CurveFS Client 概要设计（已实现）

时间	作者	内容
2021-04-27	许超杰	初稿

背景

CurveFS初步设计见 CurveFS方案设计（总体设计，只实现了部分）， 目前需细化Client端设计

概述

CurveFS client 向上提供两层接口，分别是

- 对接fuse，提供通用文件系统接口。对于fuse接口，先前进行了一些调研，见FUSE调研
- 提供lib库，提供对接分布式数据库接口，这一部分，可参考polarfs的接口，如下图所示。

```
int     pfs_mount(const char *volname, int host_id)
int     pfs_umount(const char *volname)
int     pfs_mount_growfs(const char *volname)

int     pfs_creat(const char *volpath, mode_t mode)
int     pfs_open(const char *volpath, int flags, mode_t mode)
int     pfs_close(int fd)
ssize_t pfs_read(int fd, void *buf, size_t len)
ssize_t pfs_write(int fd, const void *buf, size_t len)
off_t   pfs_lseek(int fd, off_t offset, int whence)
ssize_t pfs_pread(int fd, void *buf, size_t len, off_t offset)
ssize_t pfs_pwrite(int fd, const void *buf, size_t len, off_t offset)
int     pfs_stat(const char *volpath, struct stat *buf)
int     pfs_fstat(int fd, struct stat *buf)
int     pfs_posix_fallocate(int fd, off_t offset, off_t len)
int     pfs_unlink(const char *volpath)
int     pfs_rename(const char *oldvolpath, const char *newvolpath)
int     pfs_truncate(const char *volpath, off_t len)
int     pfs_ftruncate(int fd, off_t len)
int     pfs_access(const char *volpath, int amode)

int     pfs_mkdir(const char *volpath, mode_t mode)
DIR*    pfs_opendir(const char *volpath)
struct dirent *pfs_readdir(DIR *dir)
int     pfs_readdir_r(DIR *dir, struct dirent *entry,
              struct dirent **result)
int     pfs_closedir(DIR *dir)
int     pfs_rmdir(const char *volpath)
int     pfs_chdir(const char *volpath)
int     pfs_getwd(char *buf)
```

Figure 3: libpfs interfaces

根据讨论，我们首先对接fuse的lowlevel operators，对于数据库的lib库接口，后续可以在此基础上再做一层对接。lowlevel operators接口一共45个，如下：

```
+init
   +destroy
   +lookup
   +forget
   +getattr
   +setattr
   +readlink
   +mknod
```

```
+mkdir
+unlink
+rmdir
+symlink
+rename
+link
+open
+read
+write
+flush
+release
+fsync
+opendir
+readdir
+releasedir
+fsyncdir
+statfs
+setxattr chubaofs
+getxattr chubaofs
+listxattr chubaofs
+removexattr chubaofs
+access
+create
+getlk
+setlk
+bmap
+ioctl
+ioctl
+poll
+write_buf
+retrieve_reply
+forget_multi
+flock
+fallocate
```

```
+readdirplus
+copy_file_range
+lseek
```

关键接口分析

init

```
void (*init) (void *userdata, struct fuse_conn_info *conn);
```

- 根据挂载信息，从mds获取文件系统信息（或superblock），块分配器（bitmap）和root inode所在的copyset、 metaserver ip等信息
- 去metaserver获取文件系统信息（super block），缓存到client端。

destroy

```
void (*destroy) (void *userdata);
```

- 清理init缓存的文件系统信息。

lookup

```
void (*lookup) (fuse_req_t req, fuse_ino_t parent, const char *name);
```

- 根据parent inode id和name从denty缓存中找到对应的denty结构；
- 如果dentry缓存中不存在对应的inode，则从mds根据parent inode id获取parent inode 所在copyset，metaserver ip等信息 ，然后从metaserver获取denty（这里有两种方式，一种是只获取当前需要的denty，一种是list整个目录的denty，这个需要考虑用哪个接口）
- 根据找到的denty结构，获取inodeid，设置 fuse_entry_param，返回给fuse

write

```
void (*write) (fuse_req_t req, fuse_ino_t ino, const char *buf, size_t size, off_t off, struct fuse_file_info *fi);
```

- 首先根据inode id 从缓存中查找到对应inode结构；
- 如果inode缓存中不存在对应的inode，则从mds获取inode所在copyset，metaserver ip等信息，然后从metaserver获取inode结构，缓存之；
- 判断inode结构中，对应请求[off, size]位置的空间是否有分配：如果未分配或只有部分分配空间，则调用空间分配器分配空间，并根据空间分配器返回结果，修改inode结构（包括file length）；inode修改需要持久化到底层并修改本地cache；
- 调用curve client接口，写curve卷对应[offset, len] 数据。 （这里涉及到一个问题，是否从fuse下来的请求是4k对齐的，如果不是，那么这里还需要修改为read merge write，即读出未对齐缺少的部分，然后整个[offset,len] 调用curve client写）；
- 修改inode结构，如果上述区域存在先前未写过的区域，则需要去掉unwritten，具体方式根据inode结构而定；inode修改需要持久化到底层并修改本地cache；

read

void (*read) (fuse_req_t req, fuse_ino_t ino, size_t size, off_t off, struct fuse_file_info *fi);

- 首先根据inode id 从缓存中查找到对应inode结构；
- 如果inode缓存中不存在对应的inode，则从mds获取inode所在copyset，metaserver ip等信息，然后从metaserver获取inode结构，缓存之；
- 根据inode结构，拆分unwritten/未分配的区域与写过的区域，未写过的区域填0，其他区域继续读取
- 根据inode结构中信息，调用curve client接口，读取对应的[offset, len]数据。（这里同样要考虑4k对齐的问题，如果不对齐，则需要读取对齐的区域，然后去掉多读的部分）（读写可以做数据缓存，当前先不考虑）。

open

void (*open) (fuse_req_t req, fuse_ino_t ino, struct fuse_file_info *fi);

posix语义中open支持的oflag主要有：

- O_RDONLY 只读打开
- O_WRONLY 只写打开
- O_RDWR 读写打开

以上3个必须指定且只能指定一个

- O_APPEND 只追加写
- O_CREAT 文件不存在时创建
- O_EXCL 如果同时指定了O_CREAT，而文件已存在，则会出错，用此测试文件是否存在，如果不存在则创建此文件。
- O_TRUNC 如果文件存在，且为只写或只读打开，则将其文件长度截短为0
- O_DSYNC sync数据和必要元数据（不影响读取刚写入的数据）
- O_SYNC sync数据和所有元数据
- O_DIRECTORY 目录

- O_DIRECT 直接IO

open的主要逻辑：

- 根据inode id，从mds获取inode所在copyset，metaserver ip等信息，然后从metaserver获取inode结构，缓存之；
- 判断上述各种oflag，执行相应的操作。（对于目前阶段来说，open可以什么都不做）

create & mknod

void (*create) (fuse_req_t req, fuse_ino_t parent, const char *name, mode_t mode, struct fuse_file_info *fi);

void (*mknod) (fuse_req_t req, fuse_ino_t parent, const char *name, mode_t mode, dev_t rdev);

这两个函数的功能是类似，都用来创建文件。

- 根据parent inode id 和name，向mds查询创建dentry和inode的位置，去meta server创建dentry和inode
- 预分配一些空间？可先不做

mkdir

```
void (*mkdir) (fuse_req_t req, fuse_ino_t parent, const char *name,
mode_t mode);
```

- 根据parent inode id 和name，向mds查询创建dentry和inode的位置，去meta server创建dentry和inode

forget

```
void (*forget) (fuse_req_t req, fuse_ino_t ino, uint64_t nlookup);
```

- 根据inodeid找到对应的inode结构，lookup count值减少nlookup。 （这里涉及到一个lookup count 存在哪里的问题）

unlink

```
void (*unlink) (fuse_req_t req, fuse_ino_t parent, const char *name);
```

- 根据parent inode id 和 name找到当前文件的inode和denty结构
- 根据lookup count 值，如果非0，则需要延迟删除文件，如果为0，则真正删除文件。 （这里需要做标记删除）
- 删除时需要从缓存或mds查询删除inode和dentry的位置，并去metaserver删除，然后清除本地缓存

rmdir

```
void (*rmdir) (fuse_req_t req, fuse_ino_t parent, const char *name);
```

- 根据parent inode id 和 name找到当前文件的inode和denty结构
- 根据lookup count 值，如果非0，则需要延迟删除目录，如果为0，则删除目录。
- 删除时需要从缓存或mds查询删除inode和dentry的位置，并去metaserver删除，然后清除本地缓存

opendir

- 可先不支持，返回ENOSYS

readdir

```
void (*readdir) (fuse_req_t req, fuse_ino_t ino, size_t size, off_t off, struct fuse_file_info *fi);
```

- 根据inode id 找到当前dir的inode struct结构，从缓存或metaserver获取当前dir的dentry列表。
- 读取当前[off,size]对应位置的dentry，调用fuse_add_direntry，生成dir buffer，返回

getattr & setattr

```
void (*getattr) (fuse_req_t req, fuse_ino_t ino, struct fuse_file_info *fi);
```

- 根据inode id 找到inode id 对应的inode 结构（从缓存或者metaserver）

- 根据inode结构调用fuse_reply_attr填充attr字段，返回

```
void (*setattr) (fuse_req_t req, fuse_ino_t ino, struct stat *attr, int to_set, struct fuse_file_info *fi);
```

- 根据inode id 找到inode id 对应的inode 结构（从缓存或者metaserver）
- 根据to_set字段设置相应的attr字段，然后持久化到metaserver，并更新本地缓存。

access

- 可先不支持，返回ENOSYS

rename

```
void (*rename) (fuse_req_t req, fuse_ino_t parent, const char *name, fuse_ino_t newparent, const char *newname, unsigned int flags);
```

rename有两种做法：

一是，向metaserver发起inode和dentry迁移，从parent迁移到new parent，并修改name为newname。

二是，在new parent创建新的inode和dentry，然后删除旧的parent下的inode和dentry

两者都涉及到rename的事务性的问题？（这里可能还需要详细分析到底是否需要完整的事务的4个特性acid，还是只需要实现其中部分）

目前阶段rename可先不实现，但是可以先考虑一些方案。

symlink & readlink

```
void (*symlink) (fuse_req_t req, const char *link, fuse_ino_t parent, const char *name);
```

- 根据parent inode id 和name，向mds查询创建dentry和inode的位置，去meta server创建dentry和inode

```
void (*readlink) (fuse_req_t req, fuse_ino_t ino);
```

- 根据inodeid，向mds查询创建inode的位置，去meta server获取inode结构，并缓存
- 从inode结构中取出link contents，调用fuse_reply_readlink向上返回link contents。

软链接相关接口目前可先不实现。

link

```
void (*link) (fuse_req_t req, fuse_ino_t ino, fuse_ino_t newparent, const char *newname);
```

- 这个涉及到下文中”重要问题讨论“，目前暂时无法设计

硬链接相关目前可先不实现。

flush & fsync

- 缓存的问题暂时先不考虑太细，目前默认数据和元数据直接存储到底层，这两个也可先不实现

其他

- xattr系列接口，chubaofs都没实现，目前先不考虑
- fuse高版本新增的接口如lseek等，在低版本中没有，因此不是必须接口，也先不实现。

功能分析

根据上述接口的分析，可以把client端的功能进行汇总，client需实现的功能主要有：

- 缓存文件系统元数据（包括super block， bitmap & allocator等）
- 缓存文件和目录信息（包括inode struct，dentry struct）
- 缓存metaserver copyset 和 topo信息（目前先支持单metaserver的情况下，只需要一个metaserver的ip就可以）
- 与mds 交互，调用mds接口获取metaserver copyset 和 topo信息，这部分可先不实现（目前先支持单metaserver的情况下，可先不实现，由配置文件加载metaserver的ip）
- 与metaserver交互，调用meta server接口获取文件系统元数据信息，调用meta server接口获取文件和目录信息等
- 与现有块设备client交互，调用块设备接口，对卷进行读写。
- 向上对接fuse接口，协调上述模块交互，实现功能。
- main 主函数模块，类似deamon，接收mount消息并处理（fuse session）。

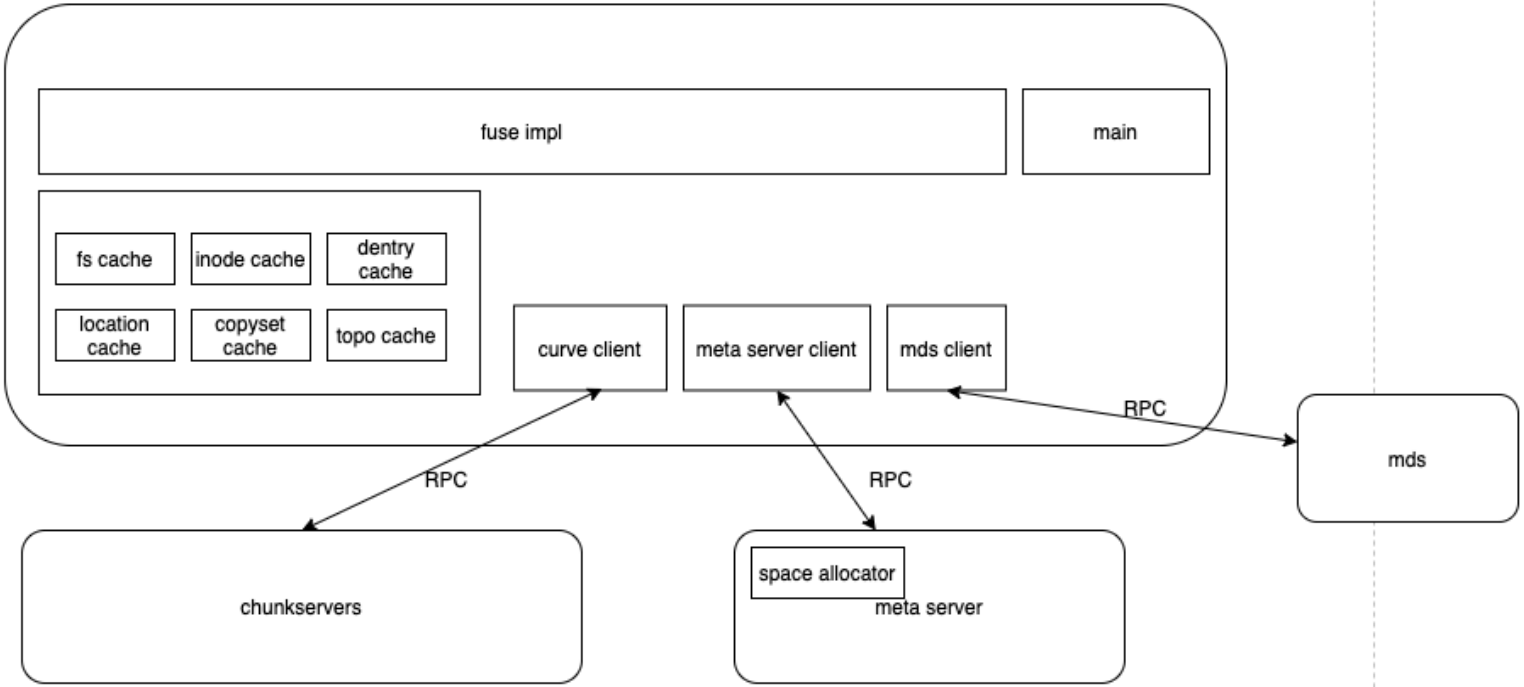
除上述功能以外，还需实现文件系统创建和fuse挂载工具，功能主要是：

- 创建文件系统，指定文件系统的名字、卷大小（多文件系统）、
- 扩展文件系统?
- 挂载fuse文件系统，指定挂载点、文件系统名字或fsID、server ip（可从配置文件读取）。

模块划分

根据上述功能模块，将client划分模块

- fs cache
- inode cache
- dentry cache
- location cache （inode location、dentry location、fs location、bitmap location，目前都在同一meta server，可先不实现）
- copyset cache & topo cache （可先不实现）
- mds client
- meta server client
- curve client （适配层）
- fuse impl
- main



接口设计

相关接口设计，见curve文件系统元数据proto（代码接口定义，已实现）

Cache设计

Client的重要部分，就是上述这些cache的组织，基于以下几点考虑cache的组织方式：

1. 由于cache不命中情况下，损失了cache查找这部分时间，因此，应当选用cache查找尽可能快的结构，这里考虑采用hash_map。

2. 由于fuse一次mount是一个独立的进程，因此，不需要考虑在同一个进程中支持多文件系统，每个文件系统对应独立进程。

3. cache应当有淘汰策略，元数据cache如果不淘汰，那么缓存到client端的cache会越来越多，这部分可以采用LRU。

4. 考虑到“dirty” inode和 dentry的flush过程，还需方便的过滤出dirty的部分的结构，当然，这部分可以后续使用一个dirty链表实现（实际可能需要多个链表）。