对Hadoop HDFS进行学习和研究，整理成文：

概述：HDFS即Hadoop Distributed File System分布式文件系统，它的设计目标是把超大数据集存储到分布在网络中的多台普通商用计算机上，并且能够提供高可靠性和高吞吐量的数据存储服务。分布式文件系统要比普通磁盘文件系统复杂，因为它要引入网络编程，分布式文件系统要容忍节点故障也是一个很大的挑战。

# 一、HDFS设计目标

## 适合的场景

* **专为存储超大数据集而设计：**HDFS应该能够支持GB级别大小的文件；它应该能够提供很大的数据带宽并且能够在集群中拓展到成百上千个节点；它的一个实例应该能够支持千万数量级别的文件。
* **适用于流式的数据访问：**HDFS一般适用于批处理的情况而不是交互式处理；它的重点是保证高吞吐量而不是低延迟的用户响应。
* **容错性：**完善的冗余备份机制。
* **支持简单的一致性模型：**HDFS需要支持一次写入多次读取的模型，而且写入文件不会发生经常变化。
* **移动计算优于移动数据：**HDFS提供了使应用计算移动到离它最近数据位置的接口。
* **兼容各种硬件和软件平台**

## 不适合的场景

* **大量小文件：**文件的元数据都存储在NameNode内存中，大量小文件会占用大量内存。
* **低延迟数据访问：**HDFS是专门针对高数据吞吐量而设计的。
* **多用户写入，任意修改文件**

# ****二、HDFS整体架构****

HDFS主要由3个组件构成，分别是**NameNode**、**SecondaryNameNode**和**DataNode**，HDFS是以master/slave模式运行的，其中NameNode、SecondaryNameNode 运行在master节点，DataNode运行slave节点。

整体架构引用官网截图如下：



上图展示了HDFS的NameNode、DataNode以及客户端的存储访问关系，解释如下：

* **MetaData ops：**NameNode接收用户的文件访问请求，完成元数据的更新操作，并返回文件的元数据信息。
* **Block ops：**NameNode决定block到具体DataNode节点的映射，Datanode在Namenode的指挥下进行block的创建、删除和复制操作。
* **Read、Write：**对HDFS存储的用户文件的读、写操作直接在DataNode上完成，不需经过NameNode。即NameNode只管理HDFS系统的元数据信息，不存储具体的用户数据。
* **Replication：**出于可靠性的考虑，每个块都会复制到多个DataNode服务器上。

## **File System NameSpace**

任何一个文件系统几乎都有命名空间（Namespace）的概念，HDFS也不例外。它采用类似于Linux文件系统目录树的结构来实现，并由NameNode对命名空间进行管理和维护。通过命名空间来表现和定位一个文件，来支持创建、移动、删除、拷贝、重命名文件或者目录等操作。

更多关于Linux文件系统的知识请移步参考：<http://vbird.dic.ksu.edu.tw/linux_basic/0230filesystem.php#harddisk>

## **NameNode**

NameNode负责维护所有文件或目录的打开、关闭、移动、重命名等操作。

正如上面所述，NameNode节点管理和维护着文件系统树及树中的所有文件和目录的元信息。因此任何客户端的文件存取请求，首先要经过NameNode获取文件的元信息，然后由客户端直接与相应的DataNode进行交互，因此实际文件数据的传输不需要经过NameNode。

此外，NameNode还会负责监控集群内所有DataNode节点，并管理所有数据块的备份（可靠性）与迁移（负载均衡），这是通过它与集群内每个DataNode节点维持心跳，并接收DataNode发送过来的Blockreport来实现。

### **1) NameNode中的元信息**

NameNode中维护的元信息不仅仅是文件系统命名空间，由于HDFS自身支持大数据集和高容错等特性，它将一个文件在物理上分成多个块存储，同时为了保证高容错性，它还会将每个块进行多个副本备份，因此NameNode在维护命名空间的同时，还需要维护这些信息，记录每个文件副本有多少个、文件的所有数据块有哪些、分别在哪些节点等。

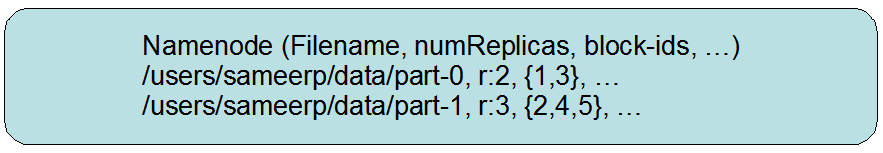
当一个客户端请求一个文件或者存储一个文件时，它首先需要知道具体到哪个DataNode上去存取，获得这些信息后，客户端才能和这个DataNode进行交互。

NameNode中元信息的内容包括：

* 文件名目录名及它们之间的层级关系
* 文件目录的所有者及其权限
* 文件副本数量以及由哪些块组成、每个文件块的名称

需要注意的是，**NameNode元信息并不包含每个块的位置信息**，这些信息会在NameNode启动时从各个DataNode获取并保存在内存中，因为这些信息会在系统启动时由数据节点重建。把块位置信息放在内存中，在读取数据时会减少查询时间，增加读取效率。NameNode也会实时通过心跳机制和DataNode进行交互，实时检查文件系统是否运行正常。不过NameNode元信息会保存各个块的名称及文件由哪些块组成。

以下是从官网上截取的NameNode元信息模样：



一般来说，一条元信息记录会占用200byte内存空间。假设块大小为64MB，备份数量是3 ，那么一个1GB大小的文件将占用16\*3=48个文件块。如果现在有1000个1MB大小的文件，则会占用1000\*3=3000个文件块（多个文件不能放到一个块中）。我们可以发现，如果文件越小，存储同等大小文件所需要的元信息就越多，所以，Hadoop更喜欢大文件。

此外，由于客户端的读写请求都要访问元数据，因此需要将HDFS元数据放在NameNode节点的内存中，以快速响应客户端的请求。

### **2) NameNode元信息的持久化**

在HDFS中NameNode上的元信息是整个系统的核心数据，全部放在内存中也存在一些问题：如果NameNode节点宕机就会丢失所有的HDFS元数据信息，会牵连影响到整个系统的所有数据。因此，很容易想到的就是将这些元数据在磁盘中也存储一份，当元数据改变时，同步更新内存和磁盘上的元数据。

NameNode会在本地磁盘中持久化两个文件：

* **fsimage**是元数据的镜像文件，存储某一时段NameNode内存中的元数据信息
* **edits**是一个操作日志文件，记录文件系统的改动信息

在系统运行期间所有对元信息的操作都保存在内存中并被持久化到edits文件中。并且edits文件和fsimage文件会被SecondaryNameNode周期性的合并（合并过程会在SecondaryNameNode中详细介绍）。

### **3) 其它问题**

运行NameNode会占用大量内存和I/O资源，一般NameNode不会存储用户数据或执行MapReduce任务。

为了简化系统的设计，Hadoop只有一个NameNode，这也就导致了hadoop集群的单点故障问题。因此，对NameNode节点的容错尤其重要，hadoop提供了如下两种机制来解决：

* 将hadoop元数据写入到本地文件系统的同时再实时同步到一个远程挂载的网络文件系统（NFS）。
* 运行一个SecondaryNameNode，它的作用是与NameNode进行交互，定期通过编辑日志文件合并命名空间镜像，当NameNode发生故障时它会通过自己合并的命名空间镜像副本来恢复fsimage。需要注意的是SecondaryNameNode保存的状态总是滞后于NameNode，所以这种方式难免会导致丢失部分数据（后面会详细介绍）。

## **DataNode**

DataNode是HDFS中的worker节点，负责存储数据块，也负责为系统客户端提供数据块的读写服务，同时还会根据NameNode的指示来进行创建、删除、和复制等操作。此外，它还会通过心跳定期向NameNode发送所存储文件块列表信息。当对HDFS文件系统进行读写时，NameNode告知客户端每个数据驻留在哪个DataNode，客户端直接与DataNode进行通信，DataNode还会与其它DataNode通信，复制这些块以实现冗余。

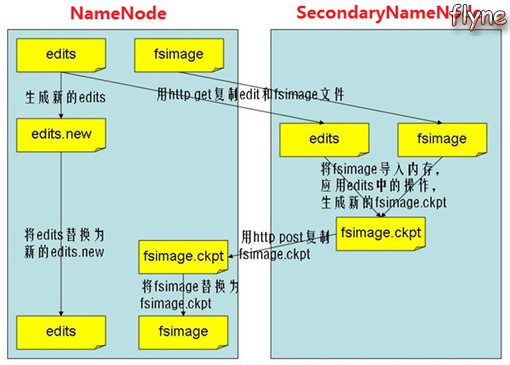
## **SecondaryNameNode**

SecondaryNameNode并不是NameNode的备份。我们从前面的介绍已经知道，所有HDFS文件的元信息都保存在NameNode的内存中。在NameNode启动时，它首先会加载fsimage到内存中，在系统运行期间，所有对NameNode的操作也都保存在了内存中，同时为了防止数据丢失，这些操作又会不断被持久化到本地edits文件中。

Edits文件存在的目的是为了提高系统的操作效率，NameNode在更新内存中的元信息之前都会先将操作写入edits文件。在NameNode重启的过程中，edits会和fsimage合并到一起，但是合并的过程会影响到Hadoop重启的速度，SecondaryNameNode就是为了解决这个问题而诞生的。

SecondaryNameNode的角色就是定期的合并edits和fsimage文件，我们来看一下合并的步骤：

1. secondary通知namenode切换edits文件（新的日志写入edits.new文件）
2. secondary从namenode获得fsimage和edits（通过http协议）
3. secondary将fsimage载入内存，然后开始执行edits中的操作
4. secondary将新的fsimage发回给namenode
5. namenode用新的fsimage替换旧的fsimage



最后再总结一下整个过程中涉及到NameNode中的相关文件

* fsimage ：保存的是上个检查点的HDFS的元信息
* edits ：保存的是从上个检查点开始发生的HDFS元信息状态改变信息
* fstime：保存了最后一个检查点的时间戳

# ****三、HDFS底层存储****

## **HDFS底层以数据块形式存储文件数据**

磁盘数据块是磁盘读写的基本单位，与普通文件系统类似，hdfs也会把文件分块来存储。hdfs默认数据块大小为64MB，磁盘块一般为512Byte，hdfs块为何如此之大呢？块增大可以减少寻址时间与文件传输时间的比例，若寻址时间为10ms，磁盘传输速率为100MB/s，那么寻址与传输比仅为1%。当然，磁盘块太大也不好，因为一个MapReduce通常以一个块作为输入，块过大会导致整体任务数量过小，降低作业处理速度。

数据块是存储在DataNode中的，为了能够容错数据块是以多个副本的形式分布在集群中的，副本数量默认为3，后面会专门介绍数据块的复制机制。

按块存储还有如下好处：

* 文件可以任意大，也不用担心单个结点磁盘容量小于文件的情况
* 简化了文件子系统的设计，子系统只存储文件块数据，而文件元数据则交由其它系统（NameNode）管理
* 有利于备份和提高系统可用性，可以以块为单位进行备份，hdfs默认备份数量为3
* 有利于负载均衡

# ****四、HDFS网络通信及协议****

## **网络通信**

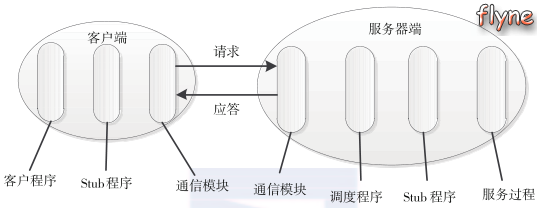
HDFS集群中各节点（无论是NameNode、SecondaryNameNode还是DataNode）都存在进程间的网络通信。所有的HDFS中的网络通信协议都是基于tcp/ip协议，一个客户端通过指定的tcp端口与NameNode机器建立连接，并通过ClientProtocol协议与NameNode交互。而DataNode则通过DataNode Protocol协议与NameNode进行沟通。HDFS的RCP(远程过程调用)对ClientProtocol和DataNode Protocol做了封装。按照HDFS的设计，NameNode不会主动发起任何请求，只会被动接受来自客户端或DataNode的请求。

## **RPC框架**

HDFS通过Hadoop RPC（在Hadoop Common模块中）将底层通信细节对上层系统透明化。

RPC是属于典型的C/S结构，提供服务的一方称为server，请求服务的一方称为client。server端提供对象方法供client端调用，被调用的对象方法的执行发生在server端。

一个典型的RPC框架的架构如下（摘自《Hadoop技术内幕——YARN》）：



在该架构中，主要包括如下几个部分：

* **通信模块：**可以理解成Socket，在客户端和服务端之间传递请求和应答消息。（请求 – 响应）
* **Stub程序：**我将它理解成代理对象，用于保证RPC的透明性。在客户端，它表现得就像一个本地程序，但不直接执行本地调用，而是将请求信息通过网络模块发送给服务端，当服务端发送应答后，它会解码对应结果。在服务端，Stub程序依次进行解码请求消息中的参数、调用相应的服务过程和编码应答结果的返回值等处理。
* **调度程序：**调度程序接受来自通信模块的请求消息，并根据其中的标识选择一个Stub程序进行处理。
* **客户程序/服务过程：**请求的发出者和请求的处理者。

通常而言，一个RPC请求从发送到获取响应结果，所经历的步骤如下：

1. 客户程序以本地方式调用系统产生的Stub程序
2. 该Stub程序将函数调用信息按照网络通信模块的要求封装成消息包，并交给通信模块发送到远程服务器端。
3. 远程服务器端接收此消息后，将此消息发送给相应的Stub程序
4. Stub程序拆封消息，形成被调过程要求的形式，并调用对应函数
5. 服务端执行被调用函数，并将结果返回给Stub程序
6. Stub程序将此结果封装成消息，通过网络通信模块逐级地传送给客户程序。

## **协议**

在HDFS中，每个角色的节点通过各自的协议向外提供交互服务，主要协议如下：

* ClientProtocol：用于和客户端（DFSClient类）通信
* NamenodeProtocol：用于和SecondaryNameNode节点通信
* DatanodeProtocol：用于和DataNode节点通信

NameNodeRpcServer类会实现上述协议，对外提供远程调用服务。

## **序列化**

在网络传输层，采用ProtocolBuffer实现二进制序列化。

更多关于ProtocolBuffer细节可以参考之前我的知识分享或官网参考文档。

## **网络拓扑**

在海量数据处理过程中，主要限制因素是节点之间的带宽。因此Hadoop提出*[“Moving Computation is Cheaper than Moving Data”](http://hadoop.apache.org/docs/r2.6.4/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html" \l "aMoving_Computation_is_Cheaper_than_Moving_Data)*（ 即，优先移动计算而不是数据）的理念。因此，我们需要在集群网络拓扑中计算距离客户端请求的数据最近的节点，然后把计算或存取请求发送到该节点上。如何衡量两个节点在网络拓扑中的距离？在这里hadoop采取了一个简单的方法，它把网络拓扑看成是一棵树，两个节点的距离=它们到最近共同祖先距离的总和，而树的层次可以这么划分：

* 同一节点中的进程
* 同一机架上的不同节点
* 同一数据中心的不同机架
* 不同数据中心

若数据中心d1中一个机架r1中一个节点n1表示为d1/r1/n1,则：

* distance(d1/r1/n1,d1/r1/n1)=0;
* distance(d1/r1/n1,d1/r1/n2)=2;
* distance(d1/r1/n1,d1/r2/n3)=4;
* distance(d1/r1/n1,d2/r3/n4)=6;

# ****五、HDFS安全性****

## **可靠性保证**

DataNode可以允许失败。DataNode会定期（默认3秒）向NameNode发送心跳，若NameNode在指定时间间隔内没有收到心跳，它就认为此节点已经失败。此时，NameNode把失败节点的数据（从另外的副本节点获取）备份到另外一个健康的节点。这保证了集群始终维持指定的副本数。

可以检测到数据块损坏。在读取数据块时，HDFS会对数据块和保存的校验和文件匹配，如果发现不匹配，NameNode同样会重新备份损坏的数据块。

## **数据备份**

HDFS通过备份数据块的形式来实现容错，除了文件的最后一个数据块外，其它所有数据块大小都是一样的。数据块的大小和备份因子都是可以配置的。NameNode负责各个数据块的备份，DataNode会通过心跳的方式定期的向NameNode发送自己节点上的Block 报告，这个报告中包含了DataNode节点上的所有数据块的列表。

文件副本的分布位置直接影响着HDFS的可靠性和性能。一个大型的HDFS文件系统一般都是需要跨很多机架的，不同机架之间的数据传输需要经过网关，并且，同一个机架中机器之间的带宽要大于不同机架机器之间的带宽。如果把所有的副本都放在不同的机架中，这样既可以防止机架失败导致数据块不可用，又可以在读数据时利用到多个机架的带宽，并且也可以很容易的实现负载均衡。但是，如果是写数据，各个数据块需要同步到不同的机架，会影响到写数据的效率。

而在Hadoop中，如果副本数量是3的情况下，Hadoop默认是这么存放的：***把第一个副本放到机架的一个节点上，另一个副本放到同一个机架的另一个节点上，把最后一个节点放到不同的机架上***。在读的过程中，HDFS会选择距离客户端节点最近的一个副本给请求者。这种策略减少了跨机架副本的个数，提高了写的性能，也能够允许一个机架失败的情况，算是一个很好的权衡。

## **安全模式**

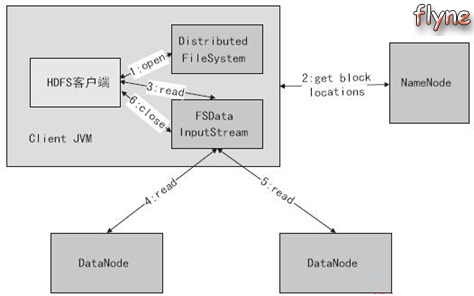
当 Hadoop的NameNode节点启动时，会进入安全模式阶段。在此阶段，DataNode会向NameNode上传它们数据块的列表，让 NameNode得到块的位置信息，并对每个文件对应的数据块副本进行统计。当最小副本条件满足时，即一定比例的数据块都达到最小副本数，系统就会退出安全模式，而这需要一定的延迟时间。当最小副本条件未达到要求时，就会对副本数不足的数据块安排DataNode进行复制，直至达到最小副本数。***而在安全模式下，系统会处于只读状态，NameNode不会处理任何块的复制和删除命令。***

# ****六、HDFS实现剖析****

HDFS原理通俗易懂，但是具体实现却又并不简单，系统中各个角色组件（Client、NameNode、ScendaryNameNode、DataNode）之间的如何交互通信、如何处理容错等等细节才是保证了HDFS的高可靠和高性能的真正秘诀。了解客户端与NameNode和DataNode的交互过程十分重要，有助于加深我们对hdfs架构设计的理解。

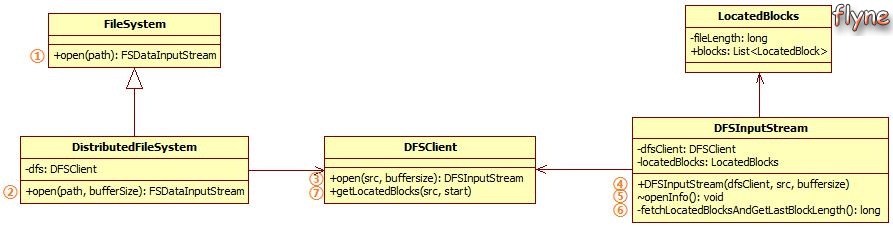
## **hdfs文件读取过程**

HDFS文件读取的主流程如下：

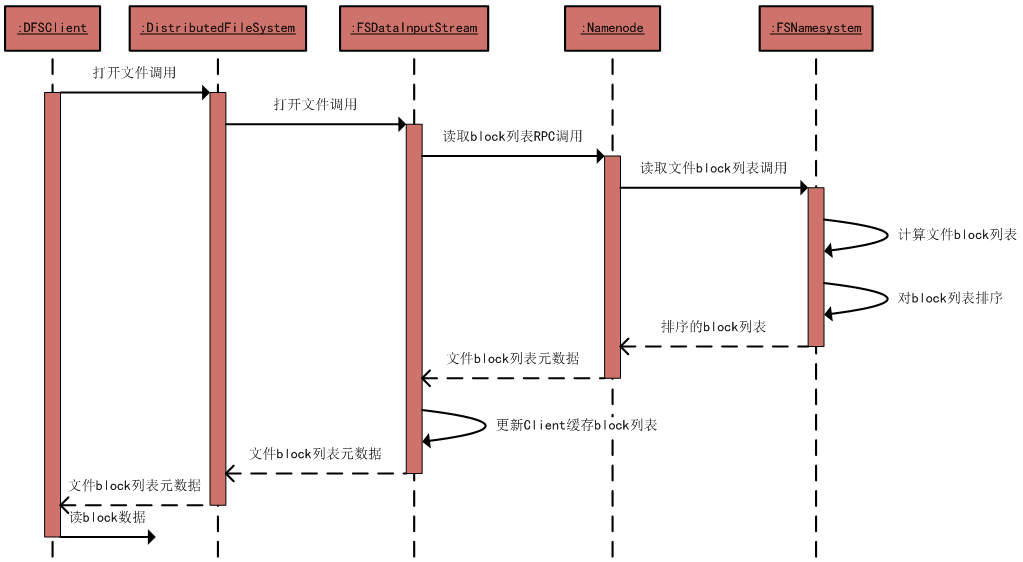


1. 客户端发起读请求，调用FileSystem.open()方法打开文件
2. 客户端与NameNode交互，调用DFSClient.getLocationBlocks()取得文件的数据块及位置信息，并返回FSDataInputStream
3. 调用FSDataInputStream.read()方法直接从DataNode读取数据
4. 调用FSDataInputStream.close()方法关闭流

具体的类图关系如下：



具体的时序图如下：



HDFS有一个FileSystem类，该类实例open()方法会返回FSDataInputStream对象。FSDataInputStream类转而封装成DFSInputStream对象。DFSInputStream对象管理着与DataNode和NameNode的I/O交互，并且在对象中封装了一个locatedBlocks字段，通过RPC调用从NameNode获取文件对应的文件块和所在DataNode节点列表等信息，并记录在该字段中。客户端在拿到该对象后，就可以直接从DataNode上读取数据了。

此外，客户端还会根据网络拓扑来确定它与每一个DataNode的位置信息，从离它最近的那个DataNode获取数据块的副本，最理想的情况是数据块就存储在客户端所在的节点上。

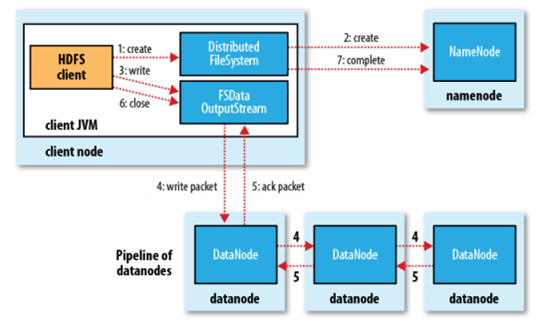
当FSDataInputStream与DataNode通信时遇到错误，它会选取另一个较近的DataNode，并为出故障的DataNode做标记以免重复向其读取数据。FSDataInputStream还会对读取的数据块进行校验和确认，发现块损坏时也会重新读取并通知NameNode。

这样设计的巧妙之处：

* 让客户端直接联系DataNode检索数据，可以使hdfs扩展到大量的并发客户端，因为数据流就是分散在集群的每个节点上的，在运行MapReduce任务时，每个客户端就是一个DataNode节点。
* NameNode仅需相应块的位置信息请求（位置信息在内存中，速度极快），否则随着客户端的增加，NameNode会很快成为瓶颈。

## **hdfs文件写入过程**

HDFS文件写入的主流程如下：



1. Client调用DistributedFileSystem.create()，创建FSDataOutputStream对象
2. 通过DistributedFileSystem对象与NameNode进行一次RPC远程调用，在HDFS的Namespace中创建一个文件条目（Entry），该条目没有任何的Block
3. 通过FSDataOutputStream对象，向DataNode写入数据，数据首先被写入FSDataOutputStream对象内部的Buffer中，然后被分割成一个个Packet数据包
4. 通知NameNode分配一组DataNode节点来存储数据块（每块默认复制3块），这组DataNode节点组成了一个pipeline。以Packet为单位，将数据块写入pipeline中的第一个DataNode，第一个DataNode节点将数据块写到给第二个数据节点，第二个写第三个……依次类推
5. 这组DataNode组成的Pipeline反方向上，发送ack，最终由Pipeline中第一个DataNode节点将Pipeline ack发送回Client
6. 完成向文件写入数据，Client调用FSDataOutputStream.close()关闭流
7. 调用DistributedFileSystem.complete()，通知NameNode文件写入成功

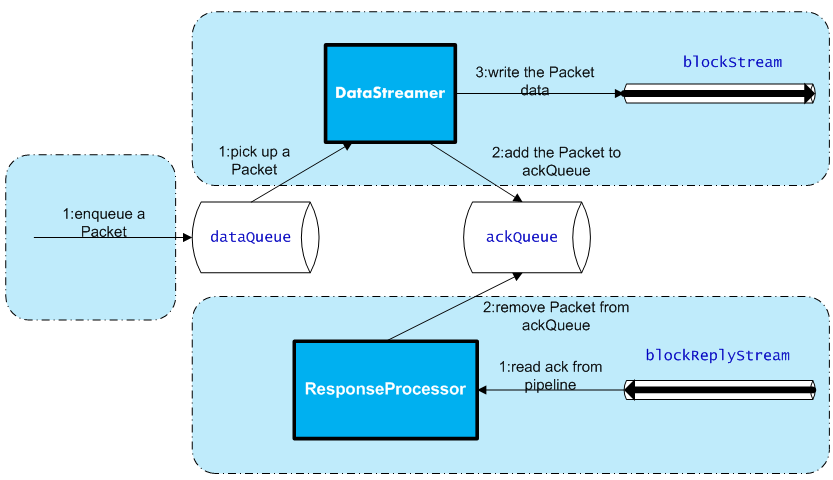
hdfs有一个DistributedFileSystem实例，客户端通过调用这个实例的create()方法就可以创建文件。DistributedFileSystem会发送给NameNode一个RPC调用，在文件系统的命名空间创建一个新文件，在创建文件前NameNode会做一些检查，如文件是否存在，客户端是否有创建权限等，若检查通过，NameNode会为创建文件写一条记录到本地磁盘的edit日志文件，若不通过会向客户端抛出IOException。创建成功之后DistributedFileSystem会返回一个FSDataOutputStream对象，客户端由此开始写入数据。同文件读取过程类似，FSDataOutputStream类转而封装成DFSDataOutputStream对象，具体的数据写入过程由该对象负责。

**DFSOutputStream内部原理**

打开一个DFSOutputStream流，Client会写数据到流内部的一个缓冲区中，然后数据被分解成多个Packet，每个Packet大小为64k字节，每个Packet又由一组chunk和这组chunk对应的checksum数据组成，默认chunk大小为512字节，每个checksum是对512字节数据计算的校验和数据。

当Client写入的字节流数据达到一个Packet的长度，这个Packet会被构建出来，然后会被放到队列dataQueue中，接着DataStreamer线程会不断地从dataQueue队列中取出Packet，发送到复制Pipeline中的第一个DataNode上，并将该Packet从dataQueue队列中移到ackQueue队列中。ResponseProcessor线程接收从Datanode发送过来的ack，如果是一个成功的ack，表示复制Pipeline中的所有Datanode都已经接收到这个Packet，ResponseProcessor线程将packet从队列ackQueue中删除。  
在发送过程中，如果发生错误，所有未完成的Packet都会从ackQueue队列中移除掉，然后重新创建一个新的Pipeline，排除掉出错的那些DataNode节点，接着DataStreamer线程继续从dataQueue队列中发送Packet。

下面是DFSOutputStream的结构及其原理，如图所示：



**三个内部流程具体描述如下：**

1. **创建Packet**

Client写数据时，会将字节流数据缓存到内部的缓冲区中，当长度满足一个Chunk大小（512B）时，便会创建一个Packet对象，然后向该Packet对象中写Chunk Checksum校验和数据，以及实际数据块Chunk Data，校验和数据是基于实际数据块计算得到的。每次满足一个Chunk大小时，都会向Packet中写上述数据内容，直到达到一个Packet对象大小（64K），就会将该Packet对象放入到dataQueue队列中，等待DataStreamer线程取出并发送到DataNode节点。

1. **发送Packet**

DataStreamer线程从dataQueue队列中取出Packet对象，放到ackQueue队列中，然后向DataNode节点发送这个Packet对象所对应的数据。

1. **接收ack**

发送一个Packet数据包以后，会有一个用来接收ack的ResponseProcessor线程，如果收到成功的ack，则表示一个Packet发送成功。如果成功，则ResponseProcessor线程会将ackQueue队列中对应的Packet删除。

## **hdfs文件删除过程**

hdfs文件删除过程一般需要如下几步：

1. 一开始删除文件，NameNode只是重命名被删除的文件到/trash目录，因为重命名操作只是元信息的变动，所以整个过程非常快。在/trash中文件会被保留一定间隔的时间（可配置，默认是6小时），在这期间文件可以很容易地恢复，恢复只需要将文件从/trash移出即可。
2. 当指定的时间到达，NameNode将会把文件从命名空间中删除
3. 标记删除的文件块释放空间，HDFS文件系统显示空间增加

# ****未完待续的部分****

* **NameNode启动过程分析**
* [**Cluster Rebalancing**](http://hadoop.apache.org/docs/r2.6.4/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Cluster_Rebalancing)**（集群负载均衡）**
* [**Space Reclamation**](http://hadoop.apache.org/docs/r2.6.4/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Space_Reclamation)**（空间回收）**
* **Centralized Cache Management in HDFS（中央缓存）**

# ****参考文档****

[官方设计文档](http://hadoop.apache.org/docs/r2.6.4/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html)

[Hadoop核心之HDFS 架构设计](http://mp.weixin.qq.com/s?__biz=MzAwNjQwNzU2NQ==&mid=2650342600&idx=1&sn=79a7be9043efa7c93784fad6cc80a3c8&scene=0#wechat_redirect)

[Hadoop RPC机制及HDFS源码分析](http://www.flyne.org/article/1095/2)

[HDFS体系结构及操作(命令行、Java程序)](http://www.flyne.org/article/1082)

[HDFS读文件过程分析：获取文件对应的Block列表](http://shiyanjun.cn/archives/925.html)

[HDFS写文件过程分析](http://shiyanjun.cn/archives/942.html)