

大数据存储与处理

HADOOP基础与HDFS



浙江大学
ZHEJIANG UNIVERSITY

Part.1 Hadoop基础

Part.2 HDFS组件

Part.3 HDFS存储

Part.4 HDFS读写流程

Part.5 HDFS数据安全性与高可用



Part1. Hadoop基础



浙江大学
ZHEJIANG UNIVERSITY



Hadoop的特点



浙江大学
ZHEJIANG UNIVERSITY

分布式系统基础架构，主要是为了解决海量数据的存储和海量数据的分析计算问题

扩容能力

- Hadoop是在可用的计算机集群间分配数据并完成计算任务的，这些集群可用方便的扩展到数以千计的节点中。

成本低

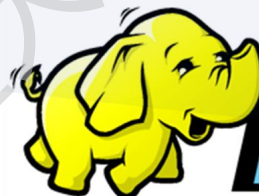
- Hadoop通过普通廉价的机器组成服务器集群来分发以及处理数据，以至于成本很低。

效率高

- 通过并发数据，Hadoop可以在节点之间动态并行的移动数据，使得速度非常快。

可靠性

- 能自动维护数据的多份复制，并且在任务失败后能自动地重新部署计算任务。

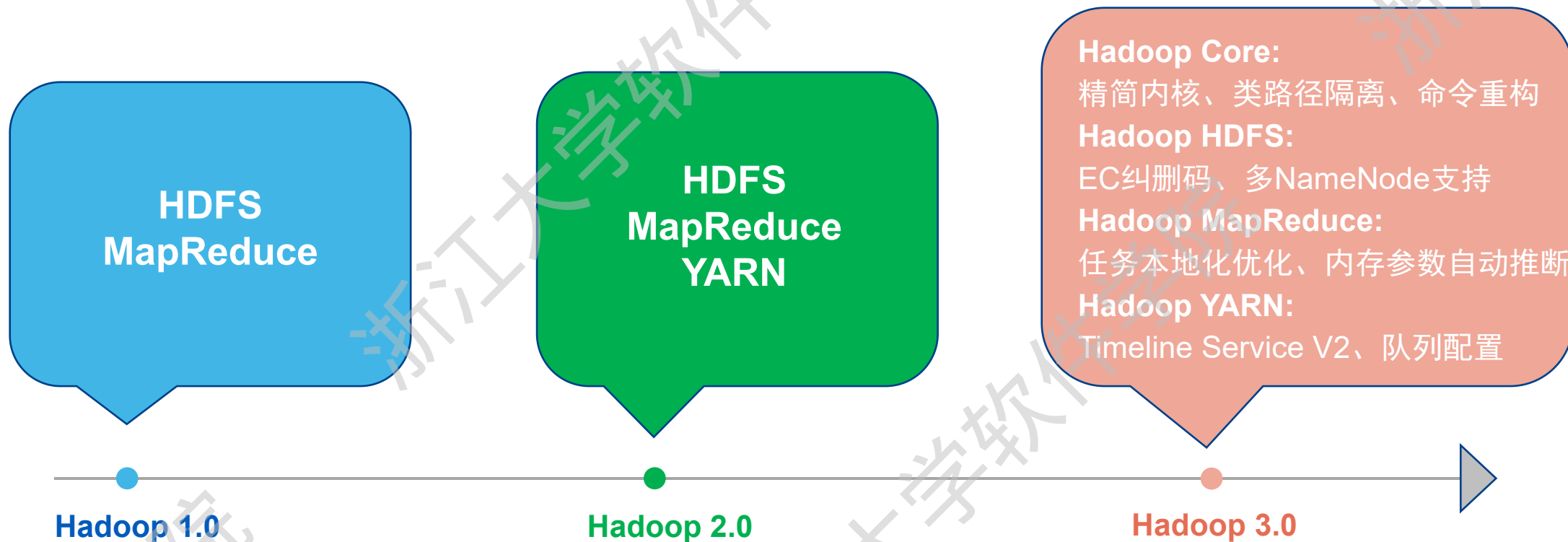


hadoop

Hadoop的1. x, 2. x, 3. x



- Hadoop架构变迁



Hadoop 3.0架构组件和Hadoop 2.0类似,3.0着重于性能优化。

Standalone mode 单机模式

1个机器运行1个java进程，所有角色在一个进程中运行，主要用于调试

Pseudo-Distributed mode 伪分布式

一个机器运行多个进程，每个角色一个进程，主要用于调试

Cluster mode 集群模式

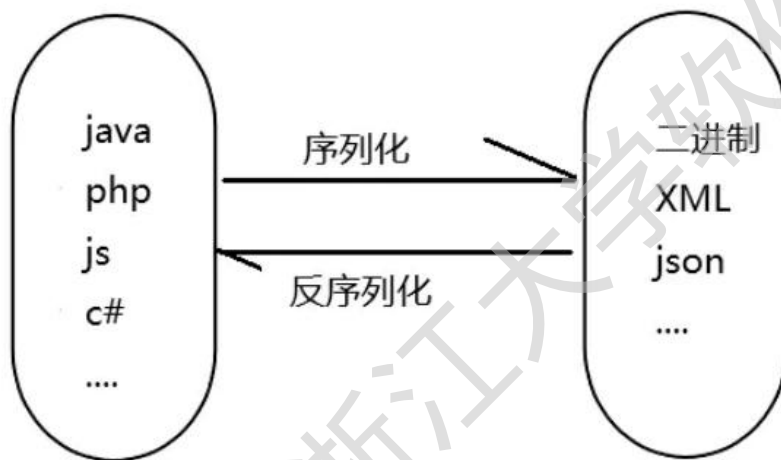
集群模式主要用于生产环境部署。会使用N台主机组成一个Hadoop集群。这种部署模式下，主节点和从节点会分开部署在不同的机器上。

HA mode 高可用

在集群模式的基础上为单点故障部署备份角色，形成主备架构，实现容错

- 何为序列化

- 序列化：将内存中的对象——>字节序列（或其他数据传输协议），以便于存储到磁盘（持久化）和网络传输
- 反序列化：将收到的字节序列（或其他数据传输协议）——> 内存中的对象



紧凑

- 高效使用存储空间

快速

- 读写数据的额外开销小

互操作

- 支持多语言的交互

□ 为什么要序列化：序列化可以存储“活的”对象，将“活的”对象发送到远程计算机

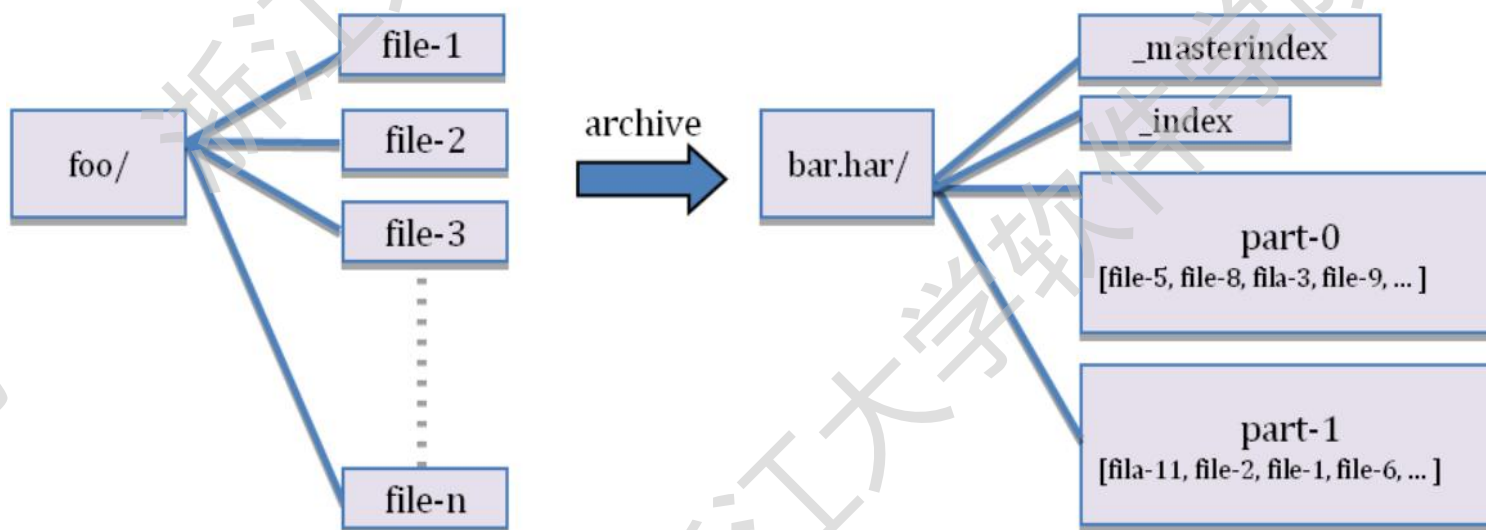
□ 为什么不用Java的序列化：Java的序列化是一个重量级序列化框架（Serializable），一个对象被序列化后，会附带很多额外的信息（如：校验信息，Header，继承体系），不便于在网络中高效传输。Hadoop因此开发了一套自己的序列化机制（Writable）

- 为了支持多种压缩/解压算法，Hadoop引入了编码/解码器，如下所示：

缩格式	对应的编码/解码器	文件扩展名	是否可切分
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec	.deflate	否
gzip	org.apache.hadoop.io.compress.GzipCodec	.gz	否
bzip2	org.apache.hadoop.io.compress.BZip2Codec	.bz2	是
LZO	com.hadoop.compression.lzo.LzopCodec	.lzo	是
Snappy	org.apache.hadoop.io.compress.SnappyCodec	.snappy	否

● Hadoop Archive文件归档

Hadoop Archives可以有效的处理小文件的问题，它可以把多个文件归档成为一个文件，归档成一个文件后还可以透明的访问每一个文件。



Part2. HDFS组件



浙江大学
ZHEJIANG UNIVERSITY



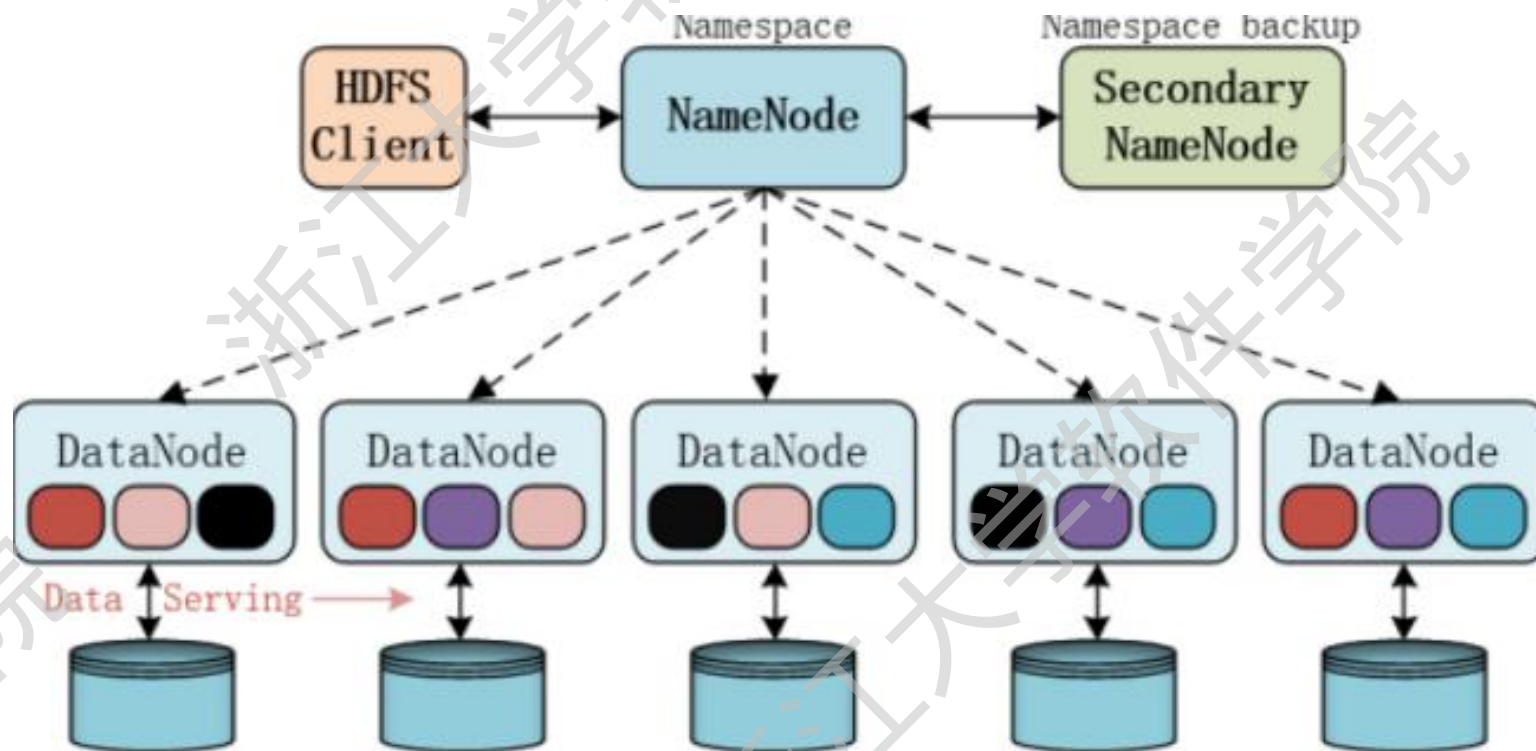
- HDFS (Hadoop Distributed File System), 意为: **Hadoop分布式文件系统**。是Apache Hadoop核心组件之一, 作为大数据生态圈最底层的分布式存储服务而存在。
- 分布式文件系统**解决大数据如何存储问题**。分布式意味着是**横跨在多台计算机**上的存储系统。
- HDFS是一种能够在普通硬件上运行的分布式文件系统, 它是**高度容错**的, 适应于具有大数据集的应用程序, 它非常适于存储大型数据 (比如 TB 和 PB)。
- HDFS使用多台计算机存储文件, 并且提供**统一的访问接口**, 像是访问一个普通文件系统一样使用分布式文件系统。



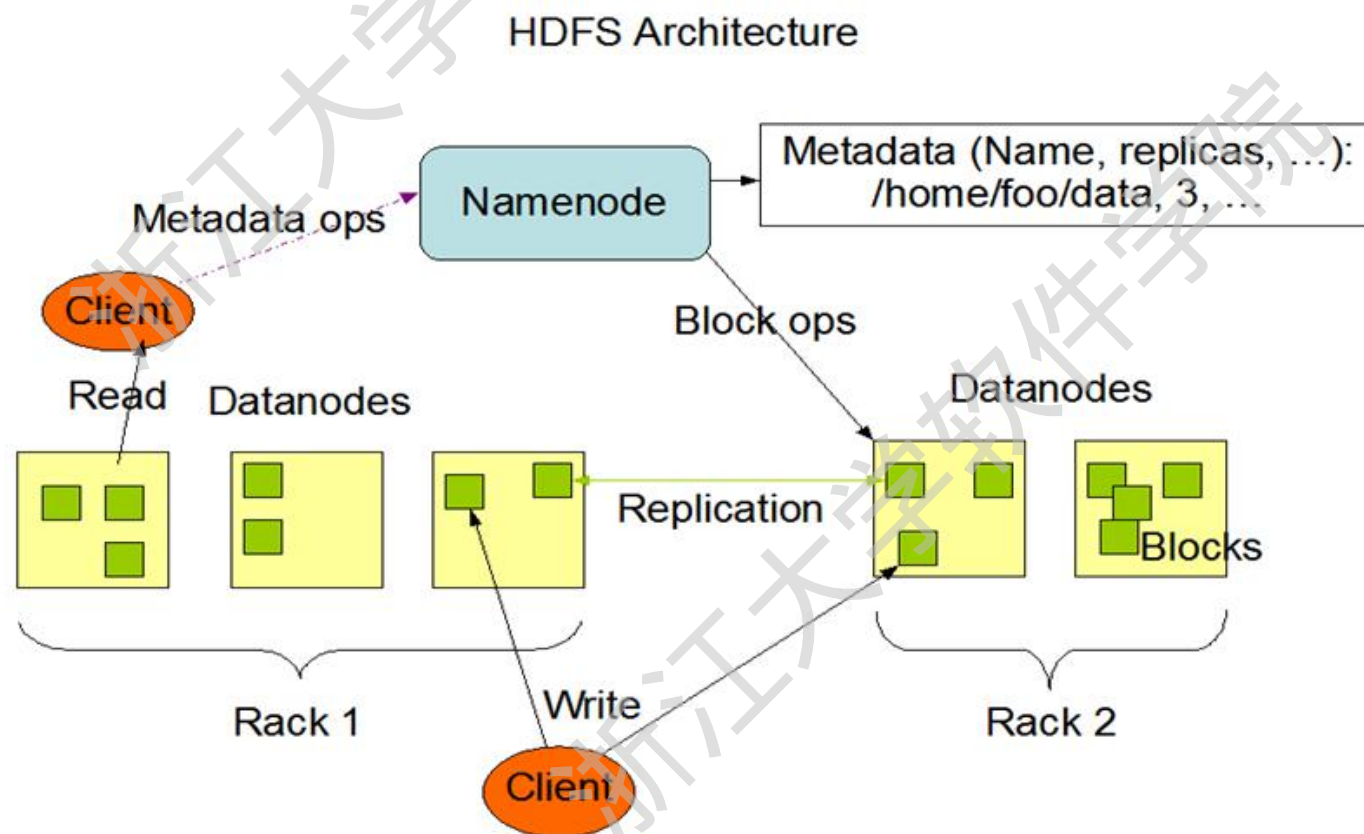
四大组件



- HDFS是经典的主从模式架构，其组成包括四个部分：HDFS Client、NameNode、DataNode和Secondary NameNode。



- **HDFS Client (客户端)**：用户与HDFS交互的接口。它负责文件系统操作，如文件读取、写入、删除等。客户端通过与NameNode和DataNode的交互来完成这些操作。

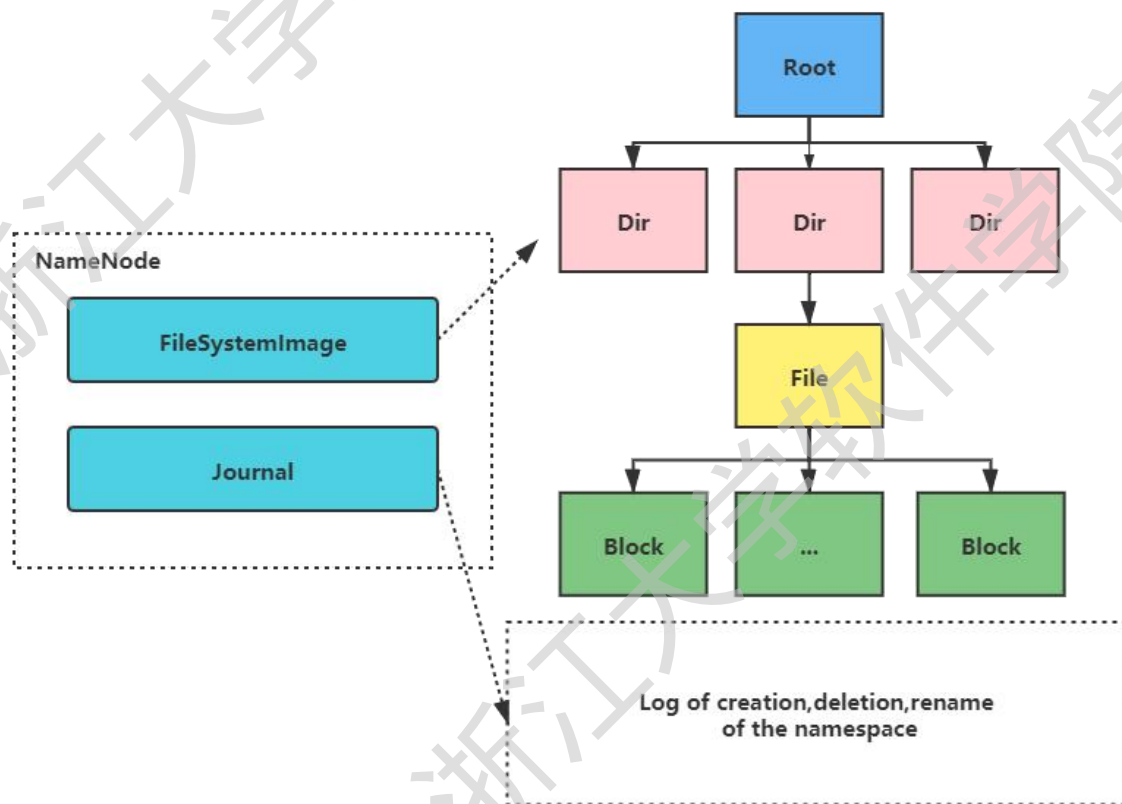


NameNode (NN) : Master



■ **NameNode (主角色)** : 管理文件系统的元数据，如文件名、目录结构、文件块的位置等。

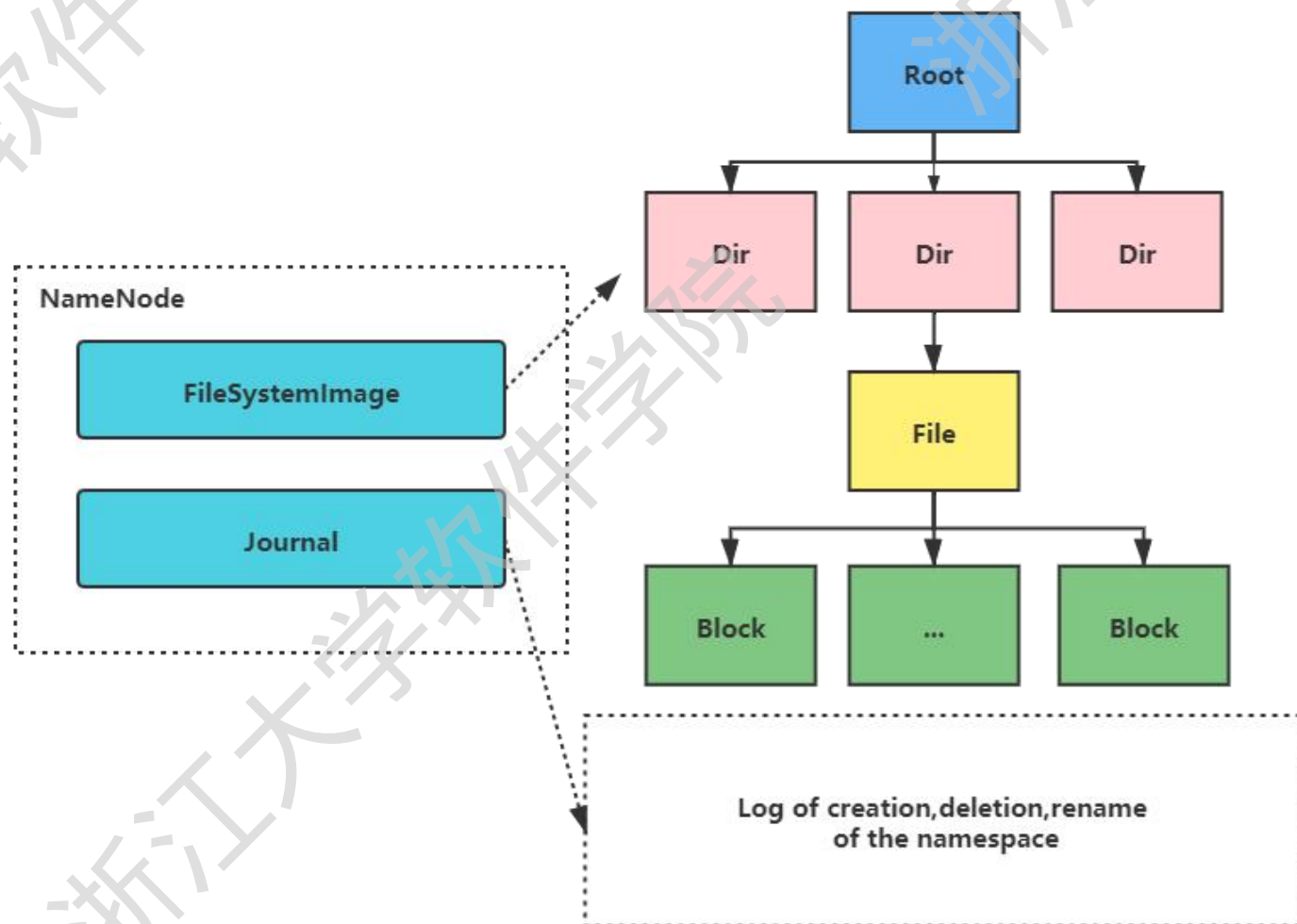
它是HDFS的核心，负责协调和管理所有文件操作。



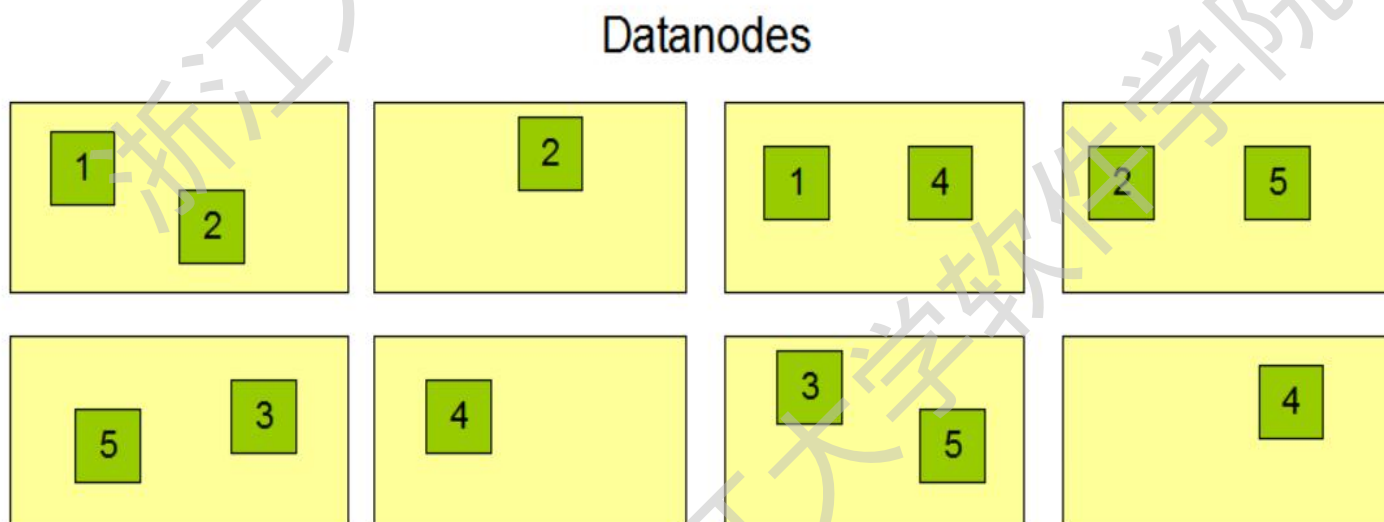
NameNode (NN) : Master



- ❑ NameNode内部通过**内存**和**磁盘文件**两种方式管理元数据。
- ❑ 其中磁盘上的元数据文件包括Fsimage内存元数据镜像文件和edits log (Journal) 编辑日志。
- ❑ 在Hadoop2之前, NameNode是单点故障。Hadoop 2中引入的高可用性。Hadoop群集体系结构允许在群集中以热备配置运行两个或多个NameNode。

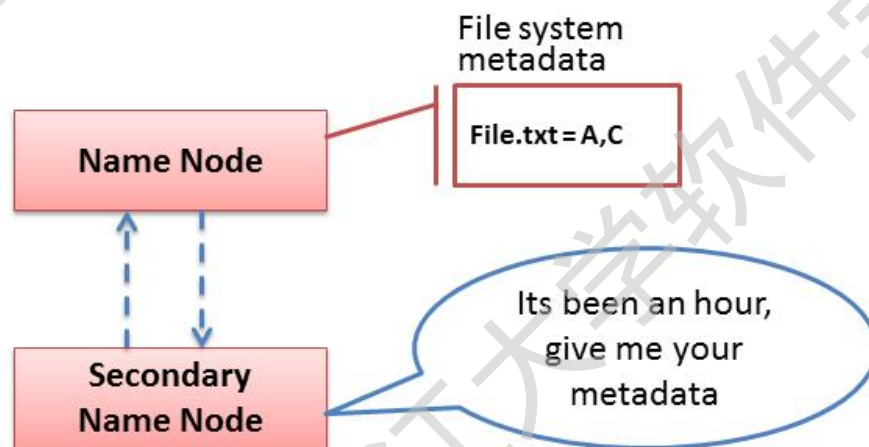


- **DataNode (数据节点)** : 负责存储实际的文件数据。每个文件在HDFS中会被分成多个块，这些块分布在不同的DataNode上。另外，DataNode还负责执行来自NameNode的读写请求。



- **SecondaryNameNode (辅助主节点)**：主要用于定期获取NameNode的元数据快照，并合并日志，减少NameNode崩溃时的恢复时间。

Secondary Name Node



Part3. HDFS存储



浙江大学
ZHEJIANG UNIVERSITY



■ Block（分块）概念

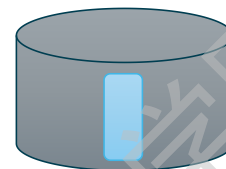
首先，磁盘有一个Block size的概念，它是磁盘读/写数据的最小单位。文件系统的块通常是磁盘块的整数倍HDFS也有Block的概念，像磁盘中的文件系统一样，HDFS中的文件按块大小进行分解，HDFS中一个块只存储一个文件的内容。

temp.txt=300M

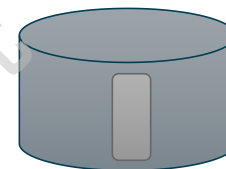


block
size=128M

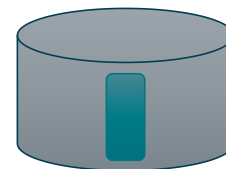
blk-1 0-128
blk-2 128-256
blk-3 256-300



node1

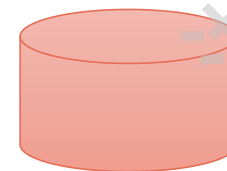


node2



node3

分块存储



HDFS块抽象后的好处：

- 1、一个文件的大小，可以大于网络中任意一个硬盘的大小
- 2、使用抽象块而非整个文件作为存储单元，简化系统设计
- 3、用块数据进行备份，提供数据容错能力、系统可用性

寻址时间：

HDFS中找到目标文件Block块所花费的时间- 文件块越大，寻址时间越短，磁盘传输时间越长- 文件块越小，寻址时间越长，磁盘传输数据越短

Block块合理的设置：

- 块设置过大：磁盘传输数据明显大于寻址时间，程序处理数据非常慢。MapReduce中的map任务通常一次只处理一个块中的数据，块过大运行速度慢
- 块设置过小：寻址时间增长，程序一直在找Block块的开始位置。大量小文件会占有NN中的内存。
- 合适：寻址时间占传输数据的1%

a) Text File

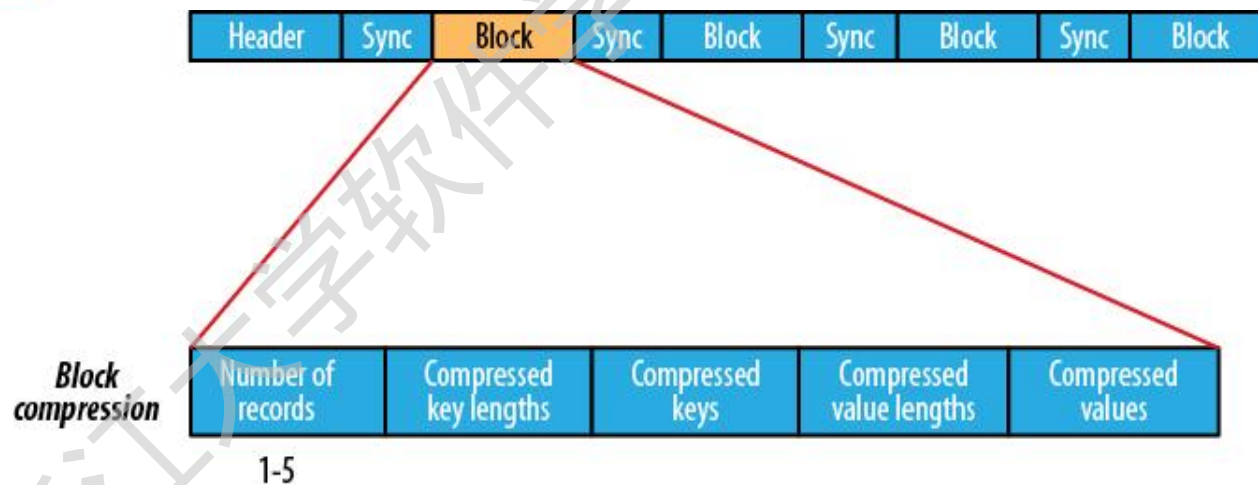
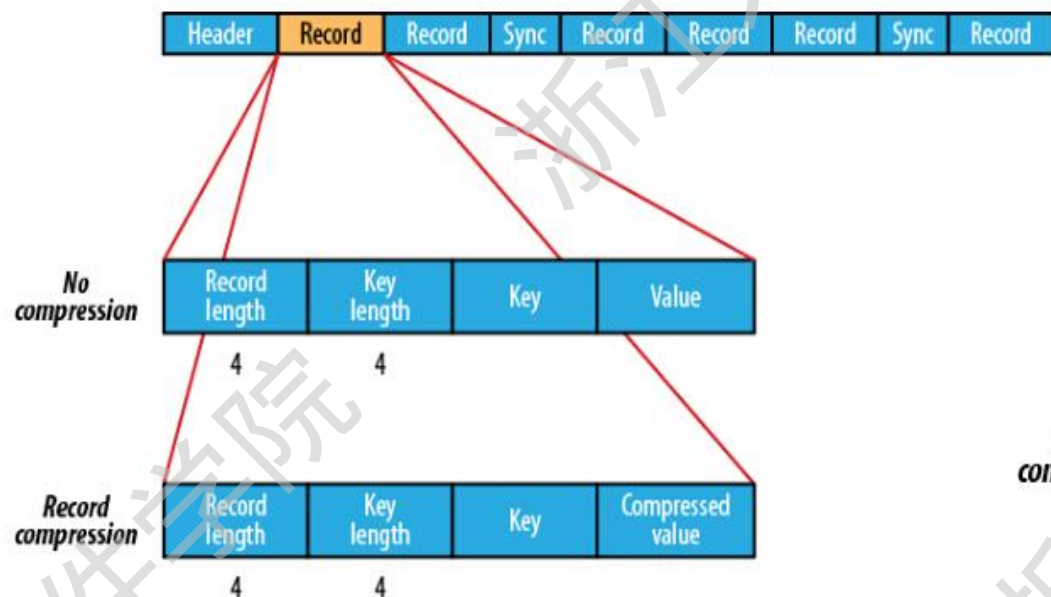
- 文本格式是Hadoop生态系统内部和外部的最常见格式。通常按行存储，以回车换行符区分不同行数据。
- 最大缺点是，它不支持块级别压缩，因此在进行压缩时会带来较高的读取成本。
- 解析开销一般会比二进制格式高，尤其是XML 和JSON，它们的解析开销比Textfile还要大。
- 易读性好。

b) Sequence File

- Sequence File，每条数据记录（record）都是以key、value键值对进行序列化存储（二进制格式）。
- 序列化文件与文本文件相比更紧凑，支持record级、block块级压缩。压缩的同时支持文件切分。
- 通常把Sequence file作为中间数据存储格式。例如：将大量小文件合并放入到一个Sequence File中。

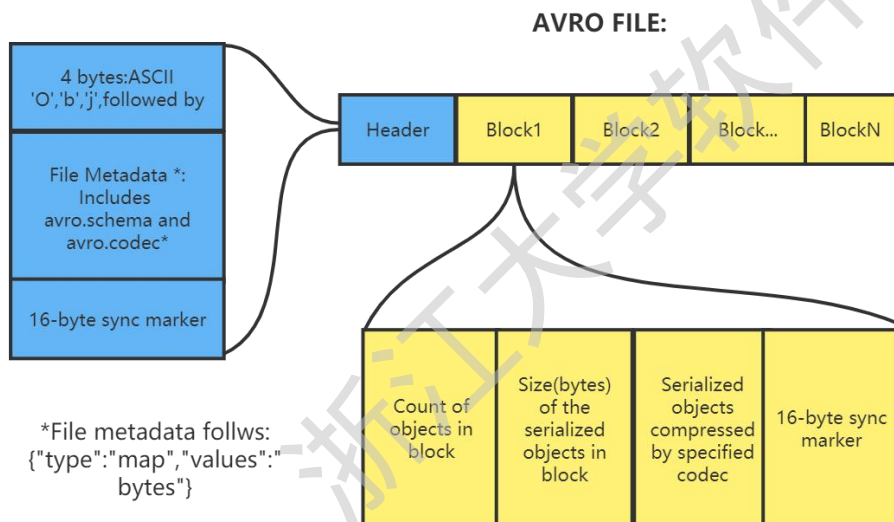
关于Sequence File中的Record和Block。

- record就是一个kv键值对。其中数据保存在value中。可以选择是否针对value进行压缩。
- block就是多个record的集合。block级别压缩性能更好。



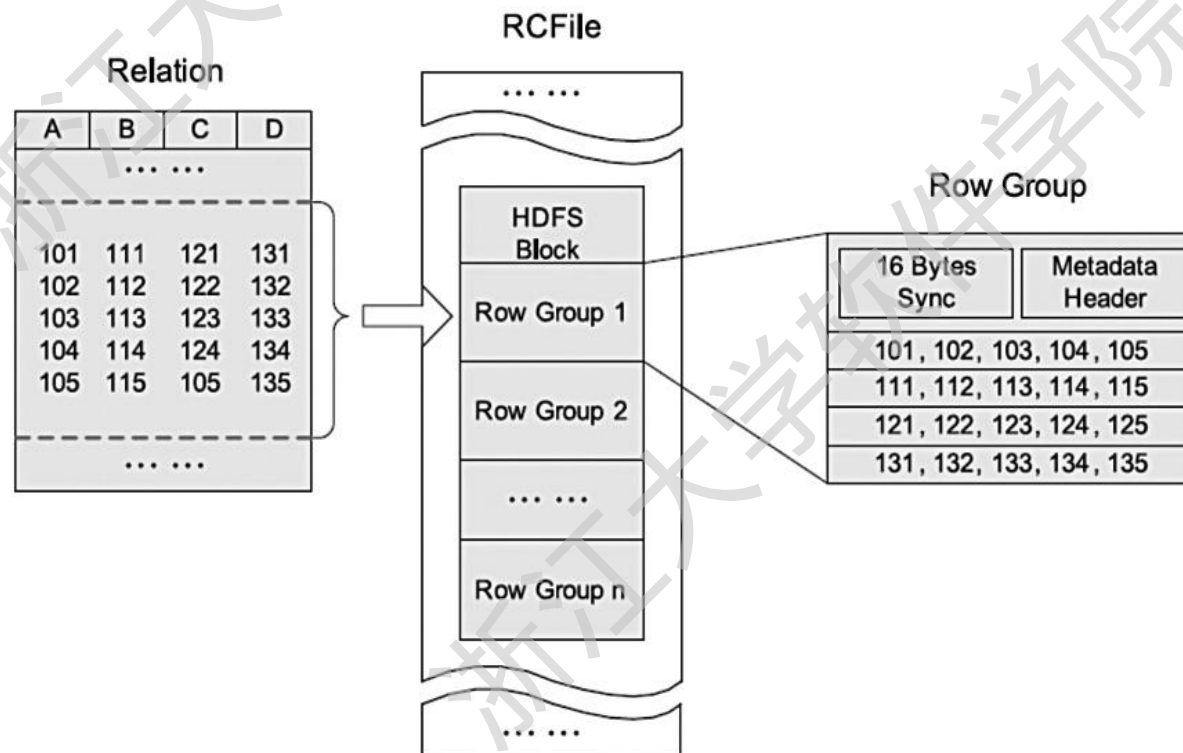
c) Avro File

- Apache Avro是与语言无关的序列化系统，由Hadoop创始人 Doug Cutting开发。
- Avro是**基于行**的存储格式，它在每个文件都包含**JSON格式的schema定义**，从而提高了互操作性并允许schema的变化（删除列、添加列）。
- Avro直接将一行数据序列化在一个block中。
- 适合于大量频繁写入宽表数据（字段多列多）的场景，其**序列化反序列化很快**。



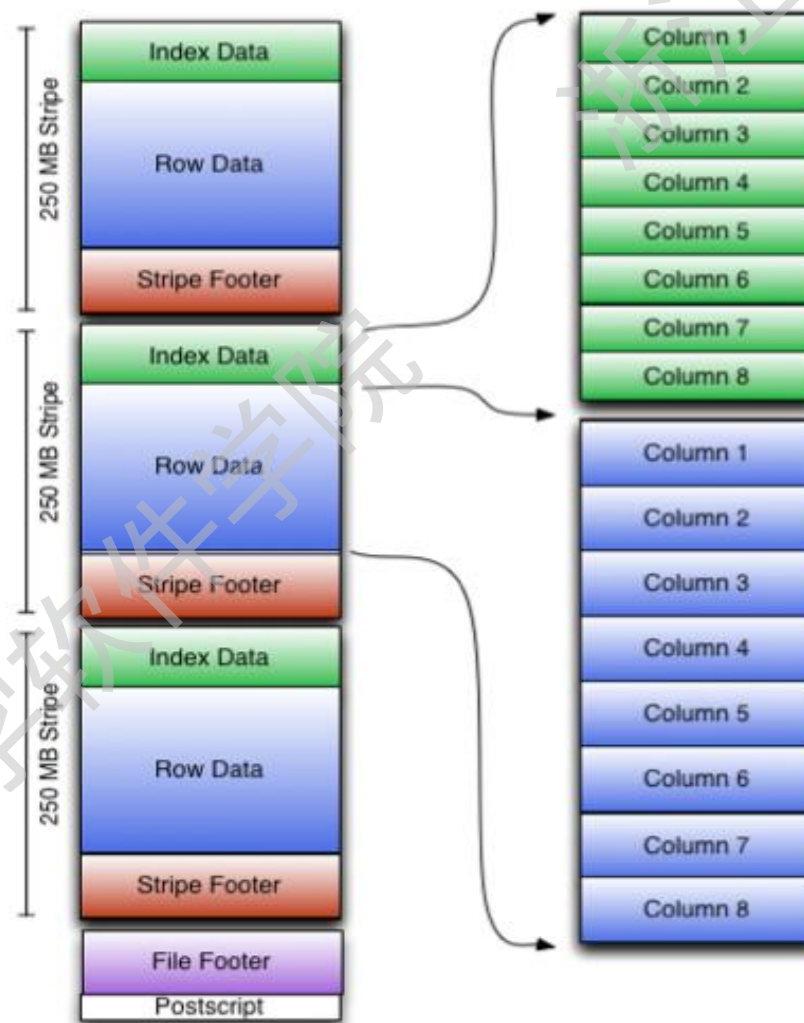
d) RCFile

- Hive **Record Columnar** File（记录列文件），这种类型的文件首先将数据按行划分为行组，然后在行组内部将数据存储在列中。很适合在数仓中执行分析。且支持**压缩**、**切分**
- 但不支持schema扩展，如果要添加新的列，则必须重写文件，这会降低操作效率。



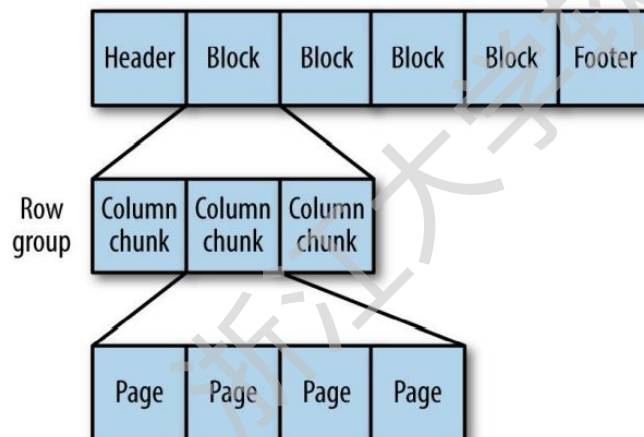
e) ORC File

- ORC File (Optimized Row Columnar) 提供了比RC File更有效的文件格式。它在内部将数据划分为默认大小为250M的Stripe。每个条带均包含索引，数据和页脚。索引存储每列的最大值和最小值以及列中每一行的位置。
- 它并不是一个单纯的列式存储格式，仍然是首先根据Stripe分割整个表，在每一个Stripe内进行按列存储。
- ORC有多种文件压缩方式，并且有着很高的压缩比。文件是可切分 (Split) 的。
- ORC文件是以二进制方式存储的，所以是不可以直接读取。



f) Parquet File

- Parquet是面向分析型业务的**列式存储**格式，由Twitter和Cloudera合作开发，2015年5月从Apache的孵化器里毕业成为Apache顶级项目。
- Parquet文件是以**二进制方式存储**的，所以是不可以直接读取的，文件中包括该文件的数据和元数据，因此Parquet格式文件是自解析的。
- 支持块压缩。
- Parquet 的存储模型主要由**行组（Row Group）**、**列块（Column Chunk）**、**页（Page）**组成。



新一代存储格式Apache Arrow

- Apache Arrow是一个跨语言平台，是一种**列式内存数据结构**，主要用于构建数据系统。
- Apache Arrow在2016年2月17日作为顶级Apache项目引入。

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

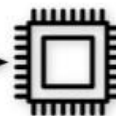
Traditional Memory Buffer

	1331246660
Row 1	3/8/2012 2:44PM
	99.155.155.225
	1331246351
Row 2	3/8/2012 2:38PM
	65.87.165.114
	1331244570
Row 3	3/8/2012 2:09PM
	71.10.106.181
	1331261196
Row 4	3/8/2012 6:46PM
	76.102.156.138

Arrow Memory Buffer

	1331246660
session_id	1331246351
	1331244570
	1331261196
	3/8/2012 2:44PM
timestamp	3/8/2012 2:38PM
	3/8/2012 2:09PM
	3/8/2012 6:46PM
	99.155.155.225
source_ip	65.87.165.114
	71.10.106.181
	76.102.156.138

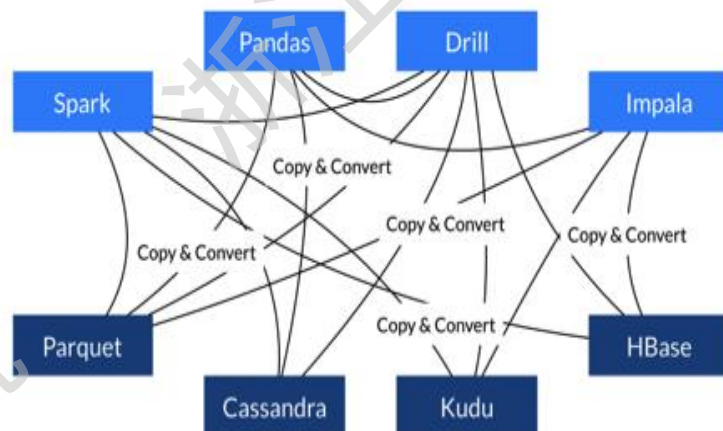
```
SELECT * FROM clickstream  
WHERE session_id = 1331246351
```



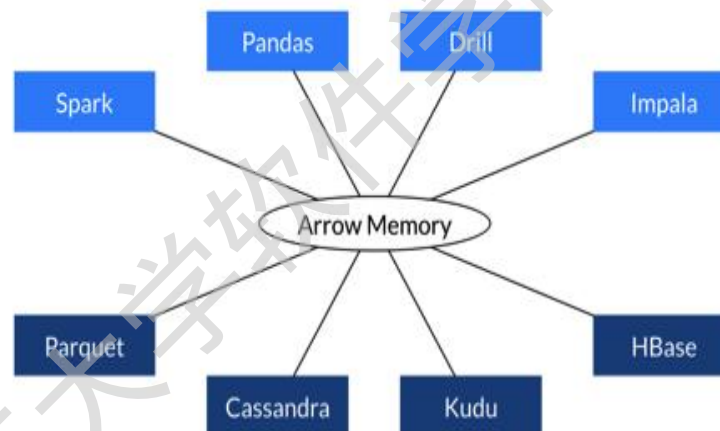
Intel CPU

新一代存储格式Apache Arrow

- Arrow促进了许多组件之间的通信。
- 极大的缩减了通信时候序列化、反序列化所浪费的时间。



Without Arrow



With Arrow

新一代存储格式Apache Arrow

- **Arrow如何提升数据移动性能**
- 利用Arrow作为**内存中数据**表示的两个过程可以将数据从一种方法“重定向”到另一种方法，而无需序列化或反序列化。例如，Spark可以使用Python进程发送Arrow数据来执行用户定义的函数。
- **无需进行反序列化**，可以直接从启用了Arrow的数据存储系统中接收Arrow数据。例如，Kudu可以将Arrow数据直接发送到Impala进行分析。
- Arrow的设计针对嵌套结构化数据（例如在Impala或Spark Data框架中）的分析性能进行了优化。

HDFS的列式存储、行式存储



- 行式存储：一条数据保存为一行，读取一行中的任何值都需要把整行数据都读取处理，如：Sequence Files, Map File, Avro Data Files, 磁盘读取开销比较大
- 列式存储：整个文件被切割为若干列数据，每一列中数据保存在一起，如：Parquet, RC Files, ORC Files, Carbon Data, IndexR, 会占有更多的内存空间，需要将行数据缓存起来

Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Row layout

a1	b1	c1	a2	b2	c2	a3	b3	c3	a4	b4	c4	a5	b5	c5
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Column layout

a1	a2	a3	a4	a5	b1	b2	b3	b4	b5	c1	c2	c3	c4	c5
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

写入：

- 行存储一次完成，效率高，保证数据的完整性
- 列存储把一行记录拆分成单列保存，写入次数多。

读取：

- 行存储将一行数据完全读出，如果只需要其中几列数据，存在冗余列
- 列存储每次读取一段或者全部，存储的数据是同质的，使得数据解析容易。

■ HDFS异构存储类型

- 冷、热、温、冻数据

通常，公司或者组织总是有相当多的历史数据占用昂贵的存储空间。典型的数据使用模式是新传入的数据被应用程序大量使用，从而该数据被标记为"热"数据。随着时间的推移，存储的数据每周被访问几次，而不是一天几次，这时认为其是"暖"数据。在接下来的几周和几个月中，数据使用率下降得更多，成为"冷"数据。如果很少使用数据，例如每年查询一次或两次，这时甚至可以根据其年龄创建第四个数据分类，并将这组很少被查询的旧数据称为"冻结数据"。

Hadoop允许将不是热数据或者活跃数据的数据分配到比较便宜的存储上，用于归档或冷存储。可以设置存储策略，将较旧的数据从昂贵的高性能存储上转移到性价比较低(较便宜)的存储设备上。

Hadoop 2.5及以上版本都支持存储策略，在该策略下，不仅可以在默认的传统磁盘上存储HDFS数据，还可以在SSD(固态硬盘)上存储数据。

■ HDFS异构存储类型

- 什么是异构存储

异构存储是Hadoop2.6.0版本出现的新特性,可以**根据各个存储介质读写特性不同进行选择**。

例如冷热数据的存储,对冷数据采取容量大,读写性能不高的存储介质如机械硬盘;对于热数据,可使用SSD硬盘存储。

在读写效率上性能差距大。异构特性允许我们**对不同文件选择不同的存储介质**进行保存,以实现机器性能的最大化。

■ HDFS异构存储类型

- HDFS中声明定义了4种异构存储类型

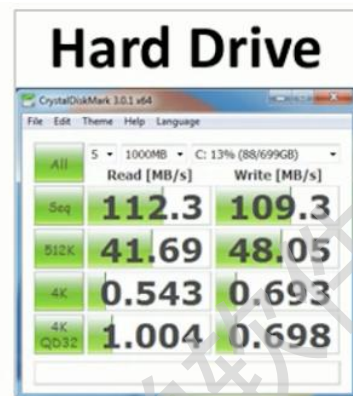
RAM_DISK(内存)

SSD(固态硬盘)

DISK(机械硬盘), 默认使用。

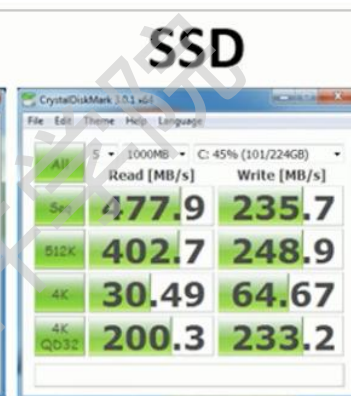
ARCHIVE(高密度存储介质, 存储档案历史数据)

Hard Drive



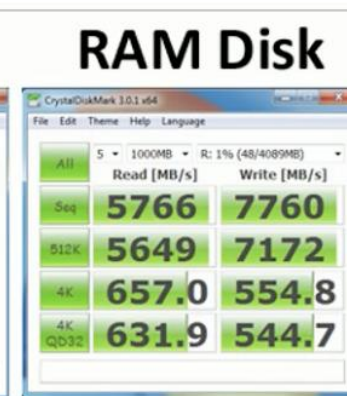
All	5	1000MB	C: 13% (88/699GB)
Seq	Read [MB/s]	Write [MB/s]	
512K	112.3	109.3	
4K	41.69	48.05	
4K	0.543	0.693	
4K QD32	1.004	0.698	

SSD



All	5	1000MB	C: 45% (101/224GB)
Seq	Read [MB/s]	Write [MB/s]	
512K	477.9	235.7	
4K	402.7	248.9	
4K	30.49	64.67	
4K QD32	200.3	233.2	

RAM Disk



All	5	1000MB	R: 1% (48/4089MB)
Seq	Read [MB/s]	Write [MB/s]	
512K	5766	7760	
4K	5649	7172	
4K	657.0	554.8	
4K QD32	631.9	544.7	

■ 块存储类型选择策略

- 块存储指的是对HDFS文件的数据块副本储存。
- 对于数据的存储介质，HDFS的BlockStoragePolicySuite 类内部定义了6种策略。

HOT（默认策略）

COLD

WARM

ALL_SSD

ONE_SSD

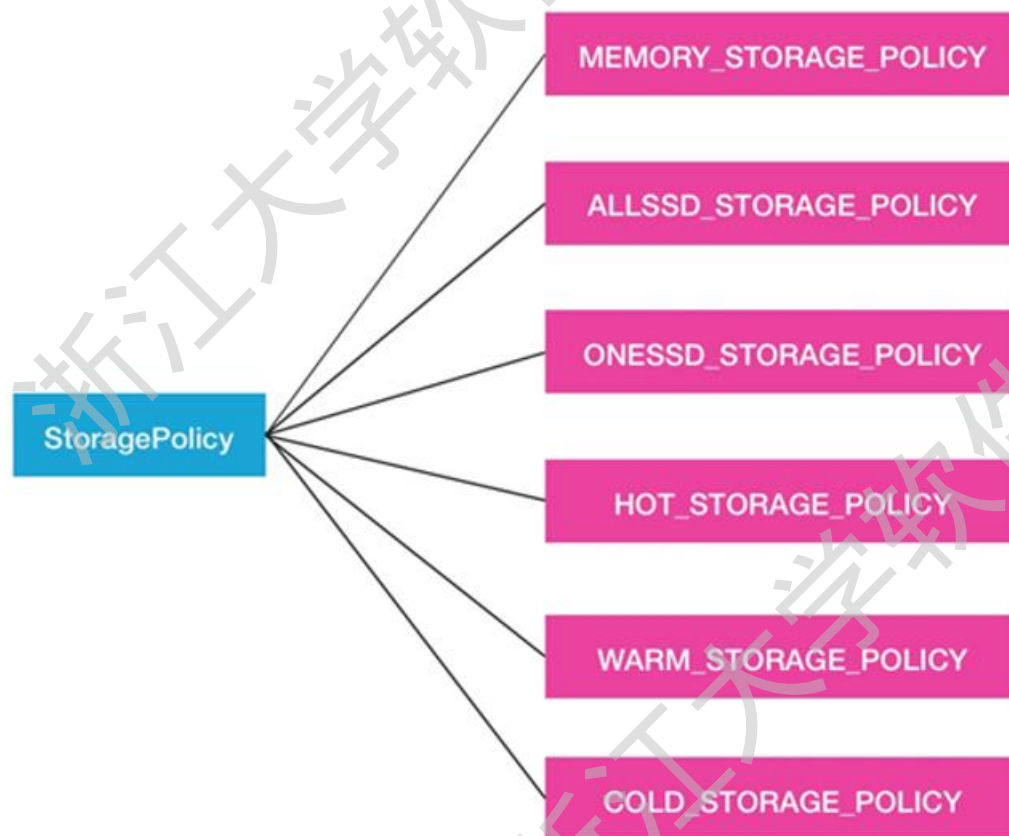
LAZY_PERSIST

- 前三种根据冷热数据区分，后三种根据磁盘性质区分。

■ 块存储类型选择策略--说明

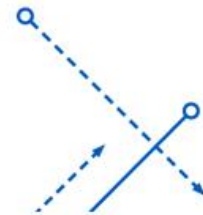
- **HOT**: 用于存储和计算。流行且仍用于处理的数据将保留在此策略中。所有副本都存储在DISK中。
- **COLD**: 仅适用于计算量有限的存储。不再使用的数据或需要归档的数据从热存储移动到冷存储。所有副本都存储在ARCHIVE中。
- **WARM**: 部分热和部分冷。热时, 其某些副本存储在DISK中, 其余副本存储在ARCHIVE中。
- **All_SSD**: 将所有副本存储在SSD中。
- **One_SSD**: 用于将副本之一存储在SSD中。其余副本存储在DISK中。
- **Lazy_Persist**: 用于在内存中写入具有单个副本的块。首先将副本写入RAM_DISK, 然后将其延迟保存在DISK中。

■ 块存储类型选择策略--速度快慢比较



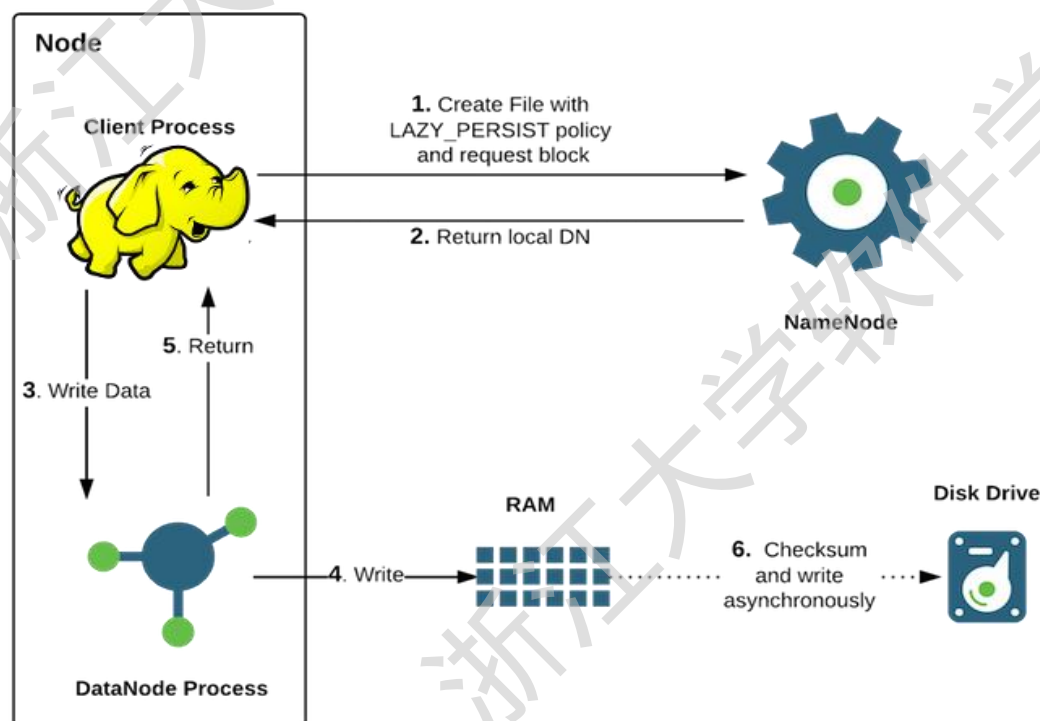
■ HDFS内存存储策略支持--LAZY PERSIST介绍

- HDFS支持把数据写入由DataNode管理的堆外内存；
- DataNode异步地将内存中数据刷新到磁盘，从而减少代价较高的磁盘IO操作，这种写入称为 Lazy Persist写入。
- 该特性从Apache Hadoop 2.6.0开始支持。



■ HDFS内存存储策略支持--LAZY PERSIST执行流程

当数据写入HDFS时，首先将数据暂时存储在内存中，而不是立即写入磁盘。随后，系统在后台异步地将这些数据从内存持久化到磁盘，确保数据最终被安全存储。





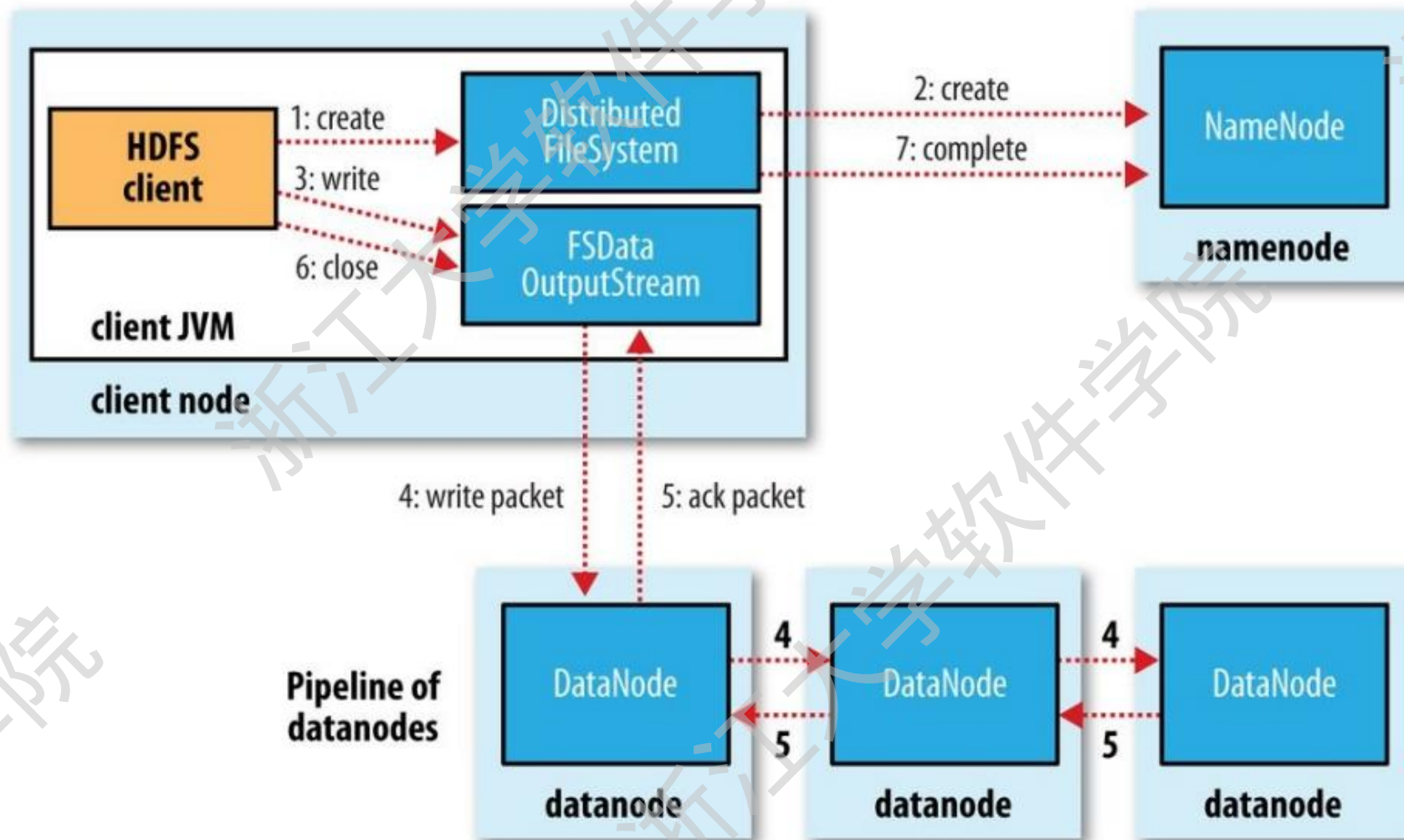
Part4. HDFS读写 流程



HDFS写流程

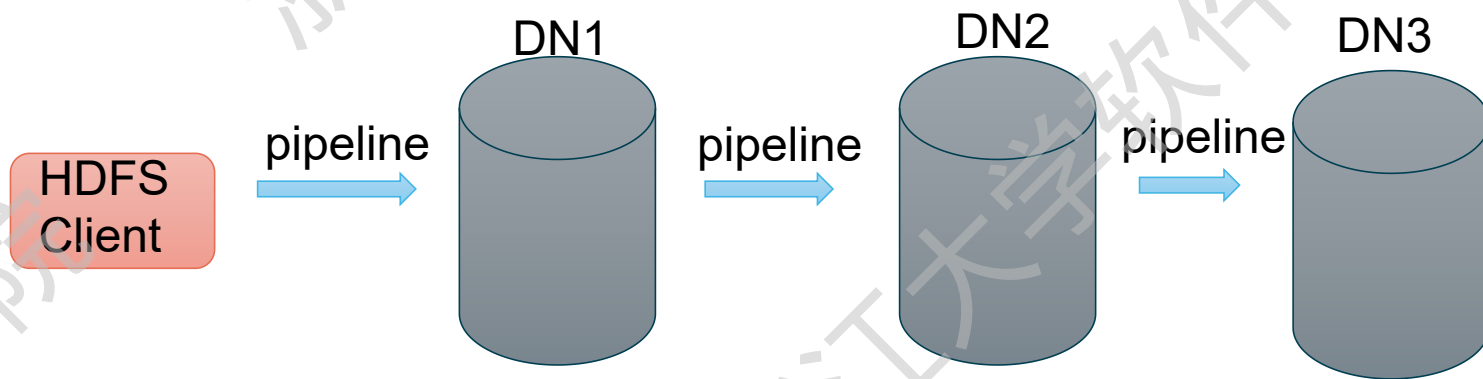


完整流程图



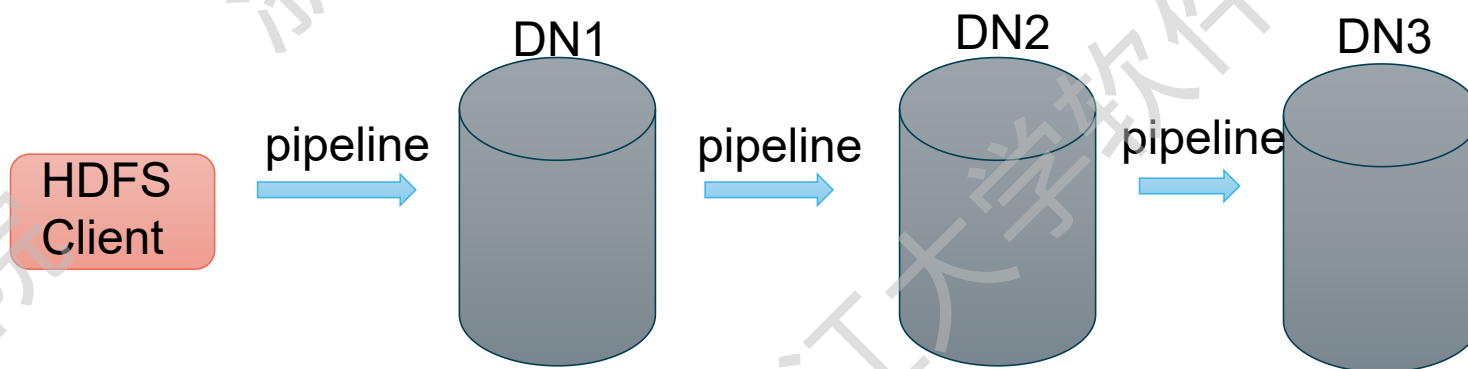
■ 核心概念--Pipeline管道

- Pipeline，中文翻译为管道。这是HDFS在上传文件写数据过程中采用的一种数据传输方式。
- 客户端将数据块写入第一个数据节点，第一个数据节点保存数据之后再将块复制到第二个数据节点，后者保存后将其复制到第三个数据节点。



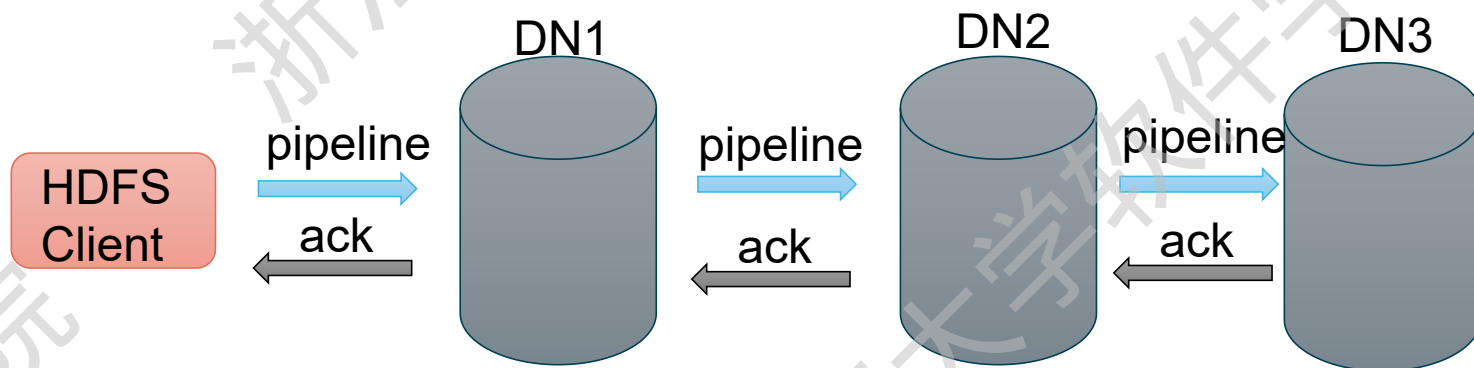
■ 核心概念--Pipeline管道

- 为什么datanode之间采用pipeline线性传输，而不是一次给三个datanode拓扑式传输呢？
- 因为数据以管道的方式，**顺序的沿着一个方向传输**，这样能够充分利用每个机器的带宽，避免网络瓶颈和高延迟时的连接，**最小化推送所有数据的延时**。在线性推送模式下，每台机器所有的出口宽带都用于以最快的速度传输数据，而不是在多个接受者之间分配带宽。



■ 核心概念--ACK应答响应

- ACK (Acknowledge character) 即是确认字符，在数据通信中，接收方发给发送方的一种传输类控制字符。表示发来的数据已确认接收无误。
- 在HDFS pipeline管道传输数据的过程中，传输的反方向会进行ACK校验，确保数据传输安全。



■ 核心概念--默认三副本存储策略

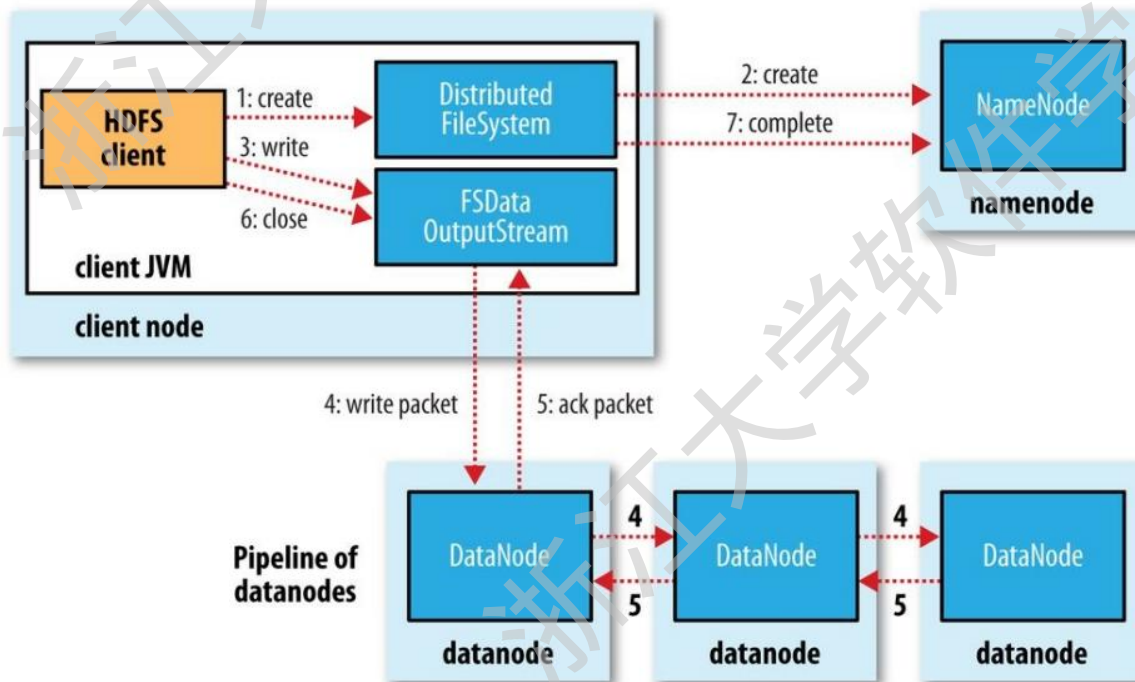
- 默认副本存储策略是由BlockPlacementPolicyDefault指定。
- 第一块副本：优先客户端本地，否则随机
- 第二块副本：不同于第一块副本的不同机架。
- 第三块副本：第二块副本相同机架不同机器。



1、HDFS客户端创建FileSystem对象实例**DistributedFileSystem**， FileSystem封装了与文件系统操作的相关方法。

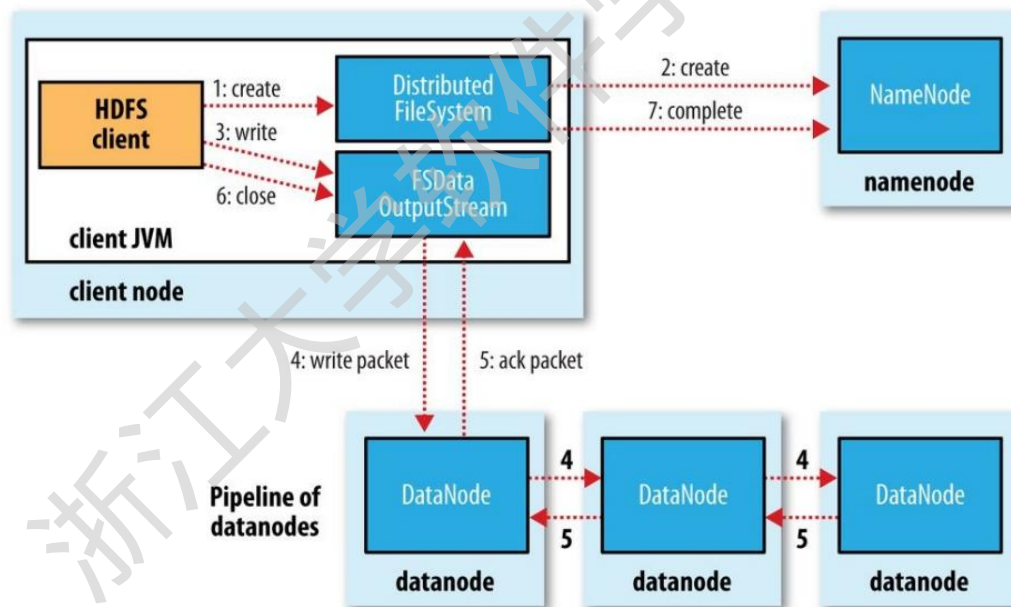
2、调用DistributedFileSystem对象的create()方法，通过**RPC请求**NameNode创建文件。

NameNode执行各种检查判断：目标文件是否存在、父目录是否存在、客户端是否具有创建该文件的权限。检查通过，NameNode就会为本次请求记下一条记录，返回**FSDDataOutputStream**输出流对象给客户端用于写数据。



- 3、客户端通过FSDDataOutputStream开始写入数据。**FSDDataOutputStream**是**DFSOutputStream**包装类。
- 4、客户端写入数据时，DFSOutputStream将数据分成一个个数据包（**packet 默认64k**），并写入一个内部数据队列（**data queue**）。

DFSOutputStream有一个内部类做**DataStreamer**，用于请求NameNode挑选出适合存储数据副本的一组DataNode，默认是3副本存储。DataStreamer将数据包流式传输到**pipeline**的第一个DataNode,该DataNode存储数据包并将它发送到pipeline的第二个DataNode。同样，第二个DataNode存储数据包并且发送给第三个（也是最后一个）DataNode。

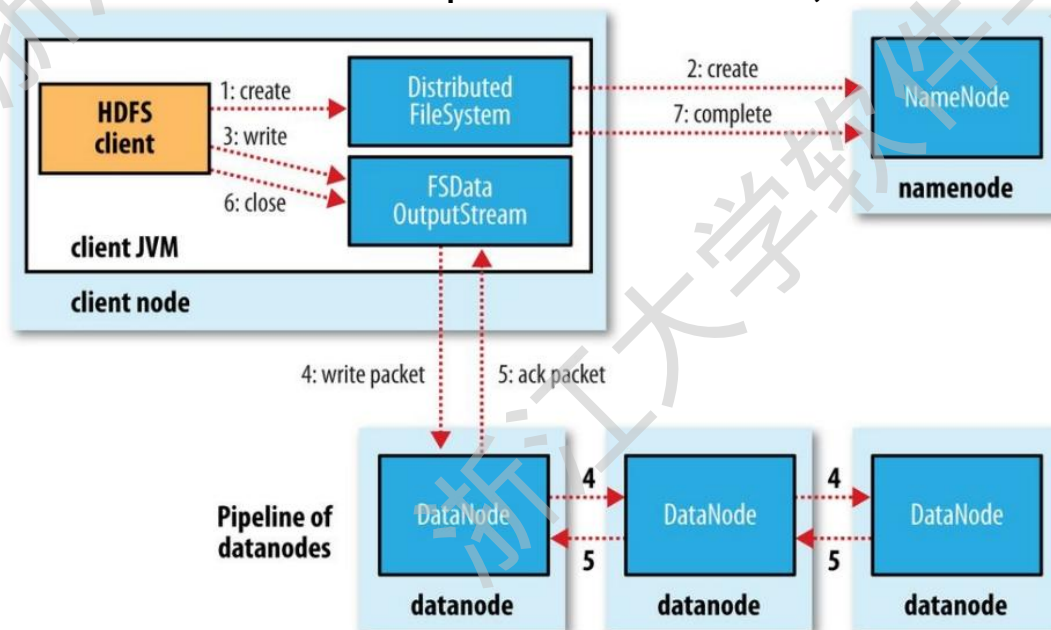


5、DFSOutputStream也维护着一个内部数据包队列来等待DataNode的收到确认回执，称之为确认队列（**ack queue**），收到pipeline中所有DataNode确认信息后，该数据包才会从确认队列删除。

6、客户端完成数据写入后，在FSDataOutputStream输出流上调用close()方法关闭。

7、DistributedFileSystem联系NameNode告知其文件写入完成，等待NameNode确认。

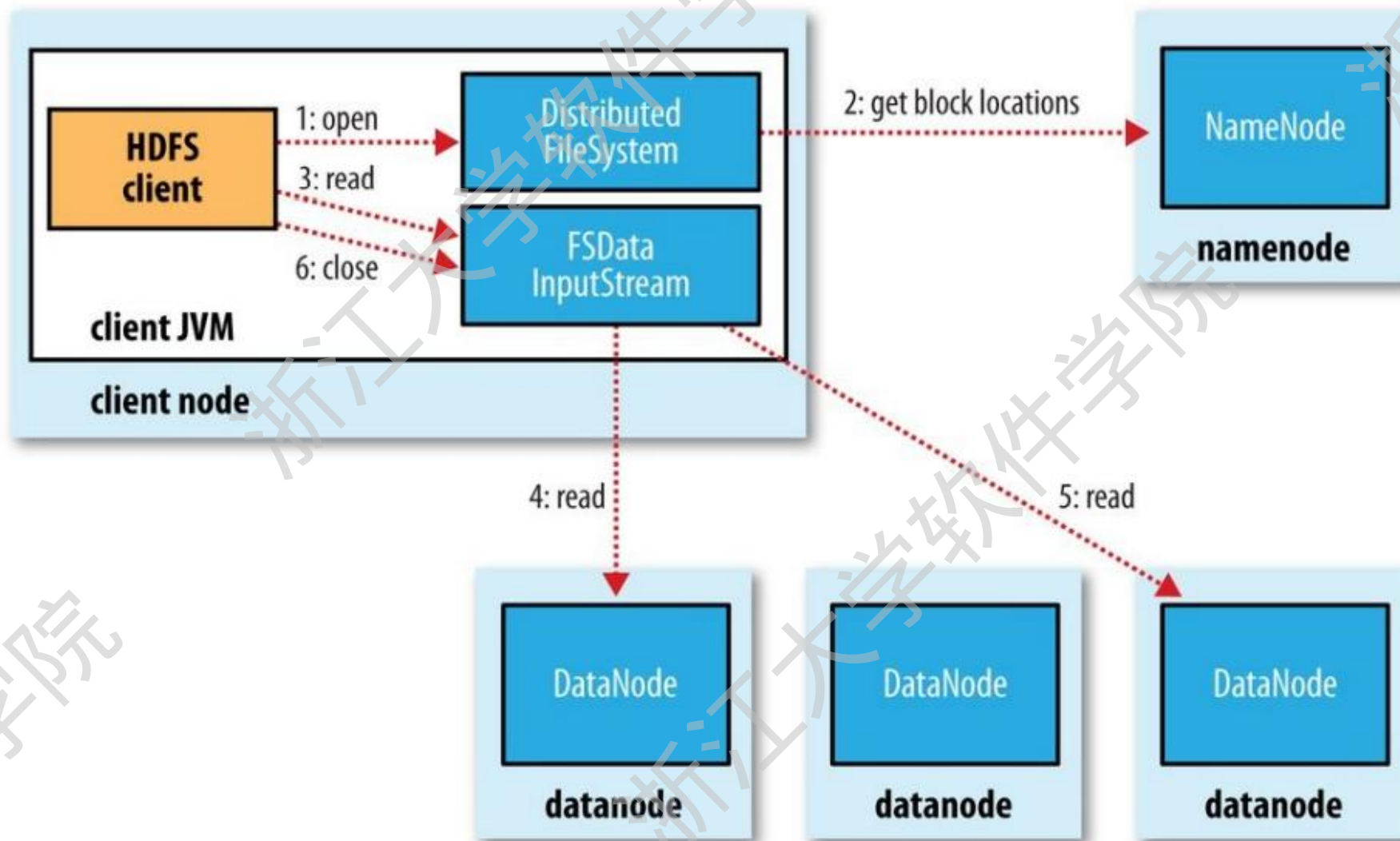
因为namenode已经知道文件由哪些块组成（DataStream请求分配数据块），因此仅需等待最小复制块即可成功返回。最小复制是由参数dfs.namenode.replication.min指定，默认是1。



HDFS读流程



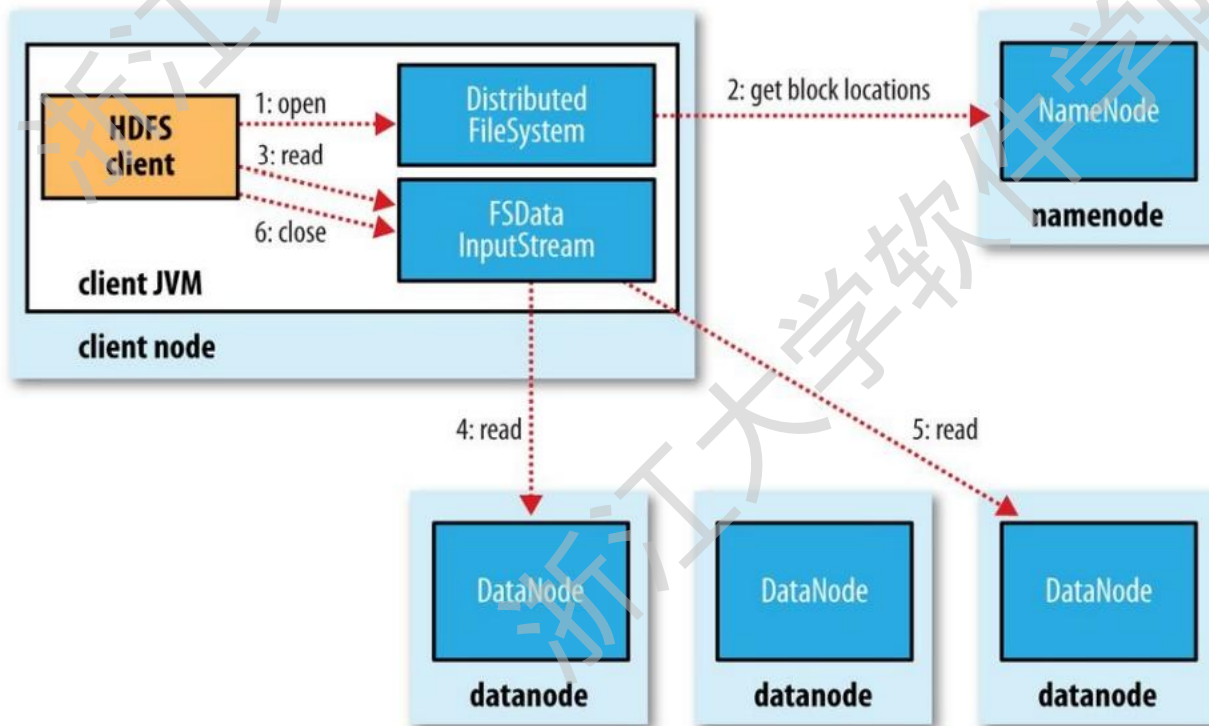
■ 完整流程图



1、HDFS客户端创建FileSystem对象实例**DistributedFileSystem**， FileSystem封装了与文件系统操作的相关方法。调用DistributedFileSystem对象的open()方法来打开希望读取的文件。

2、DistributedFileSystem使用RPC调用namenode来确定**文件中前几个块的块位置（分批次读取）信息**。

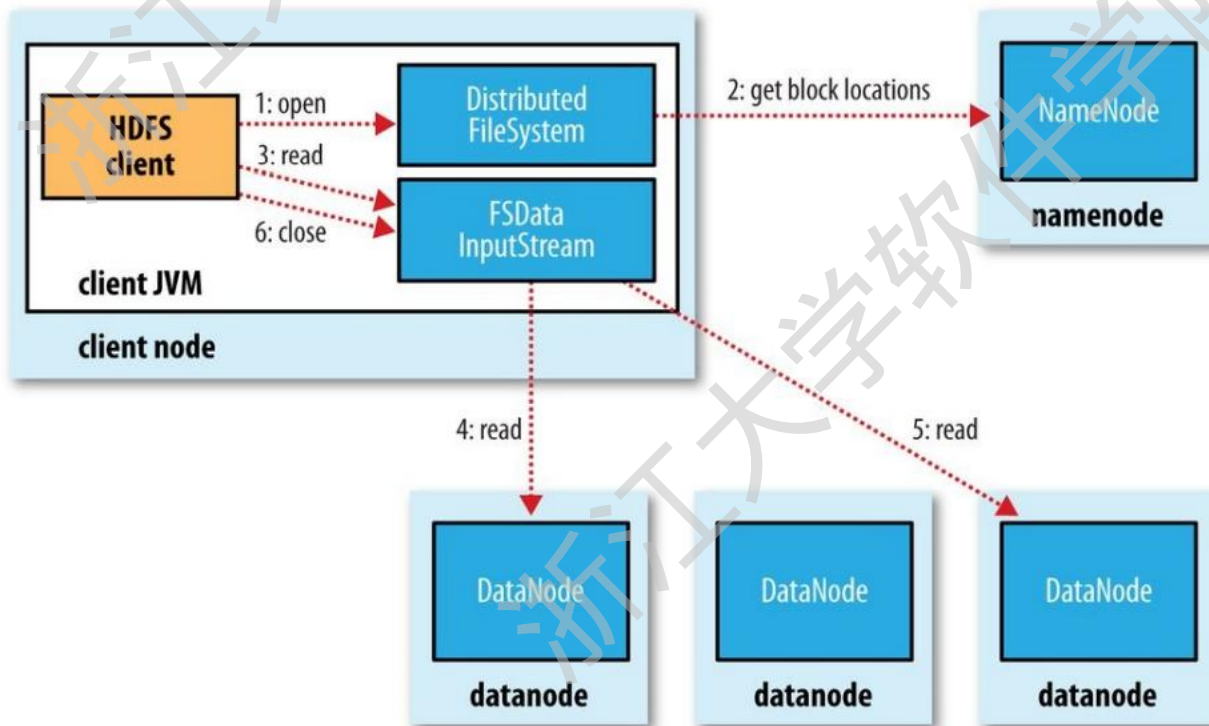
对于每个块，namenode返回具有该块所有副本的datanode位置地址列表，并且该地址列表是排序好的，与客户端的网络拓扑距离近的排序靠前。



3、DistributedFileSystem将FSDataInputStream输入流返回到客户端以供其读取数据。

FSDataInputStream类是DFSInputStream类的包装。

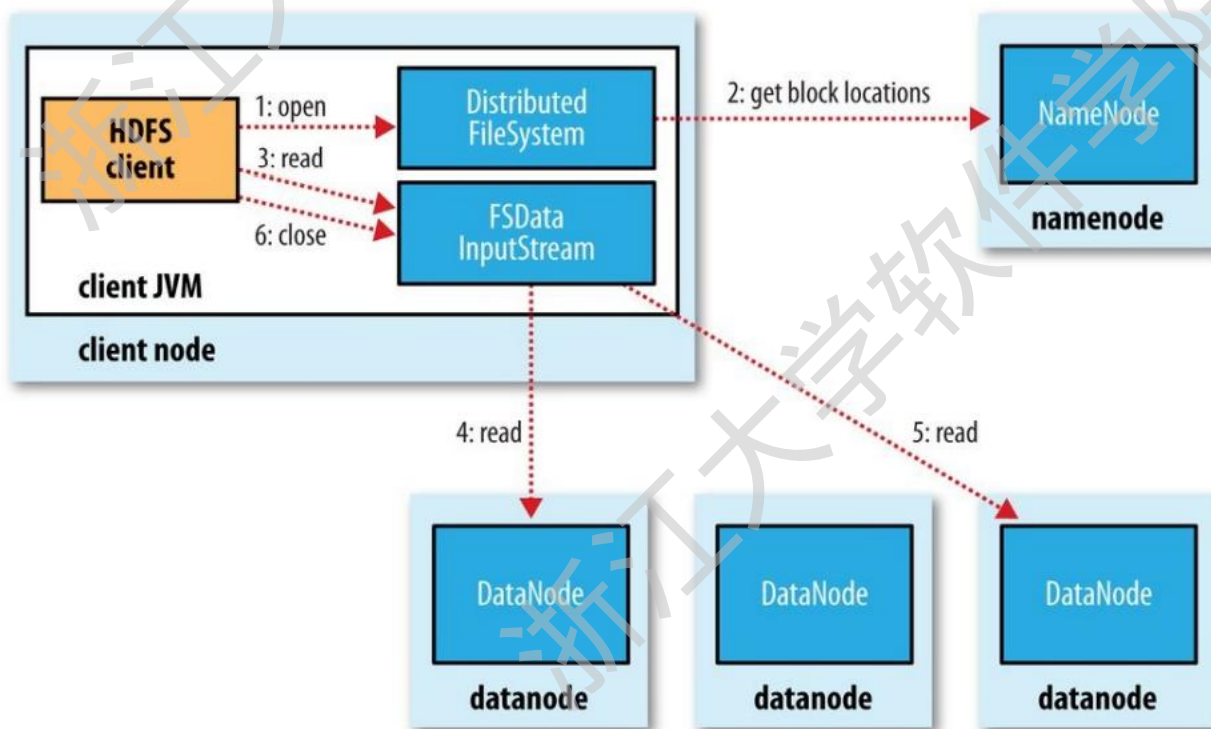
4、客户端在FSDataInputStream输入流上调用read()方法。然后，已存储DataNode地址的DFSInputStream连接到文件中第一个块的最近的DataNode。数据从DataNode流回客户端，结果客户端可以在流上重复调用read（）。



5、当该块结束时，DFSInputStream将关闭与DataNode的连接，然后寻找下一个块的最佳datanode。这些操作对用户来说是透明的。所以用户感觉起来它一直在读取一个连续的流。

客户端从流中读取数据时，也会根据需要询问NameNode来检索下一批数据块的DataNode位置信息。

6、一旦客户端完成读取，就对FSDataInputStream调用close()方法。



■ NameNode职责

- NameNode是HDFS的核心，集群的主角色，被称为Master。
- NameNode仅存储管理HDFS的元数据：文件系统namespace操作维护目录树，文件和块的位置信息。
- NameNode不存储实际数据或数据集。数据本身实际存储在DataNodes中。
- NameNode知道HDFS中任何给定文件的块列表及其位置。使用此信息NameNode知道如何从块中构建文件。
- NameNode并不持久化存储每个文件中各个块所在的DataNode的位置信息，这些信息会在系统启动时从DataNode汇报中重建。
- NameNode对于HDFS至关重要，当NameNode关闭时，HDFS / Hadoop集群无法访问。
- NameNode是Hadoop集群中的单点故障。
- NameNode所在机器通常会配置有大量内存（RAM）。

■ DataNode职责

- DataNode负责将实际数据存储在HDFS中。是集群的从角色，被称为Slave。
- DataNode启动时，它将自己发布到NameNode并汇报自己负责持有的块列表。
- 根据NameNode的指令，执行块的创建、复制、删除操作。
- DataNode会定期（dfs.heartbeat.interval配置项配置，默认是3秒）向NameNode发送心跳，如果NameNode长时间没有接受到DataNode发送的心跳，NameNode就会认为该DataNode失效。
- DataNode会定期向NameNode进行自己持有的数据块信息汇报，汇报时间间隔取参数dfs.blockreport.intervalMsec,参数未配置的话默认为6小时。
- DataNode所在机器通常配置有大量的硬盘空间。因为实际数据存储在DataNode中。

■ 元数据管理概述

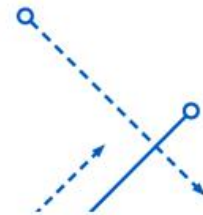
在HDFS中，文件相关元数据具有两种类型：

- 文件自身属性信息

文件名称、权限，修改时间，文件大小，复制因子，数据块大小。

- 文件块位置映射信息

记录文件块和DataNode之间的映射信息，即哪个块位于哪个节点上。



■ 元数据管理概述

按存储形式分为**内存元数据**和元数据文件两种，分别存在内存和磁盘上。

• 内存元数据

为了保证用户操作元数据交互高效，延迟低，NameNode把所有的元数据都存储在内存中，我们叫做内存元数据。**内存中的元数据是最完整的**，包括文件自身属性信息、文件块位置映射信息。

但是内存的致命问题是，断点数据丢失，数据不会持久化。因此NameNode又采用元数据文件来保证元数据的安全完整。

■ 元数据管理概述

按存储形式分为内存元数据和**元数据文件**两种，分别存在内存和磁盘上。

- **元数据文件**有两种：**fsimage内存镜像文件**、Edits log编辑日志。
- **fsimage 内存镜像文件**

是内存元数据的一个持久化的检查点。但是**fsimage**中仅包含Hadoop文件系统中文件自身属性相关的**元数据信息**，但不包含文件块位置的信息。文件块位置信息只存储在内存中，是由datanode启动加入集群的时候，向namenode进行数据块的汇报得到的，并且后续间断指定时间进行数据块报告。

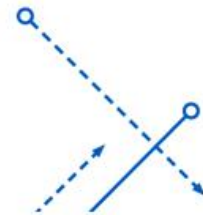
持久化的动作是数据从内存到磁盘的IO过程。会对namenode正常服务造成一定的影响，不能频繁的进行持久化。

■ 元数据管理概述

按存储形式分为内存元数据和**元数据文件**两种，分别存在内存和磁盘上。

- **元数据文件**有两种：fsimage内存镜像文件、**Edits log编辑日志**。
- **Edits log编辑日志**

为了避免两次持久化之间数据丢失的问题，又设计了Edits log编辑日志文件。文件中记录的是HDFS所有更改操作（文件创建，删除或修改）的日志，文件系统客户端执行的更改操作首先会被记录到edits文件中。

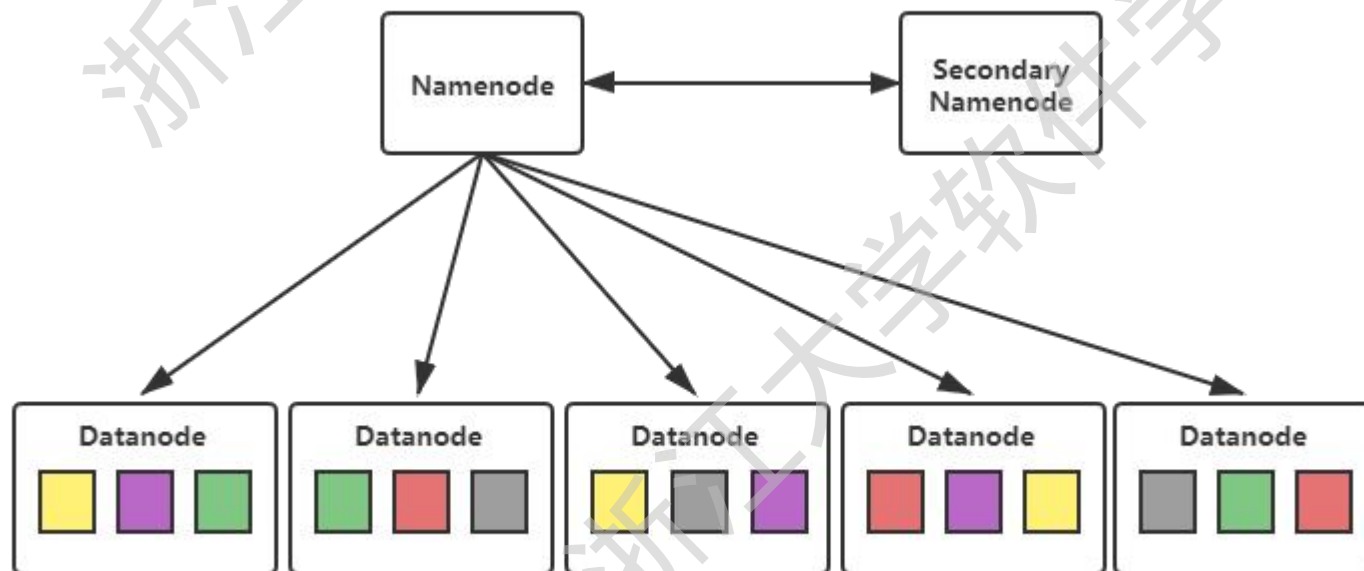


■ NameNode加载元数据文件顺序

- fsimage和edits文件都是经过序列化的，在NameNode启动的时候，它会先将fsimage文件中的内容加载到内存中，之后再执行edits文件中的各项操作，使得内存中的元数据和实际的同步，存在内存中的元数据支持客户端的读操作，也是最完整的元数据。
- 当客户端对HDFS中的文件进行新增或者修改操作，操作记录首先被记入edits日志文件中，当客户端操作成功后，相应的元数据会更新到内存元数据中。因为fsimage文件一般都很大（GB级别的很常见），如果所有的更新操作都往fsimage文件中添加，这样会导致系统运行的十分缓慢。
- HDFS这种设计实现着手于：一是内存中数据更新、查询快，极大缩短了操作响应时间；二是内存中元数据丢失风险颇高（断电等），因此辅佐元数据镜像文件（fsimage）+编辑日志文件（edits）的备份机制进行确保元数据的安全。

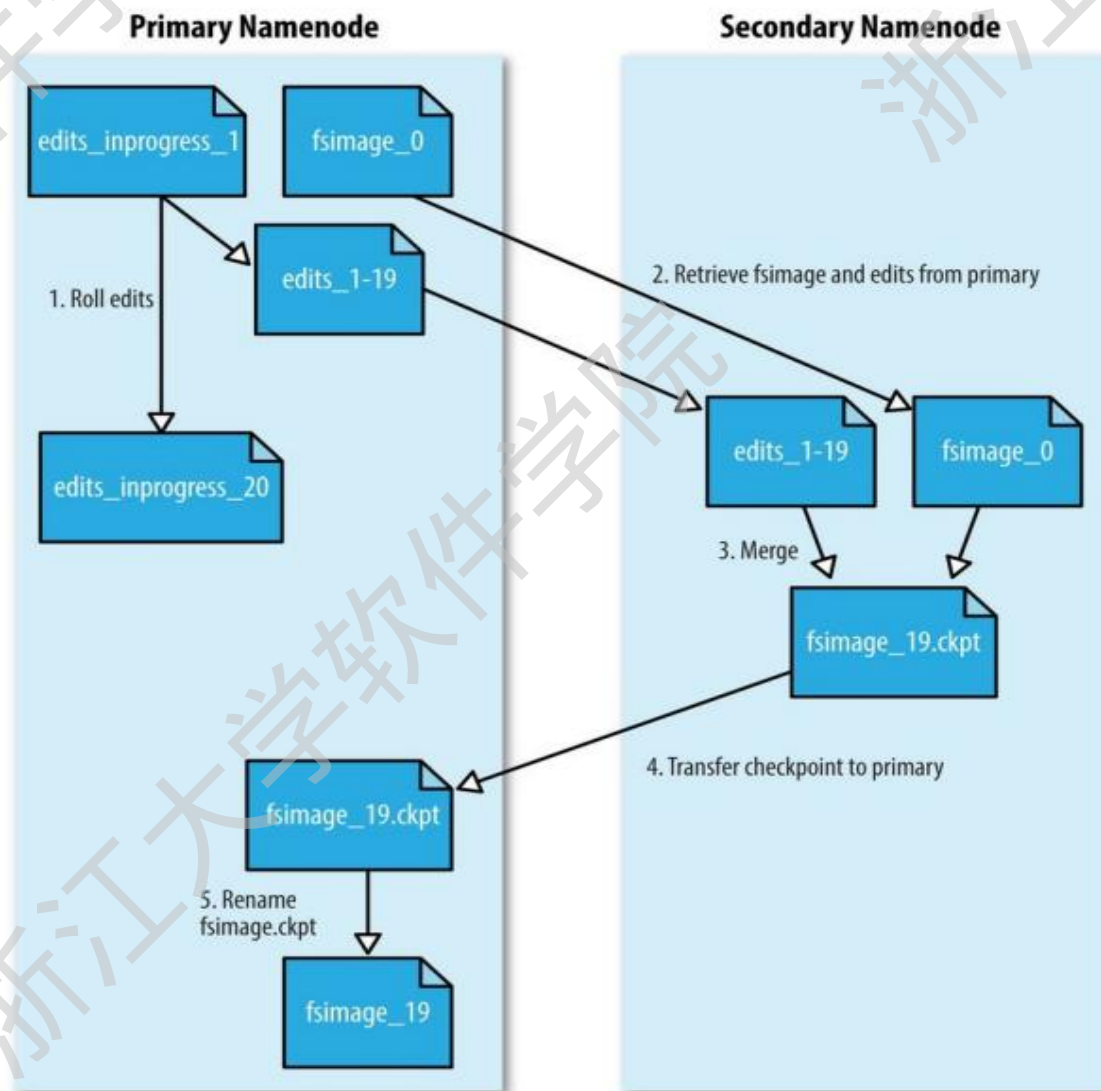
■ SNN(SecondaryNameNode)职责概述

- Hadoop需要一个易于管理的机制来帮助我们减小edit logs文件的大小和得到一个最新的fsimage文件，这样也会减小在NameNode上的压力。
- SecondaryNameNode的职责是合并NameNode的edit logs到fsimage文件中。
- 因此常把SecondaryNameNode称之为主角色的辅助角色，辅助NameNode做一些事。



■ SNN (SecondaryNameNode) Checkpoint--概述

- Checkpoint核心是把fsimage与edits log合并以生成新的fsimage的过程。
- 结果：fsimage版本不断更新不会太旧、edits log文件不会太大。



■ SNN (SecondaryNameNode) Checkpoint--流程

1、当触发checkpoint操作条件时，SNN发送请求给NN滚动edits log。

然后NN会生成一个新的编辑日志文件：edits new，便于记录后续操作记录。

2、SNN会将旧的edits log文件和上次fsimage复制到自己本地（使用HTTP GET方式）。

3、SNN首先将fsimage载入到内存，然后一条一条地执行edits文件中的操作，使得内存中的fsimage不断更新，这个过程就是edits和fsimage文件合并。合并结束，SNN将内存中的数据dump生成一个新的fsimage文件。

4、SNN将新生成的Fsimage new文件复制到NN节点。至此刚好是一个轮回，等待下一次checkpoint触发SecondaryNameNode进行工作，一直这样循环操作。

■ NameNode元数据恢复

• NameNode存储多目录

namenode元数据存储目录由参数：`dfs.namenode.name.dir`指定。

`dfs.namenode.name.dir`属性可以配置多个目录，各个目录存储的文件结构和内容都完全一样，相当于备份，这样做的好处是当其中一个目录损坏了，也不会影响到hadoop的元数据，特别是当其中一个目录是NFS（网络文件系统Network File System，NFS）之上，即使你这台机器损坏了，元数据也得到保存。

• 从SecondaryNameNode恢复

SecondaryNameNode在checkpoint的时候会将fsimage和edits log下载到自己的本机上本地存储目录下。并且在checkpoint之后也不会进行删除。

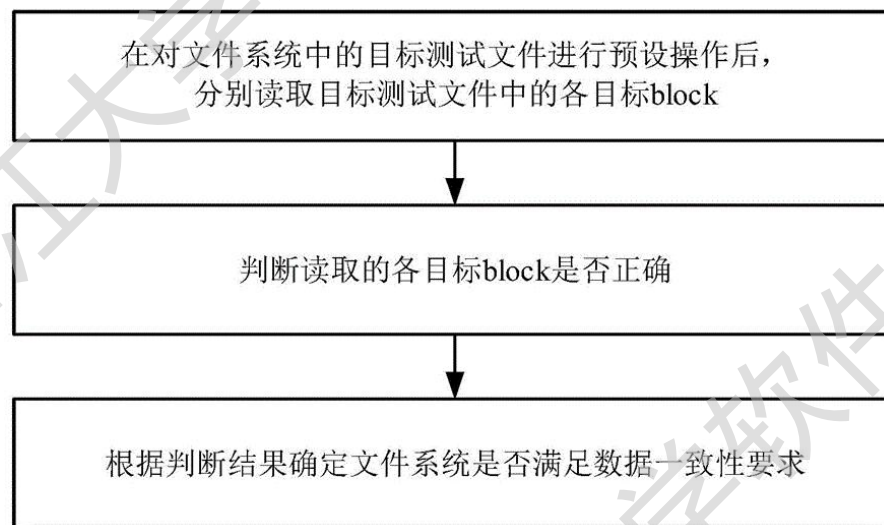
如果NameNode中的fsimage真的出问题了，还是可以用SecondaryNamenode中的fsimage替换一下NameNode上的fsimage，虽然已经不是最新的fsimage，但是我们可以将损失减小到最少



Part5. HDFS数据安全 与高可用



- 1、当DataNode读取Block时，将会计算Checksum。
- 2、若计算后的Checksum和创建Block时值不一样，说明Block损坏，则转而到其他DataNode上读取Block。

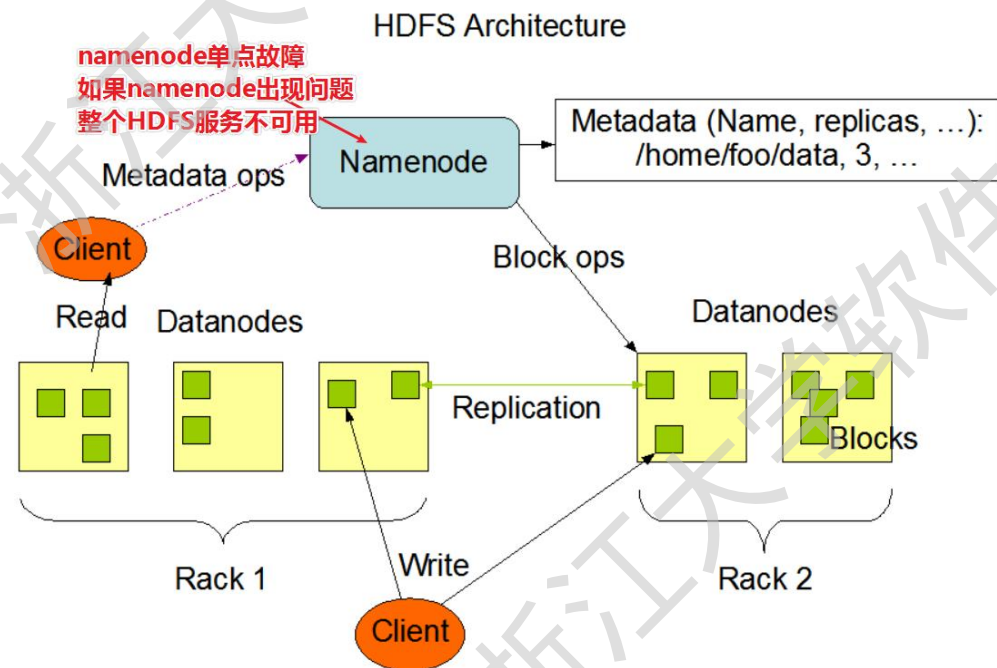


✓ DataNode将在文件创建后周期性验证Checksum

常见的校验算法：CRC(32)，md5(128)，sha1(160)

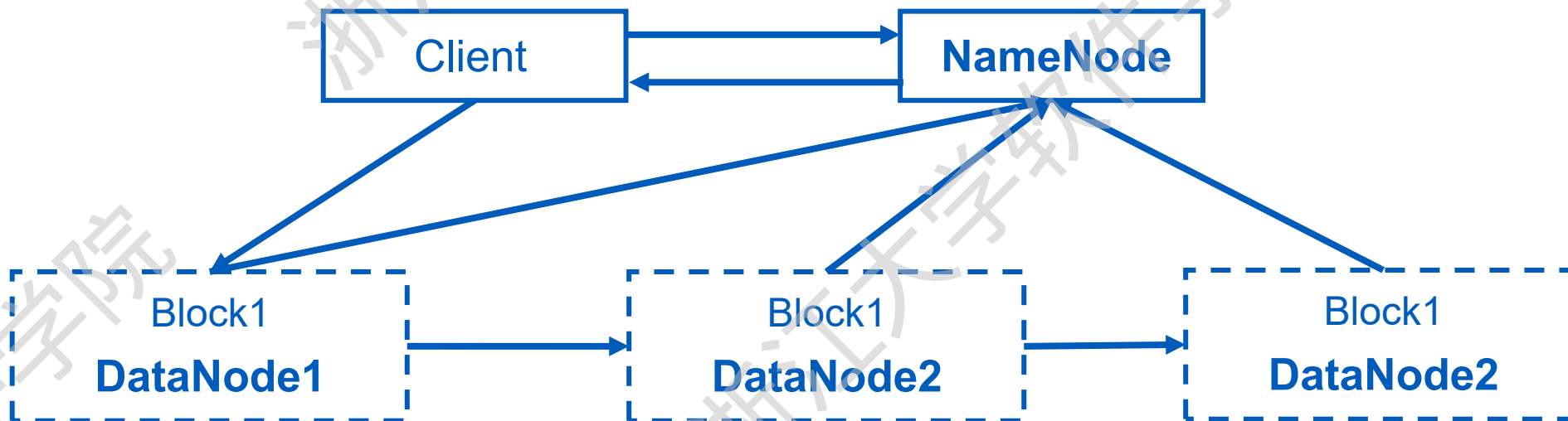
■ NAMENODE单点故障问题

- 在Hadoop 2.0.0之前，NameNode是HDFS集群中的单点故障（SPOF）。
- 每个群集只有一个NameNode，如果NameNode进程不可用，则整个HDFS群集不可用。



■ NAMENODE单点故障问题解决方法

- 在同一群集中运行两个（从3.0.0起，支持超过两个）冗余NameNode。形成主备架构。
- 这样可以在机器崩溃的情况下快速故障转移到新的NameNode，或者出于计划维护的目的由管理员发起的正常故障转移。



■ HA系统（高可用系统）设计核心问题

• 主备协调

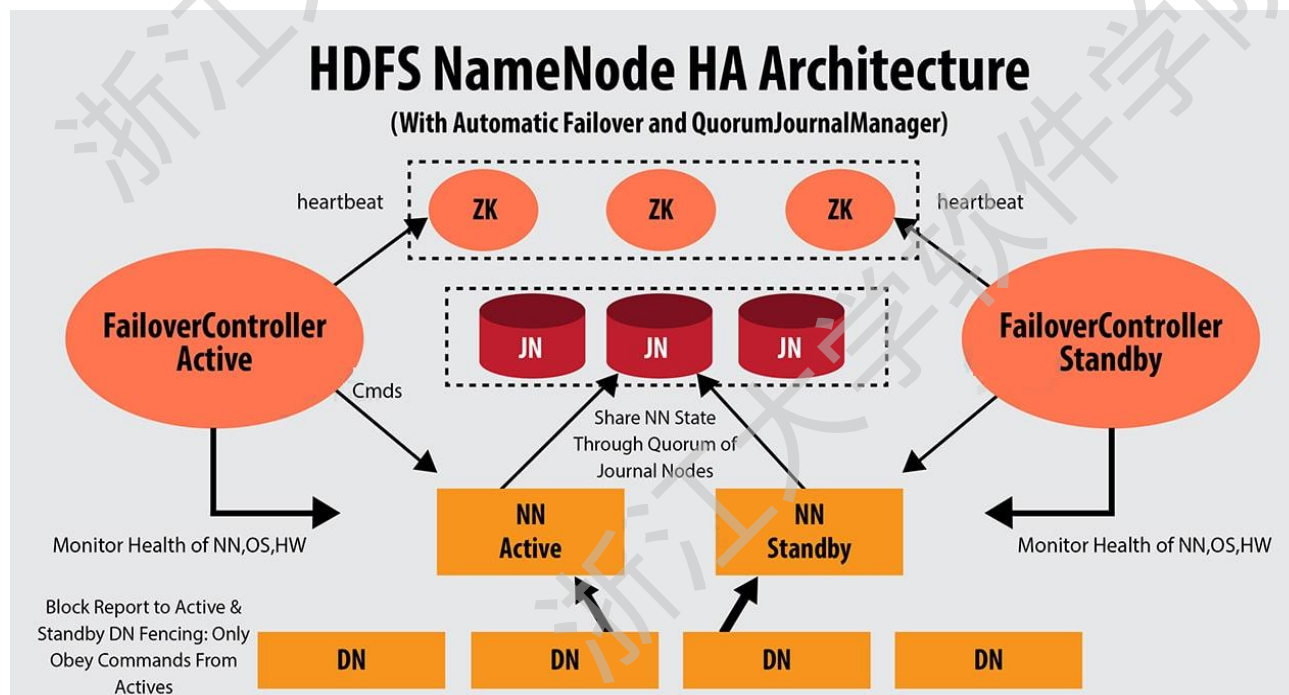
- 在HA集群中，主备节点断开联系时，本来为一个整体、动作协调的HA系统，就分裂成为两个独立的节点。由于相互失去了联系，使得整个集群处于混乱状态。
- 解决方法：保持任意时刻系统有且只有一个主角色提供服务。

• 数据状态同步

- 主备切换保证服务持续可用性的前提是主备节点之间的状态、数据是一致的，或者说准一致的。
- 数据同步常见做法是：通过日志重演操作记录。主角色正常提供服务，发生的事务性操作通过日志记录，备用角色读取日志重演操作。

■ HDFS HA解决方案--QJM

- QJM全称Quorum Journal Manager（仲裁日志管理器），是Hadoop官方推荐的HDFS HA解决方案之一。
- 使用zookeeper中ZKFC来实现主备切换；
- 使用Journal Node（JN）集群实现edits log的共享以达到数据同步的目的。



■ HDFS HA解决方案--QJM

• 主备协调问题解决--ZKFailoverController (zkfc)

ZK Failover Controller (ZKFC) 是一个ZooKeeper客户端。主要职责：

- 监视和管理NameNode健康状态

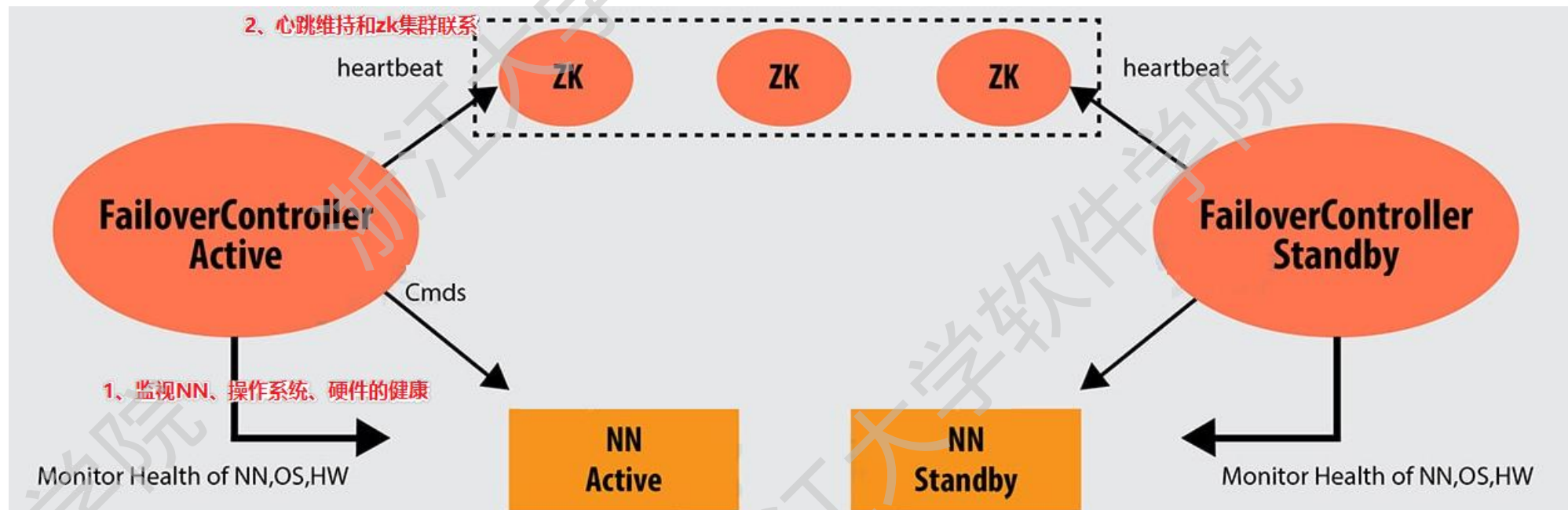
ZKFC通过命令**监视的NameNode节点及机器的健康状态**。

- 维持和ZK集群联系

如果本地NameNode运行状况良好，并且ZKFC看到当前没有其他节点持有锁znode，它将自己尝试获取该锁。如果成功，则表明它“赢得了**选举**”，并负责运行故障转移以使其本地NameNode处于Active状态。

如果已经有其他节点持有锁，zkfc选举失败，则会对该节点注册监听，等待下次继续选举。

■ HDFS HA解决方案--QJM



■ HDFS HA解决方案--QJM

• 主备协调问题解决--Fencing（隔离）机制

- 故障转移过程也就是俗称的主备角色切换的过程，切换过程中最怕的就是混乱状态的发生。因此需要Fencing机制来避免，将先前的Active节点隔离，然后将Standby转换为Active状态。
- Hadoop公共库中对外提供了两种Fencing实现，分别是sshfence和shellfence（缺省实现）。

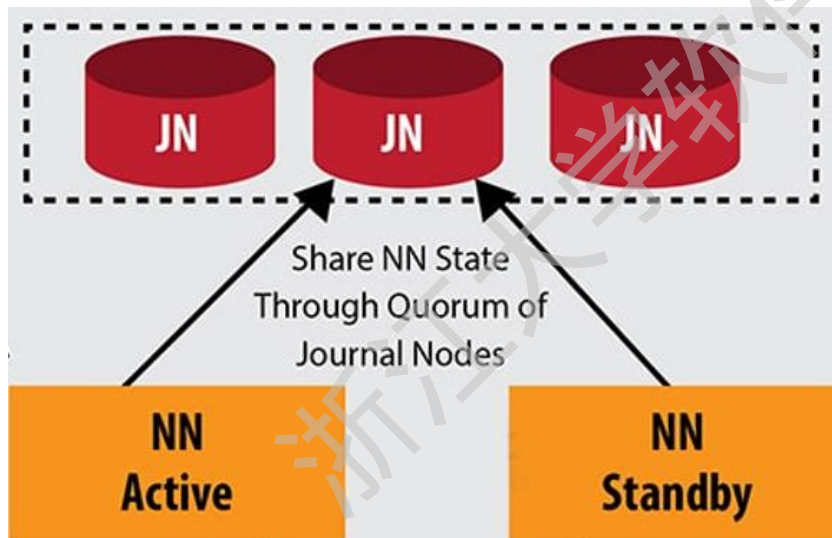
sshfence是指通过ssh登陆目标节点上，使用命令fuser将进程杀死（通过tcp端口号定位进程pid，该方法比jps命令更准确）；

shellfence是指执行一个用户事先定义的shell命令（脚本）完成隔离。

■ HDFS HA解决方案--QJM

• 主备数据状态同步问题解决

- Journal Node (**JN**) 集群是轻量级分布式系统，主要用于高速读写数据、存储数据。
- 通常使用**2N+1**台JournalNode存储共享**Edits Log**（编辑日志）。--底层类似于zk的分布式一致性算法。
- 任何修改操作在 Active NN上执行时，JournalNode进程同时也会记录edits log到**至少半数**以上的JN中，这时 Standby NN 监测到JN 里面的同步log发生了变化了会读取JN里面的edits log，然后重演操作记录同步到自己的目录镜像树里面。



- **NN机制**：NN在工作时将元数据缓存在内存中，同时备份到磁盘fsimage中。对元数据信息修改的操作会追加到editlog文件中。NN定期或者editlog达到一定数量后复制合并fsimage、editlog为一个新的fsimage，在推送给NN。
- **心跳机制**：DN定期发送元数据信息给NN，默认时3秒一次- ****安全模型****：HDFS初始化阶段会进入安全(safe)模式，此时NN不允许操作。NN同DN进行安全检查，当安全的数据块比值达到阈值才会退出安全模式。
- **回滚机制**：在hdfs升级或者数据写入时，相关的数据会被保留备份。成功则更新备份，失败则使用备份
- **安全校验**：避免网络传输造成的数据错误问题，HDFS采用了校验和机制。各个NN之间数据备份和读取需要通过校验，校验不通过则重新备份
- **回收站**：当数据文件从hdfs删除时，文件转存于/trash目录在。在超过规定的时间fs.trash.interval，NN和DN会将该文件的元数据删除

- HDFS Trash机制，叫做回收站或者垃圾桶。Trash就像Windows操作系统中的回收站一样。它的目的是防止你无意中删除某些东西。默认情况下是不开启的。
 - 启用Trash功能后，从HDFS中删除某些内容时，文件或目录不会立即被清除，它们将被移动到回收站Current目录中(/user/\${username}/.Trash/current)。
 - .Trash中的文件在用户可配置的时间延迟后被永久删除。
 - 也可以简单地将回收站里的文件移动到.Trash目录之外的位置来恢复回收站中的文件和目录。
- **Trash Checkpoint**
 - 检查点仅仅是用户回收站下的一个目录，用于存储在创建检查点之前删除的所有文件或目录。
 - 最近删除的文件被移动到回收站Current目录，并且在可配置的时间间隔内，HDFS会为在Current回收站目录下的文件创建检查点，并在过期时删除旧的检查点。

■ 快照作用

- 数据恢复

对重要目录进行创建snapshot的操作，当用户误操作时，可以通过snapshot来进行相关的恢复操作。

- 数据备份

使用snapshot来进行整个集群，或者某些目录、文件的备份。管理员以某个时刻的snapshot作为备份的起始结点，然后通过比较不同备份之间差异性，来进行增量备份。

- 数据测试

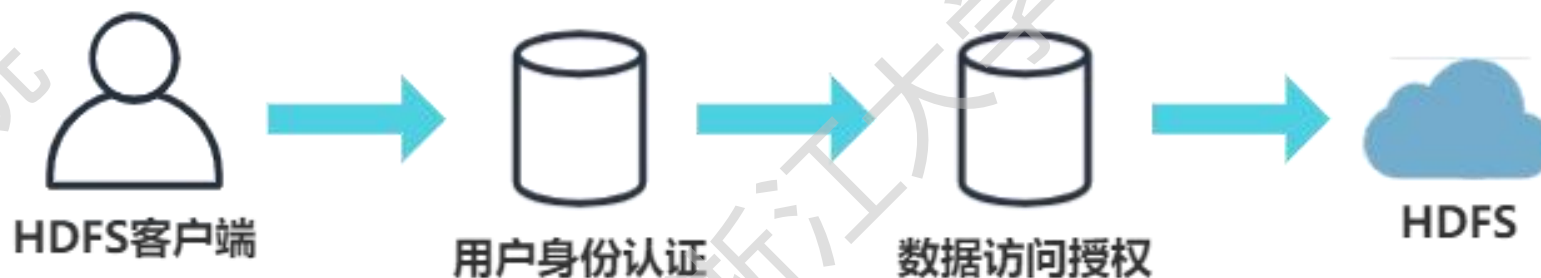
在某些重要数据上进行测试或者实验，可能会直接将原始的数据破坏掉。可以临时的为用户针对要操作的数据来创建一个snapshot，然后让用户在对应的snapshot上进行相关的实验和测试，从而避免对原始数据的破坏。

■ HDFS快照功能的实现

- HDFS快照不是数据的简单拷贝，只做差异的记录。
- 对于大多不变的数据，你所看到的数据其实是当前物理路径所指的内容，而发生变更的inode数据才会被快照额外拷贝，也就是所说的差异拷贝。
- inode指索引节点，用来存放文件及目录的基本信息，包含时间、名称、拥有者、所在组等。

■ HDFS权限管理概述

- 作为分布式文件系统，HDFS也集成了一套权限管理系统。
- 客户端在进行每次文件操作时，系统会从**用户身份认证**和**数据访问授权**两个环节进行验证。
- 客户端的操作请求会首先通过用户身份验证机制来获得“凭证”（类似于身份证），HDFS根据此“凭证”分辨出合法的用户名；
- 然后HDFS再据此查看该用户所访问的数据是否已经授权,或者说该身份凭证是否具有权限做这件事。
- 一旦这个流程中的某个环节出现异常，客户端的操作请求便会失败。



■ 拥有者、所在组、其他用户组

- HDFS文件权限与Linux/Unix系统的UGO模型类似，简单描述为：每个文件和目录都与一个拥有者和一个组相关联。
- **USER（文件的所有者）**：一般是创建该文件的用户，对该文件具有完全的权限。
- **GROUP（拥有者所在的组）**：和文件所有者属于同一组的用户。
- **OTHER（其他用户组）**：其他用户组的用户。

■ 读、写、执行权限

- HDFS文件权限也细分为：读权限（r）、写权限（w）、执行权限（x）。
- 在HDFS中，对于文件，需要r权限才能读取文件，而w权限才能写入或追加到文件。没有x可执行文件的概念。
- 在HDFS中，对于目录，需要r权限才能列出目录的内容，需要w权限才能创建或删除文件或目录，并且需要x权限才能访问目录的子级。
- 读权限（r）、写权限（w）、执行权限（x）可以使用数字表示，也可以使用字母表示。

■ umask权限掩码

- 与Linux/Unix系统类似，HDFS也提供了umask掩码，用于设置在HDFS中**默认新建的文件和目录权限位**。
- 默认umask值有属性fs.permissions.umask-mode指定，默认值022。
- 创建文件和目录时使用的umask，默认的权限就是

目录： $777-022=755$ ，也就是drwxr-xr-x

文件： $777-022=755$ ，因为HDFS中文件没有x执行权限的概念，所以是：-rw-r--r--

✓ 权限修改

可以使用命令行以及HDFS Web页面进行UGO权限的修改

■ HDFS用户身份认证概述

- 在HDFS中，用户身份认证独立于HDFS项目之外，也就说HDFS并不负责用户身份合法性检查。
- 但HDFS会通过相关接口来获取相关的用户身份，然后用于后续的权限管理。
- 用户是否合法，完全取决于集群使用认证体系。目前社区支持两种身份认证，即简单认证（Simple）和Kerberos。
- 由hadoop.security.authentication属性指定，默认simple。

■ Simple认证

- 基于HDFS客户端所在的Linux/Unix系统的登录用户名来进行认证。只要用户能正常登录就认证成功。
- 客户端与NN交互时，会将用户的登录账号作为合法用户名传递至NN。
- 意味着使用不同的账号登录到同一个客户端，会产生不同的用户名，故在多租户条件这种认证会导致权限混淆；同时恶意用户也可以伪造其他人的用户名非法获得相应的权限，对数据安全造成极大的隐患。线上生产环境一般不会使用。

■ Kerberos介绍

- 在神话里，Kerberos是Cerberus的希腊语，是一只守护地狱入口的三头巨犬，它确保没有人能在进入地狱后离开。
- 从技术角度来说，Kerberos是麻省理工学院（MIT）开发的一种网络身份**认证协议**。它旨在通过使用密钥加密技术为客户端/服务器应用程序提供强身份验证。

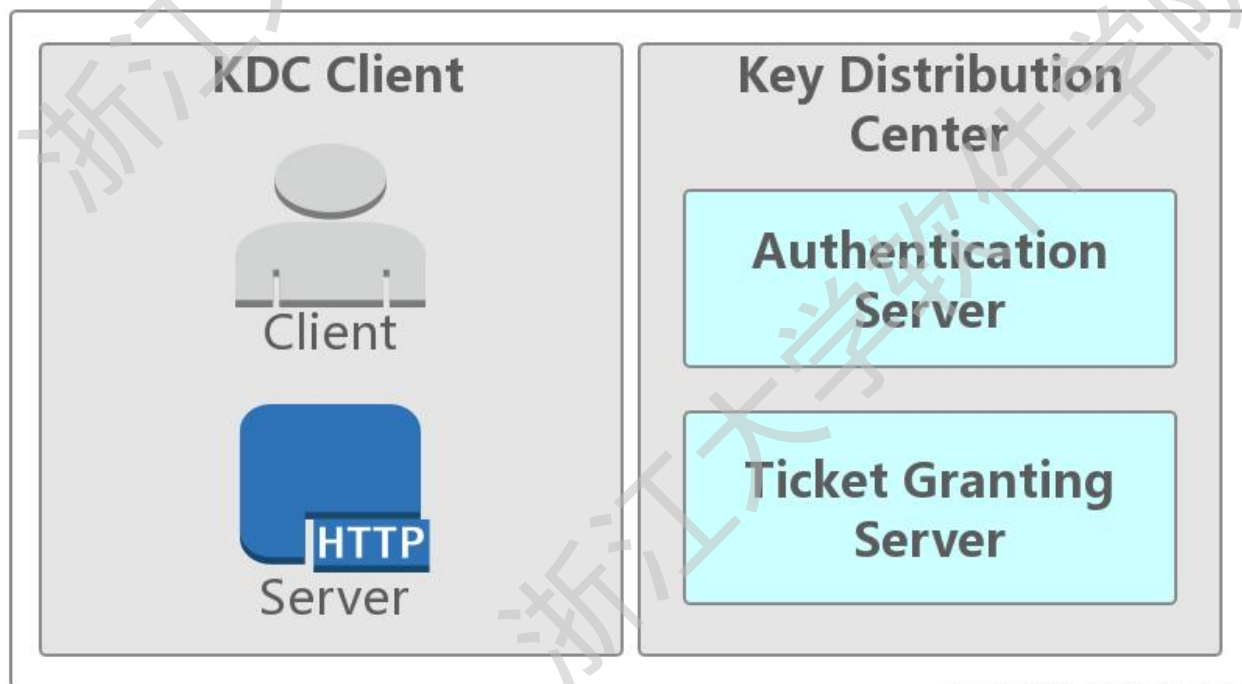


[ABOUT](#) | [NEWS](#) | [EVENTS](#) | [SOFTWARE](#) | [SPONSORS](#) | [DOCUMENTATION](#)



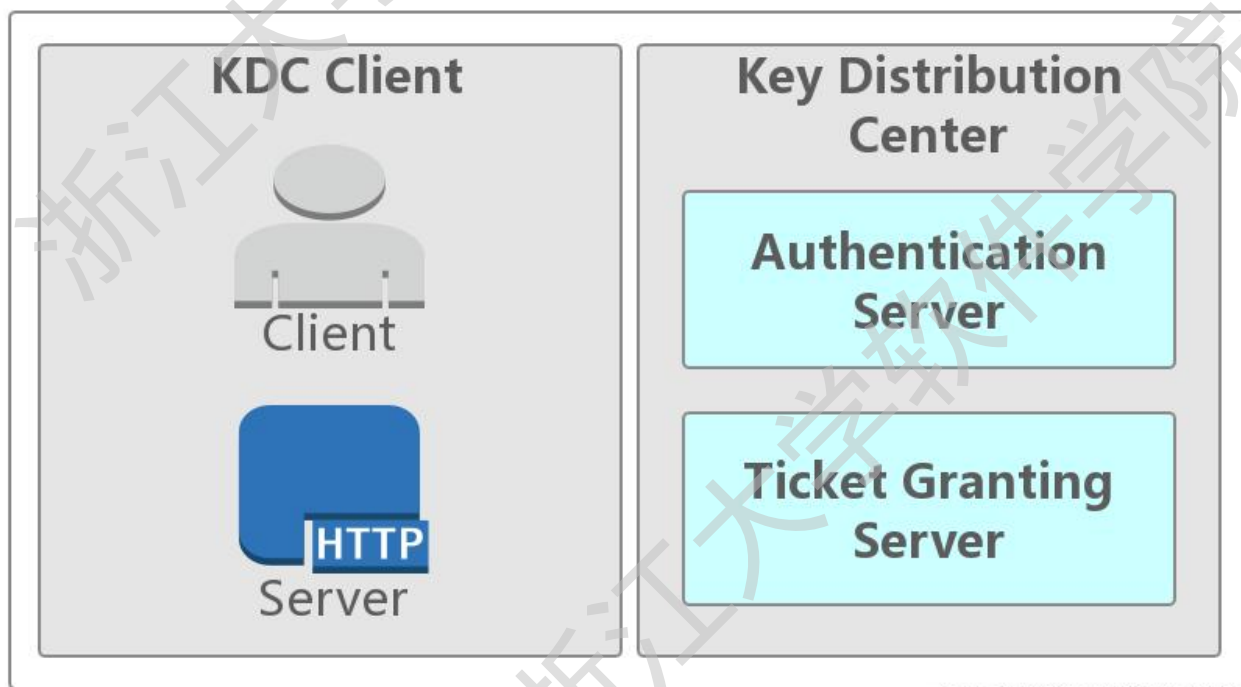
■ Kerberos角色

- 访问服务的**Client**
- 提供服务的**Server**
- **KDC** (Key Distribution Center) 密钥分发中心



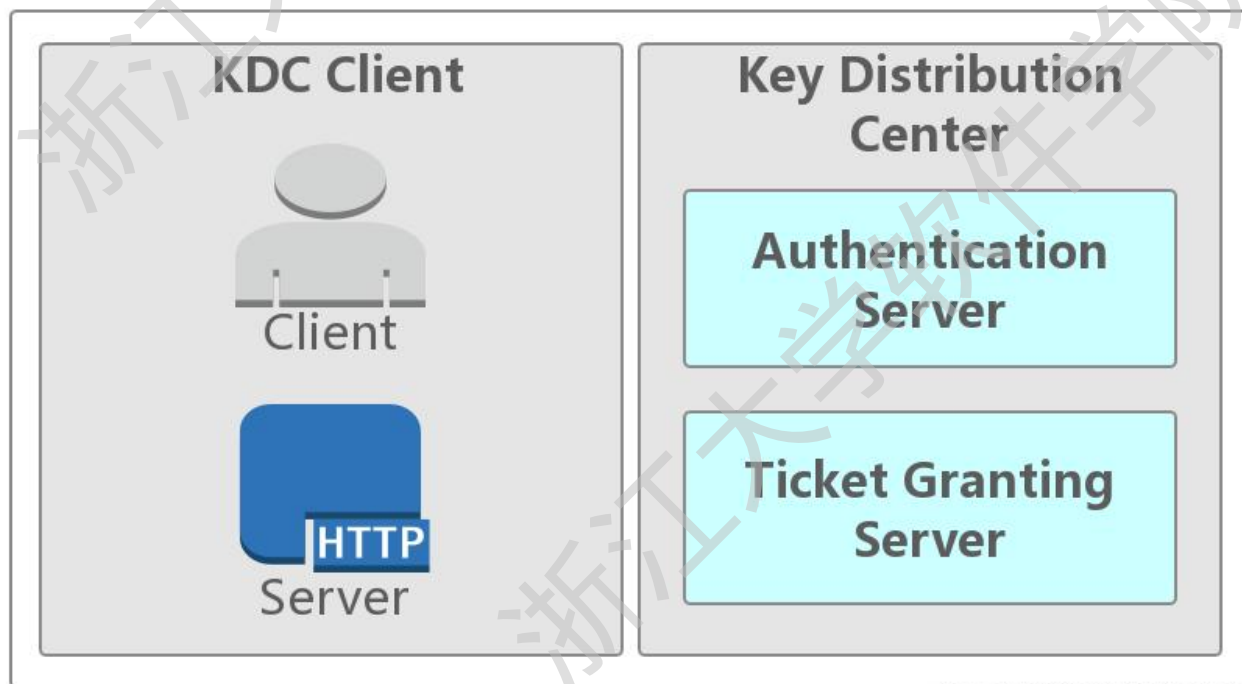
■ Kerberos-域概念

- 其中KDC服务默认会安装在一个**域**的域控中，而Client和Server为域内的用户或者是服务，如HTTP服务，SQL服务。在Kerberos中Client是否有权访问Server端的服务由KDC发放的**票据**来决定。



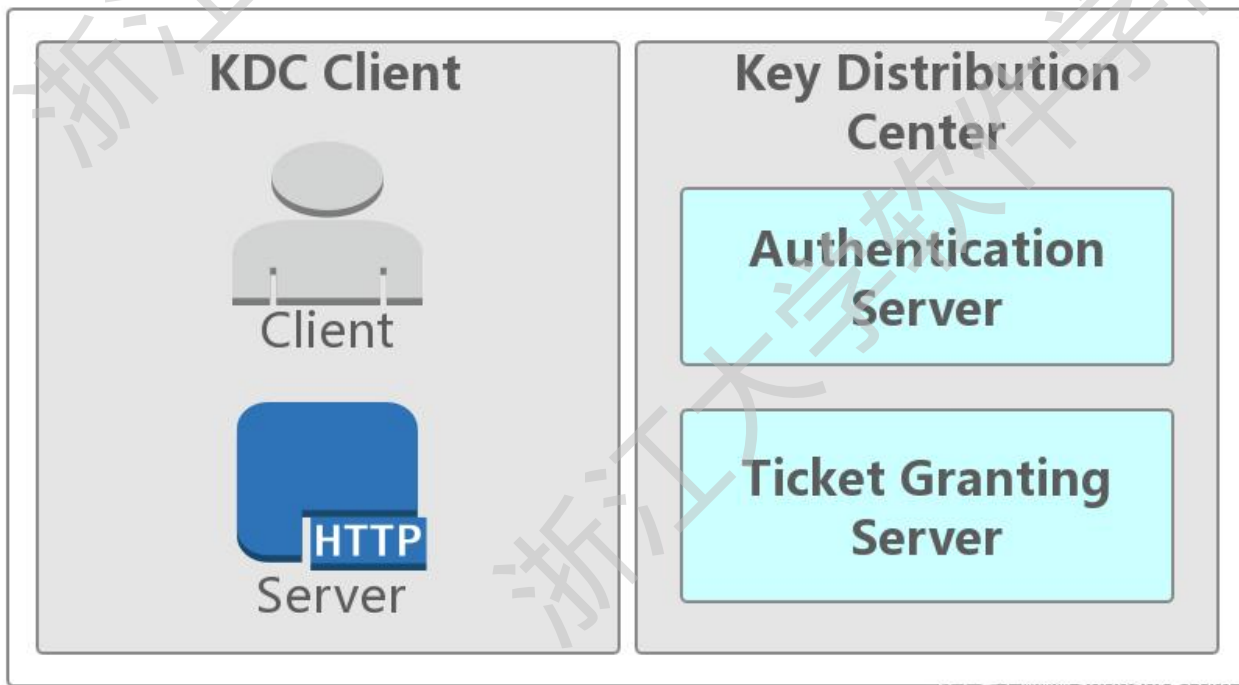
■ Kerberos--票据

- 如果把Kerberos中的票据类比为一张火车票，那么Client端就是乘客，Server端就是火车，而KDC就是车站的认证系统。如果Client端的票据是合法的（由你本人身份证购买并由你本人持有）同时有访问Server端服务的权限（车票对应车次正确）那么你能上车。当然和火车票不一样的是Kerberos中有存在两张票，而火车票从头到尾只有一张。

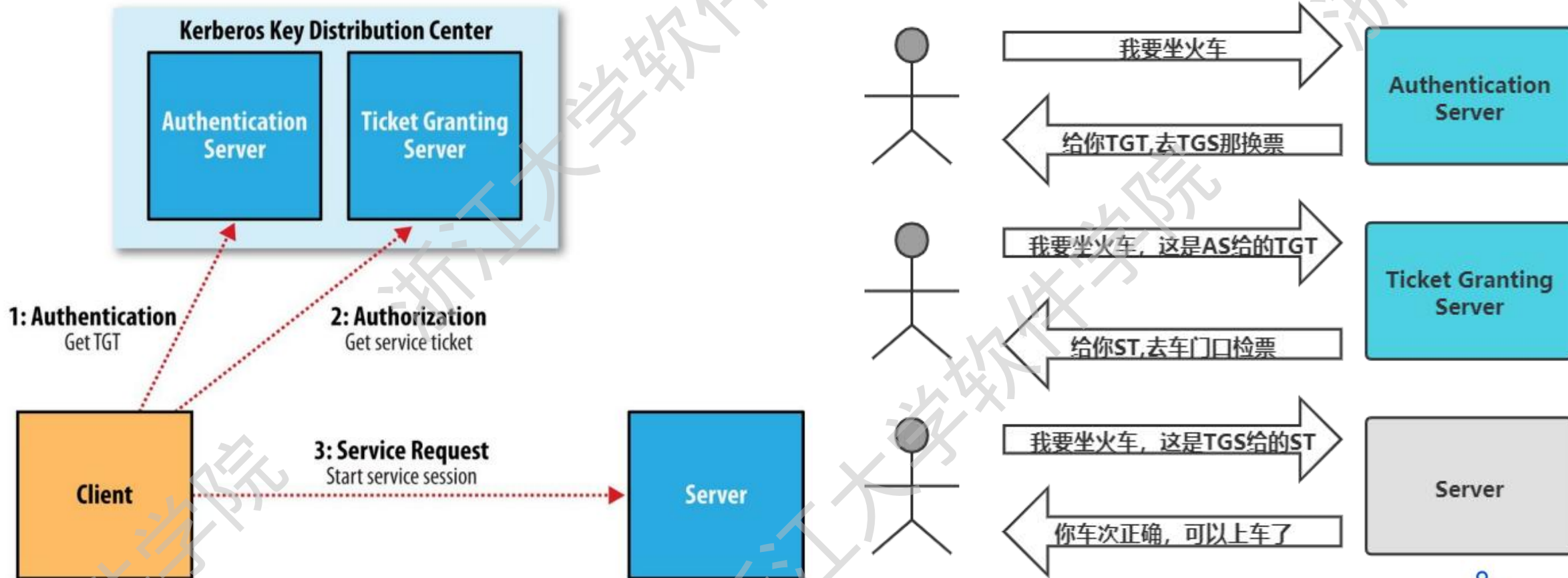


■ Kerberos-KDC

- Authentication Server: AS的作用就是**验证Client端的身份**（确定你是身份证上的本人），验证通过就会给一张TGT（Ticket Granting Ticket）票给Client。
- Ticket Granting Server: TGS的作用是**通过AS发送给Client的票（TGT）**换取访问**Server端的票**（上车的票ST）。ST（Service Ticket）也有资料称为TGS Ticket，为了和TGS区分，在这里就用ST来说明。



■ Kerberos-认证流程



■ HDFS Group Mapping组映射概述

- 在通过用户身份认证拿到用户名后之后，NameNode还需要通过用户组映射服务获取该用户所对应的用户组列表，用于后期的用户组权限校验
- HDFS中用户所属组的确认工作需要通过外部的用户组映射（Group Mapping）服务来获取。用户到组的映射可以使用系统自带的方案（使用NameNode服务器上的用户组系统），也可以通过其他实现类似功能的插件（LDAP、Ranger等）方式来代替。

■ 基于Linux/Unix系统的用户和用户组

- Linux/Unix系统上的用户和用户组信息存储在/etc/passwd和/etc/group文件中。
- 默认情况下，HDFS会通过调用外部的 Shell 命令来获取用户的所有用户组列表。
- 此方案的优点在于组映射服务十分稳定，不易受外部服务的影响。
- 但是用户和用户组管理涉及到root权限等，同时会在服务器上生成大量的用户组，后续管理，特别是自动化运维方面会有较大影响。

■ HDFS ACL权限管理

- 在UGO权限中，用户对文件只有三种身份，就是属主（user）、属组（group）和其他人（other）。
- 每种用户身份拥有读（read）、写（write）和执行（execute）三种权限。
- 但是在实际工作中，使用UGO来控制权限可以满足大部分场景下的数据安全性要求，但是对于一些复杂的权限需求则无能为力。
- ACL是Access Control List（访问控制列表）的缩写，ACL提供了一种方法，可以为特定的用户或组设置不同的权限，而不仅仅是文件的所有者和文件的组。

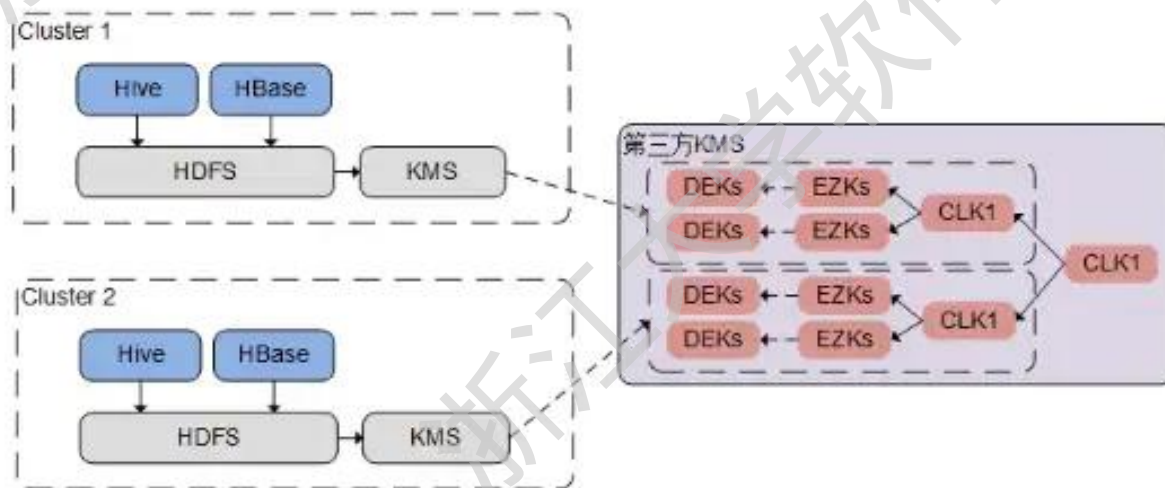
■ Proxy user代理用户

- Proxy中文称之为代理、委托。用来进行事物不想或不能进行的其他操作。
- HDFS Proxy user（代理用户）描述的是一个用户（比如超级用户）如何代表另一个用户提交作业或访问HDFS。
- 比如：用户名为“Admin”的超级用户代表用户Bill提交作业并访问HDFS。

因为超级用户Admin具有kerberos凭证，但Bill用户没有任何凭证。这些任务需要以用户Bill的身份运行，而对namenode的任何文件访问都必须以用户Bill的身份进行。要求用户Bill可以在通过Admin的kerberos凭证进行身份验证的连接上连接到namenode。换句话说，Admin正在冒充用户Bill。

■ HDFS 透明加密介绍

- HDFS透明加密（Transparent Encryption）支持端到端的透明加密，启用以后，对于一些需要加密的HDFS目录里的文件可以实现透明的加密和解密，而不需要修改用户的业务代码。**端到端是指加密和解密只能通过客户端。**
- 对于加密区域里的文件，HDFS**保存的即是加密后的文件**，文件**加密的密钥也是加密的**。让非法用户即使从操作系统层面拷走文件，也是密文，没法查看。



■ 透明加密的特点

- HDFS集群管理和密钥的管理是互相独立的职责，由不同的用户角色（HDFS管理员，密钥管理员）承担
- 只有HDFS客户端可以加密或解密数据，密钥管理在HDFS外部，HDFS无法访问未加密的数据或加密密钥
- block在操作系统是以加密的形式存储的，从而减轻了操作系统和文件系统级别的安全威胁
- HDFS使用AES-CTR加密算法。AES-CTR支持128位加密密钥（默认）

■ HDFS透明加密关键概念和架构

加密区域

- HDFS的透明加密有一个新的概念，**加密区域**（the encryption zone）。
- **加密区域就是HDFS上的一个目录**，只不过该目录比其他目录特殊。
- 加密区域里写入文件的时候会被透明加密，读取文件的时候又会被透明解密。

■ HDFS透明加密关键概念和架构

密钥

- 当加密区域被创建时，都会有一个**加密区域密钥（EZ密钥， encryption zone key）**与之对应，EZ密钥存储在HDFS外部的密钥库中。
- 加密区域里的每个文件都有其自己加密密钥，叫做**数据加密密钥（DEK， data encryption key）**。
- DEK会使用其各自的加密区域的EZ密钥进行加密，以形成**加密数据加密密钥（EDEK）**。

■ HDFS透明加密关键概念和架构

密钥库（keystore）

- **存储密钥（key）的叫做密钥库（keystore）**，将HDFS与外部企业级密钥库（keystore）集成是部署透明加密的第一步。
- 这是因为密钥（key）管理员和HDFS管理员之间的职责分离是此功能的非常重要的方面。
- 但是，大多数密钥库都不是为Hadoop工作负载所见的加密/解密请求速率而设计的。

■ HDFS透明加密关键概念和架构

KMS（密钥管理服务）

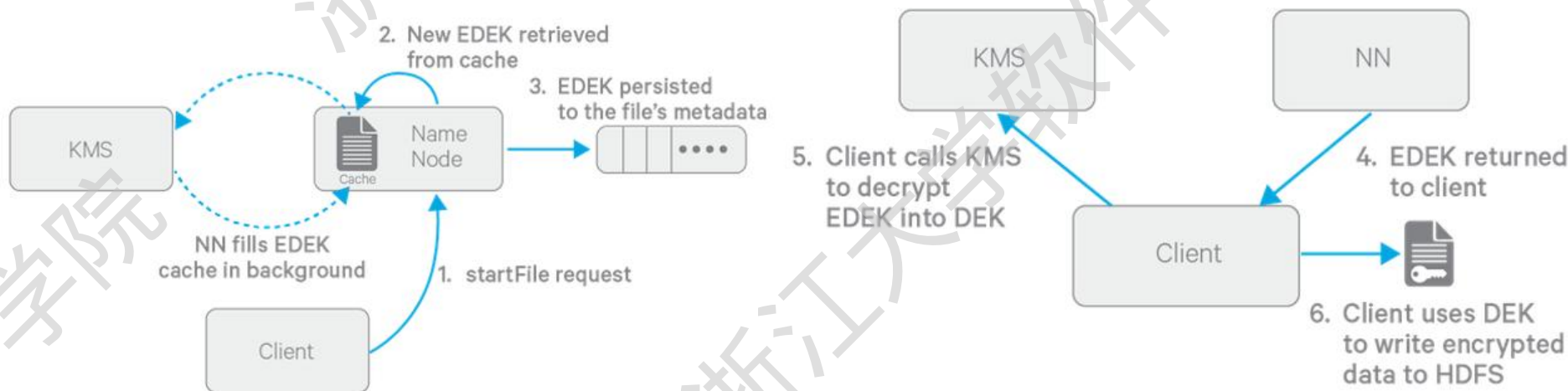
- Hadoop**密钥管理服务**（Key Management Server，简写KMS），用作HDFS客户端与密钥库之间的代理。
- KMS主要有以下几个职责：
 - 1.访问加密区域密钥（EZ key）
 - 2.生成EDEK，EDEK存储在NameNode上
 - 3.为HDFS客户端解密EDEK

■ HDFS透明加密关键概念和架构

写入加密文件过程

提前动作：创建加密区，设置加密区密钥

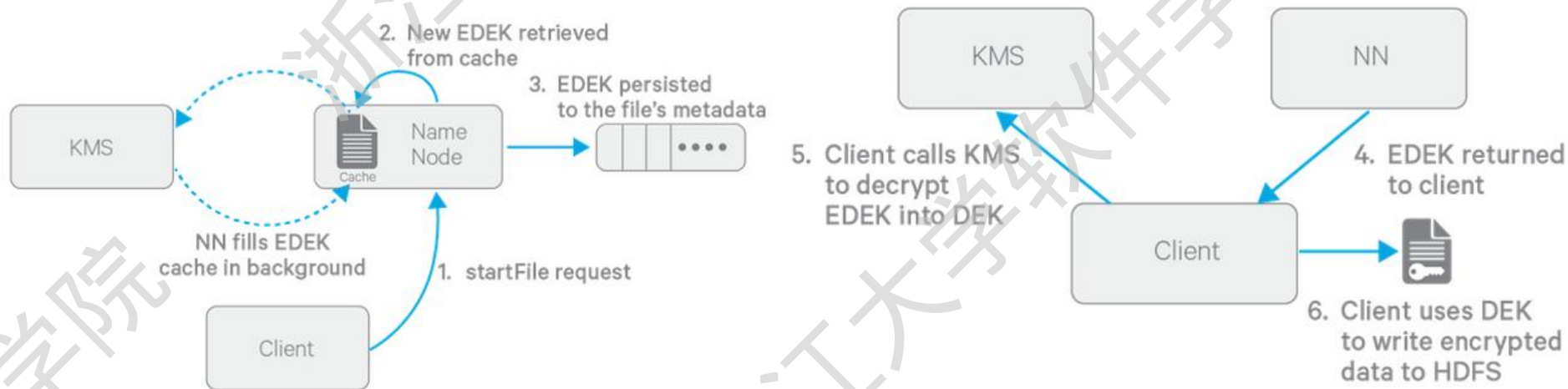
- 1、Client向NN请求在HDFS某个加密区新建文件；
- 2、NN从缓存中取出一个新的EDEK（后台不断从KMS拉取新的EDEK到缓存中）；



■ HDFS透明加密关键概念和架构

写入加密文件过程

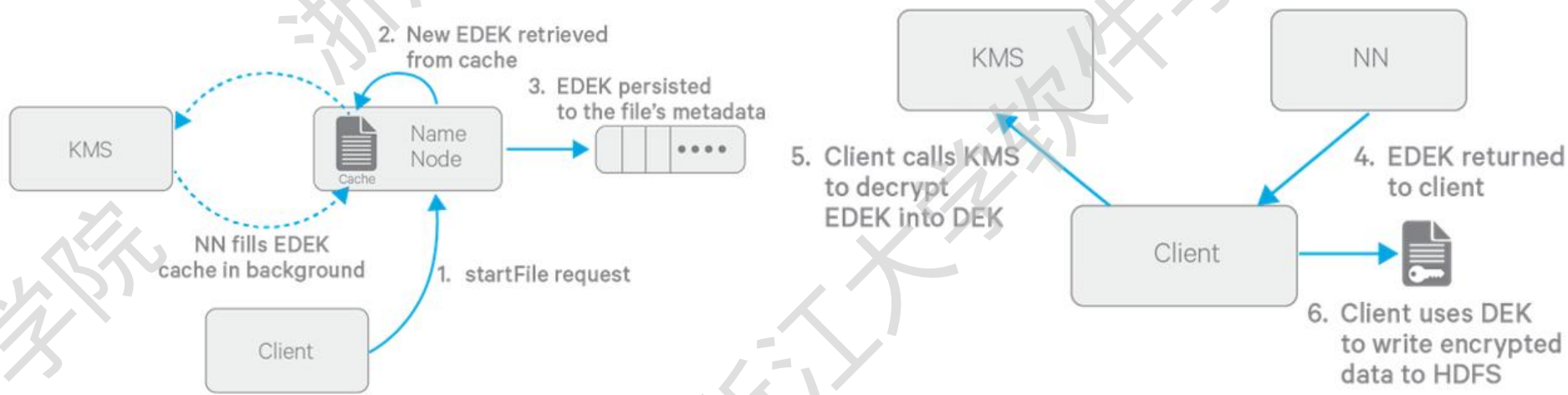
- 3、获取到EDEK会被NN保存到文件的元数据中；
- 4、然后NN将EDEK发送给Client；



■ HDFS透明加密关键概念和架构

写入加密文件过程

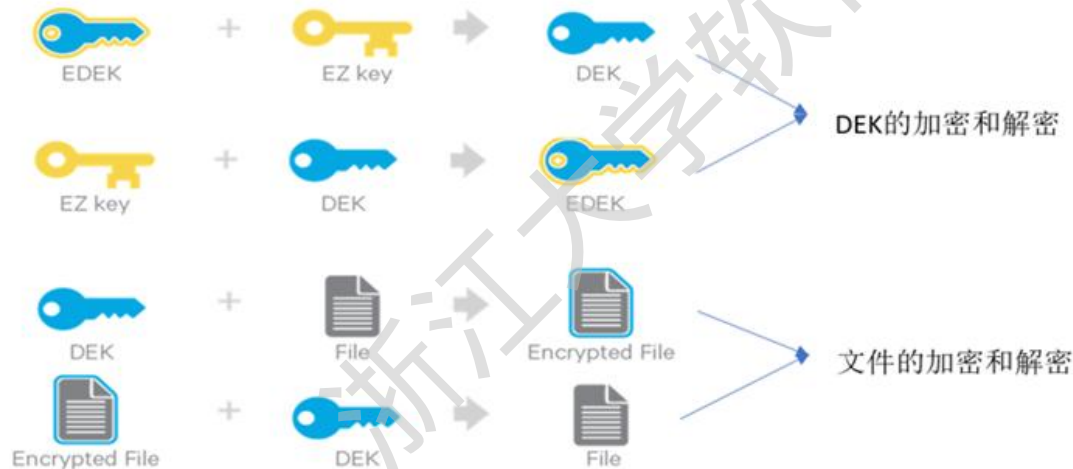
- 5、Client发送EDEK给KMS，KMS用对应的EZ key将EDEK解密出DEK发送给Client；
- 6、Client用DEK加密文件内容发送给datanode进行存储。



■ HDFS透明加密关键概念和架构

写入加密文件过程

- DEK是加解密一个文件的密钥，而KMS里存储的EZ key是用来加解密所有文件的密钥（DEK）的密钥。
- 所以，EZ Key是更为重要的数据，只在KMS内部使用（DEK的加解密只在KMS内存进行），不会被传递到外面使用；
- 而HDFS服务端只能接触到EDEK。



■ HDFS透明加密关键概念和架构

读取解密文件过程

- 读流程与写流程类型，区别就是NN直接读取加密文件元数据里的EDEK返回给客户端，客户端一样把EDEK发送给KMS获取DEK。再对加密内容解密读取。
- EDEK的加密和解密完全在KMS上进行。更重要的是，请求创建或解密EDEK的客户端永远不会处理EZ密钥。仅KMS可以根据要求使用EZ密钥创建和解密EDEK。

结 束