3.2. 数据内存模型

对于 glusterfs 来说，是使用了一些扩展属性来维护元数据信息的，那么根据前面讲的 gfid 这个扩展属性，通过隐藏目录拿到了路径和名称这些信息，那如果要打开一个文件，对于 glusterfs 来说是如何做到的呢？这个就涉及到数据在内存中的模型了，了解这部分内容，可以进一步理解一下，到底什么是 fuse，glusterfs 的 fuse 和一般的文件系统区别在哪里。这里为了区别 glusterfs 定义的数据结构与操作系统中常见的元素的区别，如 inode，那么在 glusterfs 中使用 inode_t 来指代其定义的结构，这个称呼也是从官方中继承下来的(虽然代码中命名并没有这样写，但是看到官方的资料是这样提到的，因此为了统一习惯也就这样称呼吧)。

3.2.1. loc_t 结构

对于一个文件系统来说，正常要找到一个文件，可以通过目录一层层地寻找，而对于 glusterfs 来说，则是通过隐藏目录.glusterfs 下的 gfid 来找到对应的路径，然后找到文件的 inode 信息，而有了 inode 这些信息，才能够进一步对文件进行读写操作。因此对于从文件路径再到找到文件的 inode，这里需要一个叫做 lookup 的文件操作(也就是 file operation,简称 fop,这个在后面会多次提到，可以简单理解为文件系统封装的一些操作逻辑，这里不理解没关系，后面看多了会慢慢理解的)，这个 fop 解决的就是知道一个文件路径如何找到该文件的 Inode 问题。当然为了解决这个问题，就不得不提到 glusterfs 的一些数据内存模型了，首先要讲到的就是 loc_t 结构，这个结构中就是记录了关键的一些数据

结构信息，下面一起来了解一下。

```
1.  struct _loc {
2.      const char *path;
3.      const char *name;
4.      inode_t *inode;
5.      inode_t *parent;
6.      uuid_t gfid;
7.      uuid_t pargfid;
8.  };
```

这个数据结构内容在源码中 libglusterfs/src/glusterfs/xlator.h 文件里面，这里面记录的数据结构信息也非常直接易懂，主要就是当前的路径，文件名称 ,当前文件和父目录的 inode,gfid 信息。那么通过路径和名称等等找到 inode 信息之后，就可以真正找到这个文件了。

3.2.2. inode_t 结构

那么前面了解了 loc_t 结构之后，看到 inode_t 也是一个结构，那么这里 inode_t 到底记录了什么呢？这个结构的作用又是什么？下面也来简单了解一下。

```
1.  struct _inode {
2.      //把活跃的 inode 信息记录在内存的一张表
3.      inode_table_t *table;
4.      uuid_t gfid;
5.      gf_lock_t lock;
6.      gf_atomic_t nlookup;
7.      //文件句柄 fd 的统计数量,也就是文件被打开的累计次数
8.      uint32_t fd_count;
9.     //活跃的 fd 打开统计次数
```

```
10.     uint32_t active_fd_count;
11.     //这个 Inode 被引用的统计数量
12.     uint32_t ref;
13.     //文件类型
14.     ia_type_t ia_type;
15.     //记录打开了这个文件的 fd 的 list 列表
16.     struct list_head fd_list;
17.     //此 inode 的目录项列表
18.     struct list_head dentry_list;
19.     struct list_head hash;
20.     //和 lru 相关的
21.     struct list_head list;
22.
23.     struct _inode_ctx *_ctx;
24.     //如果 inode 是不活跃的则设置这个属性
25.     bool in_invalidate_list;
26.     bool invalidate_sent;
27.    //记录 inode 是否在 lru 列表里面
28.     bool in_lru_list;
29. };
```

那么从这里也可以简单了解到，inode_t 这个结构里面，记录的主要还是和操作系统里面有关的信息比较多，主要就是该 Inode 信息是否活跃，文件类型等信息。另外通过这个 inode_t 的结构，还记录和当前这个文件被打开的文件句柄(简称 fd)相关的信息,通过这些信息，就可以比较好地在追踪文件是否还被引用或者文件是否已经关闭等等。

当然从这两个简单的数据结构中，同时对比前面提到的 vfs 里面的一些数据结构 inode 等，所谓的 fuse，就可以简单理解为对一些操作系统的内容进行自定义封装的一套文件系统接口。而 glusterfs 则也是封装了一套 glusterfs fuse

接口的，这样做的优缺点也是非常明显的，优点是可以自定义做很多特定的需求与想法，缺点就是实现难度比较大，如团队技术水平不足时，或者代码设计出现问题时，很容易造成一些额外的重要 bug,例如 glusterfs 的 fuse 则在 7.7 之前的版本，有一个文件句柄泄露的重大 bug 出现了，会导致在 mount 的时候掉线，当然关于 glusterfs 的一些版本问题等，后面也会再次提到。

3.2.3. dentry_t 结构

这个结构其实也是 vfs 中的四个基本元素中 dentry 对应的，这里 glusterfs 对其的结构定义代码在 libglusterfs/src/glusterfs/inode.h 文件中。该结构记录的信息也比较简单，而 dentry_t 和 inode_t 结构，是做打开和修改文件等操作之前，需要获取的两个数据结构的信息，而如何获取，就是后面提到的 lookup 接口实现的功能。

```
1.  struct _dentry {
2.      //记录 dentry 下的 inode 列表
3.      struct list_head inode_list;
4.      // 指向 hash 表
5.      struct list_head hash;
6.     // 当前目录的 inode
7.      inode_t *inode;
8.     // 当前目录的名称
9.      char *name;
10.     // 父目录的 inode
11.     inode_t *parent;
12. };
```

3.2.4. fd_t 结构

有了前面的 inode_t 和 dentry_t 结构，在 glusterfs 中就可以调用打开文件的相关操作了(open fop,这个会在后面提到),这里 fd_t 结构的代码文件在 libglusterfs/src/glusterfs/fd.h 中。

```
1.  struct _fd {
2.      uint64_t pid;
3.      //系统调用 open 中的 flag,如 O_CREAT 表示不存在则创建等
4.      int32_t flags;
5.      //fd 引用统计
6.      gf_atomic_t refcount;
7.      struct list_head inode_list;
8.      //打开的文件所对应的 inode
9.      struct _inode *inode;
10.      ...
11. };
```