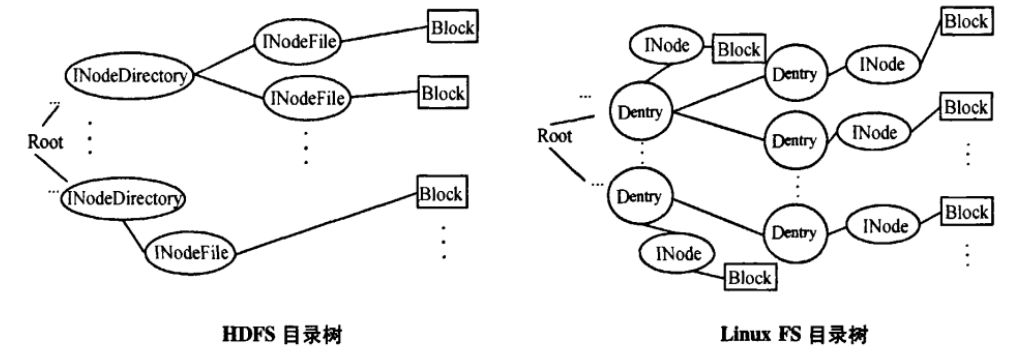
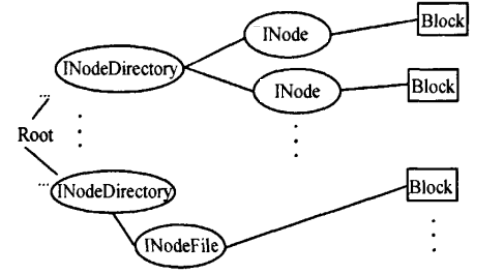
Alluxio文件组织

Linux文件系统或者Hadoop分布式文件系统都选择”树”结构来组织结构（目录树），文件存储在树叶中，其余节点都是目录，如下图所示：



在这两种文件系统中Block是文件读写的最小单元，每个文件或者目录都是由一个INode描述，在INode中定义了一组外存上的Block。文件被切分成Block后，存放到节点磁盘上，在管理节点上记录了数据块映射（BlocksMap），映射为<Block,BlockInfo>，在BlockInfo中封装了Block相关的INode和DataNode信息。目录树结构记录在管理节点内存中，在HDFS中该结构描述为FSDirectory。

在Alluxio中采用类似的树状文件组织方式，文件数据按Block来组织，元数据组织为InodeTree，文件和目录都描述为Inode，其包括基本信息（id、名称、长度等）、Block信息及备份信息等。文件及块信息保存在Alluxio Master中，块数据保存在Worker中，目录树如下所示：



其结构与Hadoop文件系统基本保存一致。

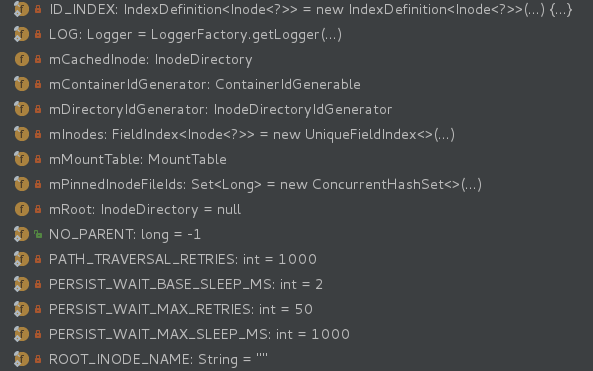
# 文件系统主要数据结构

Alluxio文件系统目录和文件在内存中以一棵树的形式存储的，该树由Alluxio Master维护，Alluxio Master会修改这个树形结构来对外提供文件添加、删除等操作功能。在文件系统目录树的节点还保存了文件与数据块的对应关系，由Alluxio Master维护。

在Alluxio FileSystem中两个重要的功能就是维护文件系统的目录树，通过InodeTree来管理，在该目录树中不管是文件还是目录都被看做是INode节点，如果是目录对应的类为InodeDirectory，文件对应的类为InodeFile，InodeDirectory以及InodeFile的类都是Inode的派生类。在InodeDirectory中包含了一个成员集合变量mChildren，如果该目录下有子目录或者文件，其子目录或者文件的INode引用会被保持到mChildren集合中，Alluxio就是通过这种方式来维护整个文件系统的目录结构。

Alluxio将文件系统目录树保持到本地工作目录下的”\_format\_”的文件中，使用这个文件AlluxioMaster每次重启都是将整个Alluxio文件结构重构，该文件的操作由JournalFormatter类负责。另外对Alluxio文件系统的操作日志都会被记录在Journal Log中，以便周期性的将该日志与镜像文件进行合并形成新的\_fsimage\_，该log文件也会在本地文件系统中保存，命名方式为：journal\_folder/version/logs/StartSequenceNumber-EndSequenceNumber，举例如下：*0x0-0x27*。文件的相关操作由JournalReader|Writer类管理。

如下图，InodeTree维护这文件系统目录树的根节点（这个根节点是整个文件系统的root，是一个InodeDiretroy类型），下面以INode为分析入口进行详细分析：



1）INode类

INode类似于Linux的inode，即索引节点，索引节点保持文件的元信息，如文件类型、权限和以字节相关的文件长度等，在Alluxio文件系统目录树中，不管是目录还是文件，都是INode类的引用。INode是一个抽象类，是InodeDirectory和InodeFile的父类，如果是父类则其实际的类为InodeDirectory，是文件则为InodeFile。在InodeDirectory中包含了一个成员变量children，如果该目录下有子目录或者文件，则引用就会保存在children集合中。INode相关的所有类，如下图所示：



Inode类是整个Inode体系的根接口，它是一个抽象类，保存了Alluxio目录和文件所有的共同属性，包含当前节点的父节点的INode对象的id号、文件名、用户组、访问权限、修改时间等。

1）InodeDirectory

InodeDirectory抽象Alluxio文件系统中的目录，目录是文件系统中的一个虚拟容器，里面保存了一组文件和其他目录。在InodeDirectory的实现中，添加了成员变量mChildren用来保存目录中所有子目录项INode对象

*private final FieldIndex<Inode<?>> mChildren = new UniqueFieldIndex<>(NAME\_INDEX);*

InodeDirectory作为目录容器，最主要的功能就是维护目录中保存的文件以及子目录，也就是维护children字段，InodeDirectory提供了创建子目录项、删除子目录项以及查询子目录项等方法，这些方法的底层实现都是操作children字段。

3）InodeFile

在文件系统目录树中，实现InodeFile抽象一个文件，保存了文件重要的信息：文件对应的数据块信息mBlocks:List<Long>。

4）Journal

文件系统目录树结构周期性的保存在本地工作目录，其执行核心类为Journal，类结构如下所示：



AsyncJournalWriter和UfsJournalLogWriter分别将文件目录树进行持久化及操作Log。

http://edu.dataguru.cn/thread-129071-1-1.htm

# 数据块管理

文件系统中另外一个重要部分是维护数据块和数据节点的对应关系，即指定数据块的副保存在哪些数据节点上的信息。InodeFile.mBlocks字段记录了一个文件中的所有数据块，该字段是List<long>类型数组，记录了Block的id号。

在Alluxio中Block的类关系如下图所示：



Alluxio Worker和Master之间通过心跳同步Block信息，在Worker中心跳的实现类为BlockMasterSync，核心方法heartbeat，代码如下：

*public void heartbeat() {*

*//获取本worker的block信息，BlockHeartbeatReporter中主要的数据是mAddedBlocks和mRemovedBlocks*

*BlockHeartbeatReport blockReport = mBlockWorker.getReport();*

*BlockStoreMeta storeMeta = mBlockWorker.getStoreMeta();*

*//向Master汇报新桃*

*Command cmdFromMaster = null;*

*try {*

*cmdFromMaster = mMasterClient*

*.heartbeat(mWorkerId.get(), storeMeta.getUsedBytesOnTiers(),*

*blockReport.getRemovedBlocks(), blockReport.getAddedBlocks());*

*handleMasterCommand(cmdFromMaster);*

*mLastSuccessfulHeartbeatMs = System.currentTimeMillis();*

*} .....*

心跳信息传给Master后，有BlockMasterWorkerServiceHandler来进行处理，具体过程不再详述。

https://github.com/gjhkael/Alluxio-Internal/blob/master/markdown/4-AlluxioBlockWrite.md