1. 启动Hadoop集群会分别启动哪些进程,各自的作用

- **NameNode：**
  - 维护文件系统树及整棵树内所有的文件和目录。这些信息永久保存在本地磁盘的两个文件中：命名空间镜像文件、编辑日志文件
  - 记录每个文件中各个块所在的数据节点信息，这些信息在内存中保存，每次启动系统时重建这些信息
  - 负责响应客户端的 数据块位置请求 。也就是客户端想存数据，应该往哪些节点的哪些块存；客户端想取数据，应该到哪些节点取
  - 接受记录在数据存取过程中，datanode节点报告过来的故障、损坏信息
- **SecondaryNameNode(非HA模式)：**
  - 实现namenode容错的一种机制。定期合并编辑日志与命名空间镜像，当namenode挂掉时，可通过一定步骤进行上顶。(**注意 并不是NameNode的备用节点**)
- **DataNode：**
  - 根据需要存取并检索数据块
  - 定期向namenode发送其存储的数据块列表
- **ResourceManager：**
  - 负责Job的调度,将一个任务与一个NodeManager相匹配。也就是将一个MapReduce之类的任务分配给一个从节点的NodeManager来执行。
- **NodeManager：**
  - 运行ResourceManager分配的任务，同时将任务进度向application master报告
- **JournalNode(HA下启用):**
  - 高可用情况下存放namenode的editlog文件

2. Hadoop1.x的缺点

1. JobTracker存在单点故障的隐患
2. 任务调度和资源管理全部是JobTracker来完成,单点负担过重
3. TaskTracker以Map/Reduce数量表示资源太过简单
4. TaskTracker 分Map Slot 和 Reduce Slot, 如果任务只需要map任务可能会造成资源浪费

3. Hadoop1.x 和Hadoop 2.x 的区别

1. **资源调度方式的改变**

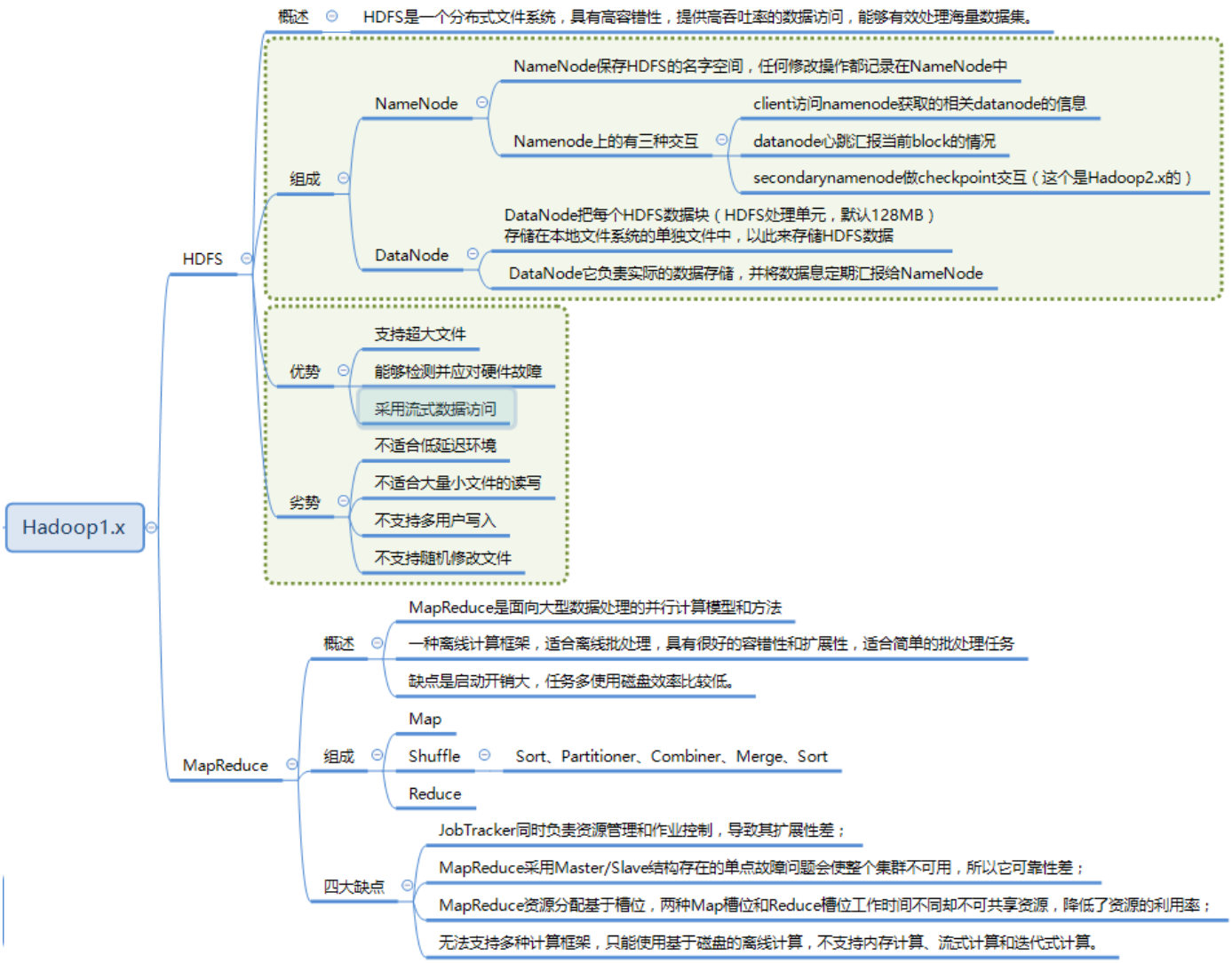
   在1.x, 使用Jobtracker负责任务调度和资源管理,单点负担过重,在2.x中,新增了yarn作为集群的调度工具.在yarn中,使用ResourceManager进行 资源管理, 单独开启一个Container作为ApplicationMaster来进行任务管理.

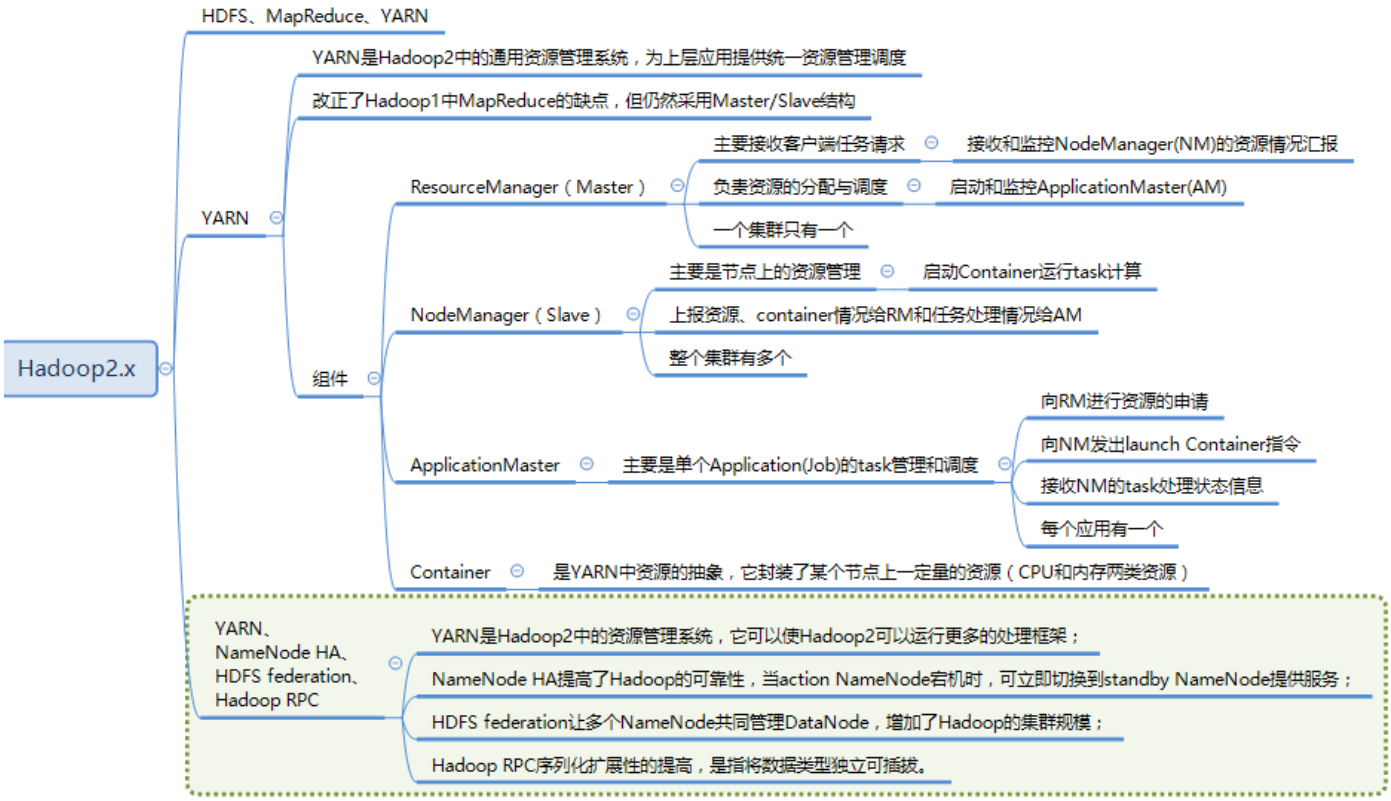
2. **HA模式**

   在1.x中没有HA模式,集群中只有一个NameNode,而在2.x中可以启用HA模式,存在一个Active NameNode 和 Standby NameNode.

3. **HDFS Federation**

Hadoop 2.0中对HDFS进行了改进，使NameNode可以横向扩展成多个，每个NameNode分管一部分目录，进而产生了HDFS Federation，该机制的引入不仅增强了HDFS的扩展性，也使HDFS具备了隔离性

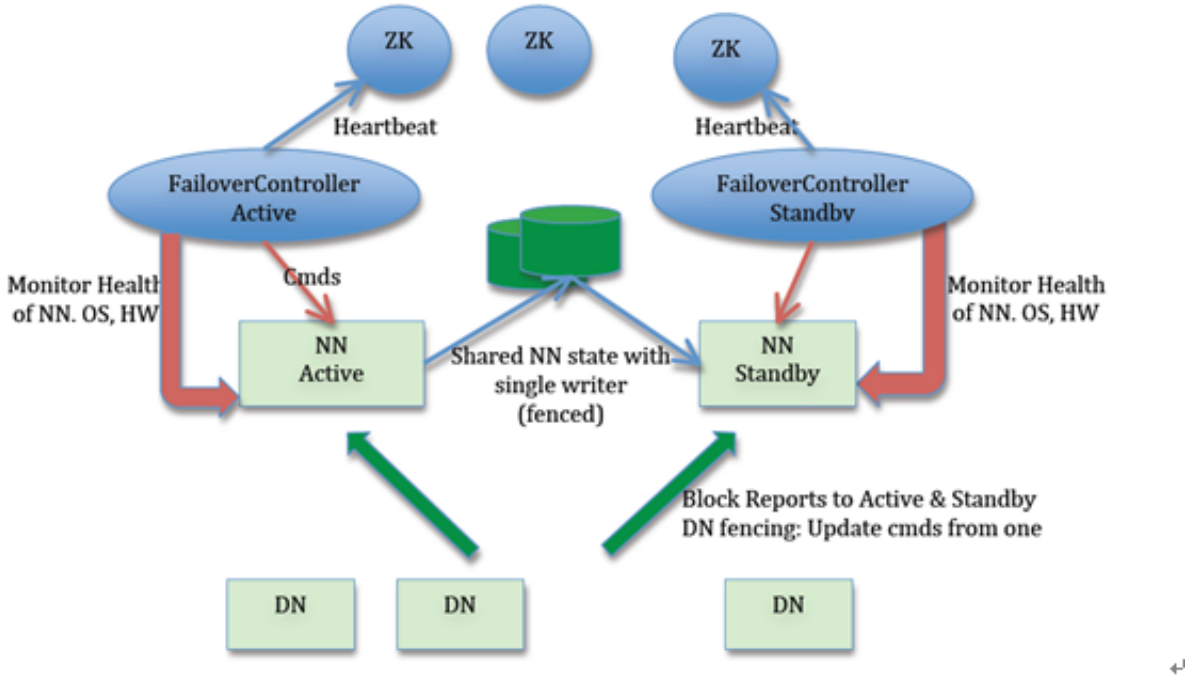




4. Hadoop 的常用配置文件有哪些

- **hadoop-env.sh**: 用于定义hadoop运行环境相关的配置信息，比如配置JAVA_HOME环境变量、为hadoop的JVM指定特定的选项、指定日志文件所在的目录路径以及master和slave文件的位置等；
- **core-site.xml**: 用于定义系统级别的参数，如HDFS URL、Hadoop的临时目录以及用于rack-aware集群中的配置文件的配置等，此中的参数定义会覆盖core-default.xml文件中的默认配置；
- **hdfs-site.xml**: HDFS的相关设定，如文件副本的个数、块大小及是否使用强制权限等，此中的参数定义会覆盖hdfs-default.xml文件中的默认配置；
- **mapred-site.xml**：HDFS的相关设定，如reduce任务的默认个数、任务所能够使用内存的默认上下限等，此中的参数定义会覆盖mapred-default.xml文件中的默认配置。

5. Hadoop HA介绍

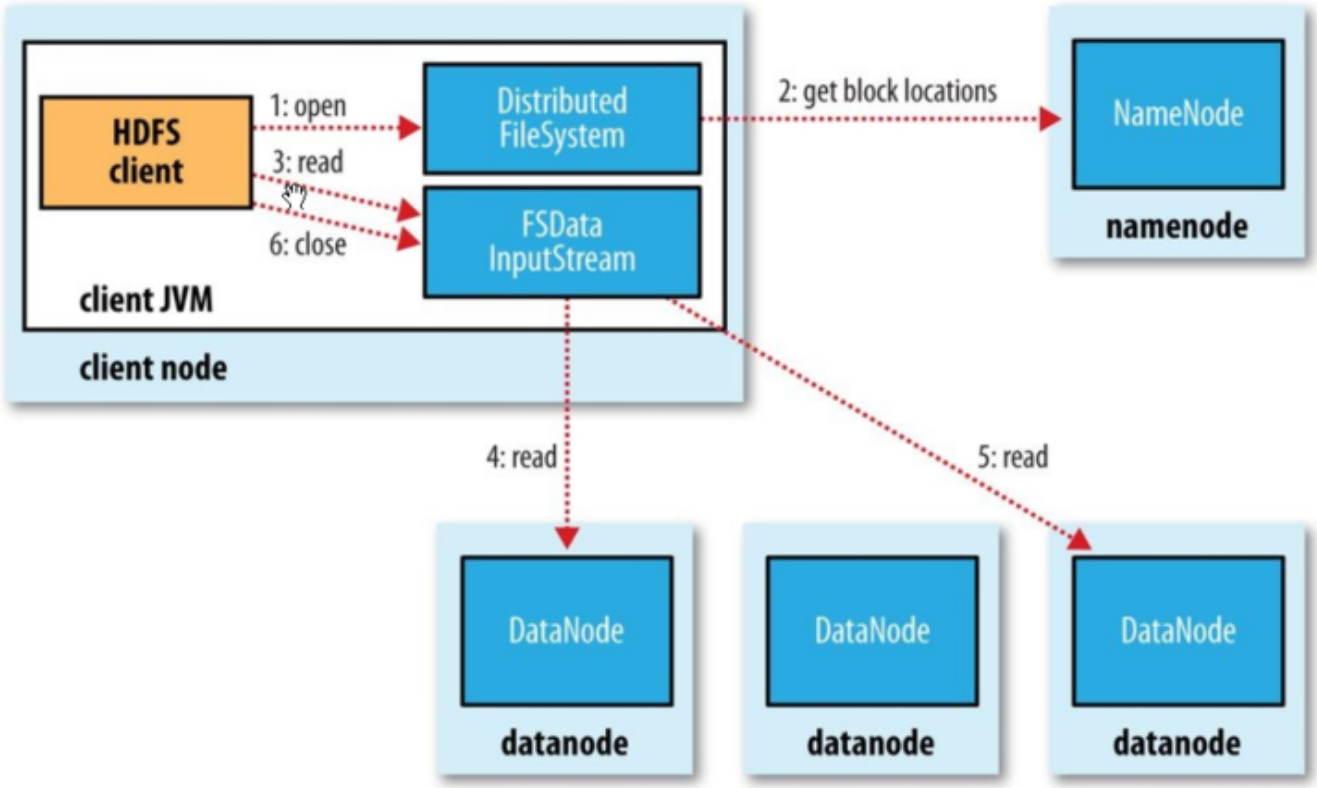


1. **Active NameNode 和 Standby NameNode**：两台 NameNode 形成互备，一台处于 Active 状态，为主 NameNode，另外一台处于 Standby 状态，为备 NameNode，只有主 NameNode 才能对外提供读写服务；
2. **ZKFailoverController（主备切换控制器，FC）**：ZKFailoverController 作为独立的进程运行，对 NameNode 的主备切换进行总体控制。ZKFailoverController 能及时检测到 NameNode 的健康状况，在主 NameNode 故障时借助 Zookeeper 实现自动的主备选举和切换（当然 NameNode 目前也支持不依赖于 Zookeeper 的手动主备切换）；
3. **Zookeeper 集群**：为主备切换控制器提供主备选举支持；
4. **共享存储系统**：共享存储系统是实现 NameNode 的高可用最为关键的部分，共享存储系统保存了 NameNode 在运行过程中所产生的 HDFS 的元数据。主 NameNode 和备 NameNode 通过共享存储系统实现元数据同步。在进行主备切换的时候，新的主 NameNode 在**确认元数据完全同步之后才能继续对外提供服务**。
5. **DataNode 节点**：因为主 NameNode 和备 NameNode 需要共享 HDFS 的数据块和 DataNode 之间的映射关系，为了使故障切换能够快速进行，DataNode 会同时向主 NameNode 和备 NameNode 上报数据块的位置信息。

6. HDFS 创建一个文件的流程

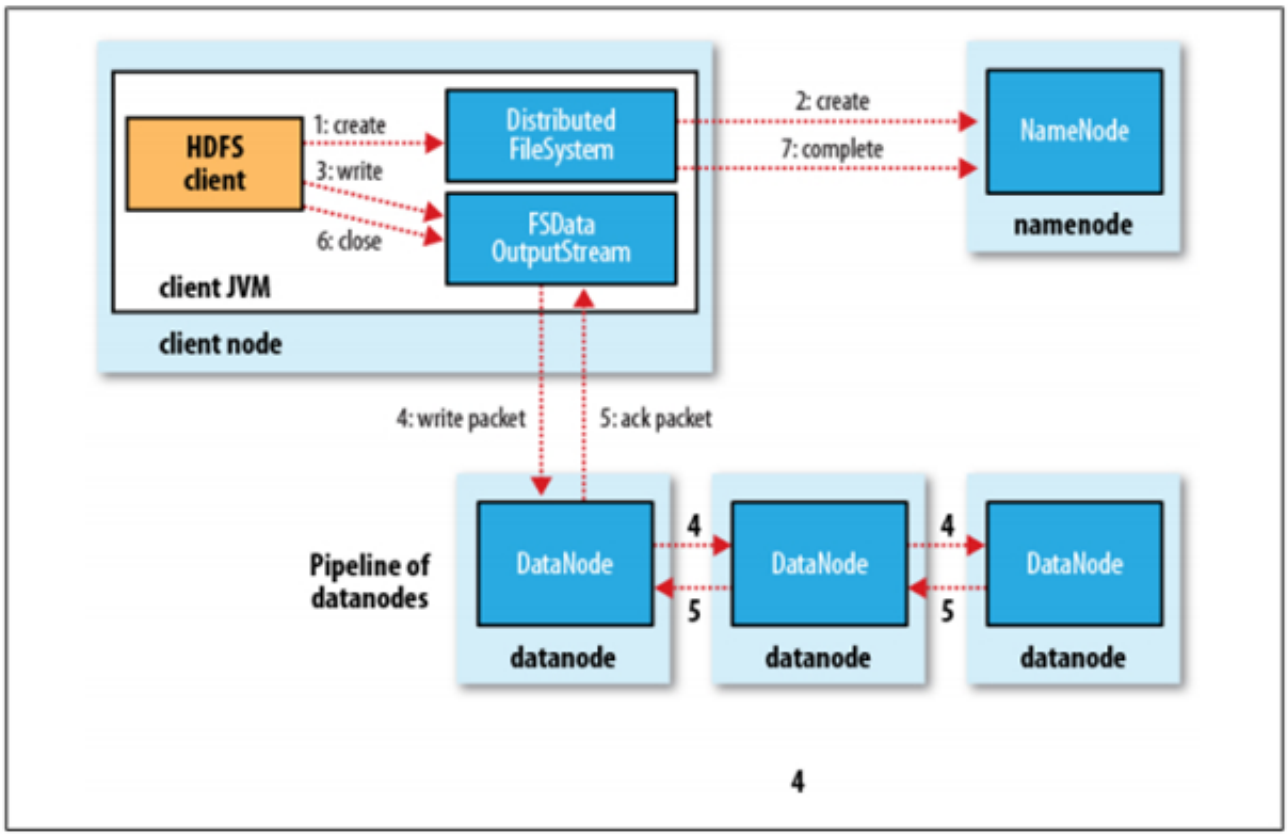
1. 客户端通过ClientProtocol协议向RpcServer发起创建文件的RPC请求。
2. FSNamesystem封装了各种HDFS操作的实现细节，RpcServer调用FSNamesystem中的相关方法以创建目录。
3. 进一步的，FSDirectory封装了各种目录树操作的实现细节，FSNamesystem调用FSDirectory中的相关方法在目录树中创建目标文件，并通过日志系统备份文件系统的修改。
4. 最后，RpcServer将RPC响应返回给客户端。

7. HDFS 读流程



1. Client 通过 DistributedFileSystem 对象与集群的 NameNode 进行一次 RPC 远程调用，获取文件 block 位置信息；
2. NameNode 返回存储的每个块的 DataNode 列表；
3. Client 将连接到列表中最近的 DataNode；
4. Client 开始从 DataNode 并行读取数据；
5. 一旦 Client 获得了所有必须的 block，它就会将这些 block 组合起来形成一个文件。

8. HDFS 写流程



1. Client 调用 DistributedFileSystem 对象的 `create` 方法，创建一个文件输出流（FSDataOutputStream）对象；
2. 通过 DistributedFileSystem 对象与集群的 NameNode 进行一次 RPC 远程调用，在 HDFS 的 Namespace 中创建一个文件条目（Entry），此时该条目没有任何的 Block，NameNode 会返回该数据每个块需要拷贝的 DataNode 地址信息；
3. 通过 FSDataOutputStream 对象，开始向 DataNode 写入数据，数据首先被写入 FSDataOutputStream 对象内部的数据队列中，数据队列由 DataStreamer 使用，它通过选择合适的 DataNode 列表来存储副本，从而要求 NameNode 分配新的 block；
4. DataStreamer 将数据包以流式传输的方式传输到分配的第一个 DataNode 中，该数据流将数据包存储到第一个 DataNode 中并将其转发到第二个 DataNode 中，接着第二个 DataNode 节点会将数据包转发到第三个 DataNode 节点；
5. DataNode 确认数据传输完成，最后由第一个 DataNode 通知 client 数据写入成功；
6. 完成向文件写入数据，Client 在文件输出流（FSDataOutputStream）对象上调用 `close` 方法，完成文件写入；
7. 调用 DistributedFileSystem 对象的 complete 方法，通知 NameNode 文件写入成功，NameNode 会将相关结果记录到 editlog 中。

9. HDFS 架构

1. HDFS 1.0 架构

HDFS 采用的是 Master/Slave 架构，一个 HDFS 集群包含一个单独的 NameNode 和多个 DataNode 节点

NameNode

NameNode 负责管理整个分布式系统的元数据，主要包括：

- 目录树结构；
- 文件到数据库 Block 的映射关系；
- Block 副本及其存储位置等管理数据；
- DataNode 的状态监控，两者通过段时间间隔的心跳来传递管理信息和数据信息，通过这种方式的信息传递，NameNode 可以获知每个 DataNode 保存的 Block 信息、DataNode 的健康状况、命令 DataNode 启动停止等（如果发现某个 DataNode 节点故障，NameNode 会将其负责的 block 在其他 DataNode 上进行备份）。

这些数据保存在内存中，同时在磁盘保存两个元数据管理文件：fsimage 和 editlog。

- fsimage：是内存命名空间元数据在外存的镜像文件；
- editlog：则是各种元数据操作的 write-ahead-log 文件，在体现到内存数据变化前首先会将操作记入 editlog 中，以防止数据丢失。

这两个文件相结合可以构造完整的内存数据。

Secondary NameNode

Secondary NameNode 并不是 NameNode 的热备机，而是定期从 NameNode 拉取 fsimage 和 editlog 文件，并对两个文件进行合并，形成新的 fsimage 文件并传回 NameNode，这样做的目的是减轻 NameNod 的工作压力，本质上 SNN 是一个提供检查点功能服务的服务点。

DataNode

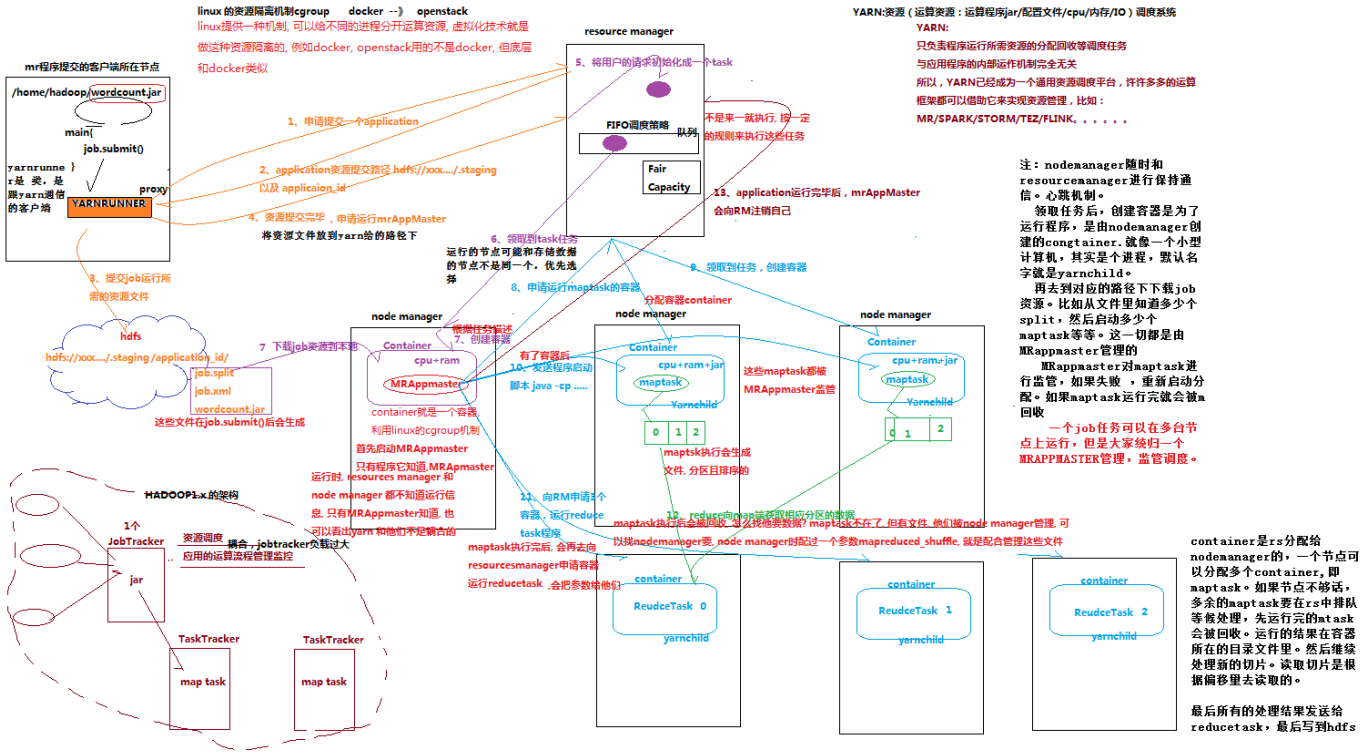
负责数据块的实际存储和读写工作，Block 默认是64MB（HDFS2.0改成了128MB），当客户端上传一个大文件时，HDFS 会自动将其切割成固定大小的 Block，为了保证数据可用性，每个 Block 会以多备份的形式存储，默认是3份。

2. HDFS 2.0 的 HA 实现

- **Active NameNode 和 Standby NameNode**：两台 NameNode 形成互备，一台处于 Active 状态，为主 NameNode，另外一台处于 Standby 状态，为备 NameNode，只有主 NameNode 才能对外提供读写服务；
- **ZKFailoverController**（主备切换控制器，FC）：ZKFailoverController 作为独立的进程运行，对 NameNode 的主备切换进行总体控制。ZKFailoverController 能及时检测到 NameNode 的健康状况，在主 NameNode 故障时借助 Zookeeper 实现自动的主备选举和切换（当然 NameNode 目前也支持不依赖于 Zookeeper 的手动主备切换）；
- **Zookeeper 集群**：为主备切换控制器提供主备选举支持；
- **共享存储系统**：共享存储系统是实现 NameNode 的高可用最为关键的部分，共享存储系统保存了 NameNode 在运行过程中所产生的 HDFS 的元数据。主 NameNode 和备 NameNode 通过共享存储系统实现元数据同步。在进行主备切换的时候，新的主 NameNode 在**确认元数据完全同步之后才能继续对外提供服务**。
- **DataNode 节点**：因为主 NameNode 和备 NameNode 需要共享 HDFS 的数据块和 DataNode 之间的映射

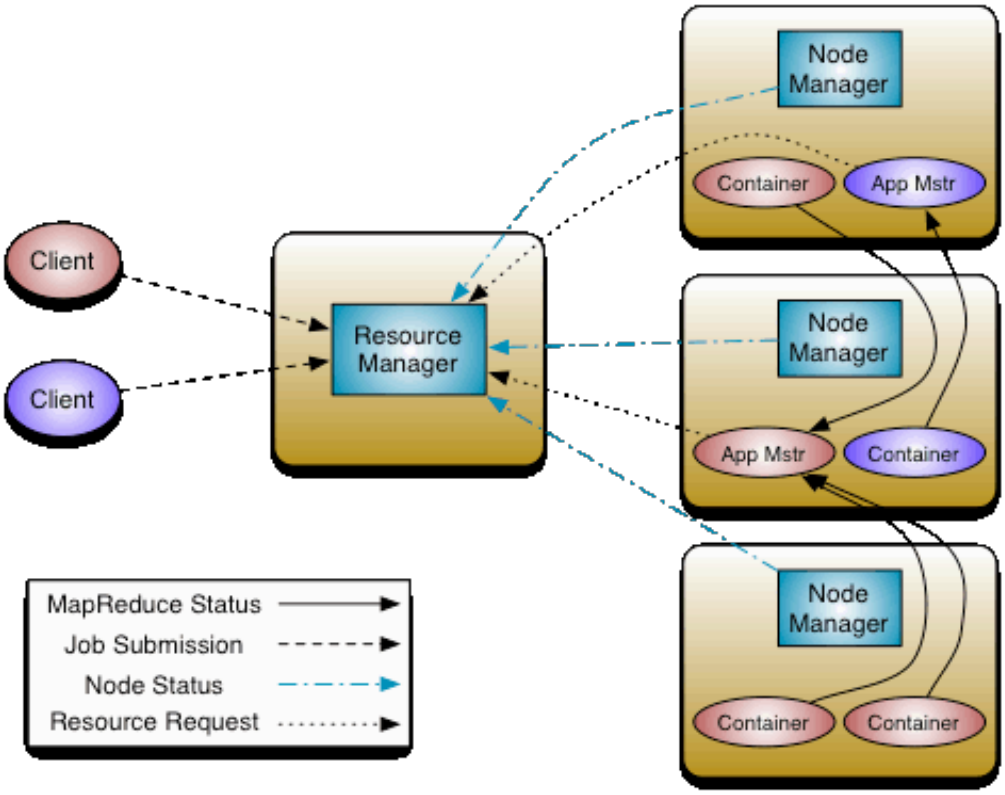
关系，为了使故障切换能够快速进行，DataNode 会同时向主 NameNode 和备 NameNode 上报数据块的位置信息。

10. Yarn 调度MapReduce过程



1. Mr程序提交到客户端所在的节点（MapReduce）
2. yarnrunner向Resourcemanager申请一个application。
3. rm将该应用程序的资源路径返回给yarnrunner
4. 该程序将运行所需资源提交到HDFS上
5. 程序资源提交完毕后，申请运行mrAppMaster
6. RM将用户的请求初始化成一个task
7. 其中一个NodeManager领取到task任务。
8. 该NodeManager创建容器Container，并产生MRAppmaster
9. Container从HDFS上拷贝资源到本地
10. MRAppmaster向RM申请运行maptask容器
11. RM将运行maptask任务分配给另外两个NodeManager，另两个NodeManager分别领取任务并创建容器.
12. MR向两个接收到任务的NodeManager发送程序启动脚本，这两个NodeManager分别启动maptask，maptask对数据分区排序。
13. MRAppmaster向RM申请2个容器，运行reduce task。
14. reduce task向maptask获取相应分区的数据。
15. 程序运行完毕后，MR会向RM注销自己。

11.Yarn架构



1. ResourceManager（RM）

RM 是一个全局的资源管理器，负责整个系统的资源管理和分配，它主要有两个组件构成：

1. 调度器：Scheduler；
2. 应用程序管理器：Applications Manager，ASM。

调度器

调度器根据容量、队列等限制条件（如某个队列分配一定的资源，最多执行一定数量的作业等），将系统中的资源分配给各个正在运行的应用程序。要注意的是，该调度器是一个纯调度器，它不再从事任何与应用程序有关的工作，比如不负责重新启动（因应用程序失败或者硬件故障导致的失败），这些均交由应用程序相关的 ApplicationMaster 完成。调度器仅根据各个应用程序的资源需求进行资源分配，而资源分配单位用一个抽象概念**资源容器(Resource Container，也即 Container)**，Container 是一个动态资源分配单位，它将内存、CPU、磁盘、网络等资源封装在一起，从而限定每个任务使用的资源量。此外，该调度器是一个可插拔的组件，用户可根据自己的需求设计新的调度器，YARN 提供了多种直接可用的调度器，比如 Fair Scheduler 和 Capacity Schedule 等。

应用程序管理器

应用程序管理器负责管理整个系统中所有应用程序，包括应用程序提交、与调度器协商资源以 AM、监控 AM 运行状态并在失败时重新启动它等。

2. NodeManager（NM）

NM 是每个节点上运行的资源和任务管理器，一方面，它会定时向 RM 汇报本节点上的资源使用情况和各个 Container 的运行状态；另一方面，它接收并处理来自 AM 的 Container 启动/停止等各种请求。

3. ApplicationMaster（AM）

提交的每个作业都会包含一个 AM，主要功能包括：

1. 与 RM 协商以获取资源（用 container 表示）；
2. 将得到的任务进一步分配给内部的任务；
3. 与 NM 通信以启动/停止任务；
4. 监控所有任务的运行状态，当任务有失败时，重新为任务申请资源并重启任务。

MapReduce 就是原生支持 ON YARN 的一种框架，可以在 YARN 上运行 MapReduce 作业。有很多分布式应用都开发了对应的应用程序框架，用于在 YARN 上运行任务，例如 Spark，Storm、Flink 等。

4. Container

Container 是 YARN 中的资源抽象，它封装了某个节点上的多维度资源，如内存、CPU、磁盘、网络等，当 AM 向 RM 申请资源时，RM 为 AM 返回的资源便是用 Container 表示的。 YARN 会为每个任务分配一个 Container 且该任务只能使用该 Container 中描述的资源。

注：资料来源于网络。