# 前言



邓仕军

dengshijun1992@gmail.com

**Lustre2.8.0源码学习笔记**

本文是针对Lustre2.8.0，linux-3.10.0-327.3.1.el7的学习笔记，综合了前人已有的工作和自己归纳的部分。

一般分为两栏，左侧为分析主体部分，右侧为分析辅助和注释部分。

# 1 Lustre简要介绍

## 1.1 Exascale文件系统：Lustre的前世今生

去年底的“HPC TOP500只是竞赛 异构计算志在数据中心”一文中，我们谈到了Xeon Phi（至强融核）协处理器在**Exascale（百亿亿次）**计算上的应用。本文要讨论的则是支持Exascale的文件系统。在写了四篇SSD（链接为最后发布的那个）、一篇存储服务器/JBOD和一篇Xeon E5相关的IDF2013报道之后，我想尝试下自己相对不太熟悉的软件领域——可以说专门**针对高性能计算的Lustre文件系统**。排名前100位HPC项目中超过40%项目采用Lustre。由于这也是我学习的一个过程，所以如有错误之处还望读者朋友不吝指出。

|  |  |
| --- | --- |
| Lustre：Exascale文件系统的前世今生 | 根据资料，Lustre项目从**1999年起始**，2003年成立CFS公司并发布1.0版本。2007年开始属于**Sun**，并于2009年发布版本1.8。2010年，随着Sun被**Oracle**收购而加入甲骨文，发布了2.0版本。同一年，Lustre团队应该是从Oracle独立出来，并成立了Whamcloud公司，建立**OpenSFS和EOFS两个开源社区**。去年，**Whamcloud被Intel收购**，之后发布了2.3版。  我在本次大会的预热中曾经提到，收购Whamcloud获得Lustre文件系统，只是**Intel HPC布局的一部分**。去年他们还先后收购了QLogic的InfiniBand业务和CRAY的一部分HPC互连技术，并推出Xeon Phi协处理器来与NVIDIA等厂商的GPGPU竞争。  收购Whamcloud之后，Intel成立了**高性能数据部（HPDD）**并保留了100%的工程师，继续提供Lustre的最主要研发工作。同时继续保持开放性，并持续稳定的发展。  Lustre：Exascale文件系统的前世今生  Intel支持Lustre作为一个**开源**文件系统继续发展，与多供应商和世界范围内的开源社区继续合作。他们认为存储系统是所有HPC软件发展的基石，而且Lustre是全球**HPC领域中扩展性最好的文件系统**。 |
| Lustre：Exascale文件系统的前世今生 | 针对高性能计算应用的特点，Lustre文件系统**I/O聚合带宽最高可达700GB/s**，支持**大量用户并发访问**，**全局命名空间**便于共享。任何可以**运行LinuxOS并具有块设备的服务器，都可以安装运行**Lustre。  当然Lustre也有不擅长的地方，作为一种**元数据路径与数据路径分离**的文件系统，它**不适合用于小数据块I/O**（比如OLTP交易型应用）场合。  由于协议和接口的标准化，Lustre的服务器和存储可以来自不同厂商，用户可以通过增加数据服务器逐步**动态的扩充**文件系统以提升聚合带宽和整体容量。  尽管Lustre是基于GPL的开源文件系统，但**Intel可以销售专业的技术支持服务**来获利。与Hadoop、OpenStack等开源产品一样，Lustre**对使用者或者集成商调试、优化要求的门槛也比较高**，技术水平不够的话可是容易给自己挖坑的。 |
| Lustre：Exascale文件系统的前世今生 | 在Lustre生态环境中，**存储厂商**包括DDN、戴尔、EMC、富士通、HDS、LSI、NetApp、Xyratex等，**IBM和惠普**不在其中是因为他们分别有**类似用途的GPFS和IBRIX**文件系统；“运算”一栏就是**HPC制造商**了，其中**CRAY、SGI和BULL**等我们也不陌生了；**集成商**当中也包括红帽（**RedHat**）；最右边的“研发”其实指的是**用户/计算服务提供商**，其中有笔者熟悉的**TACC**（德州高级计算中心）和**NASA**（美国宇航局）。 |

## 1.2 Lustre体系结构的模块视图

|  |  |
| --- | --- |
|  | * MDS（元数据服务器）提供元数据服务。相应的，MDC（元数据客户端）则是这些服务的客户端。每个文件系统的一个MDS管理一个元数据目标（MDT）。MDT存储诸如文件名、文件夹结构、访问权限之类的文件元数据。 * MGS（管理服务器）提供Lustre文件系统的配置信息。 * OSS（对象存储服务器）expose块设备并提供数据。对应的OSC（对象存储客户端）则是这些服务的客户端。每个OSS管理一个或者多个对象存储目标（OSTs），而OSTs存储文件数据对象。   MDS/MGS和OSS/OST的集合有时称为Lustre服务前端（Lustreserver fronts），而fsfilt和ldiskfs则被称为Lustre服务后端（Luster server backends）。在接下来的探讨中，我们从Lustre的客户端开始，顺着数据和控制路线，一直到OST和MDS。这些探讨牵涉到很多的部件，为使其结构关系。  Lustre作为一个遵从POSIX标准的文件系统，为用户提供了诸如open()、read()、write()等统一的文件系统接口。在Linux中，这些接口是通过虚拟文件系统（Virtual File System,VFS）层实现的（在BSD/Solaris中，则称为vnode层）。为了提供这些接口，Lustre中有一个称为llite的薄层，与VFS相连。接着，到达llite的文件操作请求，通过整个的Lustre软件栈来访问Lustre文件系统，如Figure 2所示。  在Lustre中，诸如创建、打开、读等一般的文件操作，都需要存储在MDS上的元数据信息。这些服务通过一个称为MDC的客户端接口模块来访问。  从MDS的观点来看，每个文件都是分条（stripe）在一个或者多个OST上的多个数据对象的集合。一个文件的布局（layout）信息在索引节点（inode）的扩展属性（extended attribute, EA）中定义。从本质上说，EA描述了从文件对象ID到它对应的OST之间的映射关系。这些信息成为分条扩展属性（striping EA）。  例如，如果一个文件A的分条数目为3，那么它的EA可能类似于：  EA ---> <obj id x, ost p>  <obj id y, ost q>  <obj id z, ost r> |
| http://hi.csdn.net/attachment/201203/4/0_1330828666VCV6.gif | 所以如果分条大小是1MB，那么这意味着[0, 1M), [3M, 4M) …作为对象x存储在OST p；[1M, 2M),  [4M, 5M) …作为对象y存储在OST q; [2M,3M),  [5M, 6M) …作为对象z存储在OST r。  在读取文件之前，客户端将首先通过MDC询问MDS，从而得知它应当就这个操作，和<ost p, ost q, ostr>进行对话。这些信息组织在所谓的LSM中，而客户端的LOV（logical object volume，逻辑对象卷）则是用来解释这个信息的，这样就使得客户端能够向OST发送请求。需要重申的是，客户端通过一个称为OSC的客户端模块接口，和OST进行通信。根据上下文的不同，OSC也可以用来指称OSS客户端。  在Lustre中的所有的客户端/服务器通信，都以RPC请求和应答的形式编码。在Lustre源码中，这个中间层称为Portal RPC，或者ptl-rpc。它在文件系统请求和与之等价的PRC请求和应答之间，进行翻译和解释，而LNET模块则最后将这些RPC请求/应答发送到传输线上。  **OSS**  在OSS栈的底层，是我们所熟悉的LNET和Portal-RPC层。和客户端的栈一样，Portal RPC也翻译请求。需要注意的是，由OSS处理的请求是数据请求，而不是元数据请求。元数据请求应当由MDS栈来传递和处理，正如Figure 2中最右栏所示。  溯栈而上，OST像一个发报机（dispatcher?）一样工作：它根据请求的类型,调用不同的函数。宽泛地说，有两类请求：关乎锁的和关乎数据的。前者将被传递到ldlm（Lustredistributed lock manager）进行处理，而后者将被传递道obdfilter。obdfilter可以说是用来在Lustre栈和常规的OS栈之间相互联系的模块。通过另外一个称为fsfilt的封装API模块（wrapper API component）的帮助，它定义了一个一般性API，从而将Lustre特有的请求翻译为后端文件系统特有的请求。从概念上说，fsfilt就像一个VFS层，如果你在其中定义了适当的文件操作，它将使用这个特定的文件系统作为后端；在Lustre中，这个后端文件系统暂时为ldiskfs。将来，也将支持ZFS作为后端文件系统，而 fsfilt则可能被重新设计或者被替代为一个更好的与文件系统无关的中间层。  **MDS**  MDS软件栈与OSS的软件栈类似，但是它们之间也有一些不同。主要的不同是MDS没有obdfilter，如图2所示。这个发报机被称为MDS。当然它不仅仅只作为一个发报机，这将在第6部分作进一步分析。对于一个元数据变更请求，MDS的日志有一点不同：它在直接调用VFS API之前开始一个事务。这一特定的部件块可以称为dcache，因为它主要关心对dentrycache进行的操作，但是在大的框架上，它实际上是VFS的一部分。 |

## 1.3 Lustre I/O性能特点与最佳实践

### 1.3.1 Lustre Stripe

Lustre采用对象存储技术，将大文件分片并以类似RAID0的方式分散存储在多个OST上，一个文件对应多个OST上的对象。Lustre系统中，每个文件对应MDT上的一个元数据文件，inode以扩展属性记录了数据分片布局信息，包括stripe\_count（对象数）, stripe\_size（分片大小）, stripe\_offset（起始OST）以及每个OST对象信息。当客户数据端访问文件时，首先从MDS请求文件元数据并获得分片布局信息（stripe layout），然后直接与多个OST同时交互进行并发读写。Lustre这种数据分片策略，提高了多用户访问的并发度和聚合I/O带宽，这是Lustre获得高性能的主要因素。再者，Stripe还能够使得Lustre可以存储超大文件，突破单一OST对文件大小的限制。当然，数据分片策略同时也会带来负面影响，比如增加系统负载和数据风险。

Lustre的OST数量可以达到数千，但是出于复杂性、性能、实际存储需求等考虑，目前设计实现中将单个文件对象数限制为160个。对于EXT4后端文件系统，单个文件最大可达2TB，因此Lustre单个文件最大可以达到320TB。那么，Lustre如何在可用OST集合中选择合适的OST呢？目前有两种选择算法，即Round-Robin和随机加权算法，这两种算法调度的依据是，任意两个OST剩余存储容量相差是否超过20%的阈值。一般在系统使用之初，直接使用Round-Robin算法以顺序轮转方式选择OST，这种算法非常高效。随着文件数据量的增加，一旦达到20%的阈值，Lustre将启用随机加权算法选择OST。Lustre维护着一个剩余空间的优先列表，采用随机算法在此列表中选择OST，这种算法会产生开销并影响性能。如果任意两个OST剩余存储容量相差重新降到20%阈值之内，则重新启用Round-Robin算法选择OST。Lustre在创建文件时就按照分片模式并采用OST选择算法，预先创建好文件所需的OST对象。分片模式可以使用lfs setstripe进行设置，或者由系统自动选择缺省模式，文件目录会自动继承父目录的分片模式，但可以进行修改。数据写入后，文件分片模式就不能修改，新加入的OST只会参与新创建的文件目录OST选择调度。Lustre目前还没有实现OST存储空间的自动均衡，需要手工进行数据迁移复制达到均衡的效果。

Lustre缺省情况下，stripe\_count = 1, stripe\_size = 1MB, stripe\_offset = -1，即每个文件仅包含一个OST对象，分片大小为1MB，起始OST由Lustre自动选择。实际上这种分片模式就是不对文件进行分片存储，显然不能满足许多应用的存储需求，实际应用时需要在分析数据特点、网络环境、访问行为的基础上进行适当配置。分片不是越多越好，在满足存储需求的前提下，应该使得OST对象数量尽可能少。应用lustre Stripe时，应该考虑如下因素：

1. 提供高带宽访问。Lustre文件分片并存储于多个OSS，对于单一大文件来说，它可以提供远大于单一OSS提供的聚合I/O带宽。在HPC环境中，成百上千的客户端会同时并发读写同一个文件，当文件很大时，分散与多个OSS能够获得非常高的聚合带宽。Lustre文件系统理论上可以提供2.5 TB/s的带宽，经过验证的带宽达到240 GB/s。当然对于小于1GB的文件来说，分片数量不宜多于4个，更多分片不会带来更高的性能提升，还会引入额外开销。对于小文件，文件大小本身可能小于分片大小，实际上是不作分片，对性能不会有提升。
2. 改善性能。如果聚合的客户端带宽超过单个OSS的带宽，文件分片存储策略可以充分利用聚合的OSS带宽，极大提高性能，为应用程序提供高速的数据读写访问。合理的分片数量可以估算，客户端聚合I/O带宽除以单个OSS I/O性能即可得到。
3. 提供超大容量文件。Lustre后端文件系统采用改进的EXT3文件系统（接近于EXT4），单个文件最大为2TB。如果不进行分片，则单个Lustre文件最大只能为2TB。Lustre目前分片最多可达到160个，因此文件最大可以达到320TB，这是容量是非常大的，基本上可以满足所有单一文件存储容量的需求。
4. 提高存储空间利用率。当Lustre剩余存储空间有限时，每个OSS的剩余空间也就更加有限，这时再写入一个的大文件至单一OSS很大可能会由于空间不足而失败。采用分片策略，写入单个OSS的对象容量会成倍减小，如果OSS数量选择合适，文件仍然可以写入Lustre系统。这使得Lustre存储空间利用更为充分，有效提高了利用率。
5. 增加负载。Stripe会导致额外的锁和网络操作消耗，比如stat, unlink，虽然这些操作可以并发执行，但仍会对性能产生影响。另外，分片多会造成服务器的开销。设想这样一个情形:Lustre中有100个OSS，100个客户端，100个文件，每个客户端访问一个文件。如果不分片，则每个客户端仅与一个OSS相互，可以进行顺序I/O读写。如果每个文件分成100片，则每个客户端都需要分别与100个OSS进行相交，并发访问时，OSS上的磁盘I/O为随机读写。这些都是额外的负载开销，一定程度上影响性能。
6. 增加风险。从概率的角度看，多个OSS发生故障的概率要高出单个OSS许多。文件分片存储于多个OSS上，一个分片不可用就会导致整个文件不可访问，即使其他分片仍然是完好的。因此，分片大大增加了数据发生丢失的风险，需要采用适当的措施进行保护，比如RAID5/6或者Failover。

### 1.3.2 Lustre I/O性能特征

* 写性能优于读性能

Lustre系统中通常写性能会优于读性能。首先，对于写操作，客户端是以异步方式执行的，RPC调用分配以及写入磁盘顺序按到达顺序执行，可以实现聚合写以提高效率。而对于读，请求可能以不同的顺序来自多个客户端，需要大量的磁盘seek与read操作，显著影响吞吐量。其次，目前Lustre没有实现OST read cache，仅仅在客户端实现了Readahead。这样的设计也是有充分理由的，每个OST有可能会有大量客户端并发访问，如果进行数据预读，内存消耗将会非常大，而且这个是不可控制的。Writecache是在客户端上实现的，内存占用不会太大并且是可控的。再者，对于TCP/IP网络而言，读会占用更多的CPU资源。读操作，Lustre需要从网络接口缓存进行数据Copy而获得所需数据，而写操作可以通过sendfile或Zero Copy避免额外的数据复制。

* 大文件性能表现好

Lustre的元数据与数据分离、数据分片策略、数据缓存和网络设计非常适合大文件顺序I/O访问，大文件应用下性能表现非常好。这些设计着眼于提高数据访问的并行性，实现极大的聚合I/O带宽，这其中关键得益于数据分片设计（具体见上面的分析）。另外，后端改进的EXT3文件系统本身也非常适合大文件I/O。

* 小文件性能表现差

然而，Lustre的设计却非常不利于小文件I/O，尤其是LOSF（Lots of small files）。Lustre在读写文件前需要与MDS交互，获得相关属性和对象位置信息。与本地文件系统相比，增加了一次额外的网络传输和元数据访问开销，这对于小文件I/O而言，开销是相当大的。对于大量频繁的小文件读写，Lustre客户端Cache作用会失效，命中率大大降低。如果文件小于物理页大小，则还会产生额外的网络通信量，小文件访问越频繁开销越大，对Lustre总体I/O性能影响就越大。OST后端采用改进的EXT3文件系统，它对小文件的读写性能本身就不好，其元数据访问效率不高，磁盘寻址延迟和磁盘碎片问题严重。这也是大多数磁盘文件系统的缺点，Reiserfs是针对小文件设计的文件系统，性能表现要好很多。Lustre的设计决定了它对小文件I/O性能表现差，实际I/O带宽远低于所提供的最大带宽。在4个OSS的千兆网络配置下，单一客户端小文件读写性能不到4MB/s。

### 1.3.3 Lustre小文件优化

实际上前面已经提到，Lustre并不适合小文件I/O应用，性能表现非常差。因此，建议不要将Lustre应用于LOSF场合。不过，Lustre操作手册仍然给出了一些针对小文件的优化措施。

1. 通过应用聚合读写提高性能，比如对小文件进行Tar，或创建大文件或通过loopback mount来存储小文件。小文件系统调用开销和额外的I/O开销非常大，应用聚合优化可以显著提高性能。另外，可以使用多节点、多进程/多线程尽可能通过聚合来提高I/O带宽。
2. 应用采用O\_DIRECT方式进行直接I/O，读写记录大小设置为4KB，与文件系统保持一致。对输出文件禁用locking，避免客户端之间的竞争。
3. 应用程序尽量保证写连续数据，顺序读写小文件要明显优于随机小文件I/O。
4. OST采用SSD或更多的磁盘，提高IOPS来改善小文件性能。创建大容量OST，而非多个小容量OST，减少日志、连接等负载。
5. OST采用RAID 1+0替代RAID 5/6，避免频繁小文件I/O引起的数据校验开销。

Lustre提供了强大的系统监控与控制接口用于进行性能分析与调优，对于小文件I/O，也可以通过调整一些系统参数进行优化。

1. 禁用所有客户端LNET debug功能：缺省开启多种调试信息，sysctl -w lnet.debug=0，减少系统开销，但发生错误时将无LOG可询。
2. 增加客户端Dirty Cache大小：lctl set\_param osc./\*.max\_dirty\_mb=256，缺省为32MB，增大缓存将提升I/O性能，但数据丢失的风险也随之增大。
3. 增加RPC并行数量：echo 32 > /proc/fs/lustre/osc/\*-OST000\*/max\_rpcs\_in\_flight，缺省为8，提升至32将提高数据和元数据性能。不利之处是如果服务器压力很大，可能反而会影响性能。
4. 控制Lustre striping：lfs setstripe -c 0/1/-1 /path/filename，如果OST对象数大于1，小文件性能会下降，因此将OST对象设置为1。
5. 客户端考虑使用本地锁：mount -t lustre -o localflock，如果确定多个进程从同一个客户端进行写文件，则可用localflock代替flock，减少发送到MDS的RPC数量。
6. 使用loopback mount文件：创建大Lustre文件，与loop设备关联并创建文件系统，然后将其作为文件系统进行mount。小文件作用其上，则原先大量的MDS元数据操作将转换为OSS读写操作，消除了元数据瓶颈，可以显著提高小文件性能。这种方法应用于scratch空间可行，但对于生产数据应该谨慎使用，因为Lustre目前工作在这种模式下还存在问题。操作方法如下：

dd if=/dev/zero of=/mnt/lustre/loopback/scratch bs=1048576 count=1024

losetup /dev/loop0 /mnt/lustre/loopback/scratch

mkfs -t ext4 /dev/loop0

mount /dev/loop0 /mnt/losf

### 1.3.4 Lustre I/O最佳实践

Lustre具有鲜明的I/O特点，并具有非常高的扩展性和大文件I/O性能。如果进行适当的配置和操作，Lustre则会展现更高的性能。下面给出一些Lustre I/O最佳实践，可根据实际应用情况择优实践。

1. 使用单进程读取完整的共享小文件，需要时传输数据至其他进程。
2. 使用单进程访问容量在(1MB, 1GB)之间的小文件，将文件OST对象数设为1。
3. 使用单进程访问大于1GB的中等文件，文件OST对象数不超过4个。
4. 远大于1GB的大文件OST对象数应设为＞4，这种文件不要采用顺序I/O或file-per-process的I/O访问模式。
5. 限制单个目录下的文件数量，包含大量小文件的目录stripe\_count设置为1。
6. 小文件存放在单一OST上，单进程文件创建和读写性能会得到提高。
7. 包含大量小文件的目录存放在单一OST上，文件创建性能会提到极大提升。
8. 尽量避免频繁地打开和关闭文件。
9. 仅在必要时使用ls -l，它会与所有相关的OST交互，操作代价很大，尽可能使用ls和lfs find代替。
10. 考虑使用I/O中间件来改善性能，如ADIOS、HDF5、MPI-IO。

# 2 Lustre文件系统注册过程

在分析lustre源码的过程中，将遵循文件系统的实现过程层层深入进行分析。注意，下面涉及到的函数都是框架性分析，并没有涉及到每个细节。

1. 有些虚拟文件系统（如pipe、共享内存区等），要由内核通过kern\_mount()安装，而根本不允许由用户进程通过系统调用mount()来安装。这样的文件系统类型在其fs\_flag中的FS\_NOMOUNT标志位为1。虚拟文件系统类型的“设备”其实没有超级块，所以只是按特定的内容初始化，或者说生成一个super\_block结构。对于这种文件系统类型，系统调用mount()时应出错返回。
2. 一般的文件系统类型要求有物理的设备作为其物质基础，在其fs\_flags中的FS\_REQUIRES\_DEV标志位为1，这些就是“正常”的文件系统类型，如ext2、minix、ufs等等。对于这些文件系统类型，通过get\_sb\_bdev()从待安装设备上读入其超级块。
3. 有些虚拟文件系统在安装了同类型中的第一个“设备”，从而创建了超级块的super\_block数据结构以后，再安装同一类型中的其他设备时就共享已经存在的super\_block结构，而不再有其自己的超级块结构。此时相应file\_system\_type结构的fs\_flags中的FS\_SINGLE标志位为1，表示整个文件系统只有一个超级块，而不像一般的文件系统类型那样每个具体的设备上都有一个超级块。
4. 还有些文件系统类型的fs\_flags中的FS\_NOMOUNT标志位、FS\_REQUIRE\_DEV标志位以及FS\_SINGLE标志位全都为0，所以不属于上列三种情形中的任何一种。这些所谓“文件系统”其实也是虚拟的，通常只是用来实现某种机制或者规程，所以根本就没有“设备”。对于这样的“文件系统类型”都是通过get\_sb\_nodev()来生成一个super\_block结构的。

总之，每种文件系统类型都有个file\_system\_type结构，而结构中的fs\_flags则由各种标志位组成，这些标志位表明了具体文件系统类型的特性，也决定着这种文件系统的安装过程。

## 2.1 从file\_system\_type出发

|  |  |
| --- | --- |
| Lustre在osd\_mount.c中定义文件系统类型lustre\_fs\_type，并在其中声明了get\_sb 和kill\_sb函数。  代码位置/lustre/obdclass/obd\_mount.c  在obd\_mount.c中有相应的lustre\_register\_fs和lustre\_unregister\_fs：  **int lustre\_register\_fs(void)**  {  return register\_filesystem(&lustre\_fs\_type);  }  **int lustre\_unregister\_fs(void)**  {  return unregister\_filesystem(&lustre\_fs\_type);  }  register\_filesystem就已经进入Linux内核函数层面了，下面对**register\_filesystem**进行简要的分析：  static struct file\_system\_type \*\*find\_filesystem(const char \*name, unsigned len)  {  struct file\_system\_type \*\*p;  // file\_systems是定义在fs/filesystem.c中的一个全局变量static struct file\_system\_type \*file\_systems;  for (p=&file\_systems; \*p; p=&(\*p)->next)  if (strlen((\*p)->name) == len &&  strncmp((\*p)->name, name, len) == 0)  break;  return p;//如果名为name的文件系统在系统中存在，返回指向该文件系统的指针的指针，也即file\_system\_type类型二级指针；否则返回一个struct file\_system\_type \*\*类型的p，其中\*p为空，实际上也就是新文件系统的插入位置  }  /\*\*  \* register\_filesystem – register a new filesystem  \* @fs: the file system structure  \*  \* Adds the file system passed to the list of file systems the kernel  \* is aware of for mount and other syscalls. Returns 0 on success,  \* or a negative errno code on an error.  \*  \* The &struct file\_system\_type that is passed is linked into the kernel  \* structures and must not be freed until the file system has been  \* unregistered.  \*/  int register\_filesystem**(**struct file\_system\_type **\*** fs**)**  **{**  int res **=** 0**;**  struct file\_system\_type **\*\*** p**;**  BUG\_ON**(**strchr**(**fs**->**name**,** '.'**));**  **if** **(**fs**->**next**)**  **return** **-**EBUSY**;**  write\_lock**(&**file\_systems\_lock**);**  // 遍历全局file\_systems链表，尝试查找本次要注册文件系统名。  p **=** find\_filesystem**(**fs**->**name**,** strlen**(**fs**->**name**));**  **if** **(\***p**)** // 如果不为NULL，则说明找到了重名的文件系统。注册失败。  res **=** **-**EBUSY**;**  **else** // 如果返回NULL，说明已经到链表结尾，可以注册此文件系统。  **\***p **=** fs**;** // 将此文件系统链接到链表结尾。  write\_unlock**(&**file\_systems\_lock**);**  **return** res**;**  **}** | lustre/obdlcass/class\_obd.c  static int \_\_init obdclass\_init(void)  {  初始化操作  lustre\_register\_fs();  }  lustre/obdlcass/class\_obd.c  module\_init(obdclass\_init);  static struct file\_system\_type lustre\_fs\_type **=**  **{**  **.**owner **=** THIS\_MODULE**,**  **.**name **=** "lustre"**,**  #ifdef HAVE\_FSTYPE\_MOUNT  **.**mount **=** lustre\_mount**,**  #else  **.**get\_sb **=** lustre\_get\_sb**,**  #endif  **.**kill\_sb **=** lustre\_kill\_super**,**  **.**fs\_flags **=** FS\_REQUIRES\_DEV **|** FS\_HAS\_FIEMAP **|** FS\_RENAME\_DOES\_D\_MOVE**,**  **};**  注意这里要和上面是find\_filesystem函数结合起来看 |

|  |  |
| --- | --- |
| 关于lustre\_fs\_type，不同的版本，对应的成员函数不同（mount和get\_sb），注意lustre\_fs\_type的定义部分  代码位置/lustre/obdclass/obd\_mount.c  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FS registration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #ifdef HAVE\_FSTYPE\_MOUNT  static struct dentry \*lustre\_mount(struct file\_system\_type \*fs\_type, int flags,  const char \*devname, void \*data)  {  struct lustre\_mount\_data2 lmd2 = { data, NULL };  return mount\_nodev(fs\_type, flags, &lmd2, lustre\_fill\_super);  }  #else  static int lustre\_get\_sb(struct file\_system\_type \*fs\_type, int flags,  const char \*devname, void \*data, struct vfsmount \*mnt)  {  struct lustre\_mount\_data2 lmd2 = { data, mnt };  return get\_sb\_nodev(fs\_type, flags, &lmd2, lustre\_fill\_super, mnt);  }  #endif  mount\_bdev()函数   * fs\_type不用多说，携带file\_system\_type的信息，这里传递它主要是因为它携带了super block的链表和很多锁变量。 * flags文件系统的通用挂载选项。 * dev\_name是mount操作时的设备名，如/dev/sda1。后面会用到这个设备名找到对应的设备信息，从而从中获得super block。 * data是挂载时指定的挂载选项信息。// include/uapi/linux/fs.h中定义的   详情参见http://blog.csdn.net/zr\_lang/article/details/40002285  fill\_super是一个由某个fs特定实现的fill\_super方法，用来根据文件系统的特性解析mount data并继续填充super block的字段，并且初始化挂载点的根索引节点对象和目录项对象。 | static struct file\_system\_type lustre\_fs\_type = {  .owner = THIS\_MODULE,  .name = "lustre",  #ifdef HAVE\_FSTYPE\_MOUNT  .mount = lustre\_mount,  #else  .get\_sb = lustre\_get\_sb,  #endif  .kill\_sb = lustre\_kill\_super,  .fs\_flags = FS\_REQUIRES\_DEV | FS\_HAS\_FIEMAP | FS\_RENAME\_DOES\_D\_MOVE,  };  参见lustre/autoconf/lustre-core.m4的说明  #  # LC\_HAVE\_FSTYPE\_MOUNT  #  # 2.6.39 replace get\_sb with mount in struct file\_system\_type  #  AC\_DEFUN([LC\_HAVE\_FSTYPE\_MOUNT], [  LB\_CHECK\_COMPILE([if 'file\_system\_type' has 'mount' field],  file\_system\_type\_mount, [  #include <linux/fs.h>  ],[  struct file\_system\_type fst;  void \*i = (void \*) fst.mount;  ],[  AC\_DEFINE(HAVE\_FSTYPE\_MOUNT, 1,  [struct file\_system\_type has mount field])  ])  ]) # LC\_HAVE\_FSTYPE\_MOUNT  可以看到，Lustre在版本高于2.6.39时，走lustre\_mount-> mount\_nodev调用路径，否则走get\_sb -> get\_sb\_nodev调用路径。 |

|  |  |
| --- | --- |
| **get\_sb\_nodev的实现（linux内核2.6.35，不是本版本讨论的主线）**  **int** **get\_sb\_nodev(struct file\_system\_type \*fs\_type, int flags, void \*data,**  **int (\*fill\_super)(struct super\_block \*, void \*, int),**  **struct vfsmount \*mnt)**  {  int error;  struct super\_block \*s = sget(fs\_type, NULL, set\_anon\_super, NULL);  if (IS\_ERR(s))  return PTR\_ERR(s);  s->s\_flags = flags;  error = fill\_super(s, data, flags & MS\_SILENT ? 1 : 0);  if (error) {  up\_write(&s->s\_umount);  deactivate\_super(s);  return error;  }  s->s\_flags |= MS\_ACTIVE;  return simple\_set\_mnt(mnt, s);  }  在这里我们不能不谈get\_sb\_bdev和get\_sb\_nodev的区别(我只写出这连个函数的主要逻辑)：这里的分析主要参考http://blog.csdn.net/dog250/archive/2010/02/09/5302909.aspx  **int get\_sb\_bdev(struct file\_system\_type \*fs\_type, int flags, const char \*dev\_name, void \*data, int (\*fill\_super)(struct super\_block \*, void \*, int),struct vfsmount \*mnt)**  **{**  struct block\_device \*bdev;  struct super\_block \*s;  bdev = open\_bdev\_exclusive(dev\_name, mode, fs\_type);// 这里和底层的驱动和通用磁盘gendisk联系起来  down(&bdev->bd\_mount\_sem);  s =sget(fs\_type, test\_bdev\_super, set\_bdev\_super, bdev);//这里将bdev设置进fs\_type相关的super\_block，因为之前的open\_bdev\_exclusive中已打开，所以bdev和gendisk联系上了  sget用于在fs\_type对应的fs\_supers链表中查找或者创建一个super\_block，这里的test\_bdev\_super用于在fs\_supers链表中查找时比较两个是否已经存在该bdev对应的super，set\_bdev\_super则用于设置新创建的super\_block。  up(&bdev->bd\_mount\_sem);  if (s->s\_root) {  对应于“文件系统的根目录项已经存在”的情况，参见http://alanwu.blog.51cto.com/3652632/1105681  }else{ /\* 文件系统根目录项不存在，通过filler\_super函数读取磁盘上的superblock元数据信息，并且初始化superblock内存结构 \*/  sb\_set\_blocksize(s, block\_size(bdev));  error = fill\_super(s, data, flags & MS\_SILENT ? 1 : 0);//调用回调函数，填充超级块  s->s\_flags |= MS\_ACTIVE;  }  return simple\_set\_mnt(mnt, s);  }  **int get\_sb\_nodev(struct file\_system\_type \*fs\_type,**  **int flags, void \*data,**  **int (\*fill\_super)(struct super\_block \*, void \*, int),**  **struct vfsmount \*mnt)**  **{**  int error;  struct super\_block \*s = sget(fs\_type, NULL, set\_anon\_super, NULL);//  s->s\_flags = flags;  error = fill\_super(s, data, flags & MS\_SILENT ? 1 : 0);  s->s\_flags |= MS\_ACTIVE;  return simple\_set\_mnt(mnt, s);  } | **关于mount\_nodev的实现（linux 内核3.10）**  注意将上面的参数对应过来mount\_nodev(fs\_type, flags, &lmd2, lustre\_fill\_super);  struct dentry **\***mount\_nodev**(**struct file\_system\_type **\***fs\_type**,**  int flags**,** void **\***data**,**  int **(\***fill\_super**)(**struct super\_block **\*,** void **\*,** int**))**  **{**  int error**;**  **//** **find or create a superblock**  struct super\_block **\***s **=** sget**(**fs\_type**,** **NULL,** set\_anon\_super**,** flags**,** **NULL);**  **if** **(**IS\_ERR**(**s**))**  **return** ERR\_CAST**(**s**);**  error **=** fill\_super**(**s**,** data**,** flags **&** MS\_SILENT **?** 1 **:** 0**);**  **if** **(**error**)** **{**  deactivate\_locked\_super**(**s**);**  **return** ERR\_PTR**(**error**);**  **}**  s**->**s\_flags **|=** MS\_ACTIVE**;**  **return** dget**(**s**->**s\_root**);**  **}** |

|  |  |
| --- | --- |
| **关于lustre\_fill\_super**  lustre\_fill\_super是两条路径都会使用的函数，分析如下：  /\*\* This is the entry point for the mount call into Lustre.  \* This is called when a server or client is mounted,  \* and this is where we start setting things up.  \* @param data Mount options (e.g. -o flock,abort\_recov)  \*/  static int lustre\_fill\_super**(**struct super\_block **\***sb**,** void **\***data**,** int silent**)**  **{**  struct lustre\_mount\_data **\***lmd**;**  struct lustre\_mount\_data2 **\***lmd2 **=** data**;**  struct lustre\_sb\_info **\***lsi**;**  int rc**;**  ENTRY**;**  CDEBUG**(**D\_MOUNT**|**D\_VFSTRACE**,** "VFS Op: sb %p\n"**,** sb**);**  lsi **=** lustre\_init\_lsi**(**sb**);**  **if** **(!**lsi**)**  RETURN**(-**ENOMEM**);**  lmd **=** lsi**->**lsi\_lmd**;**  /\*  \* Disable lockdep during mount, because mount locking patterns are  \* `special'.  \*/  lockdep\_off**();**  /\*  \* LU-639: the obd cleanup of last mount may not finish yet, wait here.  \*/  obd\_zombie\_barrier**();**  /\* Figure out the lmd from the mount options \*/  **if** **(**lmd\_parse**((**char **\*)(**lmd2**->**lmd2\_data**),** lmd**))**  **{**  lustre\_put\_lsi**(**sb**);**  GOTO**(**out**,** rc **=** **-**EINVAL**);**  **}**  **if** **(**lmd\_is\_client**(**lmd**))**  **{**  CDEBUG**(**D\_MOUNT**,** "Mounting client %s\n"**,** lmd**->**lmd\_profile**);**  **if** **(**client\_fill\_super **==** **NULL)**  request\_module**(**"lustre"**);**  **if** **(**client\_fill\_super **==** **NULL)//** 注释①  **{**  LCONSOLE\_ERROR\_MSG**(**0x165**,** "Nothing registered for "  "client mount! Is the 'lustre' "  "module loaded?\n"**);**  lustre\_put\_lsi**(**sb**);**  rc **=** **-**ENODEV**;**  **}**  **else**  **{**  rc **=** lustre\_start\_mgc**(**sb**);**  **if** **(**rc**)**  **{**  lustre\_put\_lsi**(**sb**);**  GOTO**(**out**,** rc**);**  **}**  /\* Connect and start \*/  /\* (should always be ll\_fill\_super) \*/  rc **=** **(\***client\_fill\_super**)(**sb**,** lmd2**->**lmd2\_mnt**);**  /\* c\_f\_s will call lustre\_common\_put\_super on failure \*/  **}**  **}**  **else**  **{**  #ifdef HAVE\_SERVER\_SUPPORT  CDEBUG**(**D\_MOUNT**,** "Mounting server from %s\n"**,** lmd**->**lmd\_dev**);**  rc **=** server\_fill\_super**(**sb**);**  /\* s\_f\_s calls lustre\_start\_mgc after the mount because we need  the MGS nids which are stored on disk. Plus, we may  need to start the MGS first. \*/  /\* s\_f\_s will call server\_put\_super on failure \*/  #else  CERROR**(**"This is client-side-only module, "  "cannot handle server mount.\n"**);**  rc **=** **-**EINVAL**;**  #endif  **}**  /\* If error happens in fill\_super() call, @lsi will be killed there.  \* This is why we do not put it here. \*/  GOTO**(**out**,** rc**);**  out**:**  **if** **(**rc**)**  **{**  CERROR**(**"Unable to mount %s (%d)\n"**,**  s2lsi**(**sb**)** **?** lmd**->**lmd\_dev **:** ""**,** rc**);**  **}**  **else**  **{**  CDEBUG**(**D\_SUPER**,** "Mount %s complete\n"**,**  lmd**->**lmd\_dev**);**  **}**  lockdep\_on**();**  **return** rc**;**  **}** | lustre\_mount函数中第一句：struct lustre\_mount\_data2 lmd2 = { data, NULL };  注释① client\_fill\_super是在obd\_mount.c中定义的全局变量，主要负责填充lustre client super\_block，同样的还有kill\_super\_cb。cient\_fill\_super和kill\_super\_cb分别在lustre\_register\_client\_fill\_super和lustre\_register\_kill\_super\_cb中被初始化，见下面代码。而lustre\_register\_client\_fill\_super和lustre\_register\_kill\_super\_cb都是在lustre/lite/super25.c中的init\_lustre\_lite中被调用。调用形式如下：  lustre\_register\_client\_fill\_super(ll\_fill\_super);  lustre\_register\_kill\_super\_cb(ll\_kill\_super);  所以这里的client\_fill\_super就是ll\_fill\_super。  启动mgc  server\_fill\_super**(**sb**)**  server\_start\_mgs(sb);  lustre\_start\_mgc(sb);  server\_start\_targets(sb); |

/\* We can't call ll\_fill\_super by name because it lives in a module that

must be loaded after this one. \*/

**void lustre\_register\_client\_fill\_super(int (\*cfs)(struct super\_block \*sb))**

**{**

client\_fill\_super = cfs;

}

**void lustre\_register\_kill\_super\_cb(void (\*cfs)(struct super\_block \*sb))**

**{**

kill\_super\_cb = cfs;

}

至此，我们**总结**一下lustre mount和填充super\_block的过程。

**cfs\_module(obdclass, LUSTRE\_VERSION\_STRING, init\_obdclass, cleanup\_obdclass);**

**----init\_obdclass（void）**

**------lustre\_register\_fs（void）**

**--------register\_filesystem(&lustre\_fs\_type)**

**----------lustre\_get\_sb //这里针对内核的不同版本，有两种实现，参数不同，返回值也不同**

**------------get\_sb\_nodev/mount\_nodev //在两个函数参数中均传递了super\_block填充函数lustre\_fill\_super**

**--------------lustre\_fill\_super//注意这里的lustre\_fill\_super是一个接口，区别对待client和server，并分别调用client\_fill\_super（其实是ll\_fill\_super）和server\_fill\_super**

下面我们看一下ll\_fill\_super（lustre\llite\llite\_lib.c）的实现：

|  |  |
| --- | --- |
| **int** **ll\_fill\_super(struct super\_block \*sb)**  **{**  struct lustre\_profile \*lprof;  struct lustre\_sb\_info \*lsi = s2lsi(sb);  struct ll\_sb\_info \*sbi;  char \*dt = NULL, \*md = NULL;  char \*profilenm = get\_profile\_name(sb);// (s2lsi(sb)->lsi\_lmd->lmd\_profile)  struct config\_llog\_instance cfg = {0, };  char ll\_instance[sizeof(sb) \* 2 + 1];  int err;  ENTRY;  CDEBUG(D\_VFSTRACE, "VFS Op: sb %p\n", sb);  cfs\_module\_get();  /\* client additional sb info \*/  lsi->lsi\_llsbi = sbi = ll\_init\_sbi();  if (!sbi) {  cfs\_module\_put(THIS\_MODULE);  RETURN(-ENOMEM);  }  err = ll\_options(lsi->lsi\_lmd->lmd\_opts, &sbi->ll\_flags);//注意，前面在**lustre\_fill\_super中，lsi->lsi\_lmd是由lmd\_parse解析的，这里由ll\_options解析**lsi->lsi\_lmd->lmd\_opts，这个解析出来的参数是为client的  if (err)  GOTO(out\_free, err);  /\* Generate a string unique to this super, in case some joker tries  to mount the same fs at two mount points.  Use the address of the super itself.\*/  sprintf(ll\_instance, "%p", sb);  cfg.cfg\_instance = ll\_instance;  cfg.cfg\_uuid = lsi->lsi\_llsbi->ll\_sb\_uuid;  /\* set up client obds （关于该函数的分析最终没有走完，以后再说吧）\*/  err = lustre\_process\_log(sb, profilenm, &cfg);//见右边分析  if (err < 0) {  CERROR("Unable to process log: %d\n", err);  GOTO(out\_free, err);  }  /\* Profile set with LCFG\_MOUNTOPT so we can find our mdc and osc obds \*/  lprof = class\_get\_profile(profilenm);//get luster\_profile with given name @profilenm  if (lprof == NULL) {  LCONSOLE\_ERROR\_MSG(0x156, "The client profile '%s' could not be"  " read from the MGS. Does that filesystem "  "exist?\n", profilenm);  GOTO(out\_free, err = -EINVAL);  }  CDEBUG(D\_CONFIG, "Found profile %s: mdc=%s osc=%s\n", profilenm,  lprof->lp\_md, lprof->lp\_dt);  //分别初始化dt和md（两者都是字符串）  OBD\_ALLOC(dt, strlen(lprof->lp\_dt) +strlen(ll\_instance) + 2);  if (!dt)  GOTO(out\_free, err = -ENOMEM);  sprintf(dt, "%s-%s", lprof->lp\_dt, ll\_instance);  OBD\_ALLOC(md, strlen(lprof->lp\_md) +strlen(ll\_instance) + 2);  if (!md)  GOTO(out\_free, err = -ENOMEM);  sprintf(md, "%s-%s", lprof->lp\_md, ll\_instance);  /\* connections, registrations, sb setup \*/  err = client\_common\_fill\_super(sb, md, dt);  out\_free:  if (md)  OBD\_FREE(md, strlen(md) + 1);  if (dt)  OBD\_FREE(dt, strlen(dt) + 1);  if (err)  ll\_put\_super(sb);  else  LCONSOLE\_WARN("Client %s has started\n", profilenm);  RETURN(err);  } /\* ll\_fill\_super \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* config llog \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  struct lustre\_cfg {  \_\_u32 lcfg\_version;  \_\_u32 lcfg\_command;  \_\_u32 lcfg\_num;  \_\_u32 lcfg\_flags;  \_\_u64 lcfg\_nid;  \_\_u32 lcfg\_nal; /\* not used any more \*/  \_\_u32 lcfg\_bufcount;  \_\_u32 lcfg\_buflens[0];  };  struct lustre\_cfg\_bufs {  void \*lcfg\_buf[LUSTRE\_CFG\_MAX\_BUFCOUNT];  \_\_u32 lcfg\_buflen[LUSTRE\_CFG\_MAX\_BUFCOUNT];  \_\_u32 lcfg\_bufcount;  }; | struct lustre\_profile {  cfs\_list\_t lp\_list;  char \*lp\_profile;  char \*lp\_dt;  char \*lp\_md;  };  struct lustre\_sb\_info {  int lsi\_flags;  struct obd\_device \*lsi\_mgc; /\* mgc obd \*/  struct lustre\_mount\_data \*lsi\_lmd; /\* mount command info \*/  struct lustre\_disk\_data \*lsi\_ldd; /\* mount info on-disk \*/  struct ll\_sb\_info \*lsi\_llsbi; /\* add'l client sbi info \*/  struct vfsmount \*lsi\_srv\_mnt; /\* the one server mount \*/  cfs\_atomic\_t lsi\_mounts; /\* references to the srv\_mnt \*/  };  /\*\* Get a config log from the MGS and process it.  \* This func is called for both clients and servers.  \* Continue to process new statements appended to the logs  \* (whenever the config lock is revoked) until lustre\_end\_log  \* is called.  \* @param sb The superblock is used by the MGC to write to the local copy of the config log  \* @param logname The name of the llog to replicate from the MGS  \* @param cfg Since the same mgc may be used to follow multiple config logs  \* (e.g. ost1, ost2, client), the config\_llog\_instance keeps the state for  \* this log, and is added to the mgc's list of logs to follow.  \*/  **int lustre\_process\_log(struct super\_block \*sb, char \*logname,**  **struct config\_llog\_instance \*cfg)**  **{**  struct lustre\_cfg \*lcfg;  struct lustre\_cfg\_bufs bufs;  struct lustre\_sb\_info \*lsi = s2lsi(sb);  struct obd\_device \*mgc = lsi->lsi\_mgc;  int rc;  ENTRY;  /\* mgc\_process\_config \*/  //设置bufs[0]为mgc->obd\_name  lustre\_cfg\_bufs\_reset(&bufs, mgc->obd\_name); //见下面分析  //设置bufs[1]为logname  lustre\_cfg\_bufs\_set\_string(&bufs, 1, logname); //见下面分析  lustre\_cfg\_bufs\_set(&bufs, 2, cfg, sizeof(\*cfg));//见下面分析  lustre\_cfg\_bufs\_set(&bufs, 3, &sb, sizeof(sb));  lcfg = lustre\_cfg\_new(LCFG\_LOG\_START, &bufs); //见下面分析，根据bufs和cmd来初始化lcfg  rc = obd\_process\_config(mgc, sizeof(\*lcfg), lcfg); //见下面分析，结合下面的obd\_process\_config我们可知，这里使用的是mgc\_process\_config，下面分析  lustre\_cfg\_free(lcfg);  RETURN(rc);  }  //置bufs为全0，并为bufs->lcfg\_buf[0]和bufs->lcfg\_buflen[0]赋值  **static inline void lustre\_cfg\_bufs\_reset(struct lustre\_cfg\_bufs \*bufs, char \*name)**  **{**  memset((bufs), 0, sizeof(\*bufs));  if (name)  lustre\_cfg\_bufs\_set\_string(bufs, 0, name);  }  static inline void lustre\_cfg\_bufs\_set\_string(struct lustre\_cfg\_bufs \*bufs,  \_\_u32 index,char \*str)  {  lustre\_cfg\_bufs\_set(bufs, index, str, str ? strlen(str) + 1 : 0);  }  //为bufs->lcfg\_buf[index]和bufs->lcfg\_buflen[index]赋值  static inline void lustre\_cfg\_bufs\_set(struct lustre\_cfg\_bufs \*bufs,\_\_u32 index,  void \*buf, \_\_u32 buflen)  {  if (index >= LUSTRE\_CFG\_MAX\_BUFCOUNT)  return;  if (bufs == NULL)  return;  if (bufs->lcfg\_bufcount <= index)  bufs->lcfg\_bufcount = index + 1;  bufs->lcfg\_buf[index] = buf;  bufs->lcfg\_buflen[index] = buflen;  }  **static inline struct lustre\_cfg \*lustre\_cfg\_new(int cmd,struct lustre\_cfg\_bufs \*bufs)**  **{**  struct lustre\_cfg \*lcfg;  char \*ptr;  int i;  ENTRY;  OBD\_ALLOC(lcfg, lustre\_cfg\_len(bufs->lcfg\_bufcount,bufs->lcfg\_buflen));  lcfg->lcfg\_version = LUSTRE\_CFG\_VERSION;  lcfg->lcfg\_command = cmd;  lcfg->lcfg\_bufcount = bufs->lcfg\_bufcount;  ptr = (char \*)lcfg + LCFG\_HDR\_SIZE(lcfg->lcfg\_bufcount);//见下面，获取lcfg\_buflens[(lcfg->lcfg\_bufcount)]在lustre\_cfg中的偏移（这个偏移是对齐处理了的）  for (i = 0; i < lcfg->lcfg\_bufcount; i++) {  lcfg->lcfg\_buflens[i] = bufs->lcfg\_buflen[i];  LOGL((char \*)bufs->lcfg\_buf[i], bufs->lcfg\_buflen[i], ptr);//拷贝，见下面，注意在上面的OBD\_ALLOC中为lcfg留了足够的空间哦  }  RETURN(lcfg);  }  **static inline int lustre\_cfg\_len(\_\_u32 bufcount, \_\_u32 \*buflens)**  **{**  int i;  len = LCFG\_HDR\_SIZE(bufcount);  for (i = 0; i < bufcount; i++)  len += cfs\_size\_round(buflens[i]);  RETURN(cfs\_size\_round(len));  }  **#define LCFG\_HDR\_SIZE(count) \**  cfs\_size\_round(offsetof (struct lustre\_cfg, lcfg\_buflens[(count)]))  **#define LOGL(var,len,ptr)**  \  do { \  if (var) \  memcpy((char \*)ptr, (const char \*)var, len); \  ptr += cfs\_size\_round(len); \  } while (0)  **static inline int obd\_process\_config(struct obd\_device \*obd, int datalen, void \*data)**  **{**  int rc;  DECLARE\_LU\_VARS(ldt, d);//声明两个变量lu\_device\_type \*ldt和lu\_device \*d；  ENTRY;  OBD\_CHECK\_DEV(obd);  obd->obd\_process\_conf = 1;//device is processing mgs config  ldt = obd->obd\_type->typ\_lu;//get the lu\_device\_type  d = obd->obd\_lu\_dev;//get the lu\_device  if (ldt != NULL && d != NULL) {  struct lu\_env env;  rc = lu\_env\_init(&env, ldt->ldt\_ctx\_tags);  if (rc == 0) {  rc = d->ld\_ops->ldo\_process\_config(&env, d, data);//这里我们需要谈一下ldo\_process\_config，见下面。  lu\_env\_fini(&env);  }  } else {  OBD\_CHECK\_DT\_OP(obd, process\_config, -EOPNOTSUPP);  rc = OBP(obd, process\_config)(obd, datalen, data);  }  OBD\_COUNTER\_INCREMENT(obd, process\_config);  obd->obd\_process\_conf = 0;  RETURN(rc);  } |

根据lcfg->lcfg\_command的值，会调用很多函数

🡪class\_setup: obd->obd\_starting = 1;

lcfg = lustre\_cfg\_new(cmd, &bufs);

do\_lcfg(obdname,0, CFG\_SETUP, s1, s2, s3, s4);

cmd: LCFG\_SETUP / LCFG\_ATTACH /…

mgc\_fs\_setup

本地的一些设置

mgc\_target\_register

client\_common\_fill\_super(sb, md, dt, mnt);

obd\_set\_info\_async

d\_make\_root(root);

server\_fill\_super\_common(sb);

server\_start\_targets(sb);

server\_start\_mgs(sb);//调用的函数类似于

lustre\_start\_mgc(sb);

osd\_start(lsi, b->s\_flags);

ptlrpc\_request\_alloc(imp, RQF\_MDS\_CONNECT);

…...

ptlrpcd\_add\_req(request);

ptlrpc\_connect\_import

ptlrpc\_first\_transno

import\_select\_connection

从lustre\_fill\_super开始，调用关系图

lustre\_fill\_super

lustre\_start\_mgc(sb);

(\*client\_fill\_super)(sb, md2->lmd2\_mnt);

实质是ll\_fill\_super函数

server\_fill\_super(sb);

lustre\_start\_simple(mgcname, LUSTRE\_MGC\_NAME,

(char \*)uuid->uuid, LUSTRE\_MGS\_OBDNAME, niduuid, NULL, NULL);

call class\_attach and class\_setup. These methods in turn call obd type-specific methods

obd\_connect(NULL, &exp, obd, &(obd->obd\_uuid), data, NULL);

OBP(obd, connect)(env, exp, obd, cluuid, data, localdata);

//#define OBP(dev, op) (dev)->obd\_type->typ\_dt\_ops->o\_ ## op

client\_connect\_import

注释①

class\_process\_config(lcfg);

class\_attach(lcfg);

obd = class\_newdev(typename, name);

obd\_devs[i] = result; // obd\_devs是一个全局变量

OBP(exp->exp\_obd, set\_info\_async)(env, exp, keylen, key, vallen,val, set);

mgc\_set\_info\_async

注释①

lustre\_start\_simple

server\_mgc\_set\_fs

server\_register\_target

server\_register\_mount

|  |  |
| --- | --- |
| 注释① | echo.c (lustre/obdecho): .o\_connect = echo\_connect,  echo\_client.c (lustre/obdecho): .o\_connect = echo\_client\_connect,  ldlm\_lib.c (lustre/ldlm):/\* ->o\_connect() method for client side (OSC and MDC and MGC) \*/  lmv\_obd.c (lustre/lmv): .o\_connect = lmv\_connect,  lod\_dev.c (lustre/lod): \* Implementation of obd\_ops::o\_connect() for LOD  lod\_dev.c (lustre/lod): .o\_connect = lod\_obd\_connect,  lov\_obd.c (lustre/lov): .o\_connect = lov\_connect,  lwp\_dev.c (lustre/osp): \* Implementation of OBD device operations obd\_ops::o\_connect.  lwp\_dev.c (lustre/osp): .o\_connect = lwp\_obd\_connect,  mdc\_request.c (lustre/mdc): .o\_connect = client\_connect\_import,  mdd\_device.c (lustre/mdd): .o\_connect = mdd\_obd\_connect,  mdt\_handler.c (lustre/mdt): .o\_connect = mdt\_obd\_connect,  mgc\_request.c (lustre/mgc): .o\_connect = client\_connect\_import,  mgs\_handler.c (lustre/mgs): .o\_connect = mgs\_obd\_connect,  obd.h (lustre/include): int (\*o\_connect)(const struct lu\_env \*env,  ofd\_obd.c (lustre/ofd): .o\_connect = ofd\_obd\_connect,  osc\_request.c (lustre/osc): .o\_connect = client\_connect\_import,  osd\_handler.c (lustre/osd-ldiskfs): .o\_connect = osd\_obd\_connect,  osd\_handler.c (lustre/osd-zfs): .o\_connect = osd\_obd\_connect,  osp\_dev.c (lustre/osp): \* Implementation of obd\_ops::o\_connect  osp\_dev.c (lustre/osp): .o\_connect = osp\_obd\_connect,  qmt\_dev.c (lustre/quota): .o\_connect = qmt\_device\_obd\_connect,  module\_init(mgc\_init);  module\_exit(mgc\_exit); |
|  |  |
|  |  |

|  |  |
| --- | --- |
| /\*\*  \* Attempt to (re)connect import \a imp. This includes all preparations,  \* initializing CONNECT RPC request and passing it to ptlrpcd for  \* actual sending.  \* Returns 0 on success or error code.  \*/  int **ptlrpc\_connect\_import(**struct obd\_import **\***imp**)**  **{**  struct obd\_device **\***obd **=** imp**->**imp\_obd**;**  int initial\_connect **=** 0**;**  int set\_transno **=** 0**;**  \_\_u64 committed\_before\_reconnect **=** 0**;**  struct ptlrpc\_request **\***request**;**  char **\***bufs**[]** **=** **{** **NULL,**  obd2cli\_tgt**(**imp**->**imp\_obd**),**  obd**->**obd\_uuid**.**uuid**,**  **(**char **\*)&**imp**->**imp\_dlm\_handle**,**  **(**char **\*)&**imp**->**imp\_connect\_data  **};**  struct ptlrpc\_connect\_async\_args **\***aa**;**  int rc**;**  ENTRY**;**  spin\_lock**(&**imp**->**imp\_lock**);**  **if** **(**imp**->**imp\_state **==** LUSTRE\_IMP\_CLOSED**)**  **{**  spin\_unlock**(&**imp**->**imp\_lock**);**  CERROR**(**"can't connect to a closed import\n"**);**  RETURN**(-**EINVAL**);**  **}**  **else** **if** **(**imp**->**imp\_state **==** LUSTRE\_IMP\_FULL**)**  **{**  spin\_unlock**(&**imp**->**imp\_lock**);**  CERROR**(**"already connected\n"**);**  RETURN**(**0**);**  **}**  **else** **if** **(**imp**->**imp\_state **==** LUSTRE\_IMP\_CONNECTING**)**  **{**  spin\_unlock**(&**imp**->**imp\_lock**);**  CERROR**(**"already connecting\n"**);**  RETURN**(-**EALREADY**);**  **}**  IMPORT\_SET\_STATE\_NOLOCK**(**imp**,** LUSTRE\_IMP\_CONNECTING**);**  imp**->**imp\_conn\_cnt**++;**  imp**->**imp\_resend\_replay **=** 0**;**  **if** **(!**lustre\_handle\_is\_used**(&**imp**->**imp\_remote\_handle**))**  initial\_connect **=** 1**;**  **else**  committed\_before\_reconnect **=** imp**->**imp\_peer\_committed\_transno**;**  ptlrpc\_first\_transno获得第一个事务编号  set\_transno **=** ptlrpc\_first\_transno**(**imp**,**  **&**imp**->**imp\_connect\_data**.**ocd\_transno**);**  spin\_unlock**(&**imp**->**imp\_lock**);**  rc **=** import\_select\_connection**(**imp**);**  **if** **(**rc**)**  GOTO**(**out**,** rc**);**  rc **=** sptlrpc\_import\_sec\_adapt**(**imp**,** **NULL,** **NULL);**  **if** **(**rc**)**  GOTO**(**out**,** rc**);**  /\* Reset connect flags to the originally requested flags, in case  \* the server is updated on-the-fly we will get the new features. \*/  imp**->**imp\_connect\_data**.**ocd\_connect\_flags **=** imp**->**imp\_connect\_flags\_orig**;**  /\* Reset ocd\_version each time so the server knows the exact versions \*/  imp**->**imp\_connect\_data**.**ocd\_version **=** LUSTRE\_VERSION\_CODE**;**  imp**->**imp\_msghdr\_flags **&=** **~**MSGHDR\_AT\_SUPPORT**;**  imp**->**imp\_msghdr\_flags **&=** **~**MSGHDR\_CKSUM\_INCOMPAT18**;**  rc **=** obd\_reconnect**(NULL,** imp**->**imp\_obd**->**obd\_self\_export**,** obd**,**  **&**obd**->**obd\_uuid**,** **&**imp**->**imp\_connect\_data**,** **NULL);**  **if** **(**rc**)**  GOTO**(**out**,** rc**);**  request **=** ptlrpc\_request\_alloc**(**imp**,** **&**RQF\_MDS\_CONNECT**);**  **if** **(**request **==** **NULL)**  GOTO**(**out**,** rc **=** **-**ENOMEM**);**  rc **=** ptlrpc\_request\_bufs\_pack**(**request**,** LUSTRE\_OBD\_VERSION**,**  imp**->**imp\_connect\_op**,** bufs**,** **NULL);**  **if** **(**rc**)**  **{**  ptlrpc\_request\_free**(**request**);**  GOTO**(**out**,** rc**);**  **}**  /\* Report the rpc service time to the server so that it knows how long  \* to wait for clients to join recovery \*/  lustre\_msg\_set\_service\_time**(**request**->**rq\_reqmsg**,**  at\_timeout2est**(**request**->**rq\_timeout**));**  /\* The amount of time we give the server to process the connect req.  \* import\_select\_connection will increase the net latency on  \* repeated reconnect attempts to cover slow networks.  \* We override/ignore the server rpc completion estimate here,  \* which may be large if this is a reconnect attempt \*/  request**->**rq\_timeout **=** INITIAL\_CONNECT\_TIMEOUT**;**  lustre\_msg\_set\_timeout**(**request**->**rq\_reqmsg**,** request**->**rq\_timeout**);**  request**->**rq\_no\_resend **=** request**->**rq\_no\_delay **=** 1**;**  request**->**rq\_send\_state **=** LUSTRE\_IMP\_CONNECTING**;**  /\* Allow a slightly larger reply for future growth compatibility \*/  req\_capsule\_set\_size**(&**request**->**rq\_pill**,** **&**RMF\_CONNECT\_DATA**,** RCL\_SERVER**,**  **sizeof(**struct obd\_connect\_data**)+**16**\*sizeof(**\_\_u64**));**  ptlrpc\_request\_set\_replen**(**request**);**  request**->**rq\_interpret\_reply **=** ptlrpc\_connect\_interpret**;**  CLASSERT**(sizeof** **(\***aa**)** **<=** **sizeof** **(**request**->**rq\_async\_args**));**  aa **=** ptlrpc\_req\_async\_args**(**request**);**  memset**(**aa**,** 0**,** **sizeof** **\***aa**);**  aa**->**pcaa\_peer\_committed **=** committed\_before\_reconnect**;**  aa**->**pcaa\_initial\_connect **=** initial\_connect**;**  **if** **(**aa**->**pcaa\_initial\_connect**)**  **{**  spin\_lock**(&**imp**->**imp\_lock**);**  imp**->**imp\_replayable **=** 1**;**  spin\_unlock**(&**imp**->**imp\_lock**);**  lustre\_msg\_add\_op\_flags**(**request**->**rq\_reqmsg**,**  MSG\_CONNECT\_INITIAL**);**  **}**  **if** **(**set\_transno**)**  lustre\_msg\_add\_op\_flags**(**request**->**rq\_reqmsg**,**  MSG\_CONNECT\_TRANSNO**);**  DEBUG\_REQ**(**D\_RPCTRACE**,** request**,** "(re)connect request (timeout %d)"**,**  request**->**rq\_timeout**);**  ptlrpcd\_add\_req**(**request**);**  rc **=** 0**;**  out**:**  **if** **(**rc **!=** 0**)**  IMPORT\_SET\_STATE**(**imp**,** LUSTRE\_IMP\_DISCON**);**  RETURN**(**rc**);**  **}** |  |

关于lu\_device---》lu\_device\_operations ---》ldo\_process\_config（注意这要和obd\_ops---）o\_process\_config区分开来）

Cmm\_device.c (lustre\cmm): .ldo\_process\_config = cmm\_process\_config,

Lovsub\_dev.c (lustre\lov): .ldo\_process\_config = NULL,

Lov\_dev.c (lustre\lov): .ldo\_process\_config = lov\_process\_config,

Mdc\_device.c (lustre\cmm): .ldo\_process\_config = mdc\_process\_config

Mdd\_device.c (lustre\mdd): .ldo\_process\_config = mdd\_process\_config,

Mdt\_handler.c (lustre\mdt): .ldo\_process\_config = mdt\_process\_config,

Osc\_dev.c (lustre\osc): .ldo\_process\_config = osc\_cl\_process\_config,

Osd\_handler.c (lustre\osd-ldiskfs): .ldo\_process\_config = osd\_process\_config,

**lustre\_process\_log 🡪 obd\_process\_config 🡪 mgc\_process\_config（注意我们这里走的只是某条路线，并不意味着obd\_process\_config一定调用mgc\_process\_config，只是在这条主线中是这样的调用关系）。下面分析一下mgc\_process\_config:**

/\*\* Called from lustre\_process\_log.

\* LCFG\_LOG\_START gets the config log from the MGS, processes it to start

\* any services, and adds it to the list logs to watch (follow).

\*/

**static int mgc\_process\_config(struct obd\_device \*obd, obd\_count len, void \*buf)**

**{**

struct lustre\_cfg \*lcfg = buf;

int cmd;

int rc = 0;

ENTRY;

switch(cmd = lcfg->lcfg\_command) {根据不同的config命令，区别处理

case LCFG\_LOV\_ADD\_OBD: {//add an osc to lov

/\* Overloading this cfg command: register a new target \*/

struct mgs\_target\_info \*mti;

if (LUSTRE\_CFG\_BUFLEN(lcfg, 1) !=

sizeof(struct mgs\_target\_info))

GOTO(out, rc = -EINVAL);

mti = (struct mgs\_target\_info \*)lustre\_cfg\_buf(lcfg, 1);

CDEBUG(D\_MGC, "add\_target %s %#x\n",

mti->mti\_svname, mti->mti\_flags);

rc = mgc\_target\_register(obd->u.cli.cl\_mgc\_mgsexp, mti);

break;

}

case LCFG\_LOV\_DEL\_OBD://delete an osc from lov

/\* Unregister has no meaning at the moment. \*/

CERROR("lov\_del\_obd unimplemented\n");

rc = -ENOSYS;

break;

case LCFG\_SPTLRPC\_CONF: {//security

rc = sptlrpc\_process\_config(lcfg);

break;

}

case LCFG\_LOG\_START: {//mgc\_only,process a config log

struct config\_llog\_data \*cld;

struct config\_llog\_instance \*cfg;

struct super\_block \*sb;

//下面3行代码，其实都是解析传递过来的lcfg，获取相应信息（这些信息都是在**lustre\_process\_log---）lustre\_cfg\_new中被设置的**）

char \*logname = lustre\_cfg\_string(lcfg, 1);

cfg = (struct config\_llog\_instance \*)lustre\_cfg\_buf(lcfg, 2);

sb = \*(struct super\_block \*\*)lustre\_cfg\_buf(lcfg, 3);

/\* We're only called through here on the initial mount \*/

//init the config\_llog\_data and add it into the global @config\_llog\_list

rc = config\_log\_add(obd, logname, cfg, sb);//见下面分析

//find a config log by name

cld = config\_log\_find(logname, cfg); //见下面分析

if (IS\_ERR(cld)) {

rc = PTR\_ERR(cld);

break;

}

// Get a config log from the MGS and process it.

rc = mgc\_process\_log(obd, cld);

config\_log\_put(cld);

break;

}

case LCFG\_LOG\_END: {//stop processing updates

struct config\_llog\_instance \*cfg = NULL;

char \*logname = lustre\_cfg\_string(lcfg, 1);

if (lcfg->lcfg\_bufcount >= 2)

cfg = (struct config\_llog\_instance \*)lustre\_cfg\_buf(

lcfg, 2);

rc = config\_log\_end(logname, cfg);

break;

}

default: {

CERROR("Unknown command: %d\n", lcfg->lcfg\_command);

GOTO(out, rc = -EINVAL);

}

}

out:

RETURN(rc);

}

/\* list of active configuration logs \*/

struct config\_llog\_data {

char \*cld\_logname;

struct ldlm\_res\_id cld\_resid;

struct config\_llog\_instance cld\_cfg;

cfs\_list\_t cld\_list\_chain;//link this into the list

cfs\_atomic\_t cld\_refcount;

struct config\_llog\_data \*cld\_sptlrpc;/\* depended sptlrpc log \*/

struct obd\_export \*cld\_mgcexp;

unsigned int cld\_stopping:1, /\* we were told to stop

\* watching \*/

cld\_lostlock:1, /\* lock not requeued \*/

cld\_is\_sptlrpc:1;

};

/\*\* Add this log to the list of active logs watched by an MGC.

\* Active means we're watching for updates.

\* We have one active log per "mount" - client instance or servername.

\* Each instance may be at a different point in the log.

\*/

**static int config\_log\_add(struct obd\_device \*obd, char \*logname,struct config\_llog\_instance \*cfg,struct super\_block \*sb)**

**{**

struct config\_llog\_data \*cld, \*sptlrpc\_cld;

char seclogname[20];

char \*ptr;

ENTRY;

CDEBUG(D\_MGC, "adding config log %s:%s\n", logname, cfg->cfg\_instance);

/\*

\* for each regular log, the depended sptlrpc log name is

\* <fsname>-sptlrpc. multiple regular logs may share one sptlrpc log.

\*/

ptr = strrchr(logname, '-');//找到字符串中最后一个‘-’的位置

if (ptr == NULL || ptr - logname > 8) {

CERROR("logname %s is too long\n", logname);

RETURN(-EINVAL);

}

memcpy(seclogname, logname, ptr - logname);

strcpy(seclogname + (ptr - logname), "-sptlrpc");

//这里为什么要查找呢，这就是应征了前面注释所述的：multiple regular logs may share one sptlrpc log

sptlrpc\_cld = config\_log\_find(seclogname, NULL);// Find a config log by name

if (IS\_ERR(sptlrpc\_cld)) {//-EINVAL或者 –ENOENT表示参数错误或者没找到

sptlrpc\_cld = do\_config\_log\_add(obd, seclogname, 1, NULL, NULL);

if (IS\_ERR(sptlrpc\_cld)) {

CERROR("can't create sptlrpc log: %s\n", seclogname);

RETURN(PTR\_ERR(sptlrpc\_cld));

}

}

//for non-sptlrpc config llog , init the config\_llog\_data @cld with given config\_llog\_instance @cfg,sb,logname and then add it into the global list @config\_llog\_list,here the 3rd para means it is not a sptlrpc config llog

cld = do\_config\_log\_add(obd, logname, 0, cfg, sb);

if (IS\_ERR(cld)) {

CERROR("can't create log: %s\n", logname);

config\_log\_put(sptlrpc\_cld);

RETURN(PTR\_ERR(cld));

}

cld->cld\_sptlrpc = sptlrpc\_cld;

RETURN(0);

}

/\* Find a config log by name \*/

**static**

**struct config\_llog\_data \*config\_log\_find(char \*logname,struct config\_llog\_instance \*cfg)**

**{**

struct config\_llog\_data \*cld;

char \*logid = logname;

int match\_instance = 0;

ENTRY;

if (cfg && cfg->cfg\_instance) {//这里针对的是参数传递指定了cfg的

match\_instance++;

logid = cfg->cfg\_instance;

}

if (!logid) {

CERROR("No log specified\n");

RETURN(ERR\_PTR(-EINVAL));

}

cfs\_spin\_lock(&config\_list\_lock);

//config\_llog\_list是一个全局链表，所有的config\_llog\_data通过cld\_list\_chain域连入其中。

下面的处理是这样的：遍历config\_llog\_list中的所有entry，如果在参数中指定了cfg，则比较 cfg\_instance，否则比较cld\_logname，找到就退出

cfs\_list\_for\_each\_entry(cld, &config\_llog\_list, cld\_list\_chain) {

if (match\_instance && cld->cld\_cfg.cfg\_instance &&

strcmp(logid, cld->cld\_cfg.cfg\_instance) == 0)//如果参数传递了cfg

goto out\_found;

if (!match\_instance &&

strcmp(logid, cld->cld\_logname) == 0)//如果第二个参数为NULL

goto out\_found;

}

cfs\_spin\_unlock(&config\_list\_lock);

CDEBUG(D\_CONFIG, "can't get log %s\n", logid);

RETURN(ERR\_PTR(-ENOENT));

out\_found:

cfs\_atomic\_inc(&cld->cld\_refcount);

cfs\_spin\_unlock(&config\_list\_lock);

LASSERT(cld->cld\_stopping == 0 || cld->cld\_is\_sptlrpc == 0);

RETURN(cld);

}

**static**

**struct config\_llog\_data \*do\_config\_log\_add(struct obd\_device \*obd,**

**char \*logname,unsigned int is\_sptlrpc,**

**struct config\_llog\_instance \*cfg,struct super\_block \*sb)**

{

struct config\_llog\_data \*cld;

int rc;

ENTRY;

CDEBUG(D\_MGC, "do adding config log %s:%s\n", logname,

cfg ? cfg->cfg\_instance : "NULL");

OBD\_ALLOC(cld, sizeof(\*cld));

if (!cld)

RETURN(ERR\_PTR(-ENOMEM));

OBD\_ALLOC(cld->cld\_logname, strlen(logname) + 1);

if (!cld->cld\_logname) {

OBD\_FREE(cld, sizeof(\*cld));

RETURN(ERR\_PTR(-ENOMEM));

}

strcpy(cld->cld\_logname, logname);

if (cfg)

cld->cld\_cfg = \*cfg;

cld->cld\_cfg.cfg\_last\_idx = 0;

cld->cld\_cfg.cfg\_flags = 0;

cld->cld\_cfg.cfg\_sb = sb;

cld->cld\_is\_sptlrpc = is\_sptlrpc;//

cfs\_atomic\_set(&cld->cld\_refcount, 1);

/\* Keep the mgc around until we are done \*/

cld->cld\_mgcexp = class\_export\_get(obd->obd\_self\_export);

if (cfg && cfg->cfg\_instance != NULL) {

OBD\_ALLOC(cld->cld\_cfg.cfg\_instance,

strlen(cfg->cfg\_instance) + 1);

strcpy(cld->cld\_cfg.cfg\_instance, cfg->cfg\_instance);

}

if (is\_sptlrpc) {//这一层逻辑暂时不管，不考虑security rpc

sptlrpc\_conf\_log\_start(logname);

cld->cld\_cfg.cfg\_obdname = obd->obd\_name;

}

rc = mgc\_logname2resid(logname, &cld->cld\_resid);

cfs\_spin\_lock(&config\_list\_lock);

cfs\_list\_add(&cld->cld\_list\_chain, &config\_llog\_list);//add the config\_llog\_data into the global @config\_llog\_list.

cfs\_spin\_unlock(&config\_list\_lock);

if (rc) {

config\_log\_put(cld);

RETURN(ERR\_PTR(rc));

}

if (is\_sptlrpc) {

rc = mgc\_process\_log(obd, cld);

if (rc)

CERROR("failed processing sptlrpc log: %d\n", rc);

}

RETURN(cld);

}

**void sptlrpc\_conf\_log\_start(const char \*logname)**

**{**

struct sptlrpc\_conf \*conf;

char fsname[16];

//NOTE:as this is sptlrpc log,so we know its log name is fsname-sptlrpc,so we can get the fsname from the logname.

if (logname2fsname(logname, fsname, sizeof(fsname)))

return;

cfs\_mutex\_lock(&sptlrpc\_conf\_lock);

conf = sptlrpc\_conf\_get(fsname, 1);

cfs\_mutex\_unlock(&sptlrpc\_conf\_lock);

}

/\*\* Get a config log from the MGS and process it.

\* This func is called for both clients and servers.

\* Copy the log locally before parsing it if appropriate (non-MGS server)

\*/

**int mgc\_process\_log(struct obd\_device \*mgc,**

**struct config\_llog\_data \*cld)**

**{**

struct llog\_ctxt \*ctxt, \*lctxt;

struct lustre\_handle lockh;

struct client\_obd \*cli = &mgc->u.cli;

struct lvfs\_run\_ctxt saved;

struct lustre\_sb\_info \*lsi = NULL;

int rc = 0, rcl, flags = 0, must\_pop = 0;

ENTRY;

LASSERT(cld);

cfs\_down(&llog\_process\_lock);

if (cld->cld\_stopping) {//we were told to stop watching

cfs\_up(&llog\_process\_lock);

RETURN(0);

}

//set timeout against the given id@ OBD\_FAIL\_MGC\_PAUSE\_PROCESS\_LOG

OBD\_FAIL\_TIMEOUT(OBD\_FAIL\_MGC\_PAUSE\_PROCESS\_LOG, 20);

if (cld->cld\_cfg.cfg\_sb)

lsi = s2lsi(cld->cld\_cfg.cfg\_sb);//get the luster\_sb\_info

CDEBUG(D\_MGC, "Process log %s:%s from %d\n", cld->cld\_logname,

cld->cld\_cfg.cfg\_instance, cld->cld\_cfg.cfg\_last\_idx + 1);

// get the llog\_ctxt with obd->obd\_olg and given index and then inc its refcount

ctxt = llog\_get\_context(mgc, LLOG\_CONFIG\_REPL\_CTXT);//见下面分析

if (!ctxt) {

CERROR("missing llog context\n");

cfs\_up(&llog\_process\_lock);

RETURN(-EINVAL);

}

/\* Get the cfg lock on the llog \*/

rcl = mgc\_enqueue(mgc->u.cli.cl\_mgc\_mgsexp, NULL, LDLM\_PLAIN, NULL,

LCK\_CR, &flags, NULL, NULL, NULL,

cld, 0, NULL, &lockh);

if (rcl)

CDEBUG(D\_MGC, "Can't get cfg lock: %d\n", rcl);

//as with the above code,here the index is LLOG\_CONFIG\_ORIG\_CTXT

lctxt = llog\_get\_context(mgc, LLOG\_CONFIG\_ORIG\_CTXT);

/\*

\* local copy of sptlrpc log is controlled elsewhere, don't try to

\* read it up here.

\*/

if (rcl && cld->cld\_is\_sptlrpc)

GOTO(out\_pop, rc);

/\* Copy the setup log locally if we can. Don't mess around if we're

running an MGS though (logs are already local). \*/没有看细节

if (lctxt && lsi && (lsi->lsi\_flags & LSI\_SERVER) &&

(lsi->lsi\_srv\_mnt == cli->cl\_mgc\_vfsmnt) &&

!IS\_MGS(lsi->lsi\_ldd)) {

push\_ctxt(&saved, &mgc->obd\_lvfs\_ctxt, NULL);

must\_pop++;

if (rcl == 0)

/\* Only try to copy log if we have the lock. \*/

rc = mgc\_copy\_llog(mgc, ctxt, lctxt, cld->cld\_logname);

if (rcl || rc) {

if (mgc\_llog\_is\_empty(mgc, lctxt, cld->cld\_logname)) {

LCONSOLE\_ERROR\_MSG(0x13a, "Failed to get MGS "

"log %s and no local copy."

"\n", cld->cld\_logname);

GOTO(out\_pop, rc = -ENOTCONN);

}

CDEBUG(D\_MGC, "Failed to get MGS log %s, using local "

"copy for now, will try to update later.\n",

cld->cld\_logname);

}

/\* Now, whether we copied or not, start using the local llog.

If we failed to copy, we'll start using whatever the old

log has. \*/

llog\_ctxt\_put(ctxt);

ctxt = lctxt;

}

if (cld->cld\_is\_sptlrpc)

sptlrpc\_conf\_log\_update\_begin(cld->cld\_logname);

/\* logname and instance info should be the same, so use our

copy of the instance for the update. The cfg\_last\_idx will

be updated here. \*/

if (rcl == 0 || lctxt == ctxt) //见下面分析

rc = class\_config\_parse\_llog(ctxt, cld->cld\_logname, &cld->cld\_cfg); out\_pop:

llog\_ctxt\_put(ctxt);

if (ctxt != lctxt)

llog\_ctxt\_put(lctxt);

if (must\_pop)

pop\_ctxt(&saved, &mgc->obd\_lvfs\_ctxt, NULL);

/\*

\* update settings on existing OBDs. doing it inside

\* of llog\_process\_lock so no device is attaching/detaching

\* in parallel.

\* the logname must be <fsname>-sptlrpc

\*/

if (cld->cld\_is\_sptlrpc && rcl == 0) {

sptlrpc\_conf\_log\_update\_end(cld->cld\_logname);

class\_notify\_sptlrpc\_conf(cld->cld\_logname,

strlen(cld->cld\_logname) -

strlen("-sptlrpc"));

}

/\* Now drop the lock so MGS can revoke it \*/

if (!rcl) {

rcl = mgc\_cancel(mgc->u.cli.cl\_mgc\_mgsexp, NULL,

LCK\_CR, &lockh);

if (rcl)

CERROR("Can't drop cfg lock: %d\n", rcl);

}

CDEBUG(D\_MGC, "%s: configuration from log '%s' %sed (%d).\n",

mgc->obd\_name, cld->cld\_logname, rc ? "fail" : "succeed", rc);

cfs\_up(&llog\_process\_lock);

RETURN(rc);

}

**static inline struct llog\_ctxt \*llog\_get\_context(struct obd\_device \*obd,int index){**

**//** obd->obd\_olg：**struct obd\_llog\_group类型**

return llog\_group\_get\_ctxt(&obd->obd\_olg, index);

}

**static inline struct llog\_ctxt \*llog\_group\_get\_ctxt(struct obd\_llog\_group \*olg,int index){**

struct llog\_ctxt \*ctxt;

LASSERT(index >= 0 && index < LLOG\_MAX\_CTXTS);

cfs\_spin\_lock(&olg->olg\_lock);

if (olg->olg\_ctxts[index] == NULL) {

ctxt = NULL;

} else {

ctxt = llog\_ctxt\_get(olg->olg\_ctxts[index]);//增加llog\_ctxt引用计数

}

cfs\_spin\_unlock(&olg->olg\_lock);

return ctxt;

}

**static inline struct llog\_ctxt \*llog\_ctxt\_get(struct llog\_ctxt \*ctxt){**

LASSERT(cfs\_atomic\_read(&ctxt->loc\_refcount) > 0);

cfs\_atomic\_inc(&ctxt->loc\_refcount);

CDEBUG(D\_INFO, "GETting ctxt %p : new refcount %d\n", ctxt,

cfs\_atomic\_read(&ctxt->loc\_refcount));

return ctxt;

}

**int class\_config\_parse\_llog(struct llog\_ctxt \*ctxt, char \*name,**

**struct config\_llog\_instance \*cfg)**

**{**

struct llog\_process\_cat\_data cd = {0, 0};

struct llog\_handle \*llh;

int rc, rc2;

ENTRY;

CDEBUG(D\_INFO, "looking up llog %s\n", name);

rc = llog\_create(ctxt, &llh, NULL, name);//见下面分析

if (rc)

RETURN(rc);

rc = llog\_init\_handle(llh, LLOG\_F\_IS\_PLAIN, NULL);

if (rc)

GOTO(parse\_out, rc);

/\* continue processing from where we last stopped to end-of-log \*/

if (cfg)

cd.lpcd\_first\_idx = cfg->cfg\_last\_idx;

cd.lpcd\_last\_idx = 0;

rc = llog\_process(llh, class\_config\_llog\_handler, cfg, &cd);

CDEBUG(D\_CONFIG, "Processed log %s gen %d-%d (rc=%d)\n", name,

cd.lpcd\_first\_idx + 1, cd.lpcd\_last\_idx, rc);

if (cfg)

cfg->cfg\_last\_idx = cd.lpcd\_last\_idx;

parse\_out:

rc2 = llog\_close(llh);

if (rc == 0)

rc = rc2;

RETURN(rc);

}

**static inline int llog\_create(struct llog\_ctxt \*ctxt, struct llog\_handle \*\*res,**

**struct llog\_logid \*logid, char \*name)**

**{**

struct llog\_operations \*lop;

int raised, rc;

ENTRY;

rc = llog\_obd2ops(ctxt, &lop);//ctxt->loc\_logops

if (rc)

RETURN(rc);

if (lop->lop\_create == NULL)

RETURN(-EOPNOTSUPP);

raised = cfs\_cap\_raised(CFS\_CAP\_SYS\_RESOURCE);

if (!raised)

cfs\_cap\_raise(CFS\_CAP\_SYS\_RESOURCE);

rc = lop->lop\_create(ctxt, res, logid, name);

if (!raised)

cfs\_cap\_lower(CFS\_CAP\_SYS\_RESOURCE);

RETURN(rc);

}

**int llog\_init\_handle(struct llog\_handle \*handle, int flags,struct obd\_uuid \*uuid){**

int rc;

struct llog\_log\_hdr \*llh;

ENTRY;

LASSERT(handle->lgh\_hdr == NULL);

OBD\_ALLOC(llh, sizeof(\*llh));

if (llh == NULL)

RETURN(-ENOMEM);

handle->lgh\_hdr = llh;

/\* first assign flags to use llog\_client\_ops \*/

llh->llh\_flags = flags;

rc = llog\_read\_header(handle);//见下面分析

if (rc == 0) {

flags = llh->llh\_flags;

if (uuid && !obd\_uuid\_equals(uuid, &llh->llh\_tgtuuid)) {

CERROR("uuid mismatch: %s/%s\n", (char \*)uuid->uuid,

(char \*)llh->llh\_tgtuuid.uuid);

rc = -EEXIST;

}

GOTO(out, rc);

} else if (rc != LLOG\_EEMPTY || !flags) {

/\* set a pesudo flag for initialization \*/

flags = LLOG\_F\_IS\_CAT;

GOTO(out, rc);

}

rc = 0;

handle->lgh\_last\_idx = 0; /\* header is record with index 0 \*/

llh->llh\_count = 1; /\* for the header record \*/

llh->llh\_hdr.lrh\_type = LLOG\_HDR\_MAGIC;

llh->llh\_hdr.lrh\_len = llh->llh\_tail.lrt\_len = LLOG\_CHUNK\_SIZE;

llh->llh\_hdr.lrh\_index = llh->llh\_tail.lrt\_index = 0;

llh->llh\_timestamp = cfs\_time\_current\_sec();

if (uuid)

memcpy(&llh->llh\_tgtuuid, uuid, sizeof(llh->llh\_tgtuuid));

llh->llh\_bitmap\_offset = offsetof(typeof(\*llh),llh\_bitmap);

ext2\_set\_bit(0, llh->llh\_bitmap);

out:

if (flags & LLOG\_F\_IS\_CAT) {

CFS\_INIT\_LIST\_HEAD(&handle->u.chd.chd\_head);

llh->llh\_size = sizeof(struct llog\_logid\_rec);

} else if (flags & LLOG\_F\_IS\_PLAIN) {

CFS\_INIT\_LIST\_HEAD(&handle->u.phd.phd\_entry);

} else {

CERROR("Unknown flags: %#x (Expected %#x or %#x\n",

flags, LLOG\_F\_IS\_CAT, LLOG\_F\_IS\_PLAIN);

LBUG();

}

if (rc) {

OBD\_FREE(llh, sizeof(\*llh));

handle->lgh\_hdr = NULL;

}

RETURN(rc);

}

**static inline int llog\_read\_header(struct llog\_handle \*handle)//0322**

**{**

struct llog\_operations \*lop;

int rc;

ENTRY;

rc = llog\_handle2ops(handle, &lop);//\*lop=loghandle->lgh\_ctxt->loc\_logops

if (rc)

RETURN(rc);

if (lop->lop\_read\_header == NULL)

RETURN(-EOPNOTSUPP);

rc = lop->lop\_read\_header(handle);

RETURN(rc);

}

**struct lustre\_profile \*class\_get\_profile(const char \* prof)**

**{**

struct lustre\_profile \*lprof;

ENTRY;

cfs\_list\_for\_each\_entry(lprof, &lustre\_profile\_list, lp\_list) {

if (!strcmp(lprof->lp\_profile, prof)) {

RETURN(lprof);

}

}

RETURN(NULL);

}

关于lustre\_profile\_list的增删函数是class\_add\_profile和class\_del\_profile以及class\_del\_profiles。见下面：

/\*\* Create a named "profile".

\* This defines the mdc and osc names to use for a client.

\* This also is used to define the lov to be used by a mdt.

\*/

**int class\_add\_profile(int proflen, char \*prof, int osclen, char \*osc,**

**int mdclen, char \*mdc)**

**{**

struct lustre\_profile \*lprof;

int err = 0;

ENTRY;

OBD\_ALLOC(lprof, sizeof(\*lprof));

if (lprof == NULL)

RETURN(-ENOMEM);

CFS\_INIT\_LIST\_HEAD(&lprof->lp\_list);

LASSERT(proflen == (strlen(prof) + 1));

OBD\_ALLOC(lprof->lp\_profile, proflen);

if (lprof->lp\_profile == NULL)

GOTO(out, err = -ENOMEM);

memcpy(lprof->lp\_profile, prof, proflen);//用@prof初始化lprof->lp\_profile

LASSERT(osclen == (strlen(osc) + 1));

OBD\_ALLOC(lprof->lp\_dt, osclen);

if (lprof->lp\_dt == NULL)

GOTO(out, err = -ENOMEM);

memcpy(lprof->lp\_dt, osc, osclen);

if (mdclen > 0) {

LASSERT(mdclen == (strlen(mdc) + 1));

OBD\_ALLOC(lprof->lp\_md, mdclen);

if (lprof->lp\_md == NULL)

GOTO(out, err = -ENOMEM);

memcpy(lprof->lp\_md, mdc, mdclen);

}

//add into the global list @lustre\_profile\_list

cfs\_list\_add(&lprof->lp\_list, &lustre\_profile\_list);

RETURN(err);

out:

if (lprof->lp\_md)

OBD\_FREE(lprof->lp\_md, mdclen);

if (lprof->lp\_dt)

OBD\_FREE(lprof->lp\_dt, osclen);

if (lprof->lp\_profile)

OBD\_FREE(lprof->lp\_profile, proflen);

OBD\_FREE(lprof, sizeof(\*lprof));

RETURN(err);

}

**void class\_del\_profile(const char \*prof)**

**{**

struct lustre\_profile \*lprof;

ENTRY;

lprof = class\_get\_profile(prof); //get @lprof from the given name @prof

if (lprof) {

cfs\_list\_del(&lprof->lp\_list);

OBD\_FREE(lprof->lp\_profile, strlen(lprof->lp\_profile) + 1);

OBD\_FREE(lprof->lp\_dt, strlen(lprof->lp\_dt) + 1);

if (lprof->lp\_md)

OBD\_FREE(lprof->lp\_md, strlen(lprof->lp\_md) + 1);

OBD\_FREE(lprof, sizeof \*lprof);

}

EXIT;

}

**/\* COMPAT\_146 \*/**

**void class\_del\_profiles(void)//delete all the entries in the global @lustre\_profile\_list**

**{**

struct lustre\_profile \*lprof, \*n;

ENTRY;

cfs\_list\_for\_each\_entry\_safe(lprof, n, &lustre\_profile\_list, lp\_list) {

cfs\_list\_del(&lprof->lp\_list);

OBD\_FREE(lprof->lp\_profile, strlen(lprof->lp\_profile) + 1);

OBD\_FREE(lprof->lp\_dt, strlen(lprof->lp\_dt) + 1);

if (lprof->lp\_md)

OBD\_FREE(lprof->lp\_md, strlen(lprof->lp\_md) + 1);

OBD\_FREE(lprof, sizeof \*lprof);

}

EXIT;

}

**static int client\_common\_fill\_super(struct super\_block \*sb, char \*md, char \*dt)**

**{**

struct inode \*root = 0;

struct ll\_sb\_info \*sbi = ll\_s2sbi(sb);//ll\_sb\_info和lustre\_sb\_info的区别是，前者是llite的super\_block后者是lustre的super\_block.

struct obd\_device \*obd;

struct obd\_capa \*oc = NULL;

struct obd\_statfs osfs;

struct ptlrpc\_request \*request = NULL;

struct obd\_connect\_data \*data = NULL;

struct obd\_uuid \*uuid;

struct md\_op\_data \*op\_data;

struct lustre\_md lmd;

obd\_valid valid;

int size, err, checksum;

ENTRY;

/\*

first,traverse all the possible index(0--MAX\_OBD\_DEVICES) and for each index @i call class\_num2obd(i),which return obd\_devs[i] of type struct obd\_device

then,compare the returned obd\_device's name with the given name,if match then return the obd\_device

\*/

obd = class\_name2obd(md);

if (!obd) {

CERROR("MD %s: not setup or attached\n", md);

RETURN(-EINVAL);

}

OBD\_ALLOC\_PTR(data);

if (data == NULL)

RETURN(-ENOMEM);

if (proc\_lustre\_fs\_root) {

err = lprocfs\_register\_mountpoint(proc\_lustre\_fs\_root, sb,dt, md);//见下面

if (err < 0)

CERROR("could not register mount in /proc/fs/lustre\n");

}

//下面关于data->ocd\_connect\_flags的设置暂时不管

/\* indicate the features supported by this client \*/

data->ocd\_connect\_flags=OBD\_CONNECT\_IBITS |OBD\_CONNECT\_NODEVOH |

OBD\_CONNECT\_JOIN | OBD\_CONNECT\_ATTRFID |

OBD\_CONNECT\_VERSION | OBD\_CONNECT\_MDS\_CAPA |

OBD\_CONNECT\_OSS\_CAPA | OBD\_CONNECT\_CANCELSET|

OBD\_CONNECT\_FID | OBD\_CONNECT\_AT |

OBD\_CONNECT\_LOV\_V3 | OBD\_CONNECT\_RMT\_CLIENT |

OBD\_CONNECT\_VBR | OBD\_CONNECT\_FULL20;

if (sbi->ll\_flags & LL\_SBI\_SOM\_PREVIEW)

data->ocd\_connect\_flags |= OBD\_CONNECT\_SOM;

#ifdef HAVE\_LRU\_RESIZE\_SUPPORT

if (sbi->ll\_flags & LL\_SBI\_LRU\_RESIZE)

data->ocd\_connect\_flags |= OBD\_CONNECT\_LRU\_RESIZE;

#endif

#ifdef CONFIG\_FS\_POSIX\_ACL

data->ocd\_connect\_flags |= OBD\_CONNECT\_ACL;

#endif

data->ocd\_ibits\_known = MDS\_INODELOCK\_FULL;

data->ocd\_version = LUSTRE\_VERSION\_CODE;

if (sb->s\_flags & MS\_RDONLY)

data->ocd\_connect\_flags |= OBD\_CONNECT\_RDONLY;

if (sbi->ll\_flags & LL\_SBI\_USER\_XATTR)

data->ocd\_connect\_flags |= OBD\_CONNECT\_XATTR;

#ifdef HAVE\_MS\_FLOCK\_LOCK

/\* force vfs to use lustre handler for flock() calls - bug 10743 \*/

sb->s\_flags |= MS\_FLOCK\_LOCK;

#endif

//关于struct file\_operations 的设置

if (sbi->ll\_flags & LL\_SBI\_FLOCK)

sbi->ll\_fop = &ll\_file\_operations\_flock;

else if (sbi->ll\_flags & LL\_SBI\_LOCALFLOCK)

sbi->ll\_fop = &ll\_file\_operations;

else

sbi->ll\_fop = &ll\_file\_operations\_noflock;

data->ocd\_connect\_flags |= OBD\_CONNECT\_REAL; /\* real connection\*/

if (sbi->ll\_flags & LL\_SBI\_RMT\_CLIENT)

data->ocd\_connect\_flags |= OBD\_CONNECT\_RMT\_CLIENT\_FORCE;

//这里obd是mdt，obd\_connect分析见下面,这里对应mdt\_obd\_connect？还是mdc \_connect？在mdt\_obd\_connect中有LASSERT(env!=NULL)，从此看来，这里应该对应mdc\_connect，见下面分析

err = obd\_connect(NULL, &sbi->ll\_md\_exp, obd, &sbi->ll\_sb\_uuid, data, NULL);

if (err == -EBUSY) {

LCONSOLE\_ERROR\_MSG(0x14f, "An MDT (md %s) is performing "

"recovery, of which this client is not a "

"part. Please wait for recovery to complete,"

" abort, or time out.\n", md);

GOTO(out, err);

} else if (err) {

CERROR("cannot connect to %s: rc = %d\n", md, err);

GOTO(out, err);

}

//见下面分析，这里对应的是mdc\_fid\_init

err = obd\_fid\_init(sbi->ll\_md\_exp);//

if (err) {

CERROR("Can't init metadata layer FID infrastructure, "

"rc %d\n", err);

GOTO(out\_md, err);

}

err = obd\_statfs(obd, &osfs, cfs\_time\_current\_64() - CFS\_HZ, 0);

if (err)

GOTO(out\_md\_fid, err);

size = sizeof(\*data);

err = obd\_get\_info(sbi->ll\_md\_exp, sizeof(KEY\_CONN\_DATA),

KEY\_CONN\_DATA, &size, data, NULL);

if (err) {

CERROR("Get connect data failed: %d \n", err);

GOTO(out\_md, err);

}

//初始化sb和sbi

LASSERT(osfs.os\_bsize);

sb->s\_blocksize = osfs.os\_bsize;

sb->s\_blocksize\_bits = log2(osfs.os\_bsize);

sb->s\_magic = LL\_SUPER\_MAGIC;

#if BITS\_PER\_LONG == 64

sb->s\_maxbytes = PAGE\_CACHE\_MAXBYTES >> 1;

#else

sb->s\_maxbytes = PAGE\_CACHE\_MAXBYTES;

#endif

sbi->ll\_namelen = osfs.os\_namelen;

sbi->ll\_max\_rw\_chunk = LL\_DEFAULT\_MAX\_RW\_CHUNK;

if ((sbi->ll\_flags & LL\_SBI\_USER\_XATTR) &&

!(data->ocd\_connect\_flags & OBD\_CONNECT\_XATTR)) {

LCONSOLE\_INFO("Disabling user\_xattr feature because "

"it is not supported on the server\n");

sbi->ll\_flags &= ~LL\_SBI\_USER\_XATTR;

}

if (data->ocd\_connect\_flags & OBD\_CONNECT\_ACL) {

#ifdef MS\_POSIXACL

sb->s\_flags |= MS\_POSIXACL;

#endif

sbi->ll\_flags |= LL\_SBI\_ACL;

} else {

LCONSOLE\_INFO("client wants to enable acl, but mdt not!\n");

#ifdef MS\_POSIXACL

sb->s\_flags &= ~MS\_POSIXACL;

#endif

sbi->ll\_flags &= ~LL\_SBI\_ACL;

}

if (data->ocd\_connect\_flags & OBD\_CONNECT\_RMT\_CLIENT) {

if (!(sbi->ll\_flags & LL\_SBI\_RMT\_CLIENT)) {

sbi->ll\_flags |= LL\_SBI\_RMT\_CLIENT;

LCONSOLE\_INFO("client is set as remote by default.\n");

}

} else {

if (sbi->ll\_flags & LL\_SBI\_RMT\_CLIENT) {

sbi->ll\_flags &= ~LL\_SBI\_RMT\_CLIENT;

LCONSOLE\_INFO("client claims to be remote, but server "

"rejected, forced to be local.\n");

}

}

if (data->ocd\_connect\_flags & OBD\_CONNECT\_MDS\_CAPA) {

LCONSOLE\_INFO("client enabled MDS capability!\n");

sbi->ll\_flags |= LL\_SBI\_MDS\_CAPA;

}

if (data->ocd\_connect\_flags & OBD\_CONNECT\_OSS\_CAPA) {

LCONSOLE\_INFO("client enabled OSS capability!\n");

sbi->ll\_flags |= LL\_SBI\_OSS\_CAPA;

}

obd = class\_name2obd(dt);

if (!obd) {

CERROR("DT %s: not setup or attached\n", dt);

GOTO(out\_md\_fid, err = -ENODEV);

}

data->ocd\_connect\_flags = OBD\_CONNECT\_GRANT| OBD\_CONNECT\_VERSION |

OBD\_CONNECT\_REQPORTAL | OBD\_CONNECT\_BRW\_SIZE |

OBD\_CONNECT\_CANCELSET | OBD\_CONNECT\_FID |

OBD\_CONNECT\_SRVLOCK | OBD\_CONNECT\_TRUNCLOCK|

OBD\_CONNECT\_AT | OBD\_CONNECT\_RMT\_CLIENT |

OBD\_CONNECT\_OSS\_CAPA | OBD\_CONNECT\_VBR|

OBD\_CONNECT\_FULL20;

if (sbi->ll\_flags & LL\_SBI\_SOM\_PREVIEW)

data->ocd\_connect\_flags |= OBD\_CONNECT\_SOM;

if (!OBD\_FAIL\_CHECK(OBD\_FAIL\_OSC\_CONNECT\_CKSUM)) {

/\* OBD\_CONNECT\_CKSUM should always be set, even if checksums are

\* disabled by default, because it can still be enabled on the

\* fly via /proc. As a consequence, we still need to come to an

\* agreement on the supported algorithms at connect time \*/

data->ocd\_connect\_flags |= OBD\_CONNECT\_CKSUM;

if (OBD\_FAIL\_CHECK(OBD\_FAIL\_OSC\_CKSUM\_ADLER\_ONLY))

data->ocd\_cksum\_types = OBD\_CKSUM\_ADLER;

else

/\* send the list of supported checksum types \*/

data->ocd\_cksum\_types = OBD\_CKSUM\_ALL;

}

#ifdef HAVE\_LRU\_RESIZE\_SUPPORT

data->ocd\_connect\_flags |= OBD\_CONNECT\_LRU\_RESIZE;

#endif

if (sbi->ll\_flags & LL\_SBI\_RMT\_CLIENT)

data->ocd\_connect\_flags |= OBD\_CONNECT\_RMT\_CLIENT\_FORCE;

CDEBUG(D\_RPCTRACE, "ocd\_connect\_flags: "LPX64" ocd\_version: %d "

"ocd\_grant: %d\n", data->ocd\_connect\_flags,

data->ocd\_version, data->ocd\_grant);

obd->obd\_upcall.onu\_owner = &sbi->ll\_lco;

obd->obd\_upcall.onu\_upcall = cl\_ocd\_update;

data->ocd\_brw\_size = PTLRPC\_MAX\_BRW\_PAGES << CFS\_PAGE\_SHIFT;

err = obd\_connect(NULL, &sbi->ll\_dt\_exp, obd, &sbi->ll\_sb\_uuid, data, NULL);

if (err == -EBUSY) {

LCONSOLE\_ERROR\_MSG(0x150, "An OST (dt %s) is performing "

"recovery, of which this client is not a "

"part. Please wait for recovery to "

"complete, abort, or time out.\n", dt);

GOTO(out\_md\_fid, err);

} else if (err) {

CERROR("Cannot connect to %s: rc = %d\n", dt, err);

GOTO(out\_md\_fid, err);

}

err = obd\_fid\_init(sbi->ll\_dt\_exp);

if (err) {

CERROR("Can't init data layer FID infrastructure, "

"rc %d\n", err);

GOTO(out\_dt, err);

}

cfs\_mutex\_down(&sbi->ll\_lco.lco\_lock);

sbi->ll\_lco.lco\_flags = data->ocd\_connect\_flags;

sbi->ll\_lco.lco\_md\_exp = sbi->ll\_md\_exp;

sbi->ll\_lco.lco\_dt\_exp = sbi->ll\_dt\_exp;

cfs\_mutex\_up(&sbi->ll\_lco.lco\_lock);

fid\_zero(&sbi->ll\_root\_fid);

err = md\_getstatus(sbi->ll\_md\_exp, &sbi->ll\_root\_fid, &oc);

if (err) {

CERROR("cannot mds\_connect: rc = %d\n", err);

GOTO(out\_lock\_cn\_cb, err);

}

if (!fid\_is\_sane(&sbi->ll\_root\_fid)) {

CERROR("Invalid root fid during mount\n");

GOTO(out\_lock\_cn\_cb, err = -EINVAL);

}

CDEBUG(D\_SUPER, "rootfid "DFID"\n", PFID(&sbi->ll\_root\_fid));

sb->s\_op = &lustre\_super\_operations;

sb->s\_export\_op = &lustre\_export\_operations;

/\* make root inode

\* XXX: move this to after cbd setup? \*/

valid = OBD\_MD\_FLGETATTR | OBD\_MD\_FLBLOCKS | OBD\_MD\_FLMDSCAPA;

if (sbi->ll\_flags & LL\_SBI\_RMT\_CLIENT)

valid |= OBD\_MD\_FLRMTPERM;

else if (sbi->ll\_flags & LL\_SBI\_ACL)

valid |= OBD\_MD\_FLACL;

OBD\_ALLOC\_PTR(op\_data);

if (op\_data == NULL)

GOTO(out\_lock\_cn\_cb, err = -ENOMEM);

op\_data->op\_fid1 = sbi->ll\_root\_fid;

op\_data->op\_mode = 0;

op\_data->op\_capa1 = oc;

op\_data->op\_valid = valid;

err = md\_getattr(sbi->ll\_md\_exp, op\_data, &request);

if (oc)

capa\_put(oc);

OBD\_FREE\_PTR(op\_data);

if (err) {

CERROR("md\_getattr failed for root: rc = %d\n", err);

GOTO(out\_lock\_cn\_cb, err);

}

memset(&lmd, 0, sizeof(lmd));

err = md\_get\_lustre\_md(sbi->ll\_md\_exp, request, sbi->ll\_dt\_exp,

sbi->ll\_md\_exp, &lmd);

if (err) {

CERROR("failed to understand root inode md: rc = %d\n", err);

ptlrpc\_req\_finished (request);

GOTO(out\_lock\_cn\_cb, err);

}

LASSERT(fid\_is\_sane(&sbi->ll\_root\_fid));

root = ll\_iget(sb, cl\_fid\_build\_ino(&sbi->ll\_root\_fid), &lmd);

md\_free\_lustre\_md(sbi->ll\_md\_exp, &lmd);

ptlrpc\_req\_finished(request);

if (root == NULL || IS\_ERR(root)) {

if (lmd.lsm)

obd\_free\_memmd(sbi->ll\_dt\_exp, &lmd.lsm);

#ifdef CONFIG\_FS\_POSIX\_ACL

if (lmd.posix\_acl) {

posix\_acl\_release(lmd.posix\_acl);

lmd.posix\_acl = NULL;

}

#endif

err = IS\_ERR(root) ? PTR\_ERR(root) : -EBADF;

root = NULL;

CERROR("lustre\_lite: bad iget4 for root\n");

GOTO(out\_root, err);

}

err = ll\_close\_thread\_start(&sbi->ll\_lcq);

if (err) {

CERROR("cannot start close thread: rc %d\n", err);

GOTO(out\_root, err);

}

#ifdef CONFIG\_FS\_POSIX\_ACL

if (sbi->ll\_flags & LL\_SBI\_RMT\_CLIENT) {

rct\_init(&sbi->ll\_rct);

et\_init(&sbi->ll\_et);

}

#endif

checksum = sbi->ll\_flags & LL\_SBI\_CHECKSUM;

err = obd\_set\_info\_async(sbi->ll\_dt\_exp, sizeof(KEY\_CHECKSUM),

KEY\_CHECKSUM, sizeof(checksum), &checksum,

NULL);

cl\_sb\_init(sb);

sb->s\_root = d\_alloc\_root(root);

if (data != NULL)

OBD\_FREE(data, sizeof(\*data));

sb->s\_root->d\_op = &ll\_d\_root\_ops;

sbi->ll\_sdev\_orig = sb->s\_dev;

/\* We set sb->s\_dev equal on all lustre clients in order to support

\* NFS export clustering. NFSD requires that the FSID be the same

\* on all clients. \*/

/\* s\_dev is also used in lt\_compare() to compare two fs, but that is

\* only a node-local comparison. \*/

uuid = obd\_get\_uuid(sbi->ll\_md\_exp);

if (uuid != NULL)

sb->s\_dev = get\_uuid2int(uuid->uuid, strlen(uuid->uuid));

RETURN(err);

out\_root:

if (root)

iput(root);

out\_lock\_cn\_cb:

obd\_fid\_fini(sbi->ll\_dt\_exp);

out\_dt:

obd\_disconnect(sbi->ll\_dt\_exp);

sbi->ll\_dt\_exp = NULL;

out\_md\_fid:

obd\_fid\_fini(sbi->ll\_md\_exp);

out\_md:

obd\_disconnect(sbi->ll\_md\_exp);

sbi->ll\_md\_exp = NULL;

out:

if (data != NULL)

OBD\_FREE\_PTR(data);

lprocfs\_unregister\_mountpoint(sbi);

return err;

}

struct lprocfs\_vars {

const char \*name;

cfs\_read\_proc\_t \*read\_fptr;

cfs\_write\_proc\_t \*write\_fptr;

void \*data;

struct file\_operations \*fops;

/\*\*

\* /proc file mode.

\*/

mode\_t proc\_mode;

};

**int lprocfs\_register\_mountpoint(struct proc\_dir\_entry \*parent,**

**struct super\_block \*sb, char \*osc, char \*mdc){**

struct lprocfs\_vars lvars[2];

struct lustre\_sb\_info \*lsi = s2lsi(sb);

struct ll\_sb\_info \*sbi = ll\_s2sbi(sb); //s2lsi(sb)->lsi\_llsbi

struct obd\_device \*obd;

char name[MAX\_STRING\_SIZE + 1], \*ptr;

int err, id, len, rc;

ENTRY;

memset(lvars, 0, sizeof(lvars));

name[MAX\_STRING\_SIZE] = '\0';

lvars[0].name = name;

LASSERT(sbi != NULL);

LASSERT(mdc != NULL);

LASSERT(osc != NULL);

/\* Get fsname \*/

len = strlen(lsi->lsi\_lmd->lmd\_profile);

ptr = strrchr(lsi->lsi\_lmd->lmd\_profile, '-');

if (ptr && (strcmp(ptr, "-client") == 0))

len -= 7;

/\* Mount info \*/

snprintf(name, MAX\_STRING\_SIZE, "%.\*s-%p", len,

lsi->lsi\_lmd->lmd\_profile, sb);

sbi->ll\_proc\_root = lprocfs\_register(name, parent, NULL, NULL);//下面分析

if (IS\_ERR(sbi->ll\_proc\_root)) {

err = PTR\_ERR(sbi->ll\_proc\_root);

sbi->ll\_proc\_root = NULL;

RETURN(err);

}

//下面的关于lprocfs\_seq\_create的代码都是使用seq file管理proc文件

rc = lprocfs\_seq\_create(sbi->ll\_proc\_root, "dump\_page\_cache", 0444,

&vvp\_dump\_pgcache\_file\_ops, sbi);

if (rc)

CWARN("Error adding the dump\_page\_cache file\n");

rc = lprocfs\_seq\_create(sbi->ll\_proc\_root, "extents\_stats", 0644,

&ll\_rw\_extents\_stats\_fops, sbi);

if (rc)

CWARN("Error adding the extent\_stats file\n");

rc = lprocfs\_seq\_create(sbi->ll\_proc\_root, "extents\_stats\_per\_process",

0644, &ll\_rw\_extents\_stats\_pp\_fops, sbi);

if (rc)

CWARN("Error adding the extents\_stats\_per\_process file\n");

rc = lprocfs\_seq\_create(sbi->ll\_proc\_root, "offset\_stats", 0644,

&ll\_rw\_offset\_stats\_fops, sbi);

if (rc)

CWARN("Error adding the offset\_stats file\n");

//下面的关于stats部分暂时忽略

/\* File operations stats \*/

sbi->ll\_stats = lprocfs\_alloc\_stats(LPROC\_LL\_FILE\_OPCODES,

LPROCFS\_STATS\_FLAG\_NONE);

if (sbi->ll\_stats == NULL)

GOTO(out, err = -ENOMEM);

/\* do counter init \*/

for (id = 0; id < LPROC\_LL\_FILE\_OPCODES; id++) {

\_\_u32 type = llite\_opcode\_table[id].type;

void \*ptr = NULL;

if (type & LPROCFS\_TYPE\_REGS)

ptr = "regs";

else if (type & LPROCFS\_TYPE\_BYTES)

ptr = "bytes";

else if (type & LPROCFS\_TYPE\_PAGES)

ptr = "pages";

lprocfs\_counter\_init(sbi->ll\_stats,

llite\_opcode\_table[id].opcode,

(type & LPROCFS\_CNTR\_AVGMINMAX),

llite\_opcode\_table[id].opname, ptr);

}

err = lprocfs\_register\_stats(sbi->ll\_proc\_root, "stats", sbi->ll\_stats);

if (err)

GOTO(out, err);

sbi->ll\_ra\_stats = lprocfs\_alloc\_stats(ARRAY\_SIZE(ra\_stat\_string),

LPROCFS\_STATS\_FLAG\_NONE);

if (sbi->ll\_ra\_stats == NULL)

GOTO(out, err = -ENOMEM);

for (id = 0; id < ARRAY\_SIZE(ra\_stat\_string); id++)

lprocfs\_counter\_init(sbi->ll\_ra\_stats, id, 0,ra\_stat\_string[id], "pages");

err = lprocfs\_register\_stats(sbi->ll\_proc\_root, "read\_ahead\_stats",sbi->ll\_ra\_stats);

if (err)

GOTO(out, err);

err = lprocfs\_add\_vars(sbi->ll\_proc\_root, lprocfs\_llite\_obd\_vars, sb);

if (err)

GOTO(out, err);

/\* MDC info \*/

obd = class\_name2obd(mdc);

LASSERT(obd != NULL);

LASSERT(obd->obd\_magic == OBD\_DEVICE\_MAGIC);

LASSERT(obd->obd\_type->typ\_name != NULL);

//实际上为lvars[0].name赋值，因为前面有lvars[0].name = name

snprintf(name,MAX\_STRING\_SIZE,"%s/common\_name",obd->obd\_type->typ\_name);

lvars[0].read\_fptr = lprocfs\_rd\_name;

err = lprocfs\_add\_vars(sbi->ll\_proc\_root, lvars, obd);

if (err)

GOTO(out, err);

snprintf(name, MAX\_STRING\_SIZE, "%s/uuid", obd->obd\_type->typ\_name);

lvars[0].read\_fptr = lprocfs\_rd\_uuid;

err = lprocfs\_add\_vars(sbi->ll\_proc\_root, lvars, obd);

if (err)

GOTO(out, err);

/\* OSC info \*/

obd = class\_name2obd(osc);

LASSERT(obd != NULL);

LASSERT(obd->obd\_magic == OBD\_DEVICE\_MAGIC);

LASSERT(obd->obd\_type->typ\_name != NULL);

snprintf(name, MAX\_STRING\_SIZE, "%s/common\_name",

obd->obd\_type->typ\_name);

lvars[0].read\_fptr = lprocfs\_rd\_name;

err = lprocfs\_add\_vars(sbi->ll\_proc\_root, lvars, obd);

if (err)

GOTO(out, err);

snprintf(name, MAX\_STRING\_SIZE, "%s/uuid", obd->obd\_type->typ\_name);

lvars[0].read\_fptr = lprocfs\_rd\_uuid;

err = lprocfs\_add\_vars(sbi->ll\_proc\_root, lvars, obd);

out:

if (err) {

lprocfs\_remove(&sbi->ll\_proc\_root);

lprocfs\_free\_stats(&sbi->ll\_ra\_stats);

lprocfs\_free\_stats(&sbi->ll\_stats);

}

RETURN(err);

}

/\*add proc entries specified by @list under @parent's subdir proc dir entry with given @name

@parent:the top most proc\_dir\_entry

@name:specify a middle proc\_dir\_entry in @parent->sub\_dirs

@list:a list of proc\_dir\_entry to be added\*/

**struct proc\_dir\_entry \*lprocfs\_register(const char \*name,**

**struct proc\_dir\_entry \*parent,struct lprocfs\_vars \*list, void \*data){**

struct proc\_dir\_entry \*newchild;

newchild = lprocfs\_srch(parent, name);//lprocfs\_srch(parent,name);

if (newchild != NULL) {//在parent所在的proc\_dir\_entry下找到了

CERROR(" Lproc: Attempting to register %s more than once \n",

name);

return ERR\_PTR(-EALREADY);

}

newchild = proc\_mkdir(name, parent);//proc\_mkdir

if (newchild != NULL && list != NULL) {

//这里newchild作为parent proc\_dir\_entry，list是将要被添加的proc entry数组，将list中的proc entry添加到newchild中。

int rc = lprocfs\_add\_vars(newchild, list, data);

if (rc) {

lprocfs\_remove(&newchild);

return ERR\_PTR(rc);

}

}

return newchild;

}

**struct proc\_dir\_entry \*lprocfs\_srch(struct proc\_dir\_entry \*head,const char \*name)**

**{**

struct proc\_dir\_entry \*temp;

if (head == NULL)

return NULL;

LPROCFS\_ENTRY();

temp = head->subdir;//temp=head->subdir;

while (temp != NULL) {

if (strcmp(temp->name, name) == 0) {

LPROCFS\_EXIT();

return temp;

}

temp = temp->next;

}

LPROCFS\_EXIT();

return NULL;

}

/\*\*

\* Add /proc entries.

\*

\* \param root [in] The parent proc entry on which new entry will be added.

\* \param list [in] Array of proc entries to be added.

\* \param data [in] The argument to be passed when entries read/write routines

\* are called through /proc file.

\*

\* \retval 0 on success

\* < 0 on error

\*/

**int lprocfs\_add\_vars(struct proc\_dir\_entry \*root, struct lprocfs\_vars \*list,void \*data)//0322**

**{**

if (root == NULL || list == NULL)

return -EINVAL;

while (list->name != NULL) {

struct proc\_dir\_entry \*cur\_root, \*proc;

char \*pathcopy, \*cur, \*next, pathbuf[64];

int pathsize = strlen(list->name) + 1;

proc = NULL;

cur\_root = root;

/\* need copy of path for strsep \*/

if (strlen(list->name) > sizeof(pathbuf) - 1) {

OBD\_ALLOC(pathcopy, pathsize);

if (pathcopy == NULL)

return -ENOMEM;

} else {

pathcopy = pathbuf;

}

next = pathcopy;

strcpy(pathcopy, list->name);

//strsep:extract token from string

while (cur\_root != NULL && (cur = strsep(&next, "/"))) {

if (\*cur =='\0') /\* skip double/trailing "/" \*/

continue;

proc = lprocfs\_srch(cur\_root, cur);

CDEBUG(D\_OTHER, "cur\_root=%s, cur=%s, next=%s, (%s)\n",

cur\_root->name, cur, next,

(proc ? "exists" : "new"));

if (next != NULL) {

//not exist,then we create a proc dentry(cur\_root is the parent proc dir entry).

cur\_root = (proc ? proc :proc\_mkdir(cur,cur\_root));

} else if (proc == NULL) {//next==NULL && proc==NULL

mode\_t mode = 0;

if (list->proc\_mode != 0000) {

mode = list->proc\_mode;

} else {

if (list->read\_fptr)

mode = 0444;

if (list->write\_fptr)

mode |= 0200;

}

proc = create\_proc\_entry(cur, mode, cur\_root);//create\_proc\_entry()

}

}

if (pathcopy != pathbuf)//pathcopy is newly allocated

OBD\_FREE(pathcopy, pathsize);

if (cur\_root == NULL || proc == NULL) {

CERROR("LprocFS: No memory to create /proc entry %s",

list->name);

return -ENOMEM;

}

if (list->fops)

proc->proc\_fops = list->fops;

else

proc->proc\_fops = &lprocfs\_generic\_fops;

proc->read\_proc = list->read\_fptr;

proc->write\_proc = list->write\_fptr;

proc->data = (list->data ? list->data : data);

list++;

}

return 0;

}

**int lprocfs\_seq\_create(cfs\_proc\_dir\_entry\_t \*parent, char \*name, mode\_t mode,**

**struct file\_operations \*seq\_fops, void \*data)**

**{**

struct proc\_dir\_entry \*entry;

ENTRY;

entry = create\_proc\_entry(name, mode, parent);

if (entry == NULL)

RETURN(-ENOMEM);

entry->proc\_fops = seq\_fops;

entry->data = data;

RETURN(0);

}

/\*\* Create a new /a exp on device /a obd for the uuid /a cluuid

\* @param exp New export handle

\* @param d Connect data, supported flags are set, flags also understood

\* by obd are returned.

\*/

**static inline int obd\_connect(const struct lu\_env \*env,**

**struct obd\_export \*\*exp,struct obd\_device \*obd,**

**struct obd\_uuid \*cluuid,**

**struct obd\_connect\_data \*d,**

**void \*localdata)**

**{**

int rc;

\_\_u64 ocf = d ? d->ocd\_connect\_flags : 0; /\* for post-condition

\* check \*/

ENTRY;

OBD\_CHECK\_DEV\_ACTIVE(obd); /\* ensure obd\_setup and !obd\_stopping \*/

OBD\_CHECK\_DT\_OP(obd, connect, -EOPNOTSUPP);//是否定义了o\_connetc函数

OBD\_COUNTER\_INCREMENT(obd, connect);

rc = OBP(obd, connect)(env, exp, obd, cluuid, d, localdata);

/\* check that only subset is granted \*/

LASSERT(ergo(d != NULL,

(d->ocd\_connect\_flags & ocf) == d->ocd\_connect\_flags));

RETURN(rc);

}

**static int mdc\_connect(const struct lu\_env \*env,**

**struct obd\_export \*\*exp,**

**struct obd\_device \*obd, struct obd\_uuid \*cluuid,**

**struct obd\_connect\_data \*data,**

**void \*localdata)**

**{**

struct obd\_import \*imp = obd->u.cli.cl\_import;

/\* mds-mds import features \*/

if (data && (data->ocd\_connect\_flags & OBD\_CONNECT\_MDS\_MDS)) {

cfs\_spin\_lock(&imp->imp\_lock);

imp->imp\_server\_timeout = 1;

cfs\_spin\_unlock(&imp->imp\_lock);

imp->imp\_client->cli\_request\_portal = MDS\_MDS\_PORTAL;

CDEBUG(D\_OTHER, "%s: Set 'mds' portal and timeout\n",

obd->obd\_name);

}

return client\_connect\_import(env, exp, obd, cluuid, data, NULL);

}

**static inline int obd\_fid\_init(struct obd\_export \*exp)**

**{**

int rc;

ENTRY;

OBD\_CHECK\_DT\_OP(exp->exp\_obd, fid\_init, 0);

EXP\_COUNTER\_INCREMENT(exp, fid\_init);

rc = OBP(exp->exp\_obd, fid\_init)(exp);

RETURN(rc);

}

**static int mdc\_fid\_init(struct obd\_export \*exp)**

**{**

struct client\_obd \*cli = &exp->exp\_obd->u.cli;

char \*prefix;

int rc;

ENTRY;

OBD\_ALLOC\_PTR(cli->cl\_seq);

if (cli->cl\_seq == NULL)

RETURN(-ENOMEM);

OBD\_ALLOC(prefix, MAX\_OBD\_NAME + 5);

if (prefix == NULL)

GOTO(out\_free\_seq, rc = -ENOMEM);

snprintf(prefix, MAX\_OBD\_NAME + 5, "srv-%s",

exp->exp\_obd->obd\_name);

/\* Init client side sequence-manager \*/

rc = seq\_client\_init(cli->cl\_seq, exp,

LUSTRE\_SEQ\_METADATA,

prefix, NULL);

OBD\_FREE(prefix, MAX\_OBD\_NAME + 5);

if (rc)

GOTO(out\_free\_seq, rc);

RETURN(rc);

out\_free\_seq:

OBD\_FREE\_PTR(cli->cl\_seq);

cli->cl\_seq = NULL;

return rc;

}

/\* @max\_age is the oldest time in jiffies that we accept using a cached data.

\* If the cache is older than @max\_age we will get a new value from the

\* target. Use a value of "cfs\_time\_current() + HZ" to guarantee freshness. \*/

**static inline int obd\_statfs(struct obd\_device \*obd, struct obd\_statfs \*osfs,**

**\_\_u64 max\_age, \_\_u32 flags)**

**{**

int rc = 0;

ENTRY;

if (obd == NULL)

RETURN(-EINVAL);

OBD\_CHECK\_DT\_OP(obd, statfs, -EOPNOTSUPP);

OBD\_COUNTER\_INCREMENT(obd, statfs);

CDEBUG(D\_SUPER, "osfs "LPU64", max\_age "LPU64"\n",

obd->obd\_osfs\_age, max\_age);

if (cfs\_time\_before\_64(obd->obd\_osfs\_age, max\_age)) {

rc = OBP(obd, statfs)(obd, osfs, max\_age, flags);

if (rc == 0) {

cfs\_spin\_lock(&obd->obd\_osfs\_lock);

memcpy(&obd->obd\_osfs, osfs, sizeof(obd->obd\_osfs));

obd->obd\_osfs\_age = cfs\_time\_current\_64();

cfs\_spin\_unlock(&obd->obd\_osfs\_lock);

}

} else {

CDEBUG(D\_SUPER,"%s: use %p cache blocks "LPU64"/"LPU64

" objects "LPU64"/"LPU64"\n",

obd->obd\_name, &obd->obd\_osfs,

obd->obd\_osfs.os\_bavail, obd->obd\_osfs.os\_blocks,

obd->obd\_osfs.os\_ffree, obd->obd\_osfs.os\_files);

cfs\_spin\_lock(&obd->obd\_osfs\_lock);

memcpy(osfs, &obd->obd\_osfs, sizeof(\*osfs));

cfs\_spin\_unlock(&obd->obd\_osfs\_lock);

}

RETURN(rc);

}

/\* ->o\_connect() method for client side (OSC and MDC and MGC) \*/

**int client\_connect\_import(const struct lu\_env \*env,**

**struct obd\_export \*\*exp,**

**struct obd\_device \*obd, struct obd\_uuid \*cluuid,**

**struct obd\_connect\_data \*data, void \*localdata)**

**{**

struct client\_obd \*cli = &obd->u.cli;

struct obd\_import \*imp = cli->cl\_import;

struct obd\_connect\_data \*ocd;

struct lustre\_handle conn = { 0 };

int rc;

ENTRY;

\*exp = NULL;

cfs\_down\_write(&cli->cl\_sem);

if (cli->cl\_conn\_count > 0 )

GOTO(out\_sem, rc = -EALREADY);

rc = class\_connect(&conn, obd, cluuid);

if (rc)

GOTO(out\_sem, rc);

cli->cl\_conn\_count++;

\*exp = class\_conn2export(&conn);//调用class\_handle2object（&（conn->cookie））。关于class\_handle2object见下面分析

LASSERT(obd->obd\_namespace);

imp->imp\_dlm\_handle = conn;//client’s ldlm export

//imp->generation++; imp->imp\_state = LUSTRE\_IMP\_NEW;

rc = ptlrpc\_init\_import(imp);

if (rc != 0)

GOTO(out\_ldlm, rc);

ocd = &imp->imp\_connect\_data;

if (data) {

\*ocd = \*data;

imp->imp\_connect\_flags\_orig = data->ocd\_connect\_flags;

}

rc = ptlrpc\_connect\_import(imp, NULL);

if (rc != 0) {

LASSERT (imp->imp\_state == LUSTRE\_IMP\_DISCON);

GOTO(out\_ldlm, rc);

}

LASSERT((\*exp)->exp\_connection);

if (data) {

LASSERTF((ocd->ocd\_connect\_flags & data->ocd\_connect\_flags) ==

ocd->ocd\_connect\_flags, "old "LPX64", new "LPX64"\n",

data->ocd\_connect\_flags, ocd->ocd\_connect\_flags);

data->ocd\_connect\_flags = ocd->ocd\_connect\_flags;

}

ptlrpc\_pinger\_add\_import(imp);

EXIT;

if (rc) {

out\_ldlm:

cli->cl\_conn\_count--;

class\_disconnect(\*exp);

\*exp = NULL;

}

out\_sem:

cfs\_up\_write(&cli->cl\_sem);

return rc;

}

/\* A connection defines an export context in which preallocation can

be managed. This releases the export pointer reference, and returns

the export handle, so the export refcount is 1 when this function

returns. \*/

**int class\_connect(struct lustre\_handle \*conn, struct obd\_device \*obd,**

**struct obd\_uuid \*cluuid)**

**{**

struct obd\_export \*export;

LASSERT(conn != NULL);

LASSERT(obd != NULL);

LASSERT(cluuid != NULL);

ENTRY;

//creates a new export,add it into the hash table and return the pointer to it.

export = class\_new\_export(obd, cluuid);

if (IS\_ERR(export))

RETURN(PTR\_ERR(export));

conn->cookie = export->exp\_handle.h\_cookie;

class\_export\_put(export);

CDEBUG(D\_IOCTL, "connect: client %s, cookie "LPX64"\n",

cluuid->uuid, conn->cookie);

RETURN(0);

}

//用于lustre\_handle的数据结构

typedef void (\*portals\_handle\_addref\_cb)(void \*object);

typedef struct rcu\_head cfs\_rcu\_head\_t;

struct portals\_handle {

cfs\_list\_t h\_link;

\_\_u64 h\_cookie;

portals\_handle\_addref\_cb h\_addref;

/\* newly added fields to handle the RCU issue. -jxiong \*/

cfs\_spinlock\_t h\_lock;

void \*h\_ptr;

void (\*h\_free\_cb)(void \*, size\_t);

cfs\_rcu\_head\_t h\_rcu;

unsigned int h\_size;

\_\_u8 h\_in:1;

\_\_u8 h\_unused[3];

};

static struct handle\_bucket {

cfs\_spinlock\_t lock;

cfs\_list\_t head;

} \*handle\_hash;

**void \*class\_handle2object(\_\_u64 cookie)**

**{**

struct handle\_bucket \*bucket;

struct portals\_handle \*h;

void \*retval = NULL;

ENTRY;

LASSERT(handle\_hash != NULL);

/\* Be careful when you want to change this code. See the

\* rcu\_read\_lock() definition on top this file. - jxiong \*/

bucket = handle\_hash + (cookie & HANDLE\_HASH\_MASK);

rcu\_read\_lock();

list\_for\_each\_entry\_rcu(h, &bucket->head, h\_link) {

if (h->h\_cookie != cookie)

continue;

cfs\_spin\_lock(&h->h\_lock);

if (likely(h->h\_in != 0)) {

h->h\_addref(h);

retval = h;

}

cfs\_spin\_unlock(&h->h\_lock);

break;

}

rcu\_read\_unlock();

RETURN(retval);

}

/\*\*

\* Attempt to (re)connect import \a imp. This includes all preparations,

\* initializing CONNECT RPC request and passing it to ptlrpcd for

\* actual sending.

\* Returns 0 on success or error code.

\*/

**int ptlrpc\_connect\_import(struct obd\_import \*imp, char \*new\_uuid)**

**{**

struct obd\_device \*obd = imp->imp\_obd;

int initial\_connect = 0;

int set\_transno = 0;

\_\_u64 committed\_before\_reconnect = 0;

struct ptlrpc\_request \*request;

char \*bufs[] = { NULL,

obd2cli\_tgt(imp->imp\_obd),

obd->obd\_uuid.uuid,

(char \*)&imp->imp\_dlm\_handle,

(char \*)&imp->imp\_connect\_data };

struct ptlrpc\_connect\_async\_args \*aa;

int rc;

ENTRY;

cfs\_spin\_lock(&imp->imp\_lock);

//针对imp的不同状态进行处理

if (imp->imp\_state == LUSTRE\_IMP\_CLOSED) {

cfs\_spin\_unlock(&imp->imp\_lock);

CERROR("can't connect to a closed import\n");

RETURN(-EINVAL);

} else if (imp->imp\_state == LUSTRE\_IMP\_FULL) {

cfs\_spin\_unlock(&imp->imp\_lock);

CERROR("already connected\n");

RETURN(0);

} else if (imp->imp\_state == LUSTRE\_IMP\_CONNECTING) {

cfs\_spin\_unlock(&imp->imp\_lock);

CERROR("already connecting\n");

RETURN(-EALREADY);

}

//设置当前状态为LUSTRE\_IMP\_CONNECTING

IMPORT\_SET\_STATE\_NOLOCK(imp, LUSTRE\_IMP\_CONNECTING);

imp->imp\_conn\_cnt++;

imp->imp\_resend\_replay = 0;

/\*imp->imp\_remote\_handle:remote export handle,this is how remote side knows what export we are talking to.filled from response to connect request.\*/

if (!lustre\_handle\_is\_used(&imp->imp\_remote\_handle))

initial\_connect = 1;

else //imp->imp\_peer\_committed\_transno:last trasno committed on remote side

committed\_before\_reconnect = imp->imp\_peer\_committed\_transno;

/\*ptlrpc\_first\_transno():get the transno of the first entry in imp->imp\_replay\_list。ocd\_transno:first transno from client to be replayed\*/

set\_transno=ptlrpc\_first\_transno(imp,&imp->imp\_connect\_data.ocd\_transno); cfs\_spin\_unlock(&imp->imp\_lock);

if (new\_uuid) {

struct obd\_uuid uuid;

obd\_str2uuid(&uuid, new\_uuid);

rc = import\_set\_conn\_priority(imp, &uuid);//call import\_set\_conn见下面

if (rc)

GOTO(out, rc);

}

rc = import\_select\_connection(imp);

if (rc)

GOTO(out, rc);

rc = sptlrpc\_import\_sec\_adapt(imp, NULL, 0);

if (rc)

GOTO(out, rc);

/\* Reset connect flags to the originally requested flags, in case

\* the server is updated on-the-fly we will get the new features. \*/

imp->imp\_connect\_data.ocd\_connect\_flags = imp->imp\_connect\_flags\_orig;

imp->imp\_msghdr\_flags &= ~MSGHDR\_AT\_SUPPORT;

imp->imp\_msghdr\_flags &= ~MSGHDR\_CKSUM\_INCOMPAT18;

rc = obd\_reconnect(NULL, imp->imp\_obd->obd\_self\_export, obd,

&obd->obd\_uuid, &imp->imp\_connect\_data, NULL);

if (rc)

GOTO(out, rc);

request = ptlrpc\_request\_alloc(imp, &RQF\_MDS\_CONNECT);

if (request == NULL)

GOTO(out, rc = -ENOMEM);

rc = ptlrpc\_request\_bufs\_pack(request, LUSTRE\_OBD\_VERSION,

imp->imp\_connect\_op, bufs, NULL);

if (rc) {

ptlrpc\_request\_free(request);

GOTO(out, rc);

}

/\* Report the rpc service time to the server so that it knows how long

\* to wait for clients to join recovery \*/

lustre\_msg\_set\_service\_time(request->rq\_reqmsg,

at\_timeout2est(request->rq\_timeout));

/\* The amount of time we give the server to process the connect req.

\* import\_select\_connection will increase the net latency on

\* repeated reconnect attempts to cover slow networks.

\* We override/ignore the server rpc completion estimate here,

\* which may be large if this is a reconnect attempt \*/

request->rq\_timeout = INITIAL\_CONNECT\_TIMEOUT;

lustre\_msg\_set\_timeout(request->rq\_reqmsg, request->rq\_timeout);

#ifndef \_\_KERNEL\_\_

lustre\_msg\_add\_op\_flags(request->rq\_reqmsg, MSG\_CONNECT\_LIBCLIENT);

#endif

lustre\_msg\_add\_op\_flags(request->rq\_reqmsg, MSG\_CONNECT\_NEXT\_VER);

request->rq\_no\_resend = request->rq\_no\_delay = 1;

request->rq\_send\_state = LUSTRE\_IMP\_CONNECTING;

/\* Allow a slightly larger reply for future growth compatibility \*/

req\_capsule\_set\_size(&request->rq\_pill, &RMF\_CONNECT\_DATA, RCL\_SERVER,

sizeof(struct obd\_connect\_data)+16\*sizeof(\_\_u64));

ptlrpc\_request\_set\_replen(request);

request->rq\_interpret\_reply = ptlrpc\_connect\_interpret;

CLASSERT(sizeof (\*aa) <= sizeof (request->rq\_async\_args));

aa = ptlrpc\_req\_async\_args(request);

memset(aa, 0, sizeof \*aa);

aa->pcaa\_peer\_committed = committed\_before\_reconnect;

aa->pcaa\_initial\_connect = initial\_connect;

if (aa->pcaa\_initial\_connect) {

cfs\_spin\_lock(&imp->imp\_lock);

imp->imp\_replayable = 1;

cfs\_spin\_unlock(&imp->imp\_lock);

lustre\_msg\_add\_op\_flags(request->rq\_reqmsg,

MSG\_CONNECT\_INITIAL);

}

if (set\_transno)

lustre\_msg\_add\_op\_flags(request->rq\_reqmsg,

MSG\_CONNECT\_TRANSNO);

DEBUG\_REQ(D\_RPCTRACE, request, "(re)connect request (timeout %d)",

request->rq\_timeout);

ptlrpcd\_add\_req(request, PSCOPE\_OTHER);

rc = 0;

out:

if (rc != 0) {

IMPORT\_SET\_STATE(imp, LUSTRE\_IMP\_DISCON);

}

RETURN(rc);

}

/\*\*

\* Structure to single define portal connection.

\*/

struct ptlrpc\_connection {

/\*\* linkage for connections hash table \*/

cfs\_hlist\_node\_t c\_hash;

/\*\* Our own lnet nid for this connection \*/

lnet\_nid\_t c\_self;

/\*\* Remote side nid for this connection \*/

lnet\_process\_id\_t c\_peer;

/\*\* UUID of the other side \*/

struct obd\_uuid c\_remote\_uuid;

/\*\* reference counter for this connection \*/

cfs\_atomic\_t c\_refcount;

};

/\*\* Client definition for PortalRPC \*/

struct ptlrpc\_client {

/\*\* What lnet portal does this client send messages to by default \*/

\_\_u32 cli\_request\_portal;

/\*\* What portal do we expect replies on \*/

\_\_u32 cli\_reply\_portal;

/\*\* Name of the client \*/

char \*cli\_name;

};

/\*\*

\* Definition of import connection structure

\*/

**struct obd\_import\_conn {**

/\*\* Item for linking connections together \*/

cfs\_list\_t oic\_item;

/\*\* Pointer to actual PortalRPC connection \*/

struct ptlrpc\_connection \*oic\_conn;

/\*\* uuid of remote side \*/

struct obd\_uuid oic\_uuid;

/\*\*

\* Time (64 bit jiffies) of last connection attempt on this connection

\*/

\_\_u64 oic\_last\_attempt;

};

/\* @priority: if non-zero, move the selected to the list head

\* @create: if zero, only search in existed connections

\*/

/\* 1:first get the ptlrpc\_connection from the given @uuid with ptlrpc\_uuid\_to\_connection()

2:if @create is set,then allocate imp\_conn of type struct obd\_import\_conn with OBD\_ALLOC()

3:walk thru the @imp->imp\_conn\_list to find an obd\_import\_conn with given @uuid,and if the @priority is set,then move it to the head of @imp->imp\_conn\_list and then goes out

4:when come here,we have not find an obd\_import\_conn with the given @uuid,so,if @create is set then init @imp\_conn and add it into the @imp->imp\_conn\_list,and if @priority is set,then add it to the head of @imp->imp\_conn\_list,otherwise to the tail of the @imp->imp\_conn\_list

\*/

**static int import\_set\_conn(struct obd\_import \*imp, struct obd\_uuid \*uuid,int priority, int create)**

**{**

struct ptlrpc\_connection \*ptlrpc\_conn;

struct obd\_import\_conn \*imp\_conn = NULL, \*item;

int rc = 0;

ENTRY;

if (!create && !priority) {

CDEBUG(D\_HA, "Nothing to do\n");

RETURN(-EINVAL);

}

ptlrpc\_conn = ptlrpc\_uuid\_to\_connection(uuid);

if (!ptlrpc\_conn) {

CDEBUG(D\_HA, "can't find connection %s\n", uuid->uuid);

RETURN (-ENOENT);

}

if (create) {

OBD\_ALLOC(imp\_conn, sizeof(\*imp\_conn));

if (!imp\_conn) {

GOTO(out\_put, rc = -ENOMEM);

}

}

cfs\_spin\_lock(&imp->imp\_lock);

cfs\_list\_for\_each\_entry(item, &imp->imp\_conn\_list, oic\_item) {

if (obd\_uuid\_equals(uuid, &item->oic\_uuid)) {

if (priority) {

cfs\_list\_del(&item->oic\_item);

cfs\_list\_add(&item->oic\_item,

&imp->imp\_conn\_list);

item->oic\_last\_attempt = 0;

}

CDEBUG(D\_HA, "imp %p@%s: found existing conn %s%s\n",

imp, imp->imp\_obd->obd\_name, uuid->uuid,

(priority ? ", moved to head" : ""));

cfs\_spin\_unlock(&imp->imp\_lock);

GOTO(out\_free, rc = 0);

}

}

/\* not found \*/

if (create) {

imp\_conn->oic\_conn = ptlrpc\_conn;

imp\_conn->oic\_uuid = \*uuid;

imp\_conn->oic\_last\_attempt = 0;

if (priority)

cfs\_list\_add(&imp\_conn->oic\_item, &imp->imp\_conn\_list);

else

cfs\_list\_add\_tail(&imp\_conn->oic\_item,

&imp->imp\_conn\_list);

CDEBUG(D\_HA, "imp %p@%s: add connection %s at %s\n",

imp, imp->imp\_obd->obd\_name, uuid->uuid,

(priority ? "head" : "tail"));

} else {

cfs\_spin\_unlock(&imp->imp\_lock);

GOTO(out\_free, rc = -ENOENT);

}

cfs\_spin\_unlock(&imp->imp\_lock);

RETURN(0);

out\_free:

if (imp\_conn)

OBD\_FREE(imp\_conn, sizeof(\*imp\_conn));

out\_put:

ptlrpc\_connection\_put(ptlrpc\_conn);

RETURN(rc);

}

Luster 文件系统分析

1. Lustre\_super\_operations分析

struct super\_operations lustre\_super\_operations =

{

.alloc\_inode = ll\_alloc\_inode,

.destroy\_inode = ll\_destroy\_inode,

.clear\_inode = ll\_clear\_inode,

.delete\_inode = ll\_delete\_inode,

.put\_super = ll\_put\_super,

.statfs = ll\_statfs,

.umount\_begin = ll\_umount\_begin,

.remount\_fs = ll\_remount\_fs,

.show\_options = ll\_show\_options,

};

* 1. ll\_alloc\_inode

**static struct inode \*ll\_alloc\_inode(struct super\_block \*sb)//0329**

**{**

struct ll\_inode\_info \*lli;

ll\_stats\_ops\_tally(ll\_s2sbi(sb), LPROC\_LL\_ALLOC\_INODE, 1);

OBD\_SLAB\_ALLOC\_PTR\_GFP(lli, ll\_inode\_cachep, CFS\_ALLOC\_IO);

if (lli == NULL)

return NULL;

inode\_init\_once(&lli->lli\_vfs\_inode);

ll\_lli\_init(lli);//初始化ll\_inode\_info

return &lli->lli\_vfs\_inode;

}

**void ll\_lli\_init(struct ll\_inode\_info \*lli)**

**{**

lli->lli\_inode\_magic = LLI\_INODE\_MAGIC;

cfs\_sema\_init(&lli->lli\_size\_sem, 1);// sema\_init（）in kernel

cfs\_sema\_init(&lli->lli\_write\_sem, 1);

cfs\_sema\_init(&lli->lli\_trunc\_sem, 1);

lli->lli\_flags = 0;

lli->lli\_maxbytes = PAGE\_CACHE\_MAXBYTES;

cfs\_spin\_lock\_init(&lli->lli\_lock);

CFS\_INIT\_LIST\_HEAD(&lli->lli\_close\_list);

lli->lli\_inode\_magic = LLI\_INODE\_MAGIC;

cfs\_sema\_init(&lli->lli\_och\_sem, 1);

lli->lli\_mds\_read\_och = lli->lli\_mds\_write\_och = NULL;

lli->lli\_mds\_exec\_och = NULL;

lli->lli\_open\_fd\_read\_count = lli->lli\_open\_fd\_write\_count = 0;

lli->lli\_open\_fd\_exec\_count = 0;

CFS\_INIT\_LIST\_HEAD(&lli->lli\_dead\_list);

lli->lli\_remote\_perms = NULL;

lli->lli\_rmtperm\_utime = 0;

cfs\_sema\_init(&lli->lli\_rmtperm\_sem, 1);

CFS\_INIT\_LIST\_HEAD(&lli->lli\_oss\_capas);

cfs\_spin\_lock\_init(&lli->lli\_sa\_lock);

CFS\_INIT\_LIST\_HEAD(&lli->lli\_sa\_dentry);

}

**static void ll\_destroy\_inode(struct inode \*inode)**

**{**

struct ll\_inode\_info \*ptr = ll\_i2info(inode);

OBD\_SLAB\_FREE\_PTR(ptr, ll\_inode\_cachep);

}