2. 第二章 文件系统的那些事

2.1.1. 文件系统的层次结构

生产环境的服务器中使用的基本上是 linux 操作系统，而操作系统中会涉及到进程模块，内存管理和文件系统等各大模块，那么对于一个正常的文件系统，如常见的 ext4 和 xfs 这些，一个普通的读写会经历哪些层次呢？这些都是非常值得关心的内容，下面请先看一张图。

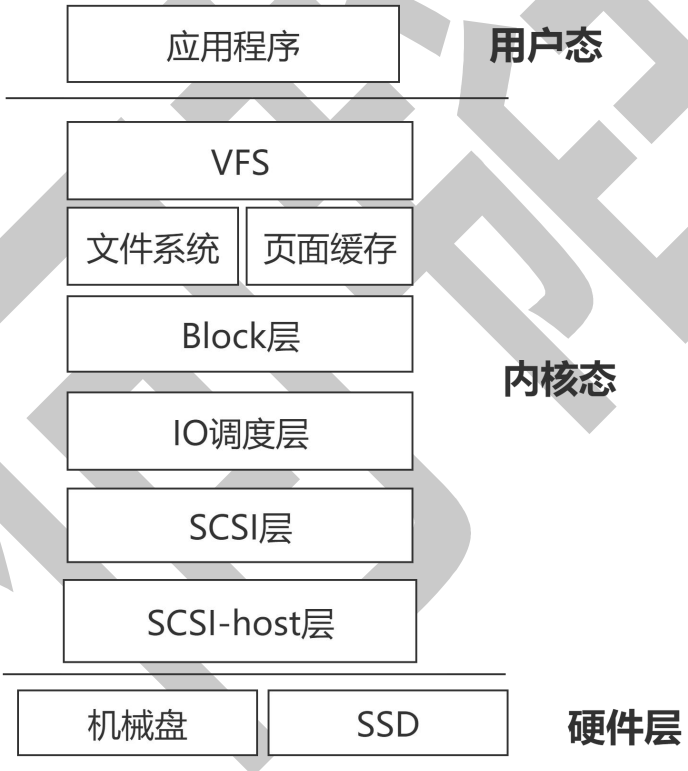

图 2.1-1 linux 文件系层次图

对于一个正常的应用服务，如果要读写本地磁盘数据，会向本地的文件系统发起请求，例如使用 mkdir 或者 touch 这样的 shell 命令进行操作，那么首先会经过一个叫 VFS 层，这个其实是 linux 操作系统中对于文件系统的一个规划，也就是说，不管是自定义的操作系

统，还是一些 ext4 这些，都需要遵守的一些规范，而 VFS 中最核心的元素就是 inode,dentry,superblock 和 file 四个元素。

inode 是负责记录每个文件的一些元数据信息的(在 linux 系统中，一切皆文件，不管是目录还是 devices 设备，都会封装看成一个文件一样调用)。dentry 对象则是用于记录文件的结构关系的，也就是不同目录的上下级层级结构关系和树状关系等。 superblock 就是超级块，用于管理整个操作系统中 inode 资源整体使用情况等，如果把文件系统比作一个图书馆，那么图书馆里面的每一本书就是 block，而书的分类和标签信息这些就是 inode,superblock 就是统计整理整个图书馆的资源情况的。file 对象就是用于记录一下每一本书的租借情况，对于文件来说，这个文件是否被打开过，是否产生了文件句柄 fd(所谓的文件句柄可以理解为，在 file 对象和打开的进程的数据结构中做了一些标记)等。

定义了 VFS 的规范之后，所有的文件系统的都要使用这四个元素进行管理文件系统，而每一个文件系统，在注册到操作系统里面时，还需要定义该文件系统的 fileOptions(这里可以理解为，每个文件系统的一些操作如何定义与具体实现，例如打开一个文件时，是否使用缓存，或者实现一些特殊的操作定义，在 glusterfs 中也有该实现，叫做 FOP)。 那么有了这些 FOP 之后，一个正常的读写请求就会到达该文件系统，根据调用的函数不同，这里就会区分到底请求是否使用缓存了，也就是是否使用操作系统的缓存，而使用这个缓存的优势与缺点也是非常明显的，如果想达到快速读取，那么如果缓存中有该数据，就不需要从磁盘中进行读取了，但是对于写入来说，尤其是数据库的一些写入，因为很多在应用的内存里面已经做了一次缓存，因此并不需要再次做缓存，而且数据库是对于写入 IO 是比较敏感的服务，为了加快写入速度，通常会不使用操作系统的缓存的。

不管是否使用缓存，对于一个读写请求来说，最后都是要到达 block 层的，而这一层的出现，其实就是为了封装所有的读写请求为一个 bio 对象。而为什么需要 block 层，一

个很重要的原因就是，操作系统上面，可能会有多个不同的文件系统，不管任何文件系统最后想要对磁盘进行读写之前，都要进行统一的格式封装，而这就是 block 层的 bio 对象所需要做的事情了，同时在这里，对于读写请求来说，还有考虑能否合并请求，如果两个请求的写入磁盘位置是前后关联的，那么这里就可以进行合并操作了，如果请求可以进行合并的话，那么封装后的 bio 对象就会传递给调度队列了，接下来就是常见的调度算法进行操作了，有 Deadline 和 CFQ 等常见的算法。

当然对于 glusterfs 来说，也实现了一套 fuse，GlusterFS 是一个用户空间文件系统。GluserFS 开发人员选择这种方法是为了避免在 Linux 内核中使用模块。下图将简单展示一下 glusterfs 的 fuse，结合 linux 文件系统的内容，其实两者是有非常多的共同点的。

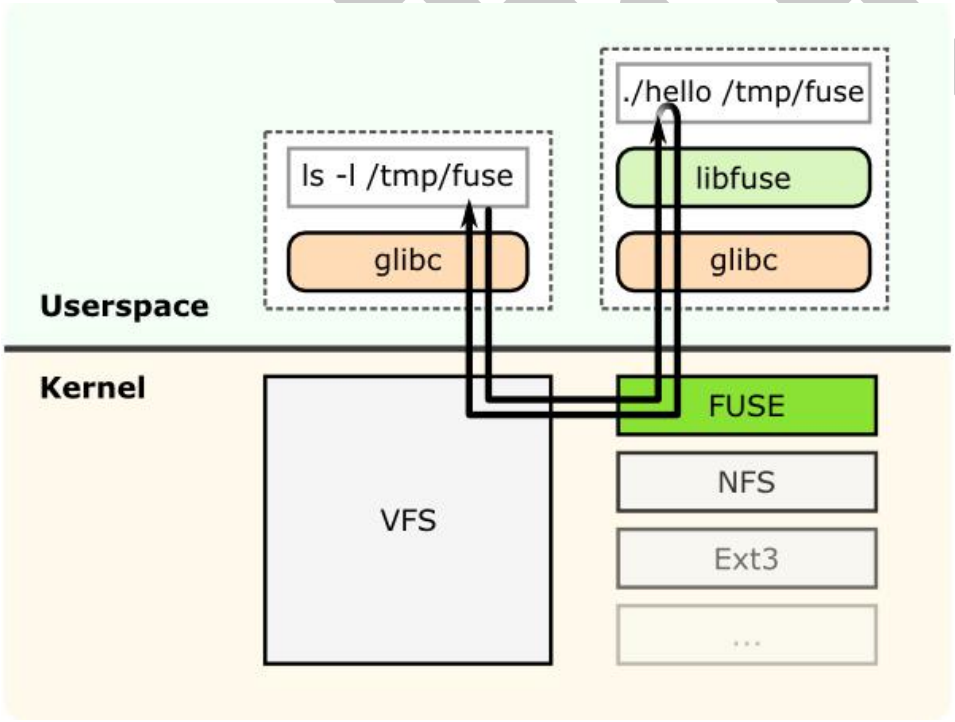

图 2.1-2 glusterfs fuse 架构图

接下来，将结合代码，具体介绍一下上面提到的内容，该部分内容是为了让大家更好地理解一些概念，如果已经熟悉该部分内容，可以跳过阅读。