料/133\_使用流对考的方式将本地磁盘文件读取出来通过网络发送出去/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/046\_使用内存缓存来优化editslog同步和写入的锁争用（一）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/043\_对editslog同步机制的代码进行梳理以及重构优化（三）/笔记.doc**

BackupNode同步editslog的代码基本上写完了，pull模型，他觉得自己可以的时候就去pull editslog，而不要去让NameNode主动push，editslog就存在于N多个磁盘文件里，要么就是存在于内存缓冲里

push模型去写，更加的复杂，更加的麻烦，对NameNode这一块的影响更大。pull模型下，在NameNode端，仅仅是在内存中会缓存比如说一小块数据，比如就内存缓冲里的数据会缓存起来，或者是某个磁盘文件里的数据，会缓冲起来

NameNode而言，内存缓存占用其实是不多的，而且是可控的，每次就缓存一小段数据，让人家来拉取就可以了，如果拉完了这一小段数据，再次缓存下一段数据就可以了，比如说下一个磁盘文件的数据

push模型，代码一样很恶心的；如果BackupNode万一宕机了，此时你要自动感知到，还要进行各种内存级别的缓冲啊什么的，内存缓冲不下了，此时还要落地磁盘；还要尝试自动感知到BackupNode什么时候恢复了；还要可能从磁盘和内存里去加载数据发送给BackupNode继续去同步

NameNode这一块可能会积压很多editslog在内存里，弄不好还有内存溢出的风险

如果说用push模型，一般来说应该怎么做会比较健壮一些，BackupNode要部署3个或者5个节点，奇数个节点，然后每次要一条数据发送给大多数的BackuupNOde才算成功，如果有3个BackupNOde，主要发送2个出去就算成功，其中如果有1个BackupNode宕机了，此时是不影响人家的push的

宕机的BackupNode如果一旦重启了之后也得找你重新同步数据，还要读磁盘

用我们设计的这个pull模型，相对来说还算比较简单的了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/044\_全面测试editslog同步机制以确保其正常运行（一）/笔记.doc**

初步的测试，基本上都ok了，其实的话呢，因为我们BackupNode拉取的性能是非常非常高的，速度极快，所以我们推测刚才那次测试可能全部都是基于内存的方式来拉取的，backupnode拉取到的txid肯定是远远的快于磁盘文件里的txid范围的

全部基于内存来拉取，比如说此时开始要刷磁盘了，两块缓冲区置换了一下，你再过来此时基本上来说，你要拉取的下一条的数据，应该都不在磁盘文件的范围里，肯定是继续从下一块内存缓冲里去拉取数据

下一讲，测试，先写数据，让他有几个磁盘文件，接着再运行BackupNode让他来拉取，他肯定会从磁盘文件里去读取数据的；如果都可以的话；BackupNOde拉取数据的这个过程就完成了

几乎是没有bug的，做了代码的重构，哪怕是用上周的那个重复度很高的代码，基本上也可以跑通，只不过他的代码太烂了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/048\_采用CopyOnWriteArrayList解决磁盘文件索引更新的并发问题/笔记.doc**

唯一还要解决的一个小问题，其实就是磁盘文件索引的更新和读取的并发问题

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/052\_触发Checkpoint时基于读写锁模式保证数据快照完整/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/051\_基于后台线程实现定时触发checkpoint检查点操作/笔记.doc**

可以开始来做一个后台线程，他需要定时的触发checkpint机制，把内存文件目录树，打成一个大的JSON，然后基于NIO给写入到磁盘文件里去，每个小时做一个checkpoint快照，然后接下来就保留最近一小时的editslog即可

下一个小时再做一个checkpoint快照，然后之前的editslog清空，再保留最近一小时的editslog即可，每个小时都有一个快照，大不了如果NameNode宕机了，就读取上一次checkpoint快照，接着把最近积累的1小时内的editslog读进内存，回放一遍，就可以恢复出来最新的文件目录树了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/050\_内存文件目录树在磁盘文件持久化方式的设计/笔记.doc**

其实非常简单，hadoop的话呢，fsimage格式是设计的非常复杂的，其实如果用简单点的方式来实现，也就是一个JSON就可以了，直接用fastjson把他转换为一个超大的json，写入磁盘文件就可以了

这个东西就代表了核心的文件目录树

很多同学会想，是不是会文件目录树里的内容特别特别的多，是否会导致fsimage文件过于的大呢？

目录是没有多少的，文件，每天上传5万个新文件，一个新文件就是一个名字，几十个字节，1000000字节，1MB，一个月大概也就是30MB；一年才300多MB；几年可能才一两个GB；自己的电商平台能否存活超过几年

150万个文件，2000万个，1亿个文件，fsimage可能就一两个GB，几百MB，看你具体你每个文件存储多少个字段，信息，一个大文件，1GB，也可以。很多我们下载的一些代码压缩包，或者是软件包，软件安装的镜像，虚拟机镜像，游戏安装包

几十MB，几百MB，还算好，不算特别特别的大

每次fsimage代表了一颗文件目录树，每次启动直接把这个数据读取到NameNode或者是BackupNode内存里，有没有问题？人家都是部署在物理机上，32GB，里面放一两个GB的核心文件目录树的数据，有问题吗？

如果说真的，就是你的文件目录树特别特别的大，几十亿个图片，几百亿个图片，几千亿个图片，文件目录树可能都要几十个GB，hadoop hdfs来存放数据，除非是顶级的互联网超大大公司的，大数据平台，hdfs，才有可能让元数据达到几十个GB

hadoop hdfs提供的一个方案，就是Federation联邦架构，其实就是把文件目录树给拆散，分布式存储在多个NameNode上，就会对应多台BackupNode，每个NameNode里存放一部分的目录和文件

保证每个NameNode内存里就几个GB的文件目录树，磁盘上的fsimage可能也就几个GB而已

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/042\_对editslog同步机制的代码进行梳理以及重构优化（二）/笔记.doc**

BackupNode同步editslog的代码基本上写完了，pull模型，他觉得自己可以的时候就去pull editslog，而不要去让NameNode主动push，editslog就存在于N多个磁盘文件里，要么就是存在于内存缓冲里

push模型去写，更加的复杂，更加的麻烦，对NameNode这一块的影响更大。pull模型下，在NameNode端，仅仅是在内存中会缓存比如说一小块数据，比如就内存缓冲里的数据会缓存起来，或者是某个磁盘文件里的数据，会缓冲起来

NameNode而言，内存缓存占用其实是不多的，而且是可控的，每次就缓存一小段数据，让人家来拉取就可以了，如果拉完了这一小段数据，再次缓存下一段数据就可以了，比如说下一个磁盘文件的数据

push模型，代码一样很恶心的；如果BackupNode万一宕机了，此时你要自动感知到，还要进行各种内存级别的缓冲啊什么的，内存缓冲不下了，此时还要落地磁盘；还要尝试自动感知到BackupNode什么时候恢复了；还要可能从磁盘和内存里去加载数据发送给BackupNode继续去同步

NameNode这一块可能会积压很多editslog在内存里，弄不好还有内存溢出的风险

如果说用push模型，一般来说应该怎么做会比较健壮一些，BackupNode要部署3个或者5个节点，奇数个节点，然后每次要一条数据发送给大多数的BackuupNOde才算成功，如果有3个BackupNOde，主要发送2个出去就算成功，其中如果有1个BackupNode宕机了，此时是不影响人家的push的

宕机的BackupNode如果一旦重启了之后也得找你重新同步数据，还要读磁盘

用我们设计的这个pull模型，相对来说还算比较简单的了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/045\_全面测试editslog同步机制以确保其正常运行（二）/笔记.doc**

先让他有几个磁盘文件，然后再运行BackupNode来进行拉取

亿级流量的那个课，写代码一下子写几百行，甚至上千行，但是一般来说，我写代码出bug的概率很小，Spark的项目课，代码也是很多很多行

技术功底的问题，你对一个技术本质，内部运行原理，架构设计的思想，各种情况的考虑，如果非常的充分和全面的话，那么你动手写代码是很快的，一边讲课，一边刷刷的写代码，是一件很困难的事情

自己写代码的话，不用讲课，我写代码是极快极快，我写代码的速度大概是普通人的3倍~5倍的速度，10倍

为了去优化锁的争用问题，大家可以去优化一下，从内存缓冲读数据的代码，每次过来检查一下当前内存里缓存的数据，是否有数据可以去读的，如果有的话，就不要去争用这个锁再次拷贝了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/053\_基于NIO将内存文件目录树快照写入磁盘文件/笔记.doc**

你这次写了一个最新的fsimage文件，上一个fsimage文件就可以给删除了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/049\_editslog同步机制设计的总结：pull模型、缓存机制、避免锁争用/笔记.doc**

（1）pull模型

pull模型，不会对NameNode造成太大的影响，被动去接收请求，每次最多就是在内存里缓存一小块的数据，供BackupNode来拉取；如果BackupNode有问题，那么他是不会影响到NameNode

NameNode主动push给BackupNode，很可能会被BackupNode一些异常的情况给影响，导致NameNode可能会有不稳定的问题；pull模型，BackupNode一旦异常，大不了就是不发送请求过来拉取日志，但是对NameNode是没影响的

（2）批量拉取editslog

批量拉取，这个你可以结合具体的情况，比如说我们这里设定的是每次批量拉取10条数据，但是你也可以搞的大一些，每次批量拉取20条，30条，可配置化的，就可以提升edits log同步的性能

（3）缓存机制

每次NameNode端都会在内存里缓存一块数据，要么是内存缓冲里的数据，要么是某个磁盘文件的数据，每个磁盘文件也就25kb，所以最多就是缓存25kb的数据。大部分的拉取，都是直接从缓存里走

不用频繁的读取磁盘文件，也不用频繁的申请锁去读取内存缓冲的数据

（4）低成本解决并发冲突问题

大部分的请求都是走内存缓存，避免去竞争锁拉取内存缓冲的数据；对于ArrayList写，读，不要去加锁，CopyOnWriteArrayList，读是读快照，写的时候也是基于多个快照版本来进行更新的

（5）效果

基本上来说，90%的请求都是走内存缓存，不需要读磁盘，不需要竞争锁；少量的请求会读磁盘文件，或者竞争锁从内存缓冲加载数据出来；BackupNode几乎是可以实时的跟上NameNode editslog的变化

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/047\_使用内存缓存来优化editslog同步和写入的锁争用（二）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/055\_测试checkpoint机制能否正常运行持久化fsimage文件/笔记.doc**

1000个文件，元数据才40kb

5万文件，一天新增的数据才2mb，每天元数据也就多2mb，一个月也就多60mb，一年也就多700mb，几年的话，2GB，几个GB，一个大文件，几个GB，内存放几个GB是可以放的下的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/041~055资料/041\_对editslog同步机制的代码进行梳理以及重构优化（一）/笔记.doc**

BackupNode同步editslog的代码基本上写完了，pull模型，他觉得自己可以的时候就去pull editslog，而不要去让NameNode主动push，editslog就存在于N多个磁盘文件里，要么就是存在于内存缓冲里

push模型去写，更加的复杂，更加的麻烦，对NameNode这一块的影响更大。pull模型下，在NameNode端，仅仅是在内存中会缓存比如说一小块数据，比如就内存缓冲里的数据会缓存起来，或者是某个磁盘文件里的数据，会缓冲起来

NameNode而言，内存缓存占用其实是不多的，而且是可控的，每次就缓存一小段数据，让人家来拉取就可以了，如果拉完了这一小段数据，再次缓存下一段数据就可以了，比如说下一个磁盘文件的数据

push模型，代码一样很恶心的；如果BackupNode万一宕机了，此时你要自动感知到，还要进行各种内存级别的缓冲啊什么的，内存缓冲不下了，此时还要落地磁盘；还要尝试自动感知到BackupNode什么时候恢复了；还要可能从磁盘和内存里去加载数据发送给BackupNode继续去同步

NameNode这一块可能会积压很多editslog在内存里，弄不好还有内存溢出的风险

如果说用push模型，一般来说应该怎么做会比较健壮一些，BackupNode要部署3个或者5个节点，奇数个节点，然后每次要一条数据发送给大多数的BackuupNOde才算成功，如果有3个BackupNOde，主要发送2个出去就算成功，其中如果有1个BackupNode宕机了，此时是不影响人家的push的

宕机的BackupNode如果一旦重启了之后也得找你重新同步数据，还要读磁盘

用我们设计的这个pull模型，相对来说还算比较简单的了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/213\_封装NetworkRequest来解析与抽取一个完整的请求（3）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/218\_在Proessor的循环中将排队响应暂存到客户端连接中去/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/209\_Processor线程将均匀分配的连接注册到自己的Selector上/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/220\_整体走读Reactor模式重构的网络通信架构的代码流程/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/219\_在poll方法中实现向客户端输出响应消息的逻辑/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/212\_封装NetworkRequest来解析与抽取一个完整的请求（2）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/210\_在一个循环中以限时阻塞的方式完成客户端请求的感知/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/204\_分析一下文件上传的三个接口能否支撑几千的QPS（2）/笔记.doc**

我平时写CRUD的业务系统，用不着并发、IO、网络、磁盘、Netty、ZK一些技术，Java Web里最复杂的一块东西，其实是Tomcat，人家Tomcat作为一个Web服务器，他底层要去做网络通信，内存管理、并发控制

你写的Servlet、SSM，嵌入在Tomcat里执行的一些业务代码，你就是CRUD

互联网系统，缓存，MQ，数据库，ES，NoSQL，架构设计

自己写中间件系统，分布式文件系统，微服务注册中心给完成，两个项目搞完，第一个，把你底层的技术全部打通，基础会极度的扎实；第二个，你后面看一些开源中间件系统的源码，会非常的轻松；第三个，这两个都是工业级的项目，直接是可以在出去面试的时候写简历上的

“盘古”分布式图片系统，替换你的很Low的CRUD的一些项目经历

每秒10w的文件上传/下载的请求，在NameNode这块是没有任何的瓶颈的，有很多的中间件系统是基于zk来做元数据管理，每次更新元数据的时候，都需要走网络请求，纯内存一般就是0.001毫秒~0.01毫秒

一走网络请求，直接就到毫秒级，一个请求过来，你需要去请求zk来做一些事情，直接就会到1毫秒+，几毫秒，10毫秒，直接会导致你的NameNode承载的并发能力，可能下降到每秒几千QPS了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/211\_封装NetworkRequest来解析与抽取一个完整的请求/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/202\_看看NameNode中有哪些接口可能会被高并发的访问？/笔记.doc**

上传接口：create、allocateDataNodes、informReplicaReceived

下载接口：chooseDataNodeFromReplicas

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/208\_让NioServer将建立好的连接均匀分发给Processor线程/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/217\_完成磁盘IO之后封装响应并且放入对应的响应队列中/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/205\_DataNode的NIO网络通信架构能支撑高并发吗？/笔记.doc**

比如说假设你每秒有1万个请求，图片系统整合到电商平台里去，大量的用到了很多的图片，图片其实都应该存储在在这个分布式图片系统里，对图片的读取，主要也是走图片系统，评论晒图、商品图片

电商首页、商品详情页，可能会有很多图片读取的请求，每秒上万的请求，你肯定必须得做静态化图片的缓存，不可能说每次都从分布式图片系统里来读吧，前置的Nginx本身肯定就可以做静态图片的缓存

CDN，大量的静态资源可以在前置的很多地方，Nginx、缓存服务器、CDN做缓存和加速，不需要每次都请求到底层的分布式图片系统里去的

部署了10台机器，每台机器要每秒有1000个QPS

目前的一个DataNode架构，每台机器接收1000个连接和请求能否实现

除了解析请求之外（很快），最核心最笨重的就是执行本地磁盘的读写逻辑（很慢）

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/203\_分析一下文件上传的三个接口能否支撑几千的QPS/笔记.doc**

单机几千的QPS

NameNode，节点，一般在生产部署的时候，高配置物理机，32核128G

正常情况下，应该极限支撑个每秒几万的请求都是可以的

一秒钟来1万个请求，每个请求排队获取锁进入执行更新文件目录树的代码逻辑

基于纯内存的操作，一个请求需要多少时间，1毫秒都不会到，0.01毫秒，1毫秒可以执行100个请求，100 \* 1000 = 10w个请求

Kafka之类的中间件系统，他其实本质也是大量的基于内存来实现核心逻辑的，高配置物理机的场景下，抗下来每秒10万的QPS

虽然说有并发逻辑里会加锁，但是不要紧，只要基于纯内存，每个请求速度极快，就可以做到每秒几万个请求

一次创建文件的请求 -> 更新内存里的文件目录树 -> editlog写入内存缓冲 -> 0.01毫秒 -> 每秒执行10w次请求没问题

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/201\_在分布式文件系统中高并发主要指的是什么？/笔记.doc**

高并发和高性能的架构改造，升华的一个部分

第一块：NameNode，元数据变更，能否承载高并发

第二块：DataNode，文件的上传和下载，能否承载高并发

NameNode而言，假设高峰时期，一万个客户端，同时发起请求要创建文件，一秒钟内高峰期直接来一万个请求去访问NameNode

DataNode，1000个客户端连接到DataNode上去，同时进行文件的 上传和下载，能否扛得住

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/207\_重写DataNode的NioServer让其仅仅监听客户端连接请求/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/206\_基于Reactor模式重新设计DataNode的网络通信架构/笔记.doc**

直接可以参考Kafka服务端的网络通信架构，Reactor模式

1000个客户端同时连接过来，发送请求，比较高并发的场景下，用Reactor模式来支撑是很轻松的

10个Processor线程，每个线程也就处理100个客户端

30个IO线程

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/215\_实现IO线程从请求队列中争抢请求以及执行磁盘IO操作/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/214\_将读取完毕的网络请求分发到全局的请求队列中/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/201~220资料/216\_实现IO线程从请求队列中争抢请求以及执行磁盘IO操作（2）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/246\_文件上传的代码流程再次梳理：请求排队与发送/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/255\_分布式海量小文件存储系统：架构总结/笔记.doc**

海量数据架构：分布式存储 + 可伸缩架构

高可用架构：多副本冗余 + 读写容错机制

高并发架构：纯内存元数据管理机制 + Reactor网络模型

高性能架构：长连接机制 + 异步化机制

分布式海量小文件存储系统

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/245\_文件上传的代码流程再次梳理：长连接机制/笔记.doc**

所有的代码全部写完了

权限管理，给大家留一些作业，可以自己去做，无非都是修改元数据而已，都是在NameNode层面去做一些元数据的修改

文件上传，文件下载，其实都分为三个部分，client、namenode、datanode

P6级别的高级工程师，有没有能力写出来这样的一个复杂的中间件系统，独立设计，独立开发，考虑到里面的各种架构。答案：不行。

技术专家、架构师水平的人，才可以，涉及到大量的底层技术，网络通信，磁盘读写，分布式系统的架构设计，必须是起码精读过三五个开源分布式系统的源码的架构师，才有能力来独立设计和研发这样的分布式海量小文件存储系统

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/247\_文件上传的代码流程再次梳理：Reactor网络通信模型/笔记.doc**

在讲咱们自己写的这个系统的时候，就感觉像是在讲复杂的开源项目一样

100%完全吃透，自己可以尝试蒙着眼睛，自己去写，多练几遍，自己的技术功底大幅度的提升，而且出去完全可以拿这种中间件项目作为自己核心的项目经验

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/257\_给每一个人留下的作业：如何自行完善一些功能/笔记.doc**

1、希望大家做更多的测试，搞几台虚拟机，真实的模拟部署集群，做更多的压力测试，并发测试，集群扩容测试，高可用测试，元数据管理机制，高并发访问下的情况，压测每次请求的性能，自己找到一些存在的小问题，自己去排查和发现问题，做优化

2、希望大家更多的去完善内部代码，日志、配置、监控

3、希望大家去做更多的功能性的完善：自己玩儿一把fastdfs，看看他的文档提供了哪些功能，然后自己尝试把一些功能加入进去，封装一些脚本，启动集群，停止集群，测试集群的脚本

4、去看看自己本公司的业务里面，有哪些场景可能额涉及到大量的小文件的，把那些业务场景去梳理一下，看看自己的这个系统如何跟业务匹配起来，然后梳理一下可能支持的自己负责的系统的业务场景，中间件系统作为你的项目之一

5、基础架构部，“蝙蝠侠”，分布式海量小文件存储系统，各种各样的架构，去说你们系统支撑了多少文件的存储，可用性如何，并发能力，性能，如何支撑你们的一些业务场景的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/248\_文件上传的代码流程再次梳理：Reactor网络通信模型（下）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/250\_文件读取的代码流程再次梳理：基于Reactor模型的处理/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/251\_为最终的测试准备一个干净的本地环境/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/249\_文件读取的代码流程再次梳理：同步请求模式/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/252\_尝试一次全新架构下的文件上传流程能否跑通/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/245~257资料/256\_分布式海量小文件存储系统：未来架构演进/笔记.doc**

1、高可用架构优化

NameNode彻底的高可用，基于BackupNode + ZooKeeper技术，彻底改造为高可用的master节点

假设一亿个小文件，会导致NameNode使用几十个GB的内存来存放这些小文件的元数据，可能会导致NameNode有比较严重的FullGC的问题，研发元数据分片存储，需要部署一个NameNode集群，每个NameNode管理一部分的元数据

2、高并发架构优化

在客户端和集群的通信之间，还需要实现更多的优化，性能这块，网络通信这块，客户端现在的做法是对每个数据节点一次只能发送一个请求过去，这个效率是一般的，不是特别的高。把每个数据节点的各种不同的请求在客户端缓存起来

然后每隔一段时间，把针对一个数据节点的多个请求（上传请求、读取请求）打包成一个大的ClientRequest，一次发送多个请求到数据节点上去，数据节点然后就一次可以收到一个客户端的多个请求，发到队列去排队，后台IO Thread争抢并发处理请求

返回响应给客户端，也是打包多个请求的响应成一个ClientResponse，一次性返回

更好的处理高并发高吞吐的请求

3、高性能架构优化

磁盘读写这块，上传文件的时候，可以优先把这个数据写入os cache就可以了，不要直接落地到磁盘上去，每隔一定时间从os cache刷入磁盘中，可以提升上传文件的性能；读取文件，优先从os cache内存里读取磁盘文件，性能更高

系统的架构优秀程度，完全堪比hadoop hdfs和kafka

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/078\_如何设计一个海量小文件的高可用分布式存储架构？/笔记.doc**

回顾一下分布式文件系统的元数据管理机制

接下来，我们已经实现了分布式文件系统里面的目录的管理，“创建目录”，删除目录、重命名目录，等等诸如此类的，其实大家自己都可以做，基于我们已经做好的代码来实现这些功能，几乎没有任何的难度了

我们肯定是假设比如有电商平台，每天要上传各种商品的图片以及评论的图片，网站首页的广告图片，网站促销活动页面的图片，各种各样的图片，作为一个千万级用户的这么一个垂直类的电商平台

像这个各种图片接下来你就需要往分布式文件系统里上传，以及从分布式文件系统里下载，我们会给大家来讲解和分析基于分布式文件系统来构建一套图片服务器集群，来管理大量的海量小图片

集群，每台机器存储部分小文件，分布式存储

高可用，每个文件有多个副本冗余

数据节点自动上报存储信息到master节点

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/098\_NIO文件传输的粘包拆包问题以及解决这些问题/笔记.doc**

粘包和拆包的问题

底层的TCP这块，如果看过之前的一些网络课程的话，网络对应的是很多的数据包，TCP包，客户端发送请求到服务端，其实本质就是发送很多个TCP包过去

粘包，意思就是说本来应该是两个包，但是到了服务器那块接收到了以后，把两个不同的包粘在一起交给了你的程序来处理，此时你会发现获取到的数据粘到一起去了，尴尬了，就不知道这次要处理的是哪部分的数据

拆包，本来是一个包，结果给拆分后发送，到服务器那块的时候，看到的就是两个包，先接受一个包，再接受一个包，你需要把两个包给合并起来进行处理

Netty的时候，可以深入的去看看应该怎么来处理，在原生的NIO层面，我现在没有做完美的解决方案，我的代码里仅仅是针对的是拆包问题，进行了一个定的处理，粘包，一个连接就传输 一个图片，所以一般不会出现粘包的问题

但是有可能会有拆包的问题

你的一个图片，100kb，给拆分成了两次来调用你的handleRequest，第一次是过来了80kb，第二次是过来了20kb

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/092\_通过画图的方式阐述传统的BIO为什么是同步阻塞的？/笔记.doc**

同步和阻塞之间，异步

文件读写，InputStream；网络通信，Socket

阻塞，针对的是IO流来说的，系统内核在IO完成之前都会卡住；同步，针对的是我们的程序和JDK API之间的关系，调用了IO API之后，就必须同步等待人家完成底层的IO操作，才可以让方法返回

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/081\_实现在Master节点的文件目录树中查重后创建文件的功能/笔记.doc**

先得进行查重，然后在文件目录树里创建一个文件

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/091\_是否有必要使用管道数据流传输方式来上传文件的多副本？/笔记.doc**

现在我们手上已经有多个数据节点了，此时就可以尝试把这个图片的副本传输到各个数据节点上去，但是此时传输采用什么样的方式呢？有两种方式可以来做

第一种方式：管道数据流的方式

第二种方式：客户端多副本依次上传

hadoop hdfs采用的方式

避免客户端的网络连接资源过多，负载过重

机器之间传输数据的性能更高，比客户端到服务器之间的性能要高

咱们就是采用第二种方式即可，就是我们的定位和应对的场景，上传图片的次数并不是特别多的，网络连接的资源不会特别的频繁和负载过重，我们也就是2个副本而已，所以说你如果为了把一个副本的传递放在两台机器之间自己去做

会导致编程特别的麻烦和复杂

哪怕我们就是客户端的机器上传图片到两个数据节点上也可以的，假定中型公司没有使用那么多的机房，比如说总共就是1个机房，大家都在一起，所以说性能的损耗并不是特别的大，所以就是还好

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/085\_思考一下：为了高可用而冗余存储到底需要几个副本呢？/笔记.doc**

到底需要几个副本来冗余存储呢？

比如说hadoop hdfs，默认就是3个副本，是有一个规律的，假设你的集群的所有的机器是假设到多个机架上的，机房和机架的关系，其中2个副本是在一个机架上，另外一个副本是在别的机架上

默认情况下，如果是一个机架里的某台机器宕机了，此时可以使用这个机架上的另外一台服务器上的副本就可以了；万一说这个机架都故障了，此时别的机架上还有一个副本冗余是可以用的

提高可用性的级别，万一某个机架都故障了，其他机架上有副本可以用

其实以我们通常的一些运维的经验来看的话，机器的故障概率比较高，物理机的网络故障、磁盘坏了，机架都故障了，机架上所有的机器都故障了，这个事情就比较尴尬了，是比较少见的

3个副本的优点，就是可以实现更高的可用性，缺点，对存储空间的占用太多了

数据总共是100G，3副本，300G，其实就要耗费掉3倍的磁盘空间来存储你所有的数据；总共10TB，3副本，就需要30TB来存储；但是如果你是2副本，就只要20TB来存储即可，有没有必要用3副本呢

中等规模的公司而言，暂时还没有必要去考虑机架故障的问题

双副本，一个图片就存储两份，在两台机器上就可以了，对于可用性而言也可以保证，一般来说比较多的故障就是某台机器故障，但是其他机器上还有图片的副本可以使用，保证了系统的可用性

另外，对存储空间的占用可以减少1/3，特别是对于中等规模的公司而言，最大的好处就是可以节约很多的钱，1/3的存储，也是要钱买的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/095\_为数据节点引入数据流传输端口供NIO上传机制使用/笔记.doc**

每个数据节点要基于哪个端口去监听开放nio server呢？这个你一般可以做在配置文件里，但是对于我们来说，稍微简化一下，直接这个端口是写死的就可以了，就认为这个端口默认就是有一个端口就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/079\_为DFS客户端设计与构建一个文件上传接口/笔记.doc**

给咱们的这个DFS客户端设计一个文件上传的接口

一般来说在上传文件的时候，可以通过流的方式把数据给写入到分布式文件系统里去，通过IO流不停的读取数据，然后再把数据交给分布式文件系统的一个接口，DFS的接口就把接收到的数据给写入到对应的集群中的某一台机器上去

采取一个比较简化的设计手法，我可以让他比如说传递一个File对象过来，我DFS接口内部负责从File对象里读取数据上传到集群机器上面去，byte[]字节数组，就包含了文件所有的二进制的字节数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/090\_测试双副本对应的数据节点选择机制是否正常运行/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/087\_设计一个为文件的多个副本选择合适数据节点的RPC接口/笔记.doc**

设计一个master节点的接口，可以为图片上传的请求选择当前存储资源量最小的两台机器分配过去作为双副本存储的目标数据节点

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/094\_基于NIO设计数据节点应对多个客户端上传文件的架构/笔记.doc**

BIO，阻塞，一个客户端的连接，只能是一个线程来应对

NIO，非阻塞，N个客户端的连接，可以是一个线程来应对，大大提升了单机应对高并发连接和请求的能力

每个数据节点，实际上都有一个NIO server，基于非阻塞的方式来监视多个客户端的连接是否有请求过来，有请求就会交给后端的架构来进行处理，所以说，我们在核心的文件上传这块，就可以用NIO来进行实现

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/080\_为Master节点设计一个支持查重的创建文件RPC接口/笔记.doc**

咱们先把这个master节点的RPC接口先设计出来，支持文件创建，create file，其实就是要在文件目录树里加入一个文件，这个接口还是需要支持去重的，你的这个接口的返回值里，必须有多种状态，比如说成功，失败，重复

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/082\_创建文件editlog以及对应的日志回放功能的实现/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/093\_针对文件和网络两种场景分析NIO的同步非阻塞/笔记.doc**

NIO同步非阻塞

异步非阻塞，NIO2，AIO，每次执行一个IO请求，都必须提供一个回调函数

BIO，同步阻塞

NIO，同步非阻塞

AIO，异步非阻塞

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/078~098资料/086\_选择数据节点时如何实现集群资源均匀负载的效果？/笔记.doc**

尽可能得让集群里的各个机器上存放的数据是比较均匀的负载的

30TB的数据，4台机器

一台机器上放了20TB的数据，另外三台机器，每台机器放3TB多一点的数据。要实现集群里的存储资源的均匀使用，每台机器的存储的数据量大概在7TB~8TB之间

构思一下这个里面的思路，首先呢，必须是要知道，每台机器上放了多少数据量的文件，我现在需要知道的是机器01上存放了5TB的数据，机器02上存放了5.3TB的数据，机器03上存放了6TB的数据，机器04上存放了6.1TB的数据

比如说来了一个图片上传的请求，我应该把图片的2个副本上传到哪两台机器上去呢？把机器上面的数据量的大小正序来排序，选择数据量最小的两台机器就可以了，各个机器的存储的数据量基本是均匀的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/035\_同步editslog的时候是基于数据流还是基于RPC接口？/笔记.doc**

BackupNode还是基于pull模型去NameNode那儿拉取数据比较靠谱一些

是基于NIO来开发一套网络通信的程序，不停的通信尝试从NameNode那儿读取数据呢？还是说基于gRPC接口调用，每次发送一个请求就拉取10条edits log回来

在这个场景下，其实还是用gRPC的模式比较好一些，接口来调用，批量拉取，每次可以拉取10条~20条，1kb以内，或者1kb左右

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/034\_Backup同步editslog到底应该是pull还是push？/笔记.doc**

push模型，NameNode每次生成一条editslog之后，都会主动发送给BackupNode让他接收了以后同步到自己的内存文件目录树里去；BackupNode，如果说他挂掉了以后，那么NameNode该怎么办呢？

难道让NameNode不停的在内存里缓冲一些editslog等待人家恢复吗？

如果NameNode不停的往BackupNode去发送数据的话，就会导致BackupNode每次接收数据要写内存文件目录树，此时要加锁的，这里就会有频繁的加锁的行为，跟BackupNode后面如果执行checkpiont加锁刷数据到fsimage里去

可能会导致NameNode很多editslog需要积压在内存里，等待BackupNode的checkpoint执行完毕之后，释放了锁，人家NameNode的editslog才能继续发送过去，所以说这里有很多的不确定性

pull模型，BackupNode可以不停的批量拉取NameNode那儿的editslog，每次拉取一小批，比如说一条editslog是50字节，每次可以拉取10条数据，500字节，不到1kb的数据，这个是没问题的

NameNOde而言他不需要在内存里积压和缓冲很多的editslog去主动发送给BackupNode让他去接收，NameNode只要被动的接受人家的一个调用，如果人家调用，此时可以尝试从文件里读取一些editslog发送过去就可以了

可以提前在内存里缓冲一点点editslog就可以了

如果说BackupNode发生了checkpoint的操作之后，锁掉了内存文件目录树，此时的话呢就是BackupNode自己就不会去继续拉取editslog了，对于NameNode而言，此时根本不用去care BackupNode的事情

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/031\_基于多线程并发测试editslog机制是否正常运行（二）/笔记.doc**

测试一下如果触发了25kb的阈值，是否会有一个线程进行刷磁盘；刷磁盘的过程中，是否其他的线程可以继续写内存缓冲，一旦触发阈值，有一个线程会交换两块缓冲区，写满的一块会刷入磁盘中，另外一块缓冲区同时供其他线程继续写入

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/033\_实现NameNode优雅关闭接口保证editslog全部写入磁盘/笔记.doc**

editslog就是存在磁盘里多个文件中，内存缓冲里也有一部分，如果此时你要重启NameNode的话，是否就是需要提供优雅关闭接口，必须要把核心的editslog缓冲区的数据给主动刷入到磁盘里面去

而且如果你要重启的话，此时还应该在所有的核心接口都应该加入一个判断，如果要优雅关闭了，此时不允许别人发送请求，全部返回一个异常码，把内存缓冲里仅有的数据全部刷入磁盘中

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/037\_基于RPC接口实现edtislog同步机制的代码（二）/笔记.doc**

索引位置，就是现在这个backupnode拉取到了什么地方了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/030\_基于多线程并发测试editslog机制是否正常运行（一）/笔记.doc**

先测试多线程并发写，数据能否正常的都落入到内存缓冲区里去，先不测试触发阈值刷磁盘；触发25kb的阈值，刷磁盘，磁盘文件里的数据是否符合期望；第一次刷磁盘过后，继续不停的写数据，能否触发不断的正确的刷磁盘，磁盘文件是分段来存储，有多个文件，完整的editslog就是内存缓冲里的数据+所有磁盘文件的数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/023\_分析Hadoop源码如何实现完美的editslog双缓冲机制/笔记.doc**

删除目录，重命名目录，创建文件，重命名文件，删除文件，其实这个代码是类似的；你只要仿照mkdir那个方法对文件目录树的数据结构做出一定的维护和管理，就可以实现内存里的一颗文件目录树的变更

这一次变更日志，创建目录：editslog，OP: MKDIR，PATH: /usr/warehouse/hive

你不可能直接写入磁盘文件里去的，因为那些的话性能太差了，所以一般来说这种editslog都是写入到内存缓冲里去的，设计的是双缓冲的机制，不停的往一块缓冲区里去写，一旦写满了之后，就由一个线程把这个缓冲区的数据刷入磁盘

在刷磁盘之前，会做一个缓冲区的交换，把两块缓冲区交换一下，让后续的editslog写入另外一块空置的缓冲区里去，之前写满的一块缓冲区就可以刷入磁盘中了

之前在并发课里实现过一个简化版本的editslog双缓冲机制，实战了很多并发编程的技术，尤其是对锁的优化这一块，尽可能的缩短锁持有的时间，提高多个线程来并发写editslog的效率

分析一下，真正hadoop hdfs工业级的分布式文件系统，他是如何实现完美的editslog双缓冲机制的呢，并没有看过hadoop源码的课，在大数据架构课里讲解的，我们就只是分析他的一个editslog源码

电商平台里，可能是有大量的用户每天都会购物，每天都会发表评论，可能就会在一些高峰期，导致比较高并发的一些评论文件的上传，必然会调用NameNode来在文件目录树里创建一个文件

虽然说对文件目录树和editslog两块地方都是依靠锁机制来让多线程串行写，但是如果说每个线程只是写内存的话，可能就是在微秒的一个级别，1ms都不到，内存的操作速度是非常非常快的

每次文件目录树的修改操作，加上写editslog，如果都发生在内存里，0.01ms，1ms就可以执行100次操作，1秒就可以执行100 \*1000 = 10万次操作，所以如果说保证这个操作仅仅是发生在内存里的话

只要每次操作性能极高，速度极快，每次就耗时0.01毫秒，哪怕是多个线程串行起来，基于锁来执行，每秒也可以执行10万次操作，并发已经很高很高了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/027\_基于NIO FileChannel将内存缓冲数据写入磁盘文件中/笔记.doc**

dd

hadoop里面他的editslog设计的更加的复杂一些，所以每一条editslog可能都是几百字节，甚至一个editslog就是1kb大小，所以说他的缓冲区大小默认是给的512kb，512条editslog就会刷一次磁盘文件

一条editslog就50个字节，平均每隔500条edittslog就应该刷一次磁盘，500 \* 50 = 25000，25kb，所以我们可以把我们的缓冲区的大小给他缩小一些，让他到25kb左右，500条editslog，就可以触发一次刷盘的操作

RandomAccessFile file = new RandomAccessFile(name, "rw"); // 读写模式，数据写入缓冲区中

FileOutputStream out = new FileOutputStream(file.getFD());

FileChannel channel = out.getChannel();

channel.position(channel.size()); // 定位到文件里的最后一个位置，进行append追加写

byte[] data = ByteArrayOutputStream.toCharArray();

channel.force(false);

ByteArrayOutputStream.reset();

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/036\_基于RPC接口实现editslog同步机制的代码（一）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/038\_基于RPC接口实现edtislog同步机制的代码（三）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/025\_基于JSON设计一个简单易用的editslog数据存储格式/笔记.doc**

我们的editslog到底要设计成什么样的格式，要如何序列化之后 放到磁盘文件里去呢？比如说人家的hadoop采用的是他特定的一种序列化的机制。但是我们来说，NameNode写editslog，是会通过checkpoint检查点的操作，定时清理掉的

如果一旦你完成了一个checkpoint检查点的操作之后，在检查点之前的editslog都可以删除了。所以说一般来说，这个editslog都不会特别的多。每秒10万次的高并发的修改文件目录树的请求

电商平台的发表评论，每天能发表多少评论？3000万注册用户，日活用户300万。日订单大概是100万个订单。很多订单是没有人发表评论，发表评论的人就10万个人。每天可能就几万条评论，或者是10万条评论。

分布式文件系统，存放的主要是商品的图片，评论的晒图，每天10万个小图片进入分布式文件系统也就差不多了，一般来说是不会对这种系统造成多大的并发的压力的。10万个新的文件进入分布式文件系统

每天会执行10万次请求，来在文件目录树中新添加一个文件，每天新添加10万个文件，editslog也就10万条，但是在一天里会假设每隔1小时执行一次checkpoint，也就是说，editslog最多就是保留最近1小时的editslog的日志

一小时就1万条，editslog日志最多就保留1万条在editslog日志文件里

每条editslog可能就几十个字节，edits log日志文件一般来说也就几百kb，一个文件里可能也就最多是1万条

基于方便的JSON格式就可以了，人的肉眼是可读的，基于特殊的序列化的机制，写入磁盘文件的东西看起来跟乱码一样的，在用的时候不是很方便，压缩处理，会很节约磁盘的空间开销

kafka，大数据课程学习权限的，磁盘文件日志去存储你写入的所有的消息，每天可能上TB的，所以说他必须精心设计他的消息格式，序列化的方式，尽可能紧凑的压缩消息的存储，用最少的bit位去存储尽可能多的信息，节约磁盘开销

{“OP”: “MKDIR”, “PATH”: “/usr/warehouse/hive”}

{“OP”: “RM”, “PATH”: “/usr/warehouse/hive”}

{“OP”: “CREATE”, “PATH”: “/usr/warehouse/hive/access.log”}

每次写入磁盘的时候，每条editslog后面都跟上一个换行符，一行一个日志；读取的时候就很方便了，每一行就是一个txid对应的日志，定位到某一行，然后就去读取就可以了，一行一行的读

中间件系统在设计磁盘上的数据存储格式的时候，序列化方式的时候，考虑这个数据有多大，如果数据量很大很大的话，那必须精心设计一套数据存储格式，尽可能用二级制类的bit位来代表很多东西，long很多bit位就可以代表很多信息了，kafka就是这样，每条数据的大小就被压缩到了极致

而且在存储的时候都是紧凑格式来存储的，一条数据后面直接就跟着一条数据，都不会换行的，序列化的时候用的也是压缩的方式，数据可能都被压缩了，存放到磁盘文件里，你如果直接打开看到的都是类似乱码的东西

分布式文件系统，文件目录树对应的磁盘文件，其实不会太大，如果太大的话，也可以分成几个文件来存放，一般来说都还好。所以说不用过于讲究，直接用简单易用的JSON格式，一行放一个editslog就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/028\_editslog日志在磁盘文件上进行分段存储的机制设计/笔记.doc**

就是你的editslog如果都在一个文件里，那么如果后续做了checkpoint操作之后，你需要把checkpoint时间点之前的editslog都删了，此时怎么删呢？所有的数据都在一个文件里，你这样子是非常的不好删的

按照细粒度的方式把editslog可以拆分为很多个日志文件

每一次你刷磁盘的时候都是刷一个新的editslog日志文件，日志文件的名字就是txid~txid.log，格式，是最好的。如果你checkpoint之后要删除一些editslog日志文件，完全可以就把之前的一些文件给物理删除即可

每次在落地磁盘的时候，都是把上一次落地磁盘的最大的一个txid+1

0~500.log

501~1000.log

1001~1500.log

每次落地磁盘都记录好本次落地的时候，最大的txid是多少，下一次落地磁盘就可以取出来进行命名就可以了，每次落地磁盘都是一个新的日志文件

25kb，50字节，500条数据就会刷一次盘，500条数据就在一个日志文件里面，后面删除的时候也就方便很多了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/032\_基于多线程并发测试editslog机制是否正常运行（三）/笔记.doc**

按理来说，他应该是不停的写内存缓冲，只要满25kb就写一个磁盘文件，每次都是换一个新的磁盘文件，分段来存储。磁盘上会不停的有新的磁盘文件，内存缓冲里也永远有一批数据在

内存缓冲里的数据 + 磁盘文件里所有的数据，就是完整的一套edits log内容

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/029\_基于txid机制实现editslog日志文件分段存储/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/026\_基于字节数组IO流构建内存中的数据缓冲区/笔记.doc**

我们需要把数据不断的写入内存缓冲，可以考虑用List，但是一般来说在这种情况下都是用字节数组来存放，又不太好维护这个数组，会用一个叫做字节数组的输出流，IO流的方式，不断的往这个流里写数据

ByteArrayOutputStream

他内部就是维护了一个内存缓冲区，最后你可以从里面获取到一个字节数组，把这个字节数组通过NIO的方式，Buffer + FileChannel的方式，写入磁盘文件中，就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/024\_重构editslog双缓冲机制：仿照hadoop源码实现/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/022~040资料/022\_完善内存中的文件目录树管理功能的代码/笔记.doc**

客户端的API可以发起一个创建目录的请求，通过RPC接口调用到NameNode那儿上去，其实他应该做的第一件事情，就是维护自己的内存中的文件目录树，命名空间，元数据，分布式文件系统，文件目录树

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/012\_如何基于editslog机制实现Master重启之后的数据恢复/笔记.doc**

如果你仅仅是在NameNode内存里维护你的元数据，文件目录树，如果此时NameNode宕机，会怎么样？那内存里的数据就全部都丢失了。你可以基于磁盘文件，比如说xml格式的文件来维护一份内存里的元数据

如果你每次创建目录，上传文件，更新元数据的时候，都必须更新一个复杂的xml文件，会导致每次都直接同步随机读写磁盘文件，是很恐怖的，这个会直接导致你的系统的性能很差，你每次上传一个文件，光是更新磁盘上的xml里的元数据

就是需要在目录下面加入一个文件

<dir name=”/usr”>

<file name=”/access.log”>

</dir>

就是基于一个内存buffer做一个写缓冲

editslog，意思就是说，我本次对元数据做了什么修改

在/usr目录下创建了一个文件，access.log，edits log，他只是一条日志，修改日志，他就是说明了他本次对文件目录树做了什么修改，双缓冲机制

如果此时namenode宕机了，会怎么样？不要紧的，磁盘文件里存有一份edtislog的数据，如果你重启namenode，就可以读取全部的editslog来回访日志，重新把日志对应的操作在内存文件目录树上执行一遍，此时就可以恢复出来一份完整的数据

因为edits log会先在内存缓冲里等待一会儿，所以说此时，如果 有些edits log还没刷入磁盘，此时就宕机了，会导致内存缓冲里的部分数据会丢失，如果namenode宕机，是可能会导致部分数据丢失的

如果你需要系统保持高性能，你需要允许部分数据的丢失，elasticsearch、redis，都是可能会丢失部分数据的，同步写磁盘，会导致性能很低

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/002\_我们需要一个什么样的分布式存储系统来应对电商场景？/笔记.doc**

这个分布式文件系统出发点，就是支撑电商系统里的大量的小图片，需求，就是解决海量小图片存储，比如说，现在你电商平台一共有8000个小图片，分布式存储，那到底是怎么来存储呢？

你现在分布式文件系统部署的集群有4台机器，每台机器放2000个小图片

比如说我们现在手头有10亿个小图片，每个小图片就算他是1MB，1000000000MB，1000000GB，1000TB大小的小图片，你说如果你就一台图片服务器，能存放的下吗？当然不可能了

分布式文件系统，你现在部署1000台机器组成的一个大集群，每个机器上都有一个分布式文件系统的进程，负责管理这台机器上存储的一部分小图片，每台机器上就存放1TB的小图片就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/018\_开发分布式文件系统的客户端工程实现mkdir功能/笔记.doc**

dfs-client，作为一个客户端的工程，可以给电商平台来调用的，需要使用分布式文件系统的客户端，进而实现跟分布式文件系统的通信，目录的创建，文件的上传

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/003\_大数据领域的Hadoop能给我们带来什么样的启示？/笔记.doc**

基础架构类的分布式系统的开发，不要脱离业务，我们始终要围绕一个业务场景驱动来进行需求的分析，以及架构的设计和开发

大数据领域的hadoop hdfs，分布式文件系统，但是他是解决什么问题的，架构是如何设计的，我们从他身上可以得到一些什么样的启示呢

比如说，现在我们要采集网站或者APP的一些用户浏览行为，收集到一些用户浏览行为的日志，比如说每天收集到10亿条数据，一共是几个GB，这几个GB可能都存在于一个大文件里

一个大文件里就放了10亿条数据，大小达到了几个GB

我们需要对这个文件里的用户行为的日志数据进行分析，此时你说怎么弄呢？你不可能说对这个文件里的数据一条一条扫描和分析，这样会很慢很慢很慢，可能10亿条数据可能需要耗费几天的时间来进行分析

hadoop hdfs，凭空出现，解决的就是上面的那个场景，超大数据集

10亿条数据，一共是5GB，5GB的数据会被拆分成N多个数据块，每个数据块就128MB，一共会把这个文件拆分成40个小文件，每个小文件是128MB，这40个小文件可以分散在10台机器上，每台机器可以放4个小文件就可以了

此时你可以启动40个分布式计算的任务分发到10台机器上去，每台机器上是4个计算任务，每个计算任务针对一个128MB的小文件进行分析，最后40个计算任务的结果会被汇总起来放在一起

最多有40个计算任务在10台机器上并行的运行，你之前是对10亿条数据一行一行的扫描和分析，比如一共要耗费48个小时，2天2夜才可以。现在40个计算任务在10台机器上，利用每台机器的CPU资源，分布式并行的计算和分析，速度提高了40倍

1个小时左右，就可以把这个10亿条数据的大文件里的数据都分析完毕

大数据领域要解决的这个问题，是跟我们的电商场景下的问题是不太一样的，把一个超大的数据集分散开来存储，以便于你可以让更多的计算任务可以并行在多台机器上来进行处理，可以把大数据集的分析的性能和效率提升几十倍，甚至是几百倍

原来分析一个超大数据集需要几天几夜的时间，现在仅仅需要几个小时而已，几十分钟，就可以完成了

大量的小文件，只不过是一台图片服务器放不下去1亿张图片，10TB。就需要采用分布式文件系统，部署在多台图片服务器上，每台服务器存放一部分的图片，这样就可以利用多台机器的存储空间放下海量的图片资源了

但是大家其实都是分布式文件系统，我们完全可以参考hadoop的很多的架构来实现我们自己的这套分布式文件系统，fastdfs源码（c写的），架构设计，fastdfs和hdfs很多地方是类似的

但是我们在写的过程中，会很多地方都是参照着我们解决的电商领域的业务场景来设计和开发的，不是完全照搬hadoop hdfs。因为hdfs的源码非常的优秀，里面大量的磁盘读写和网络通信的环节

他基本上全都是用原生的Java NIO来实现的磁盘文件读写，网络通信和数据传输，仿照hdfs的架构和源码来实现，可以大量的实战NIO技术

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/013\_采用分段机制优化editslog在磁盘文件上的存储/笔记.doc**

我们是用一个文件来承载所有的edits log吗？显然不是的，假设你频繁的更新文件目录树，此时有几千万条editslog都放在一个文件里，如果一个文件过大的话，可能还会导致读写性能的下降

一般来说磁盘文件，editslog会不断的增长的话，一般都是会采取分段存储的机制，先写入一个文件，如果说这个文件的大小达到了一定的数量之后，此时就会拆分出来一个新的文件，一般来说建议单个日志文件的大小不要超过1GB

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/015\_基于fsimage和checkpoint机制优化Master重启性能/笔记.doc**

重启过慢，每次重启都要回放所有的edits log机制

假设在晚上8:00的时候，突然执行了一次checkpoint操作，NameNode把一份完整的文件目录树写入到了磁盘上的fsimage里去，接着8:10，8:00~8:10之间，一共做出了几十条editslog的变更日志

此时NameNode需要重启，他会直接读取fsimage文件加载到内存里变成文件目录树，接着把edits log文件里的几十条editslog读出来，在内存的文件目录树里回放一遍

比如说你的有一个目录，/usr目录

在fsimage里只有一条数据，但是在editslog里就不只是一条了，创建操作，更新操作，重命名操作，在editslog里有多条操作日志

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/009\_gRPC：一个轻量级RPC框架的入门讲解/笔记.doc**

google开源的一个轻量级的rpc框架，gRPC，底层分布式系统的实现，用他来作为一个rpc框架还是挺合适的

protoc-3.1.0-win32

hello.proto

hello\_service.proto

两个文件放在bin目录下

win命令行：protoc.exe --java\_out=./ \*.proto

protoc-gen-grpc-java-0.13.2-windows-x86\_64.exe放到bin目录下

win命令行：

protoc.exe --plugin=protoc-gen-grpc-java=protoc-gen-grpc-java-0.13.2-windows-x86\_64.exe --grpc-java\_out=./ \*.proto

生产一堆的代码

搞一个：distributed-filesystem-rpc，放进去自动生成的代码

<dependencies>

<dependency>

<groupId>com.google.protobuf</groupId>

<artifactId>protobuf-java</artifactId>

<version>3.1.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.grpc/grpc-stub -->

<dependency>

<groupId>io.grpc</groupId>

<artifactId>grpc-stub</artifactId>

<version>1.3.0</version>

</dependency>

<dependency>

<groupId>io.grpc</groupId>

<artifactId>grpc-protobuf</artifactId>

<version>1.3.0</version>

</dependency>

<dependency>

<groupId>io.grpc</groupId>

<artifactId>grpc-netty</artifactId>

<version>1.3.0</version>

</dependency>

</dependencies>

编写服务端的代码，引入rpc工程的依赖

public class GreetingServiceImpl implements HelloServiceGrpc.HelloService {

private static final Logger logger = Logger.getLogger(GreetingServiceImpl.class.getName());

public void sayHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {

logger.info(String.format("sayHello方法调用的请求参数信息: name={%s}, id={%d}", request.getName(), request.getId()));

HelloResponse reply = HelloResponse.newBuilder().setMessage(String.format("Hello, %s", request.getName()))

.build();

responseObserver.onNext(reply);

responseObserver.onCompleted();

}

}

public class HelloServer {

private static final Logger logger = Logger.getLogger(HelloServer.class.getName());

private static final int DEFAULT\_PORT = 50051;

private Server server = null;

private void start() throws IOException {

server = ServerBuilder.forPort(DEFAULT\_PORT).addService(HelloServiceGrpc.bindService(new GreetingServiceImpl()))

.build().start();

logger.info("Server started, listening on " + DEFAULT\_PORT);

Runtime.getRuntime().addShutdownHook(new Thread() {

@Override

public void run() {

System.err.println("\*\*\* shutting down gRPC server since JVM is shutting down");

HelloServer.this.stop();

System.err.println("\*\*\* server shut down");

}

});

}

private void stop() {

if (server != null) {

server.shutdown();

}

}

/\*\*

\* Await termination on the main thread since the grpc library uses daemon

\* threads.

\*/

private void blockUntilShutdown() throws InterruptedException {

if (server != null) {

server.awaitTermination();

}

}

public static void main(String[] args) throws IOException, InterruptedException {

final HelloServer server = new HelloServer();

server.start();

server.blockUntilShutdown();

}

}

搞一个client端

public class GreetingServiceClient {

private static final Logger logger = Logger.getLogger(GreetingServiceClient.class.getName());

private final ManagedChannel channel;

private final HelloServiceGrpc.HelloServiceBlockingStub blockingStub;

public GreetingServiceClient(String host, int port) {

channel = NettyChannelBuilder.forAddress(host, port).negotiationType(NegotiationType.PLAINTEXT).build();

blockingStub = HelloServiceGrpc.newBlockingStub(channel);

}

public void shutdown() throws InterruptedException {

channel.shutdown().awaitTermination(5, TimeUnit.SECONDS);

}

public void sayHello(String name) {

try {

System.out.println("Will try to say Hello " + name + " ...");

HelloRequest request = HelloRequest.newBuilder().setName(name).setId(12345678).build();

HelloResponse response = blockingStub.sayHello(request);

System.out.println("result from server: " + response.getMessage());

} catch (RuntimeException e) {

System.out.println("RPC failed:" + e.getMessage());

return;

}

}

public static void main(String[] args) throws Exception {

GreetingServiceClient client = new GreetingServiceClient("127.0.0.1", 50051);

try {

String name = "Eric";

logger.info(String.format("Client 调用RPC接口，参数为name = {%s}", name));

client.sayHello(name);

} finally {

client.shutdown();

}

}

}

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/014\_editslog机制的弊端分析：耗时过长的Master重启/笔记.doc**

分析一下基于editslog机制，会导致master重启的速度过慢

比如说现在你的editslog里都有几千万条日志了，此时你的NameNode一旦重启，你需要从磁盘上读出来几千万条日志，一条一条在内存里执行回放一遍，比如说一开始先是创建某个目录，创建文件，删除文件

可能会导致你的NameNode重启的时候需要耗时几个小时都有可能

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/020\_文件目录树管理和editslog内存双缓冲的代码重构的说明/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/011\_分布式文件系统的Namespace指的到底是什么/笔记.doc**

比如说你现在需要使用一个分布式文件系统，最基本的操作有哪些？

目录的管理，文件的管理

目录：创建目录、删除目录、重命名目录，目录是不是有层级结构的，是一个树形的概念

文件：上传文件到某个目录里去，删除文件，对文件进行重命名，下载文件（读取文件）

分布式文件系统，是不是需要一套元数据，专门维护说你的文件系统里有哪些目录层级的结构？每个目录下面有没有挂载哪些文件？文件目录树

/root

/usr

/local

/app

/home

/kafka

/data

/access.log

namespace，命名空间

你的一个java开发的系统，电商系统，商品服务，或者是评论服务，收到一个用户的评论，需要上传这个图片，在这个之前首先是需要先建立好商品对应的目录，以及评论对应的目录，这个目录就需要找NameNode来进行创建

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/008\_在Master节点中实现基于心跳机智的故障节点检测/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/005\_基于已有案例代码完成Master-Slave节点的启动流程/笔记.doc**

在并发课里，使用分布式存储系统作为案例搞过一块代码，master命名为“NameNode”，把slave命名为“DataNode”，用多线程来开发的时候，启用多个线程，volatile，Atomic，ThreadLocal，synchronized，ReentrantReadWriteLock，线程安全的集合（队列、Map、List），线程池

并发编程设计模式，几十种并发设计模式，要不要在这个课里来讲，会放到分布式存储系统和分布式微服务注册中心两个项目都基本做完之后，最后引入几十种并发编程设计模式，在两个底层中间件系统里，大量的基于并发设计模式来重构代码

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/016\_优化Master机器的CPU负载：Backup节点的引入/笔记.doc**

NameNode所在的机器做的磁盘IO的操作太多了

NameNode进程，他需要额外分配出来一个线程，后台线程定时的去进行磁盘IO的操作，其实这个是很影响本地CPU负载的，如果你一边大量的线程需要来更新内存的文件目录树，肯定是要加锁的

此时如果你还要每隔一段时间，耗费比如说几秒钟甚至几分钟的时间也对文件目录树来加锁读取数据写入本地磁盘，会导致更新文件目录树，和读取文件目录树写入磁盘，他们之间会产生巨大的锁的冲突

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/010\_基于gRPC实现Master-Slave节点之间的通信调用/笔记.doc**

google开源的一个轻量级的rpc框架，gRPC，底层分布式系统的实现，用他来作为一个rpc框架还是挺合适的

protoc-3.1.0-win32

hello.proto

hello\_service.proto

两个文件放在bin目录下

win命令行：protoc.exe --java\_out=./ \*.proto

protoc-gen-grpc-java-0.13.2-windows-x86\_64.exe放到bin目录下

win命令行：

protoc.exe --plugin=protoc-gen-grpc-java=protoc-gen-grpc-java-0.13.2-windows-x86\_64.exe --grpc-java\_out=./ \*.proto

生产一堆的代码

搞一个：distributed-filesystem-rpc，放进去自动生成的代码

<dependencies>

<dependency>

<groupId>com.google.protobuf</groupId>

<artifactId>protobuf-java</artifactId>

<version>3.1.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.grpc/grpc-stub -->

<dependency>

<groupId>io.grpc</groupId>

<artifactId>grpc-stub</artifactId>

<version>1.3.0</version>

</dependency>

<dependency>

<groupId>io.grpc</groupId>

<artifactId>grpc-protobuf</artifactId>

<version>1.3.0</version>

</dependency>

<dependency>

<groupId>io.grpc</groupId>

<artifactId>grpc-netty</artifactId>

<version>1.3.0</version>

</dependency>

</dependencies>

编写服务端的代码，引入rpc工程的依赖

public class GreetingServiceImpl implements HelloServiceGrpc.HelloService {

private static final Logger logger = Logger.getLogger(GreetingServiceImpl.class.getName());

public void sayHello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {

logger.info(String.format("sayHello方法调用的请求参数信息: name={%s}, id={%d}", request.getName(), request.getId()));

HelloResponse reply = HelloResponse.newBuilder().setMessage(String.format("Hello, %s", request.getName()))

.build();

responseObserver.onNext(reply);

responseObserver.onCompleted();

}

}

public class HelloServer {

private static final Logger logger = Logger.getLogger(HelloServer.class.getName());

private static final int DEFAULT\_PORT = 50051;

private Server server = null;

private void start() throws IOException {

server = ServerBuilder.forPort(DEFAULT\_PORT).addService(HelloServiceGrpc.bindService(new GreetingServiceImpl()))

.build().start();

logger.info("Server started, listening on " + DEFAULT\_PORT);

Runtime.getRuntime().addShutdownHook(new Thread() {

@Override

public void run() {

System.err.println("\*\*\* shutting down gRPC server since JVM is shutting down");

HelloServer.this.stop();

System.err.println("\*\*\* server shut down");

}

});

}

private void stop() {

if (server != null) {

server.shutdown();

}

}

/\*\*

\* Await termination on the main thread since the grpc library uses daemon

\* threads.

\*/

private void blockUntilShutdown() throws InterruptedException {

if (server != null) {

server.awaitTermination();

}

}

public static void main(String[] args) throws IOException, InterruptedException {

final HelloServer server = new HelloServer();

server.start();

server.blockUntilShutdown();

}

}

搞一个client端

public class GreetingServiceClient {

private final ManagedChannel channel;

private final HelloServiceGrpc.HelloServiceBlockingStub blockingStub;

public GreetingServiceClient(String host, int port) {

channel = NettyChannelBuilder.forAddress(host, port).negotiationType(NegotiationType.PLAINTEXT).build();

blockingStub = HelloServiceGrpc.newBlockingStub(channel);

}

public void shutdown() throws InterruptedException {

channel.shutdown().awaitTermination(5, TimeUnit.SECONDS);

}

public void sayHello(String name) {

try {

System.out.println("Will try to say Hello " + name + " ...");

HelloRequest request = HelloRequest.newBuilder().setName(name).setId(12345678).build();

HelloResponse response = blockingStub.sayHello(request);

System.out.println("result from server: " + response.getMessage());

} catch (RuntimeException e) {

System.out.println("RPC failed:" + e.getMessage());

return;

}

}

public static void main(String[] args) throws Exception {

GreetingServiceClient client = new GreetingServiceClient("127.0.0.1", 50051);

try {

String name = "Eric";

logger.info(String.format("Client 调用RPC接口，参数为name = {%s}", name));

client.sayHello(name);

} finally {

client.shutdown();

}

}

}

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/017\_引入核心数据的冷备解决方案从而保障集群数据不丢失/笔记.doc**

引入BackupNode可以作为冷备份的解决方案

冷备份：每隔一段时间可以去备份一下数据，每次恢复数据就是一部分，有些最新的数据一定是会丢失的

热备份：几乎是实时的在同步数据到其他的机器上去，如果说这台机器宕机，其他机器可以立马切换过来接管所有的操作

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/019\_基于gRPC实现目录创建接口的实现作为后续功能入口/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/001\_在大规模电商场景下为什么需要分布式文件系统？/笔记.doc**

NIO技术，磁盘读写，网络通信

仅仅只是那样来学NIO的话，其实是不够的

分布式存储：在很多台机器上通过分布式的手段来存储一些数据，这里其实是有很多种不同的分布式存储的类型的，分布式数据库，MySQL关系型数据库改成分布式的，NoSQL数据库，HBase，分布式存储数据

分布式文件系统，hadoop distributed filesystem，hdfs

你可以在他里面上传一些特别大的文件，比如说一个大文件，1TB，但是他可以把这1TB的大文件分布式存储在各个机器上，每台机器上就存储几百MB的数据，可能你的每个文件不大，都很小

比如说，你有很多的小文件，每个小文件可能就几MB，或者几百KB，几十KB，几KB。多达几百万个小文件，几千万个小文件，几亿几十亿个小文件，没办法都存储在一台机器上，此时你必须得把这些东西都放在一个多台机器组成的集群上

每台机器上可以放一部分的文件，所有的机器加起来，就可以存放所有的几十亿个小文件了

分布在多台机器上存放的文件通过什么来管理呢？分布式文件系统来进行管理，你得在多台机器上部署一套分布式文件系统，通过他来管理各种小文件

结合咱们的业务来，并发的时候，深入分析的是微服务注册中心的一个项目，还没做完，架构有很多缺陷，此外网络通信那块没跑通， 还没集成到spring cloud里去，是跟我们整个课程的大的背景是相关的

电商项目，之前的话呢，我们已经看到那个项目里的很多东西了，在那个平台里，其实是有很多的文件需要存储的，主要是各种各样的图片。其实啊，如果你是仅仅是做图片存储的话，你完全是可以基于专业的图片服务器来进行搭建的，你可以基于多台专业的图片服务器搭建一套集群，来存放 你对应的图片

你也可以尝试来自己研发一套分布式文件存储系统，来存放电商平台里海量的图片，还有系统里其他的一些文件这样子

fastdfs，c开发的一套分布式文件系统，特别适合用作大量小文件的分布式存储系统，你可以基于fastdfs来构建一个分布式文件系统集群，基于他来存储大量的电商网站里的各种图片，商品的图片，发表评论的时候可以上传图片，一般都是很小的，可能就是几百kb居多一些

你如果不用fastdfs，那么就需要你的电商系统自己不是部署在linux上，你就需要电商系统本身把上传的文件写入到本地的linux的某个文件目录下面去，但是现在有了fastdfs搭建一套图片服务器的集群，文件可以上传到分离开来的图片服务器集群上去

图片服务器集群是基于fastdfs来部署的，在一堆服务器上部署了一套fastdfs，你可以把图片上传给他，就把图片从电商系统自己本身给分离了出去，在页面上，他的图片的在html里的连接，就可以直接是指向某个专门用于显示图片的nginx，这个nginx就可以直接发送请求到后台的fastdfs集群上去读取文件

最后返回给浏览器来显示图片

就可以把图片这个事情给专门的拆分出去

为什么需要自己来开发一套分布式文件系统替代掉fastdfs呢？第一个最大的原因，fastdfs社区并不是太活跃，里面其实是有很多的bug和坑的，如果你真的用在生产环境里，fastdfs会有很多的问题

fastdfs是用c写的，你不可能看他的源码，很难去解决和处理他里面发生的一些故障什么的

真的需要用图片存储服务的话，现在很多中小型公司会选择采用云服务商，采用一些云平台上的图片存储服务，按需付费，使用了多少存储空间，就给他付费多少钱，人家云平台就是自己构建了一套分布式文件系统

但是对于我们这个课程来说，完全可以自己尝试仿照hadoop hdfs结合他的架构，参照电商平台海量小文件存储的需求，来定制开发一套属于我们自己的分布式文件系统，来支撑我们的电商平台的海量图片存储的需求

后面我们还可以在高并发那个环节，会有一块，图片服务器的分离架构，就可以把我们自己研发的分布式文件系统引入进去，把所有的文件上传到自己的分布式文件系统上去，通过nginx的配置，来展示图片出去

最大的好处是什么呢，你出去找工作之类的，完全可以用这个项目写到自己简历里去，人家是会信服的，因为你可以说我们公司需要把核心的一些图片等数据保留在自己这里，所以说选择基础架构团队自己开发了一套分布式文件系统来存放我们系统里的核心的图片数据，因为开源的fastdfs肯定是不好用的

如果用第三方的云服务，其实也暂时不能完全去信赖他们

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/004\_分布式系统的Master-Slave架构应该如何设计/笔记.doc**

分布式系统是非常非常复杂的，如果你哪怕去看一些开源项目的源码，都不一定能看得懂，何况说自己设计和开发呢，所以我们一般是采取迭代式的方式，一步一步的来做，来分析，一点一点的把他最终给做出来

我坚持一定要先讲解分布式存储系统，微服务注册中心，这两个项目呢？在后面比如学到zookeeper的时候，完全是融入到这两个项目里去实战的，因为一般来说zk常用的场景都是一些基础架构层面的分布式系统

如果是业务层面的分布式系统，基于dubbo或者是spring cloud开发的一些分布式系统，一般很少用到zookeeper，课程设计和讲解，层层递进的，把一些前面的技术先搞明白了，才能往后学习

我在前面的时候先学习sprign cloud和源码，再去讲解并发这个课，必须得依托于一些底层中间件的项目和源码来讲解，微服务注册中心这个项目去实战驱动讲解并发的一些技术，才能继续往后学

分布式文件系统里，需要两个角色，一个角色是数据管理的角色，他是一个进程，每台机器上都需要部署一个这个进程，他负责管理这台机器上的存放的那些小图片和小文件；另外一个角色，是负责管控整个集群的角色，他负责感知各个机器上的进程状态，每台机器上存放图片的情况，剩余多少存储空间

客户端需要来上传图片，就可以先找管控进程，告诉他哪台机器上有空间，客户端就上传图片到那台机器上去就可以了

另外一种不一样的架构，是类似于Peers架构，Spring Cloud Eureka采取的就是Peers架构，Eureka可以部署任意多台机器，每台机器都会互相同步注册表的数据，每台机器里的数据是一样的，每台机器都是对等的，Peers

Slave是奴隶的意思，Master是主人，一个主人管控多个奴隶来干活儿

Master-Slave架构里，最最核心的一点就是心跳机制，heartbeat机制，euerka那块的时候已经看到了，每个服务实例都是一个eureka client，他会负责不停的上报心跳给euerka server来证明自己还存活着

如果eureka server一段时间过后发现某个euerka client一直没上报心跳，就会认为那个服务实例已经死掉了，此时就会从服务注册表里摘除这个服务实例，其他的服务会不停的来拉取最新的服务注册表，此时感知到某个服务实例宕机之后

其他服务就不会再去访问那个宕机的服务实例了

Master负责管控集群，Slave负责管控某台机器上的数据

Slave必须不停的发送心跳给Master

Master如果发现某个Slave一段时间内没上报心跳，此时认为他宕机

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/006\_完善Master-Slave架构下的集群注册的代码/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/001~021资料/007\_在Slave节点中实现后台线程定时上报心跳机制/笔记.doc**

其实没多大的难度，因为整体的实现机制跟微服务注册中心是差不多的，各种分布式类的系统，心跳机制都是类似的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/197\_实现rebalance的核心算法：集群资源重平衡算法逻辑/笔记.doc**

假设现在

机器01：90GB

机器02：90GB

机器03：90GB

机器04：90GB

机器05：0GB

机器06：0GB

60G

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/186\_实现重新分配数据节点这个接口的代码业务逻辑/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/195\_到底什么时候应该从磁盘快满的机器缓慢迁移数据出去/笔记.doc**

机器01：90GB

机器02：90GB

机器03：90GB

机器04：90GB

机器05：0GB

机器06：0GB

把所有的机器全部加起来，算一个平均数，360GB，60GB

机器01、机器02、机器03、机器04 -> 机器05、机器06

机器01 -> 机器05，30GB，对应的是哪些文件，生成两种任务，第一种任务是复制任务

对于机器05而言，有复制任务，他需要从机器01复制指定的文件过来

每隔一段时间，你都可以让他去进行一次全量存储的汇报，把 这个节点原先的各种存储信息重新刷新一遍，同时在全量存储汇报的时候，就可以检查一下每个图片的副本数量，如果超出了2，就生成删除任务也可以

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/185\_定义一个新接口：重新分配数据节点以及排除故障节点/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/192\_海量数据存储：分布式存储、多副本冗余以及高可用架构/笔记.doc**

海量数据的存储，针对的各种小图片

分布式存储架构，多副本冗余，高可用架构

一边要多思考里面的架构设计思想，FastDFS和TFS，作业，FastDFS是一个国产的开源项目，c语言开发的，中小型公司，一般在分布式文件存储的场景中，都会采用FastDFS来使用，c语言开发，我们没办法阅读里面的源码

出问题的时候极坑，全部c异常，Java工程师没有能力维护FastDFS集群的

百度“FastDFS架构原理”，分布式存储、副本冗余、高可用架构，跟我们设计的这套架构类似的

TFS，淘宝内部开发的分布式文件系统，淘宝上面大量的店铺中的商品的小图片，4kb~400mb之间，分布式存储，元数据管理机制，副本冗余，高可用架构。很多小图片会被合并为一个大文件来存储，每个文件都会有一个对应的索引文件

我们这边参考了HDFS的架构，Hadoop分布式文件系统，分布式计算系统，分布式资源调度系统，尤其是里面的元数据管理架构

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/189\_改造代码实现下载文件发现网络故障重新申请一个数据节点/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/194\_上线新机器之后是否会自动优先往里面写入数据？/笔记.doc**

如果每次扩容，一般来说针对我们的这个系统，要么不扩容，要么扩容都是>=2台机器起步来进行扩容，每次扩容2台机器，那么在这里一排序，就会优先往2台空的机器里写入数据，就可以立马缓解住已有4台机器快满的压力

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/198\_实现rebalance的核心算法：集群资源重平衡算法逻辑（2）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/191\_重写为下载文件分配数据节点的接口：加入排除故障节点逻辑/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/193\_分布式文件系统的可伸缩架构值的是什么以及如何设计/笔记.doc**

收尾了，分布式存储架构，容错架构，高可用架构，可伸缩架构，高性能架构，高并发架构

可伸缩架构，集群而言，加机器去里面，或者是下线机器，要可以实现

加一台机器，你接下来如何做，假设4台机器，每台机器上的磁盘空间都快满了，这个是很常见的场景，大数据的同学，HDFS的时候，已有的几台机器的磁盘空间都满了，无法写入新的数据了

首先你得保证接下来要优先往最空的这台机器去写入数据，接下来在后台应该启动一大堆的定时任务，要慢慢的把快满的4台机器上的数据逐步逐步的迁移到空的机器上去，缓解已有4台机器的存储压力，可以让空机器放更多的数据

下线机器，有某一台机器不需要了，关闭DataNode，把机器干掉，属于机器的宕机，人家感知到，副本会自动的去进行复制，保证数据不丢失

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/184\_改造代码实现发现网络故障时重新分配一个数据节点/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/187\_在下载文件的时候发现数据节点宕机该如何进行处理？/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/190\_改造已有的旧接口：为下载文件分配一个数据节点/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/182\_在上传文件的时候发现数据节点宕机该如何进行处理？/笔记.doc**

最最典型的一个客户端上传的容错机制，就是感知到网络故障之后，就得去进行一些容错的处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/183\_在客户端的代码中找找如何感知到上传过程中的网络故障？/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/196\_定义一个新接口：手动触发集群数据的rebalance/笔记.doc**

上线了新的几台机器，立马就应该执行一个命令，提供一个用python编写的脚本，在脚本里基于python调用gRPC提供的接口，调用到Master上去，执行某个命令，gRPC本来就是支持多语言的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/199\_实现rabalance的核心算法：集群资源重平衡算法逻辑（3）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/200\_基于可伸缩架构实现集群扩容支撑海浪数据的存储/笔记.doc**

海量数据存储架构：分布式存储架构 + 可伸缩架构

高可用+高容错架构：多副本冗余 + 副本自动迁移 + 冗余副本自动删除 + 客户端容错机制

高性能架构：尽可能提升客户端文件上传和下载的性能和速度

高并发架构：尽可能让每个数据节点可以支撑更多的客户端的并发上传和下载

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/182~200资料/188\_在客户端的代码中找找如何感知到下载过程中的网络故障？/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/232\_在核心的poll方法中对机器返回的响应进行读取/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/235\_发送请求的过程中遇到了超时问题该如何处理？/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/228\_将封装好的请求放入NetworkManager的请求队列中/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/239\_解决异步化之后对于重试时遗留的一个错误程序（1）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/236\_改造请求发送为异步化：调用方设置回调函数的模式（1）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/230\_对每个数据节点获取一个请求缓存起来等待发送/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/233\_建立连接以及发送请求过程中异常了该如何返回响应？/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/223\_回头审视一下客户端的短连接模式有哪些问题？/笔记.doc**

就是他需要从其他的数据节点拷贝副本过来写入本地，短连接，也无所谓，因为这个过程都是后台慢慢执行的，但是当然最好也是重构成长连接模式

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/222\_分布式图片存储系统中的高性能指的到底是什么？/笔记.doc**

重构系统架构实现一个高性能，就要做非常完善的一个测试，对这个系统做一个总结，说说后续我们还要做一些什么东西，另外，我还要给大家留一些作业，相当于是让大家课后自己去做的，就不是完全拷贝我的代码

两块

第一块：客户端现在是短连接，每次发送请求，都需要建立 连接，然后断开连接，站在客户端的角度而言，发现每次执行一个文件上传和下载的操作，速度都很慢

第二块：文件上传，需要多副本上传，一般来说，针对kafka，多副本的时候默认情况下只要写成功一个副本，就返回了。另外其他的副本的写都是异步慢慢来执行的，采取的是副本pull数据的机制

kafka，强调高性能，生产消息的行为都是以尽快的可以完成

hdfs，不强调高性能，几个GB的大文件上传到服务器上去，主要慢慢上传就可以了，速度慢点无所谓。多个副本一定要依次上传成功，才可以说是本次文件上传成功了。上传速度肯定是很慢的，高性能。

高性能架构的重构：短连接 -> 长连接；同步上传多副本 -> 写一个副本，其他副本在后台慢慢的异步复制和拉取

文件上传和文件下载，性能至少会提升好几倍

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/225\_在NetworkManager中实现核心线程无限循环进行poll操作/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/224\_初步实现用于进行网络管理的NetworkManager组件/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/231\_在核心的poll方法中将每个机器暂存等待的请求发送出去/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/229\_如何实现异步发送请求以及同步等待响应两个接口/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/226\_在无限循环的poll方法中完成网络连接的建立/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/234\_建立连接以及发送请求过程中异常了该如何返回响应？（2）/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/222~244资料/227\_客户端的核心业务方法对要发送的请求进行封装/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/178\_将冗余副本的删除任务下发给对应的数据节点/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/161\_迄今为止已有架构通过一张图来整体梳理一下/笔记.doc**

why

HDFS的架构：文件目录树的更新、磁盘随机写

对于我们的这套架构，里面有一些高可用的问题的，我们后面给大家来分析

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/177\_接收数据节点存储上报的时候发现副本冗余生成删除任务/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/172\_通过心跳响应让目标数据节点接收一个复制任务/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/166\_感知到副本数量缺失之后触发副本复制的任务/笔记.doc**

一旦感知到了一个DataNode宕机，此时就应该触发这个DataNode上的所有的副本都需要新增一个异步复制的任务，交给其他的DataNode来进行副本复制，保证说每个图片都有2个副本

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/175\_基于NIO实现两个数据节点之间进行副本的复制/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/170\_将创建好的副本复制任务放入目标数据节点的任务队列/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/167\_完善副本数据结构，需要知道每个数据节点有哪些文件副本/笔记.doc**

每个datanode他是有哪些文件的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/169\_为副本复制任务去获取可用的源头数据节点/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/164\_如果Master节点宕机之后如何手动完成主备切换？/笔记.doc**

搞一台新的机器，然后在上面放上去一个最新的fsimage，从BackupNode上来，接着部署一个新的NameNode，然后启动一个新的NameNode，对外来提供服务，比较挫的一个办法，就是得重新部署所有的系统，包括客户端，包括DataNode，包括BackupNOde

在原来的机器上，尝试重启NameNode，所有人都可以基于原先配置的hostname之类的东西可以瞬间感知到

只能是用backupnode他的fsimage在新的一个机器上来部署，你要是实现NameNode高可用，必须得是引入ZooKeeper概念，NameNode的地址应该是写到zk里去的，然后其他的bakcupnode，datanode，客户端感知他的地址，都是通过zk来的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/163\_先来看看分布式文件系统存在哪些高可用的问题？/笔记.doc**

如果你在设计一个系统的架构的时候，高可用，你的系统如果有任何一个部分挂掉了，此时你的系统如何继续保持对外是可用的

这个分布式文件系统，讲完了以后，Netty，就会把分布式微服务注册中心给搞定，架构升级，网络通信，当时网络通信的环节都没加入进去，他的架构参考比如阿里的Nacos等等给他做一个架构升级

我们会讲解ZooKeeper，就是放在分布式文件系统里，做双NameNode热备热切换的高可用的架构设计，然后的话就是用ZooKeeper再看看在微服务注册中心里有没有使用的场景

Dubbo，把最新的版本，源码，以及底层的Netty、ZooKeeper的使用都一起讲了，结合别的业务讲一个小的项目出来

（1）副本数量的缺失，每个图片都是双副本，如果一个DataNode挂掉了，势必会导致一批图片只有单副本，如果此时再挂一台机器，就会导致部分图片就丢失掉了

（2）如果某个客户端正在向DataNode上传图片，此时他突然挂了，客户端发现图片上传失败了，此时如何处理？

（3）如果某个客户端正在从DataNode读取图片，此时他突然挂了，客户端发现图片读取失败了，此时如何处理？

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/165\_如果一个数据节点突然宕机了，Master节点如何感知到？/笔记.doc**

我们来看一下这一块，如果DataNode一旦宕机了，此时NameNode如何可以感知到？

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/173\_目标数据节点将接收到的复制任务放入本地任务队列中/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/176\_如果宕机的数据节点事后再次重启会发生什么事情？/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/168\_为宕机的数据节点上的每个副本都生成一个复制任务/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/171\_目标数据节点发送心跳时将复制任务下发给该节点/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/161~181资料/162\_再来通过一张图梳理一下分布式存储的整体架构思想/笔记.doc**

不同的分布式系统，分布式NoSQL数据库（HBase），分布式文件系统，分布式缓存系统，分布式消息系统，分布式数据库，分布式微服务注册中心，不同的分布式系统，其实他的元数据都是不一样的

在管理不同的元数据的时候，所采取的策略也是不一样的

HDFS，他已经全世界规模最大，最成熟，最优秀的一种分布式文件系统了，他的元数据管理机制的设计都是经过大量的实践和考量的，我就给大家详细分析了为什么要那样设计一套架构

对技术的理解和掌握，大致分为以下几个level：

学习过我们的亿级流量，es，面试突击，加上你平时自己会自学一些技术，比如说mq等等，各种技术都知道，初步的能用各种技术做系统

还学过springcloud、分布式事务、分布式锁、并发，如果你学习过这样的一些技术，相当于就是读过了一些开源项目的源码

你能够自研分布式系统、中间件系统，能够基于最底层的数据结构、并发编程、网络通信、磁盘读写，去研发出来适用于特定场景的分布式系统，里面的架构设计的，机制的设计，你都能够玩儿的转

MQ、数据库中间件、ES、ZooKeeper、Dubbo，一些流行的分布式系统的源码，这些技术的落地使用

结合业务，微服务架构，性能优化，高可用方案，高并发方案，把常见的技术在业务中的架构设计的方案，都能搞一遍

七八十万年薪的架构师，或者是技术专家的职位，妥妥的

高大上的架构，那个时候朝着的就是年薪百万以上的架构师，首席架构师，或者高级技术专家 的职位去走，大量的高大上的架构

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/101\_重新审视一下数据节点的NIOServer接收图片的代码/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/109\_在Master节点中实现增量上报的副本接收成功消息的处理逻辑/笔记.doc**

其实就是在内存的文件目录树里给一个文件新增一个副本所在的DataNode

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/118\_文件上传全流程测试：增量上报存储信息功能/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/116\_Master节点重启后如何指示数据节点重新注册以及全量上报？/笔记.doc**

Master节点重启了，此时会恢复文件目录树，但是此时所有的datanode信息都丢失，datanode还是会一如既往的发送心跳过来，此时如果接收到心跳，发现datanodes数据结构里没有他，就会让他重新注册以及上报全量的存储信息过来

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/121\_文件上传全流程测试：集群优雅关闭后重启的数据重建/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/103\_在二进制格式的请求中如何放入文件名、文件大小以及文件数据？/笔记.doc**

我们需要在请求中放入文件名、文件大小、文件数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/111\_先来设计Master节点的全量存储信息上报RPC接口/笔记.doc**

先给他设计出来，接下来，对照着三种要先实现的场景把这个全量上报的逻辑来写一下

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/107\_为Master节点设计一个增量存储信息上报的RPC接口/笔记.doc**

namenode的文件目录树里就有一个文件了

而且多个datanode也接收到了上传的文件副本了，就下来就是得让namenode要感知到这个文件有哪些副本，分别在哪些机器上，这里就涉及到增量上报

这个datanode存储数据量的大小，其实在之前申请分配节点的时候已经累加上去了

设计一个RPC接口，namenode可以接收一个请求，datanode说我收到了一个文件对应的副本

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/100\_完善客户端基于NIO依次对两个数据节点上传图片的代码/笔记.doc**

79讲，99讲，结果给写错了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/106\_阶段测试之文件上传以及多副本存储机制彻底打通/笔记.doc**

解决了几个bug

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/105\_阶段测试：文件创建+副本分配+网络通信+磁盘存储全流程/笔记.doc**

BufferUnderflowException

一般来说是你使用Buffer的时候忘了flip

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/114\_小范围测试以及完成数据节点启动时的全量存储信息上报/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/117\_文件上传全流程测试：审查代码流程以及加入调试日志/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/113\_实现数据节点启扫描与计算本地存储信息的代码逻辑/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/108\_在数据节点中实现接收副本之后增量上报存储信息的逻辑/笔记.doc**

一旦数据节点接收到了一个文件副本之后，就会上报存储信息给Master节点感知到

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/112\_整个集群启动时Master节点应该如何处理全量上报的存储信息？/笔记.doc**

集群比如说刚刚启动，Master恢复自己的元数据，DataNode会上报自己的存储信息过去，我们就先来处理一下这种情况下，Master收到请求之后应该如何重建自己本地的存储信息的数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/115\_数据节点快速重启的时候如何避免重复上报存储信息？/笔记.doc**

现在master和datanode都运行的好好的，结果datanode突然临时重启了一下，但是很快就恢复过来了，快到master都没感知到他的宕机，比如说两三分钟他就完成了重启，其实你可以这么来做

全量存储信息的上报，不需要你来搞，就是说是这样子的，你重启之前接收到的所有的文件和你存储的数据量的大小，都在master内存里存着呢，你的数据又没有变化，所以此时可以想一个办法避免重复全量上报

DataNode长时间停机或者是宕机，那个是在高可用的环节里来讲解

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/102\_通过文件名的同步在每个数据节点上保证图片文件名相同/笔记.doc**

第一个办法，就是在上传文件的时候，就把文件名给上传过去

第二个办法，就是在上传文件的时候，先用一个随机的文件名，接着上传成功之后，客户端再发送一次RPC的请求把文件名给传输过去，让他进行目录的构建以及改名字

可以考虑去使用第二个办法，也可以考虑使用第一个办法，在你传输的数据里，必须先有4个字节是int类型的，他是代表了文件名的长度，接着就是实际的多个字节的文件名，接着是文件的数据

你在解析的时候就必须要按照这个思路来进行解析，是比较常见的一个办法

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/119\_文件上传全流程测试：数据节点快速重启避免重复上报/笔记.doc**

DataNode短时间重启不会导致全量上报

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/120\_文件上传全流程测试：Master节点重启后指示注册和全量上报/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/099\_完善数据节点的核心配置信息以及上报拉取的过程/笔记.doc**

搭建好了客户端和数据节点传输一个图片文件的基础的代码，还没测试，主要是因为很多的代码还是没有完善好呢，所以先不要测试，等我们全部都搞定了以后我们再统一的测试一下整个完整的流程

写中间件的代码一定是不能着急的，要一点一点的写，考虑清楚很多里面的细节，就是一个是主体的架构设计、核心的运行流程（主要靠你自己把各种细节考虑清楚）、其他的一些注意的点（主要是靠想）

很多人都说，我这个去年写的代码自己现在去看都觉得很陌生

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/104\_数据节点接收完文件之后如何让Master节点感知到？/笔记.doc**

假设数据节点接收完了文件之后，他应该让master节点可以感知到说他这里存放了某个文件的一个副本这样子，每次datanode接收完毕了文件之后，就应该走一个RPC接口调用去请求master节点

让master节点知道说，某个文件在哪台机器上有一个副本

如果master节点重启了之后，他只能从自己的元数据里恢复出来有哪些目录和文件，但是恢复不出来到底每个文件的副本存储在那些机器上的，就是应该还要在每个数据节点在心跳的时候就需要去上报自己这里存储的所有的文件的信息给master节点

master节点每次收到心跳就知道datanode上当前存储了哪些文件，就是如果说每次心跳就传递全量的当前自己管理的文件的信息的话，可能涉及到了几万个文件？几十万个文件这样子？心跳的数据量会很大

正常来说如果master正在运行中，datanode就直接上报自己受到的文件即可，master在内存中就可以维护文件的每个副本在哪台机器上了。如果说master重启了，第一次收到datanode的心跳，发现datanode此时不在自己的内存数据局结构里

此时master应该指示datanode重新进行一次注册，在注册的时候才会带过来全量的文件存储的信息，包括datanode自己存储了哪些文件，还有就是datanode当前使用磁盘空间的总量，都应该带过来

但是只要注册完毕之后，后续datanode就是每次收到文件，或者是删除文件，对文件追加写才会增量上报存储信息

增量上报

全量上报：Master先启动，Datanode再启动；Master重启，DataNode重新注册

把已有的代码给先测试一下，保证说内存文件目录树里的文件的创建 + 申请多个机器上传多副本 + 实际的网络传输文件上去 + 数据节点本地磁盘的文件存储，这些东西先搞定，都是没有问题的

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/099~121资料/110\_实现数据节点重启时向Master节点全量上报存储信息的逻辑/笔记.doc**

NameNode先启动，然后DataNode来启动，此时一定会执行一个全量上报；如果是DataNode自己重启，但是NameNode可能还没感知到他故障了，此时就可以直接让DataNode重启之后上报自己全量的存储信息过去

NameNode又要提供一个专门的接口，用来接收全量上报的逻辑

第一个情况，NameNOde先启动，然后DataNode再启动：扫描自己本地目录下所有的文件，根据文件所在的目录反向拼接出来他对应的相对路径，而且计算所有文件加起来所占的磁盘空间大小，收集好，上报给NameNode

NameNode收到全量上报的请求之后，直接往自己的对应的数据结构里去塞就可以了

第二个情况，DataNode他突然重启了，重启的时间间隔很短，此时NameNode还没判定这个DataNode宕机什么的，此时DataNode他突然全量上报自己的存储信息，你觉得NameNode要怎么来处理呢？NameNode本身内存里还存储着DataNode重启之前的存储的数据

第三个情况，DataNode突然宕机，此时NameNode感知到了，要做一系列的事情，然后DataNode又重启了，全量上报，此时NameNode要做一系列的复杂的事情，我们要跟着后面的高可用架构的实现

第四种情况，NameNode自己突然重启，DataNode们在一段时间内都感知不到NameNode了，此时NameNode重启了，DataNode都会自然的发送心跳过去，NameNode发现人家发送心跳过来，需要指示他们重新注册以及重新全量上报存储信息

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/056~077资料/057\_基于NIO网络通信将fsimage文件传输到NameNode（二）/笔记.doc**

服务端的接受fsimage的代码就写完了

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/056~077资料/059\_基于NIO网络通信将fsimage文件传输到NameNode（四）/笔记.doc**

测试，namenode自己这里写editslog都成功的，backupnode同步editslog都是成功的，backupnode checkpoint写入自己本地磁盘文件也是成功的，上传fsimage到namenode去写入磁盘是否能够成功

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/056~077资料/071\_整体测试NameNode的优雅关闭以及重启恢复的全流程/笔记.doc**

先发送1000条数据：2个磁盘文件 + 内存缓冲里的部分数据

backupnode执行一次checkpoint：fsimage -> 1000条数据

再次发送1000条数据，多出来几个磁盘文件，内存里有部分数据，但是此时肯定有结果磁盘文件的数据是不包含在fsimage里的

优雅关闭namenode：内存缓冲的数据刷入磁盘上，checkpoint txid刷入磁盘

重启namenode：恢复数据（fsimage + 上次checkpoint之后的所有的数据），backupnode会继续拉取

在磁盘文件名称前面给他补很多的0

32个0

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/056~077资料/064\_NameNode的优雅关闭机制再次完善：禁止访问以及持久化/笔记.doc**

内存缓冲里的editslog必须写到磁盘上去，持久化checkpoint txid，fsimage加载到内存里去，根据checkpoint txid扫描磁盘上的editslog，txid之后的editslog回放到内存里，就可以恢复出来完整的数据了

syncedTxid，已经保留在backupNode那边，不应该保留在NameNode这儿，不停的拉取的过程中会更新，但是如果说你更新了syncedTxid以后，持久化到磁盘上去了，还没来得及返回这批数据给backupNode，你就重启了

每次发送fetch的请求过来，都应该带上一个syncedTxid，我上次拉取完的一批数据最大的一个txid是多少，这次你要从syncedTxid + 1开始继续帮我来拉取

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/056~077资料/058\_基于NIO网络通信将fsimage文件传输到NameNode（三）/笔记.doc**

完成backupnode那块的代码

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/056~077资料/056\_基于NIO网络通信将fsimage文件传输到NameNode（一）/笔记.doc**

我们可以参考之前的NIO网络通信那块的代码来做一下

在editslog、fsimage、checkpoint这块机制里，完全是两个进程之间自己在做一些低频率的写磁盘，网络通信的事情，这块东西不涉及到什么特别复杂的技术难点，一般出问题的概率很低，所以说直接基于我们之前的那套最简单的NIO代码也可以来实现

分布式文件的上传，你上传文件的时候，所有文件分散在多台机器上，分布式的存储，上传文件的量很大，每天可能都是几万个文件，每个文件还要在集群里自动传输多个文件的副本，那块网络通信就比较复杂了

很多NIO里，网络通信这块，很多高阶的技术，参数，要考虑的点，网络故障和异常，都需要在那里来考虑到，对于hadoop hdfs，在editslog、fsimage、checkpoint那块的写磁盘和网络通信的机制，都是非常简单的

checkpoint做文件上传的时候，HTTP协议，namenode启动一个http server，backupnode就是发送一个http put请求，把文件传输过去了，在文件上传和读取的时候，量很大，基于NIO来做的，很多复杂的参数，机制，设计，故障，那块代码就是非常的复杂

在文件上传和读取的过程中，来实战和演练NIO复杂的网络通信高阶的一些技术

分成两块来写，第一块就是先写服务端，第二块是写客户端，第三块就是测试，checkpoint机制串起来测一下，看看是否可以正常的把一个文件上传过去

# **File Path: /Users/xiaotingting/Downloads/学习资料/11\_支撑1亿图片的分布式存储系统项目实战unzip/056~077资料/060\_基于RPC请求通知NameNode重置editslog txid（一）/笔记.doc**

你一旦上传完毕了一个fsimage文件之后，就可以发送rpc请求，去namenode更新一下当前的fsimage对应的一个最大的txid是多少