Hadoop snapshot

HDFS Snapshot是某一时间点的文件系统的只读备份。可以对HDFS全文件系统或者文件系统的子树进行快照。快照用于数据备份，解决用户的误操作及灾难恢复的问题。一个快照是当前系统状态的一个备份，系统发生故障时，利用这个快照将系统恢复到快照时的样子。

HDFS Snapshot的实现是高效的：

1、Snapshot的创建是瞬间完成：除了节点遍历耗时之外，其创建的开销为O(1)。

    2、Snapshot后数据发生更改时，将会消耗额外的内存(记录修改操作)，内存使用量为O(M)，其中M为更改的文件或者目录的个数。

    3、Datanodes上的Blocks并不会发生Copy：snapshot文件记录了有关的block列表和文件大小，没有实际数据的Copy。(并不是计算机科学中的Full snapshot)

    4、Snapshot不会影响正常的HDFS操作：修改操作按照时间倒叙的方式记录，因此当前数据可以直接访问。snapshot数据由当前数据减去已经修改的部分。snapshot Data = current data - modification。访问数据时会经过snapshot判定(可以认为snapshot在内存中维护了一份目录子树，Namenodes)，对于修改记录中没有涉及到的文件或者目录将可以直接访问，否则从snapshot子树中获取。

# 一、Snapshottable目录和路径

通过命令hdfs dfsadmin -allowSnapshot <Path>，将一个目录设置为snapshottable。对于一个snapshotable的目录，路径部分.snapshot用来访问其snapshots。

例如/foo是一个snapshottalbe目录，/foo/.snapshot/bar即是/foo/bar的snapshot副本(在hdfs中，snapshot是一个保留字，用户不能创建.snapshot名称的目录或者文件)。

Snapshot可以在任何Snapshottable目录上进行，一个snapshottable目录可以容忍65536个snapshots,文件系统对snapshottable目录个数没有上限。管理员可以将任何目录设置为snapshottable，如果一个snapshottalbe目录下存在snapshots数据，这个目录不能被删除或者重命名，直到所有的snapshots被删除。

# 二、snapshot操作

1) 开启snapshot: 将一个目录设置为snapshottable

hdfs dfsadmin -allowSnapshot <path>

2)关闭snapshots:此目录下的所有snapshots需要首先删除，才能执行disallow指令

hdfs dfsadmin -disallowSnapshot <path>

3)创建snapshot，对一个snapshottable目录创建snapshot

hdfs dfs -createSnapshot <Path> [snapshotName]

4) 删除snapshot,从一个snapshottable目录下删除snapshots

hdfs dfs -deleteSnapshot <path> <snapshotName>

5) 重命名snapshots,修改snapshot名称

hdfs dfs -renameSnapshot <path> <oldName> <newName>

6) 列举所有的snapshottable目录

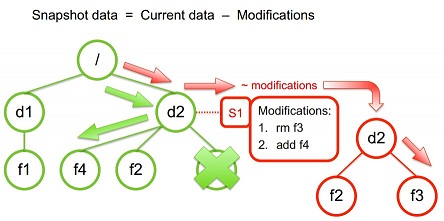
hdfs lsSnapshottalbleDir

7)获取 snapShots差别报告，获取两个snapshots的不同

# 三、snapShot源码分析

## 3.1 原理

上述的snapshot操作，均可以通过FileSystem API操作。Snapshots并不会创建实际数据的copy，不过这些Snapshots以及snapshot之后变更的操作将会添加到Namenode的metadata中。对于datanodes并不知道snapshots的任何细节，snapshots操作不会和Datanode交互，即一个文件、目录是否有snapshots，Datanodes并不需要知道，创建snapshot也不需要Datanodes做任何额外的操作。不过文件的修改擦操作均会有Namenode判定，当一个目录有snapshot时不允许删除。下图是Snapshot的操作图：



## 3.2 源码分析

snapShot本质上讲是对Namenode中的元数据进行快照，而不是对datanode中的数据进行管理。其中维护着数据的变化。

### 3.2.1 Snapshot元数据

#### 3.2.1.1 DirectoryWithSnapshotFeature

该类维护着Snapshot 变化的列表，用于存储snapshot数据，当目录变化时，Old data存储在最新的snapshot中。

在该类中定义了三个内部类，分别是ChildrenDiff,DirectoryDiff,DirectoryDiffList，

1) ChildrenDiff

该类用于表示INodeDirectory中当前和原来Snapshot中ChildrenList的变化，该类继承Diff<K,E extends Diff.Element<K>>

Diff的Element用(,)->current来表示，其中：

(0,0): 元素i不在当前的状态

(c,0): 创建i，将其增加到c-list

(0,0):在current中创建i,并删除

(c2,0):在current中创建,并在c-list中覆盖c。其余的操作涉及到增加、删除及修改等

K表示key type，<E>用于表示element type

在类Diff中维护了List<E> created,List<E> deleted两个链表

在类ChildrenDiff中，方法：

boolean replace(ListType type,INode oldChild,final INode newChild):在Diff的created/deleted list中覆盖给定的childNode，如果没有找到child，则返回错误值。

boolean removeChild(ListType,INode)，从列表中移除Inode

destroyDeletedList：清空delete列表

writeCreated(DataOutput out):将created列表写入文件

writeDeleted(DataOutput out)

void getDirsInDeleted(List<INodeDirectory> dirList): 在deleted List中获取INodeDirectory列表

List<DiffReportEntry> generateReport(byte[][] parentPath,parent,fromEarlier)：获取变化的列表

2) DirectoryDiff

该类用于表示两个snapshots当中INodeDirectory的不同，继承了类AbstractINodeDiff

AbstractINodeDiff用于表示snapshots当中Inode的变化，维护snapshot变化的列表，<N extends INode,A extends INodeAttributeds,D extends AbstractINodeDiff>

成员变量：

Snapshot snapshot;当diff被应用时，这个变量会被获取

A snapshotINode; 一个snapshot inode data，当没有变化时，这个值是空的。

D posteriorDiff;

DirecotryDiff的成员变量：

ChildDiff diff; children列表的变化

int childrenSize; snapshot创建时的children 列表变化

方法：

FSImage 在加载的时候会使用到DirectoryDiff的构造器

3) DirectoryDiffList(DirectoryDiff的列表类)

方法：

DirectoryDiff createDiff(Snapshot,INodeDirectory):根据currentDir创建DirectoryDiff

INodeDirectoryAttributes createSnapshotCopy(INodeDirectory currentDir):根据currentDir创建snapshot副本

replaceChild(ListType,INode)

removeChild(ListType,INode)

4) InodeDirecotryWithSnapshot的方法和成员变量

DirectoryDiffList diffs;根据snapshotIDS排序的Directory变化列表

方法：

computeDiffBetweenSnapshots(fromSnapshot,toSnapshot,ChildrenDiff):该方法用于计算startSnapshot和endSnapshot中出现的Inode变化

saveChild2Snapshot(INode child,Snapshot latest,INode snapshotCopy,InodeMap):将Inode存储到Snapshot当中

addChild(INode inode,boolean,Snapshot latest,INodeMap):在children diff中增加一个Inode,包括创建该Inode对应的diff元素，将该element加入到ChildrenDiff中

removeChild(......)

undoRename4SrcParent(INodeReference oldChild,newChild,latestSnapshot)

cleanSubTree()

......

其他方法不在介绍，基本上都是和DirecotryDiffList 处理相关的方法。本质上讲，INodeDirectoryWithSnapshot，维护的是该INodeDirecotry下的所有INode的变化，为相应的INode构建Diff，并根据INode的变化更新这个Diff。

#### 3.2.1.2 INodeDirectorySnapshottable

允许Snapshot的目录，通过Namesytem和FSdirectory lock进行同步。该类继承了类INodeDirctory，下面是成员变量的分析：

int SNAPSHOT\_LIMIT; //限制每个snapshottable directory的Snapshot个数

int snapshotQuota = Snapshot\_limit； 允许的snapshot个数

List<Snapshot> snapshotsByNames; 根据snapshots的名字升序排列Snapshots

内部类：SnapshotDiffInfo,用于描述snapshotable directory下snapshots之间的不同。

成员变量：

INodeDirecotrySnapshottable snapshotRoot; // snapshots的根目录

Snapshot from;

Snapshot to;

SortedMap<INode,byte[][]> diffMap;记录改变的File和InodeDirecotry，及根据snapshot root的相对路径。

Map<InodeDirectory,ChildrenDiff> dirDiffMap;文件创建、删除等细节性的区分，key值表示children变化的InodeDirectory（from和to snapshot之间的比较）

方法：

addDirDiff(INodeDirecotry dir,byte[][] relativePath,ChildrenDiff diff):增加dir-diff 对

addFileDiff(INodeFile file,byte[][] relativePaht)；添加修改的文件

SnapshotDiffReport generateReport); 产生SnapshotDiffReport

INodeDirecotrySnapshottable的方法：

\*Snapshot():这类方法对Snapshot进行的操作

computeDiff(String from,String to):计算Snapshot的不同，返回SnapshotDiffInfo

ComputeDiffRecursively(Inode inode,List<byte[]> parentPath,diffReport):计算给定目录/file下的SnapshotDiffInfo。

#### 3.2.1.3 FileWithSnapshotFeature

该类用于管理文件中和Snapshot相关的信息，成员变量：

FileDiffList diffs; //存储snapshot 数据的FileDiffs列表

成员方法：

cleanFile(INodeFile file,snapshotID,priorSnapshotId,......)

CollectBlocksBeyondMax(INodeFile file,long max,......)

collectBlocksAndClear(INodeFile file,info,removedINodes)// 收集blockList中不属于任何INode的block信息并进行处理

#### 3.2.1.4 FileDiff/FileDiffList

FileDiff用于表示INodeFile之间的不同，其继承了AbstractInodeDiff,增加的成员变量long fileSize; 文件大小

FileDiffList继承了AbstractINodeDiffList<INodeFile,INodeFileAttributeds,FileDiff

>

#### 3.2.1.5 SnapshotDiff

该类用于获取两个快照之间的差别，或者当前快照和目录（统计数据）之间的不同，该类的方法如下：

run():SnapshotDiff只能用在DistributedFileSystem中，调用DFS.getSnapshotDiffR

eprot(snapshotDir,fro,to)

### 3.2.2 Snapshot

该类用于表示namesystem中的一个子树的快照。

成员变量：

Root root;Root是Snapshot的内部类，用于表示snapshot的根目录。

int id; snapshot的ID号

ID\_COMPARATOR/ID\_INGEGER\_COMPARTOR:保证Null指示当前状态，比其他非null snapshot更优先级更高

其中Root是内部类，其继承InodeDirectory，其提供了方法getChildrenList(Snapshot snapshot)，getChild(byte[] name,Snapshot snapshot),getFullName(getParent.getFull

PathName(),getLocalName())。

### 3.2.3 SnapshotManager

这个类用于管理snapshottable的目录和snapshots，包括创建、访问、修改snapshot及相关的数据。

核心成员变量：

boolean allowNestedSnapshots = false; 默认是不允许进行快照

Map<Long,InodeDirectorySnapshottable> snapshottables; 快照的目录

FSDirectory fsdir;

AtomicInteger numSnapshots; //目录个数

方法如下：

setSnapshottable(String path,boolean checkNestedSnapshottable):设置path为可快照的目录。

\*Snapshotable()进行snapshotables的操作，包括添加、移除、重置、查找等操作

String createSnapshot(String path,String snapshotName):根据给定的path创建snapshot

\*Snapshot()：根据需要对snapshot进行操作，包括重命名，删除

SnapshotDiffInfo diff(String path,String from,String to):计算可以snapshot的目录的变化

### 3.2.4 SnapshotFSImageFormat

这个类是用于将snapshot相关信息从FSImage中写入及读取的helper类。

方法如下：

saveSnapshots(InodeDirectorySnapshotable current,out):保存directory中

snapshots和quota数据

saveInodeDiffs(AbstractInodeDiffList,out,referenceMap):将InodeDirecotryWith

Snapshot的变化保存到FSimage中

saveFileDiffs()

saveFileDiffList()

类中的load方法是和save方法对应的，不再介绍

## 3.3 Snapshot运行流程

在FSNameSystem中实例化及使用SnapshotManager，Snapshot相关工作都是通过ShotManager来完成。Snapshot是否使用，都会创建SnapshotManager。

### 3.3.1 Snapshot的启用

在FSNameSystem中，方法allowSnapshot(String path),启用path的快照，仅针对该Path下的文件。通过DFSClient.allowSnapshot完成调用。

步骤：

1)检测Namenode运行状态，如果处在safemode下不允许Snapshot操作

2)检测权限

3)调用snapshotManager.setSnapshottable(path,checkNestedSnapshotshottable)，设置该path为可快照。

获取该Path的INodesInPath及INodeDirectory，检测INodeDirecotry是否可快照，如果已经快照，则更新SnapshotQuota。

还没有快照，则将该目录转换成INodeDirectorySnapshotble

将Snapshot copy保存至最新的snapshot当中。Snapshot保存在InodeDirecotrySnapshttable的Snapshot列表中。

### 3.3.2 Snapshot的创建

将path指定的目录设置成Snapshotable以后，需要定期创建该目录的Snapshot。分析FSNameSytem下的createSnapshot(string,string)

首先检测操作、权限及Namenode的运行状态，

然后调用snapshotManager.createSnapshot(snapshotRoot,snapshotName)

SnapshotManager.createSnapshot(snapshotRoot,snapshotName)

首先获取src的INodeDirectorySnapshottable

调用addSnapshot(snapshotCounter,snapshotName)

创建snapshot实例，获取snapshotCounter与当前INodeDirectory的DirecotryDiff，设置其Diff所属的Snapshot，将snapshot添加到snapshotByNames中

### 3.3.3 Snapshot的文件操作

appendFile(String,String)

### 3.3.3 Snapshot的FSImage操作

FSImage在Saver.save()中，调用sourceNamesystem.getSnapshotManager.write(ou

t)，将snapshot写入FSImage文件中。

saveImage(INodeDirectory current,out,toSaveSubTree,inSnapshot,counter):会根据inSnapshot进行判断，是否是Snapshot目录，如果是，进行与Snapshot有关的操作。

在save()中saveImage中的参数inSnapshot赋值为true;

Loader.load(File)中首先根据NameNodeLayoutVersion，判断是否支持supportSnapshot，如果支持使用nameSystem.getSnapshotManager.read(in,this)，对snapshotMap进行初始化，并加载snapshot化的INodeFile。