3.4. 一些重要的概念与进程

前面提到了一些 glusterfs 的数据内存模型，还有一些 posxi 接口的实现，那么 glusterfs 当中其中还有一些其他比较重要的概念和进程，因为这些概念有些是代码架构设计层面的，有些是和配置文件有关的，有些进程则是 glusterfs 集群中非常重要的进程，而这些内容的关联性比较紧密，因此打算把这几样东西一起介绍一下。同时有些概念因为涉及到代码规范设计模式那样的，可能理解起来不太好一下子理解，但本书的目的也并没有计划把 glusterfs 的源码理解个通透，在源码解读方面，本书的定位还是希望保持入门带领大家认识一下 glusterfs (其实就是本人水平有限呀，很多细节和内容理解起来也没有很好掌握~)。

3.4.1. 代码模块功能 xlator

从前面的一些代码路径上面，大家可能会留意到这个名词 xlator，当然在官方文档中，还有时会看到 translators 这个名词，这两个其实都是一样的。而且 xlator 还是一个文件路径的开头，例如前面提到的文件路径 xlators/storage/posix/src/posix-entry-ops.c ，从这里可以看到，xlator 的作用应该是起到一个很重要的划分项目不同功能的作用。是的，一个项目里面 (不管是使用 java 或者 c,又或者其他 go 等开发语言开发的项目)，项目功能都会有很多层划分，而在 glusterfs 当中就使用了这种叫 xlator 的架构，这是一种模块化、堆栈式的架构，可通过灵活的配置支持高度定制化的应用环境，比如大文件存储、海量小文件存储、云存储、多传输协议应用等。每个功能以模块形式实现，然后以积木方式进行简单的组合，即可实现复杂的功能。

下面可以来看看 xlator_api 的定义。

```
1.  xlator_api_t xlator_api = {
2.      .init = init,
3.      .fini = fini,
4.      .notify = notify,
5.      .reconfigure = reconfigure,
6.      .mem_acct_init = mem_acct_init,
7.      .op_version = {1},
8.      //这里和 statedump 的内容有关
9.      .dumpops = &dumpops,
10.     //定义的 file operation
11.     .fops = &fops,
12.    // 和回调函数有关
13.     .cbks = &cbks,
14.     .options = options,
15.     .identifier = "replicate",
16.     .category = GF_MAINTAINED,
17. };
```

这是 afr 功能的 xlator 定义,afr 是 glusterfs 使用的一个数据恢复机制，关于这部分内容后面会讲。而该内容是在文件 xlator/cluster/afr/src/afr.c 中定义的。当然这里还可以拿其他模块的进行对比，例如 ec 模块的，这里路径在 xlator/cluster/ec/src/ec.c 中。

当然这里还有一个 xlator 的定义，该内容是在 libglusterfs/src/glusterfs/xlator.h 中，定义如下所示。

```
1.  struct _xlator {
2.
3.      char *name;
4.      char *type;
5.      char *instance_name;
6.
7.      //指向下一个 xlator
```

```
8.      xlator_t *next;
9.     //指向上一个 xlatro
10.     xlator_t *prev;
11.     xlator_list_t *parents;
12.     xlator_list_t *children;
13.     dict_t *options;
14.
15.     //和 linux 的 dl 库有关的
16.     void *dlhandle;
17.     struct xlator_fops *fops;
18.     struct xlator_cbks *cbks;
19.     struct xlator_dumpops *dumpops;
20.     struct list_head volume_options;
21.     ..................
22.    //用于记录 xlator 的一些特殊的信息
23.     void *private;
24.     ....
25. }
```

这里有一个信息叫做 private 的，这个在每个 xlator 中还有定义，下面来简单了解一下。

```
1.  typedef struct _afr_private {
2.      ...
3.      //用于记录一些可能影响决策一致性的因素
4.      uint32_t event_generation;
5.      char vol_uuid[UUID_SIZE + 1];
6.
7.      ...
8.      gf_boolean_t esh_granular;
9.      ...
10. }
11.
12. ...
13.
14. typedef struct afr_granular_esh_args {
15.     fd_t *heal_fd;
16.     xlator_t *xl;
```

```
17.     call_frame_t *frame;
18.     gf_boolean_t mismatch;
19.
20. } afr_granular_esh_args_t;
```

这里有一个叫 esh_granular 的变量，这个变量记录的是 afr 这个模块中和 granular 有关的功能。而在 atf.c 中就有相关的 options 参数选项，例如 granular-entry-heal，当然这里在 afr.c 中还有和该相关相关的一些设置代码，如下所示。

```
1.  int32_t
2.  init(xlator_t *this)
3.  {
4.       ...
5.       priv->granular_locks = (strcmp(locking_scheme, "gran
   ular") == 0);
6.      GF_OPTION_INIT("full-lock", priv->full_lock, bool, ou
   t);
7.       GF_OPTION_INIT("granular-entry-heal", priv->esh_granu
   lar, bool, out);
8.       ...
9.  }
```

这个函数是一个初始化的函数，这里就有一些和参数 granular 相关的设置了。

另外那么这里每个模块都有一些 fops 的，如果要重新实现的话，这里的定义同样是在该文件中的，如下所示。

```
1.  struct xlator_fops fops = {
2.      .lookup = afr_lookup,
3.      .lk = afr_lk,
4.      .flush = afr_flush,
5.      .statfs = afr_statfs,
6.      ...
7.
8.      /* inode read */
```

```
9.      .access = afr_access,
10.     .stat = afr_stat,
11.      ...
12.
13.     /* inode write */
14.     .writev = afr_writev,
15.     .truncate = afr_truncate,
16.     .ftruncate = afr_ftruncate,
17.     ...
18.
19.     /*inode open*/
20.     .opendir = afr_opendir,
21.     .open = afr_open,
22.
23.     /* dir read */
24.     .readdir = afr_readdir,
25.     .readdirp = afr_readdirp,
26.
27.     /* dir write */
28.     .create = afr_create,
29.     .mknod = afr_mknod,
30.    ...
31. };
```

但是这里并不是所有的 fop 都会重新定义实现的，那么这里就会使用 default 的内容了。那么这里 default fop 在哪里呢？就是在 libglusterfs/src/default-tmpl.c

```
1.  struct xlator_fops _default_fops = {
2.      .create = default_create,
3.      .open = default_open,
4.      .stat = default_stat,
5.      .readlink = default_readlink,
6.      .mknod = default_mknod,
7.      .mkdir = default_mkdir,
8.      .unlink = default_unlink,
9.      .rmdir = default_rmdir,
10.     ...
11.
12. }
```

这里就是定义的 default 的 fop 了，如果感兴趣的，可以对比一下 defatult

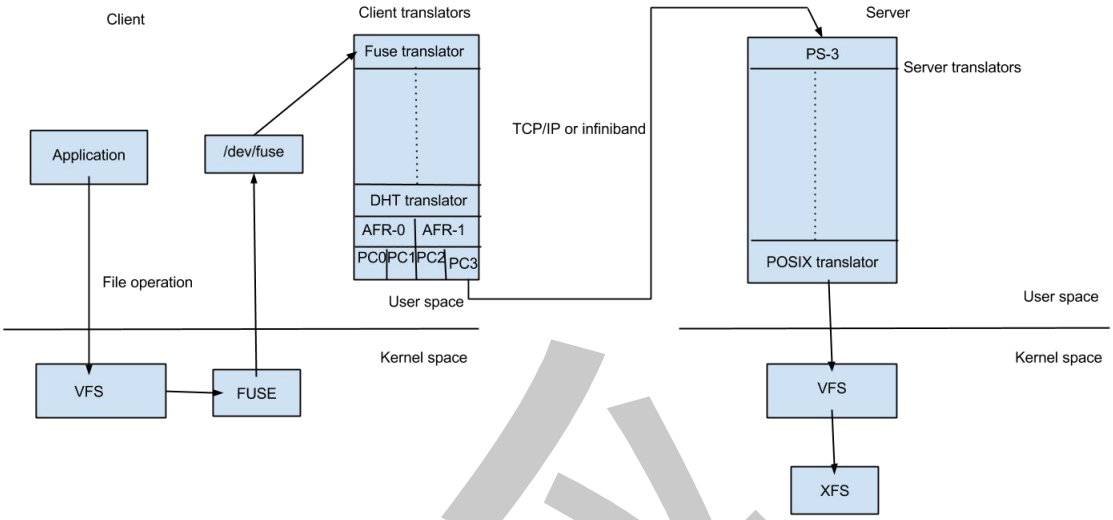
的内容与模块实现的不同。

当然对于 xlator 这里的定义的数据成员的作用，如果有些不理解的，建议暂时不需要全部深入理解，在后面的内容中，其中部分内容会详细讲解一下。另外在 afr.c 文件中，这里还可以看到很多参数的定义，因为不同的模块，涉及的参数不同，在 afr.c 中可以看到这个参数。

```
1.  {
2.          .key = {"quorum-count"},
3.          .type = GF_OPTION_TYPE_INT,
4.          .min = 1,
5.          .max = INT_MAX,
6.          .default_value = 0,
7.          .op_version = {1},
8.           .flags = OPT_FLAG_CLIENT_OPT | OPT_FLAG_SETTABLE
   | OPT_FLAG_DOC,
9.          .tags = {"replicate"},
10.         /*.option = quorum-count*/
11.         /*.validate_fn = validate_quorum_count*/
12.          .description = "If quorum-type is \"fixed\" only
   allow writes if "
13.                         "this many bricks are present.  Ot
   her quorum types "
14.                        "will OVERWRITE this value.",
15.   }
```

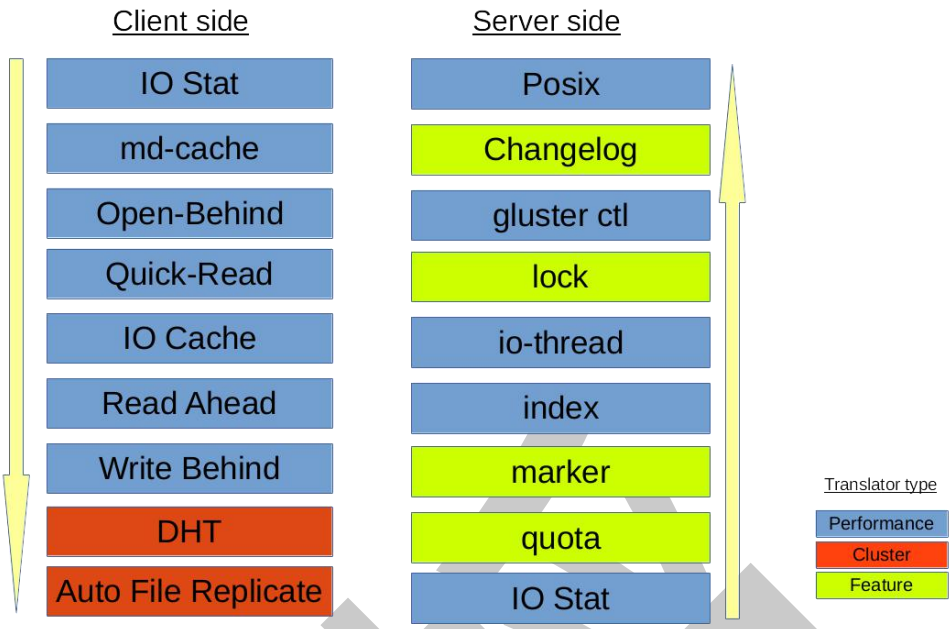
这个参数的作用就是前面仲裁节点中提到的，因此未来如果遇到一些参数，想要知道这个参数到底在哪个模块中发挥作用，就可以使用这种方式在代码中找到对应的位置进行判断了。

那么这里一个挂载从客户端到服务端，会经过哪些 xlator 呢？可以见下图所示。



在客户端挂载一个 volume 的时候 ,客户端的 glusterfs 进程会与服务端的 glusterd 进程沟通，服务端的 glusterd 进程会发送一个配置文件叫(vol file)，里面包含了一系列客户端模块(xlator)和其他信息的内容(例如 volume 每个 brick 的信息)。通过这个文件，客户端 glusterfs 进程就可以与 volume 的每一个 brick 的 glusterfsd 进程进行沟通了，然后就可以进行其他的操作了。

当然这里的 vol file 文件包含很多内容，这个内容就是 graph，简单理解就是不同的 xlator 模块的组合，也就是图中的 client translators 和 server translators 的内容，那么这里会有一些什么模块呢？可以见下图所示。



这里有很多不同的 translator(也叫 xlator)，如 read ahead 就是文件预读相关的功能，io cache 就是 IO 缓存相关的，dht 则是 glusterfs 中的一个重要的哈希算法，changlog 是一个用于记录文件操作与恢复的，quota 则是给 volume 进行容量限制的。这里其中有些功能，会在后面介绍。

那么最后关于 xlator ,这里简单总结以下 glusterfs 中使用这种方式来组织代码层次结构的一些好处优点吧。

1. 代码松耦合，这一点在日常开发中比较容易理解。
2. xlator 在不同环境下具备很好的可重用性与可插拔性，也就是增加新的 xlator 结构比较容易。
3. 可以通过 volfie 文件来实现 graph，这样不同层次结构之间的关系比较明确和清晰。graph 就是不同 xlator 的组合起来，一个 volume 中会有多个不同的 xlator 组成的，其中的关系顺序就是 graph。volfile 可以先简单理解为和 xml 一样的有一定规范的配置文件。