# 删除

不像一些本地文件系统，只需要释放元数据就可以继续在原位置继续覆盖写入新数据。CephFS 作为一个分布式存储系统，需要将数据和元数据都删除，才能保证不浪费空间。然而当有大量删除的时候，数据的删除会是一个非常耗时的操作，所以实际上 Ceph 实现了延迟删除，先删除元数据，数据并没有被马上删除，而是被移动到了一个名叫 stray 的特殊目录，后续再后台一点点删除。

# 空间统计

CephFS 使用文件的扩展属性 xattr 去实现上述需求，将目录包含的子文件和子目录的使用空间等统计信息保存在xattr 中，当需要获取某个目录的空间使用情况时，只需要通过 [getfattr](https://link.zhihu.com/?target=https%3A//man7.org/linux/man-pages/man1/getfattr.1.html" \t "_blank) 获取这个目录的xattr就可以了。

root@node-3:/# getfattr -d -m ceph.dir.\* .

# file: .

ceph.dir.entries="1"

ceph.dir.files="0"

ceph.dir.rbytes="708822611968"

ceph.dir.rctime="1457535283.09430737000"

ceph.dir.rentries="3633611"

ceph.dir.rfiles="3608532"

ceph.dir.rsubdirs="25079"

ceph.dir.subdirs="1"

各字段含义见:

表格

描述已自动生成

# Hardlink

当存在硬链接时，第一个指向Inode的Dentry被称为Primary Dentry，后续的被称为Remote Dentry。

为了访问Remote Dentry， 人们提出了Anchor，它包括Path，Inode，Parent，Ref.前三者显而易见，Ref是被Inode引用的次数（即2.4节提到的计数器）。当进行目录重命名时，可能会影响整个链上的Inode，此时就需要一个事务来保证整个链上相关的Inode同时进行更新，将旧的Ref计数减少，新的Ref计数删除。如果Ref为0，说明Inode已经没有硬链接，可以从Anchor表删除，对Dentry的增加修改同理。

我理解上面的描述应该是之前的版本，最新版本已经废弃了上述的方案？

#### hardlink整体流程

1、leader先发送prepare给对端，对端生成prepare日志，对端返回prepare ack；

Server::\_link\_remote

op = MMDSPeerRequest::OP\_LINKPREP;

req = make\_message<MMDSPeerRequest>(mdr->reqid, mdr->attempt, op);

mds->send\_message\_mds(req, linkauth); //发送prepare消息

mdr->more()->waiting\_on\_peer.insert(linkauth);

Server::dispatch\_peer\_request

Server::handle\_peer\_link\_prep // 参与者记录日志

le = new EPeerUpdate（peer\_link\_prep） 创建link\_prep日志并更新日志记录 EVENT\_PEERUPDATE

mdr->more()->peer\_commit = new C\_MDS\_PeerLinkCommit(this, mdr, targeti);

submit\_mdlog\_entry(C\_MDS\_PeerLinkPrep)

Server::\_logged\_peer\_link //参与者记录完成日志后，给回响应

reply = make\_message<MMDSPeerRequest>(mdr->reqid, mdr->attempt, MMDSPeerRequest::OP\_LINKPREPACK);

mds->send\_message\_mds

2、leader收到prepare ack后，生成EUpdate日志，并给对端发送finish消息；

Server::handle\_peer\_request\_reply //收到消息响应

case MMDSPeerRequest::OP\_LINKPREPACK:

Server::handle\_peer\_link\_prep\_ack

Server::dispatch\_client\_request

case CEPH\_MDS\_OP\_LINK:

Server::handle\_client\_link

Server::\_link\_remote

生成link\_remote日志 EVENT\_UPDATE

journal\_and\_reply

Server::\_link\_remote\_finish

Server::respond\_to\_request

reply\_client\_request

mdcache->request\_finish

MDCache::request\_finish

MDCache::request\_cleanup

MDCache::request\_drop\_locks

MDCache::request\_drop\_foreign\_locks

r = make\_message<MMDSPeerRequest>(mdr->reqid, mdr->attempt,

MMDSPeerRequest::OP\_FINISH) //发送finish消息

mds->send\_message\_mds(r, \*p)

3、对端收到finish消息后，生成commit日志，对端返回committed消息；

Server::handle\_peer\_request //收到finish消息

If（MMDSPeerRequest::OP\_FINISH）

mdcache->request\_finish // MDCache::request\_finish

fin = mdr->more()->peer\_commit

fin->complete // server->\_commit\_peer\_link

le = new EPeerUpdate(mdlog, "peer\_link\_commit" //生成commit日志 EVENT\_PEERUPDATE

submit\_mdlog\_entry（C\_MDS\_CommittedPeer）

Server::\_committed\_peer

req = make\_message<MMDSPeerRequest>(mdr->reqid, mdr->attempt, MMDSPeerRequest::OP\_COMMITTED);

mds->send\_message\_mds //返回commited响应消息

4、leader收到committed消息后，生成ECommitted日志。

Server::handle\_peer\_request\_reply //收到响应

if (MMDSPeerRequest::OP\_COMMITTED)

mdcache->committed\_leader\_peer

MDCache::log\_leader\_commit

mds->mdlog->start\_submit\_entry //日志下盘(EVENT\_COMMITTED)

MDLog::\_submit\_entry

MDCache::\_logged\_leader\_commit

hardlink整体流程：

|  |  |  |
| --- | --- | --- |
| 流程 | leader | peer |
| leader先发送prepare给对端，对端生成prepare日志，对端返回prepare ack； | leader发送完prepare后，就会加入waiting\_on\_peer队列 | remote收到prepare后：  加入uncommitted\_peers队列  生成link\_prep日志 |
| leader收到prepare ack后，生成EUpdate日志，并给对端发送finish消息； | 生成link\_remote日志  leader加入uncommitted\_leaders队列 | 生成peer\_link\_commit日志 |
| 对端收到finish消息后，生成commit日志，对端返回committed消息； |  | 从uncommitted\_peers中删除 |
| leader收到committed消息后，生成ECommitted日志。 | 从uncommitted\_leaders队列中删除记录  生成EVENT\_COMMITTED日志。 |  |
|  |  |  |

故障场景分析

|  |  |  |
| --- | --- | --- |
| 故障场景 | 说明 | 解决方案 |
| leader先发送prepare给对端，对端生成prepare日志，对端返回prepare ack；  故障：  （1）prepare ack消息丢失；（2）leader重启。 | 对端处理prepare时，会加入uncommitted\_peers队列。  我个人理解，应该有个后台定时任务定时处理uncommitted\_peers队列中超时的任务（给leader发送消息确认？）。 |  |
| leader收到prepare ack后，生成EUpdate日志，并给对端发送finish消息；  故障：  （1）在发送finish消息前leader节点重启；  （2）发送finish消息丢失； | 个人理解，应该有后台定时任务定时处理uncommitted\_leaders队列中超时的任务（给remote重新发送消息）。 |  |
| 对端收到finish消息后，生成commit日志，对端返回committed消息；  故障：  （1）返回committed消息丢失；  （2）remote或leader节点重启； |  |  |
|  |  |  |

# 日志

1）更新首先会写入到MDS的日志中；

2）将有改变的元数据标记为“dirty”，并在MDS缓存中置为“pinned”。

3）最终修改会更新到具体元数据对象中，但同时也会做延时处理直到从日志中剪掉，这使得日志可以变得非常大（数百兆）。

# 其他

chattr/lsattr 文件特殊属性 这些属性存放在文件的那个字段中？