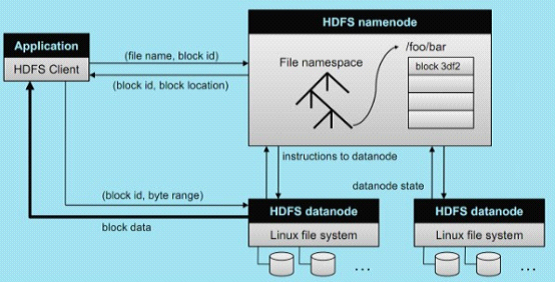
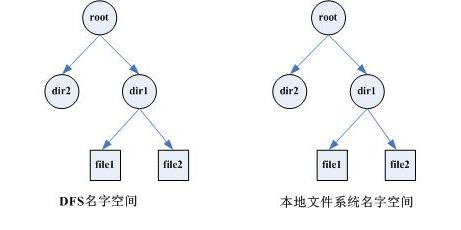
HDFS Federation

HDFS是主从体系架构的分布式系统，集群中拥有NameNode和一些DataNode，用户可以通过HDFS客户端同NameNode和DataNode交互来访问文件系统。 NameNode是Master节点，负责管理文件系统的命名空间（namespace），以及数据块到具体DN节点的映射等信息。DN负责管理它所在节点上的存储。从内部看，一个文件被分成一个或者多个数据块，这些块存储在一组DN上，DN会以本地文件形式存储这些数据块以及数据块的校验信息。如下图：



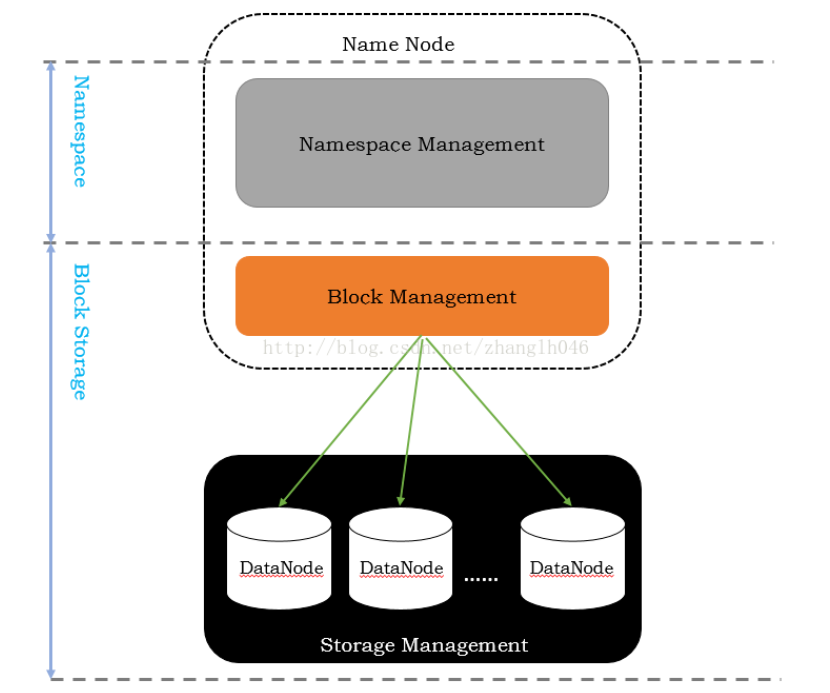
# HDFS命名空间管理

文件系统的命名空间即文件系统文件目录的组织方式，是文件系统的重要组成部分，为用户提供可视化、可理解的文件系统视图。HDFS的文件系统命名空间和传统文件系统结构类似，由目录和文件和目录组成，支持创建、删除、移动、重命名文件和目录等功能：



分布式文件系统的每一个目录或文件，在元数据服务器的本地文件系统之上对应创建元目录和元文件，分别用于保存目录属性及文件属性（访问控制信息、数据分片信息及存储位置等）信息。HDFS还支持用户配额（users quotas）和权限控制。

在HDFS中NameNode管理着整个文件系统的元数据，从架构设计上看元数据大致分成两个层次：NameSpace管理层，负责管理文件系统的树状目录结构以及文件与数据块的映射关系；块管理层，负责管理文件系统中文件物理块与实际存储位置关系的BlocksMap。如下图：



块管理层包括块管理和物理存储：块管理层获取DN向NN的注册请求及周期性心跳信息，处理DN汇报的Block信息及维护其位置信息、处理块相关的操作（创建、删除及修改等）及块的副本管理；物理存储就是DN将Block存储到本地文件系统中，负责其数据的读写。

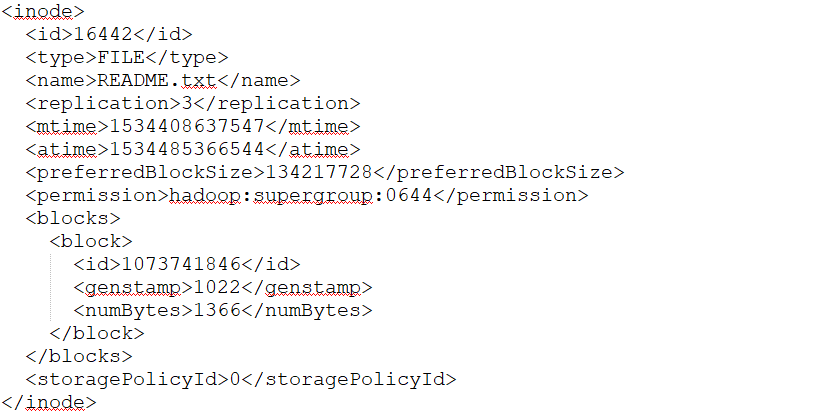
NameNode负责文件系统层次和块管理层的块管理模块，其核心数据结构为FsDirectory和FsNameSystem：

1. FsDirectory，文件系统层次结构

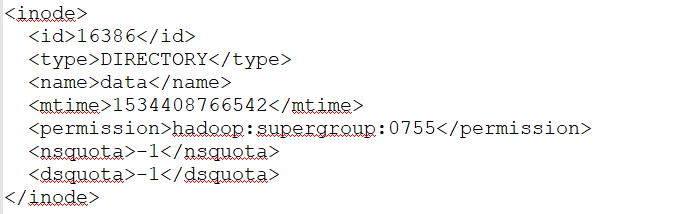
存储了整个文件系统的目录状态，目录结构的管理通过FSImage和FSEditLog的方法从本地磁盘读取和写入元数据信息，并记录对目录结构的操作日志。

FSDirectory保存了文件名和数据块的映射管理：核心的数据类为INode（目录结构中一个节点的抽象），INodeFile和INodeDirectory均继承INode类，分别表示文件节点和目录节点，以目录/data/README.txt为例：

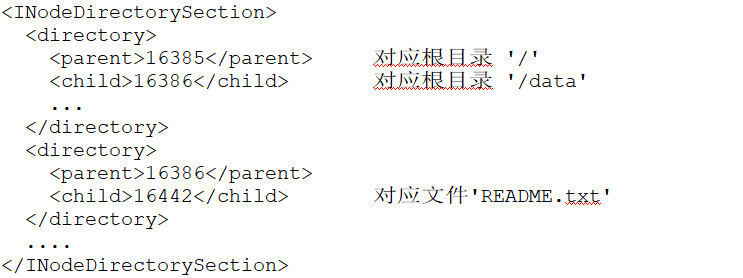
* INodeFile类中最重要的数据结构为blocks: BlockInfo[]，记录文件包含了所有的Block，README.txt文件的inode其描述如下：



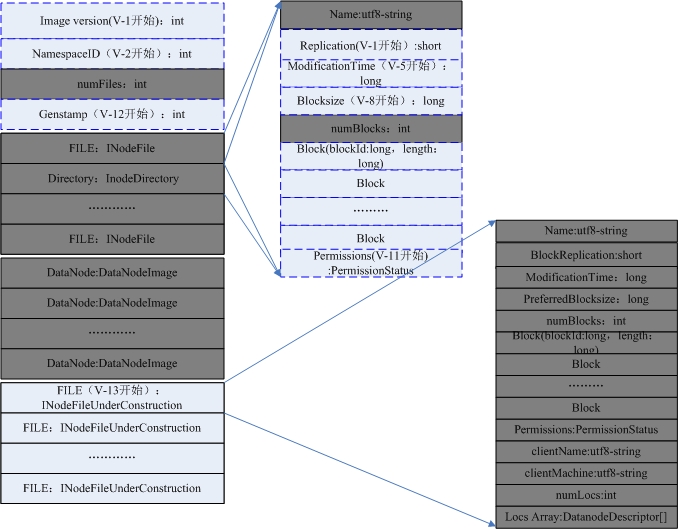
目录也是INode的一种，/data的其描述如下：



* INodeDirectory，关键数据结构是children:List<INode>，记录了目录中所有子节点信息，描述了文件系统的目录层次关系，/data/README.txt的描述如下：

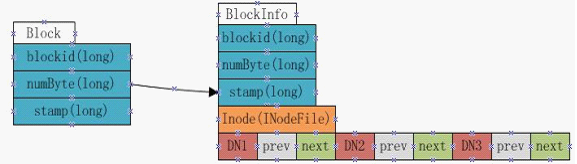


其中inode id为16385的directory为根目录。下图是FsImage文件与FsDirectory的逻辑框图



1. BlocksMap-BlockManagement

根据FsDirectory的分析，在fsimage中仅是存储了目录的层次结构，并没有持久化存储每个block存储的Datanodes列表。DN启动时会将本节点上保持的block信息汇报给NN，NN将接受到的块信息保存到内存中，其通过数据结构BlocksMap来存储，对BlocksMap的维护通过FsNameSystem#BlockManager来实现，即上图中的Block Management的功能，其逻辑描述如下：



在BlockInfo中，将block所属的DN列表保存到Object[]数组中，不仅保存了DN列表，还包含额外的信息：

IMG_256

三副本的DN构成了列表。

1. Storage Management

DataNode存储实际的数据块，Block都对应一个数据存储文件，数据存储文件以\_blk开头，同时block还对应一个元数据文件（以.meta），ReadME.txt文件的存储的目录结构如下所示：

├── blk\_1073741846

├── blk\_1073741846\_1022.meta

├── blk\_1073741851

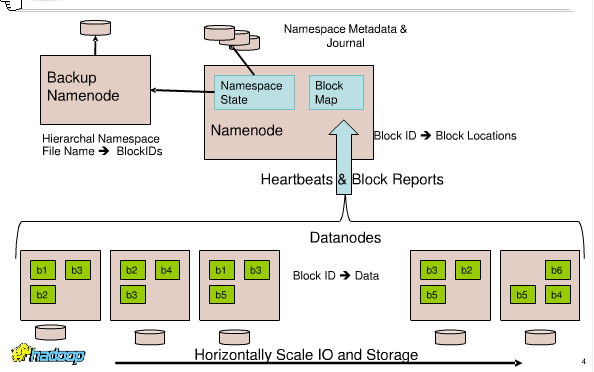
├── blk\_1073741851\_1027.meta

......

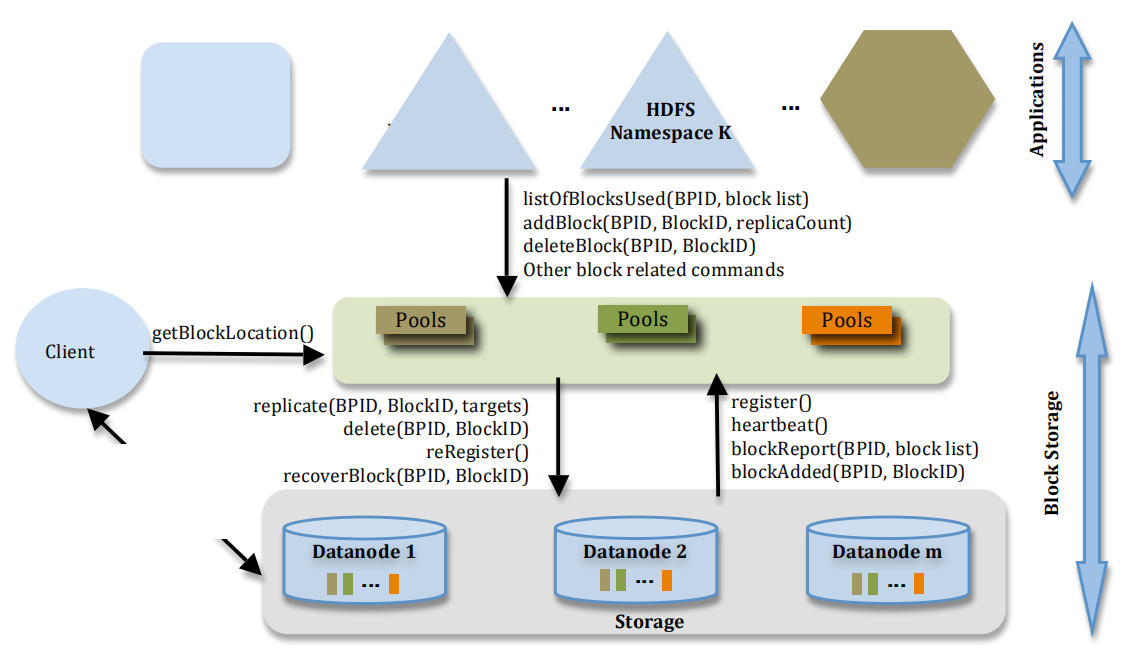
README.txt的数据的blockID为1073741846，对应磁盘上的文件为blk\_1073741846。

DN启动时遍历数据目录，得到所有block文件列表，并添加所有block的记录到FsDataSet的volumeMap属性中，这个对象保存block与DatanodeBlockInfo的映射关系，便于通过block查询具体的文件信息。

1. HDFS的Block管理架构



对于Federation HDFS来说，Block的管理架构图如下所示：



# **HDFS Federation**

随着集群的规模及存储需求的增长，单个NameNode已经不能满足整个集群的运行，其限制因素包括以下：

1）Namespace的限制

在HDFS中元数据存储在内存中，因此单个NameNode所存储的对象（文件+block）数目受到JVM的heap Size的限制。根据估算50GB的heap能够存储20亿个对象（文件平均大小40MB），大致4000个DN、12PB的存储，（每存储100万文件大致消耗1GB内存）。但是随着存储需求的增长，单个DN的磁盘容量从4TB增长到72TB，集群的规模也增加到数千节点，单个NameNode已经不能支撑整个集群的运行

1. 性能的瓶颈

在单Namenode的HDFS架构中，整个HDFS文件系统的吞吐量受限于单个NN的吞吐量，单NN当前最多支持60K的task，但是在YARN中将支持多余100K的并发任务。

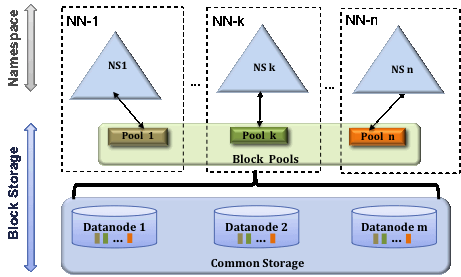
1. 扩展性

HDFS的底层存储时水平扩展（底层为Datanode，当集群存储空间不够时，可简单的添加机器进行水平扩展），但是NameSpace管理了整个分布式文件系统的元数据信息，限制了集群中数据块、文件和目录的数目

1. 隔离型

大部分Hadoop集群是共享的，不同的用户提交作业，单NN难以提供隔离型

通过纵向扩展NN(提高内存容量，例如从50GB增加到120GB)，这带来的问题：启动时间过长（50GB Heap的NN）启动大概需要30分钟到2小时；NN在Full GC时如果发生错误将影响整个集群；大JVM Heap的调试比较困难，优化NN的内存使用性价比较低。因此在Hadoop 2.6版本以后，通过HDFS Federation提高HDFS性能，其原理结构图如下所示：



HDFS Federation多个独立的NameSpace使得命名空间服务能够水平扩展，NameNode之间是联盟关系，相互独立并且不需要相互协调。HDFS Federation共用一套DataNode，DN维护了多个Block Pool存储块。

1. Block Pool，所谓Block Pool就是属于单个命名空间的一组block块。其是一个重新将block划分的逻辑概念，在集群中划分两个namespace后，对应的目录如下：

*BP-1710352838-10.139.4.98-1534393032868*

*BP-301621019-10.139.4.82-1534393187641*

当DN与NameNode开始会话后自动建立Block Pool。为了支持HDFS Federation，对Block进行扩展，增加了poolId字段。

1. DN中的block Map，维护在BlockPoolManager中

在DN的数据结构都是通过poolID进行索引，即dn中的BlockMap、Storage等都通过BPID索引，数据结构如下所示：

*private final Map<String, FoldedTreeSet<ReplicaInfo>> map = new HashMap<>();*

1. BPOfferService

DN为每一个NN创建一个BPOfferService，在BlockPoolManager中维护的数据结构如下：

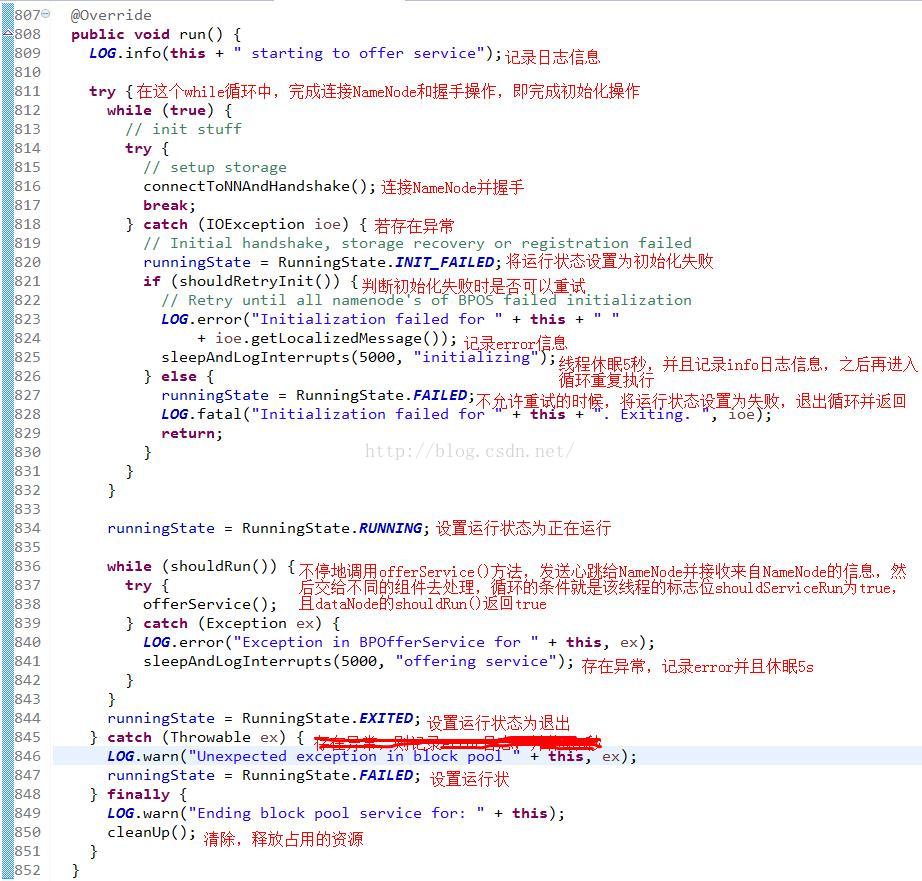
*private final Map<String, BPOfferService> bpByBlockPoolId = Maps.newHashMap();*

在BPOfferService中启动BPServiceActor线程，与对应的NN进行心跳交互，

BPOfferService在实例化时初始化bpID和NameSpaceInfo，因此只与对应的NN进行心跳

*NamespaceInfo bpNSInfo;  
volatile DatanodeRegistration bpRegistration;  
private final String nameserviceId;  
private volatile String bpId;*

1. BPServiceActor的执行



其中核心方法是offserService，发送心跳给NameNode并接收来自NN的命令，发送心跳信息中包含block信息，sendHeartbeat的执行如下：

*HeartbeatResponse sendHeartBeat(boolean requestBlockReportLease)*

*throws IOException {*

*scheduler.scheduleNextHeartbeat();*

*StorageReport[] reports =*

*dn.getFSDataset().getStorageReports(bpos.getBlockPoolId());*

*//根据block poolId获取resport*

*HeartbeatResponse response = bpNamenode.sendHeartbeat(bpRegistration,*

*reports,*

*dn.getFSDataset().getCacheCapacity(),*

*dn.getFSDataset().getCacheUsed(),*

*dn.getXmitsInProgress(),*

*dn.getXceiverCount(),*

*numFailedVolumes,*

*volumeFailureSummary,*

*requestBlockReportLease,*

*slowPeers,*

*slowDisks);*

*.....}*

1. BlockPoolSlice，在DataNode中通过FsVolumeImpl管理数据，其通过BlockPoolSize来管理

*private final String bpid;  
private final FsVolumeImpl volume; // volume to which this BlockPool belongs to  
private final File currentDir; // StorageDirectory/current/bpid/current  
// directory where finalized replicas are*

在FsVolumeImpl中维护bpId->BlockPoolSlize

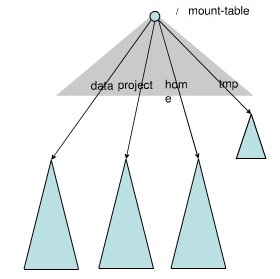
*private final Map<String, BlockPoolSlice> bpSlices  
 = new ConcurrentHashMap<String, BlockPoolSlice>();*

https://blog.csdn.net/gongxifacai\_believe/article/details/80008370

<https://issues.apache.org/jira/browse/HDFS-1052>

# **Federation多命名空间的管理**

使用HDFS Federation，集群中有多个命名空间，需要解决命名空间中数据的共享和访问的问题。使用全局唯一的命名空间是解决数据共享和访问的一种方法，其中Mount Table是常用做法，如下图所示：

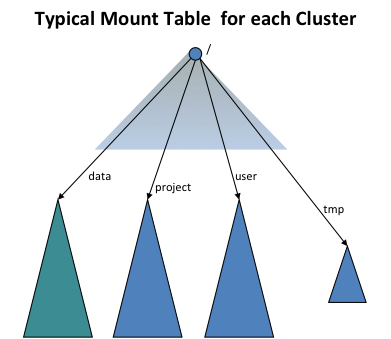


下面四个三角形代表独立的命名空间，上方浅色的三角形代表从客户端角度去访问的子命名空间。各个深色的命名空间Mount到浅色的表中，可以通过访问不同的挂载点来访问不同的命名空间，如同Linux系统访问不同的挂载点。

HDFS Fedeation中命名空间的管理也是将各个命名空间挂载到全局的mount-table，将数据全局共享，命名空间挂载到客户端的Mount-table中形成应用程序可见的命名空间视图。

## ViewFS

View FileSystem提供了管理多个Hadoop文件系统命名空间的方式，其提供客户端挂载表，借鉴Linux提供client-side mount table，其通过一层新的文件系统viewfs实现，实际上提供一种映射关系，将一个全局（逻辑）目录映射到某个具体的NN(物理)目录上：



在core-site.xml中的配置示例如下：

*<xi:include href="mount-table.xml" />*

*<property>*

*<name>fs.defaultFS</name>*

*<value>viewfs://brqviewfs</value>*

*<final>true</final>*

*</property>*

其中brqviewfs是整个HDFS集群的名称，mount-table.xml配置了全局（逻辑）目录与具体NN(物理)目录的映射关系，可以类比Linux挂载点。例如有两个NN，分别是brqrzt和brqrzt1，其中brqrzt管理/data目录，brqrzt1管理/project,/user和/tmp目录，则mount-table.xml的配置内容如下：

*<?xml version="1.0"?>*

*<configuration xmlns:xi="http://www.w3.org/2001/XInclude">*

*<property>*

*<name>fs.viewfs.mounttable.brqviewfs.link./data</name>*

*<value>hdfs://brqrzt/data</value>*

*</property>*

*<property>*

*<name>fs.viewfs.mounttable.brqviewfs.link./user</name>*

*<value>hdfs://brqrzt1/user</value>*

*</property>*

*<property>*

*<name>fs.viewfs.mounttable.brqviewfs.link./project</name>*

*<value>hdfs://brqrzt1/project</value>*

*</property>*

*<property>*

*<name>fs.viewfs.mounttable.brqviewfs.link./tmp</name>*

*<value>hdfs://brqrzt1/tmp</value>*

*</property>*

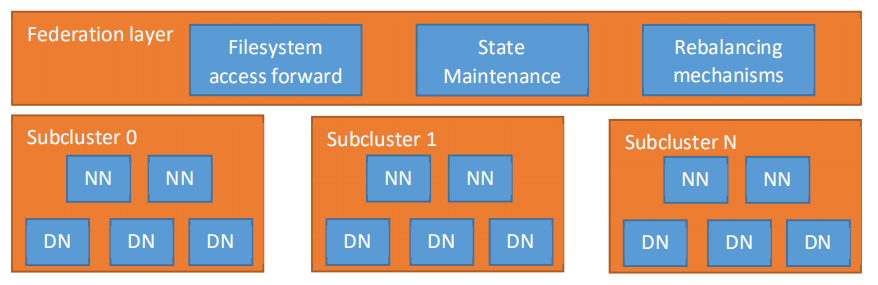
*</configuration>*

经过上面的配置后，可以向访问单NN一样访问访问Federation模式下的HDFS文件。

## **RBF(RouteBasedFederation)**

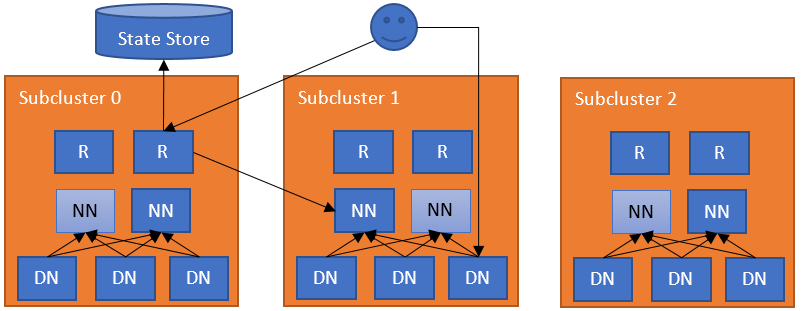
Hadoop 3.0以后添加了Router-Based Federation，添加了RPC路由层，提供了多个HDFS命名空间的联合视图，与ViewFs功能类似，不同之处在于Mount table由服务器端的路由层维护，这简化了客户端对Federated Cluster的访问。

RBF底层由多个namespace的子集群组成，可以是独立的hdfs集群、Federation集群或者混合集群。通常在每个NameNode节点上部署Router，向客户端提供NameNode接口的服务，Router是无状态的，其通过StateStore获取Federation状态信息，类似于VeiwFS中的MountTable。RBF为多个子集群提供了Federation Layer，如下图所示：



用户通过该Layer，可以透明访问任意的子集群，并且支持跨子集群的数据均衡，为了实现以上功能，RBF提供以下模块：

* Filesystem access Forward，在Router Component中提供和NameNode相同的接口，并根据State Store中的信息将客户端请求提交给相应的相应的子集群
* State Maintenance，State Store类似于ViewFS维护了Mount Table，及子集群的负载信息
* Rebalancing Mechanisms，在每个子集群中可以启动多个Routers，如下所示：



可以通过负载均衡降低Router的负载。

DFS客户端通过Router访问Federated FileSystem中的文件，Router从Route Table中获取哪个子集群有文件，从Membership table中获取该子集群的NameNode节点。Router将请求转发给对应的NameNode，并给客户端返回响应信息，即文件所在的DN，最后客户端直接和DN交互获取数据。

Router的配置样例如下：

*<property>*

*<name>dfs.nameservices</name>*

*<value>ns1,ns2,ns-fed</value>*

*</property>*

*<property>*

*<name>dfs.ha.namenodes.ns-fed</name>*

*<value>r1,r2</value>*

*</property>*

*<property>*

*<name>dfs.namenode.rpc-address.ns-fed.r1</name>*

*<value>fys1.cmss.com:8888</value>*

*</property>*

*<property>*

*<name>dfs.namenode.rpc-address.ns-fed.r2</name>*

*<value>cmhhost1.novalocal:8888</value>*

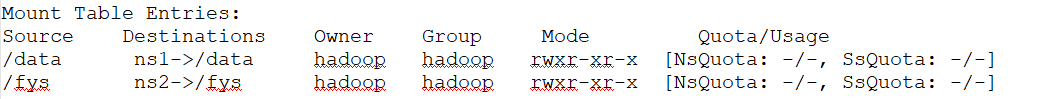
*</property>*

将目录挂载到Router中，信息会保存到StateStore中，执行命令如下：

*[hdfs]$ $HADOOP\_HOME/bin/hdfs dfsrouteradmin -add /data ns1 /data*

*[hdfs]$ $HADOOP\_HOME/bin/hdfs dfsrouteradmin -add /fys ns2 /fys*

*[hdfs]$ $HADOOP\_HOME/bin/hdfs dfsrouteradmin -ls*



默认情况下使用Zookeeper作为StateStore，实现类为StateStoreZookeeperImpl，在zk中的State Store的保存信息如下：

*[zk: localhost:2181(CONNECTED) 1] ls /hdfs-federation*

*[MountTable, MembershipState, RouterState]*

*[zk: localhost:2181(CONNECTED) 2] ls /hdfs-federation/MountTable*

*[0SLASH0data, 0SLASH0fys]*

*[zk: localhost:2181(CONNECTED) 3] ls /hdfs-federation/MembershipState*

*[null-ns2-cmhhost1.novalocal\_8888,null-ns1-fys1.cmss.com\_8888,]*

*[zk: localhost:2181(CONNECTED) 5] ls /hdfs-federation/RouterState*

*[cmhhost1.novalocal\_8888,fys1.cmss.com\_8888]*

https://issues.apache.org/jira/browse/HDFS-10467