《openEuler内核编程》

课程讲稿

第九章 第5讲

分布式文件系统Ceph解析

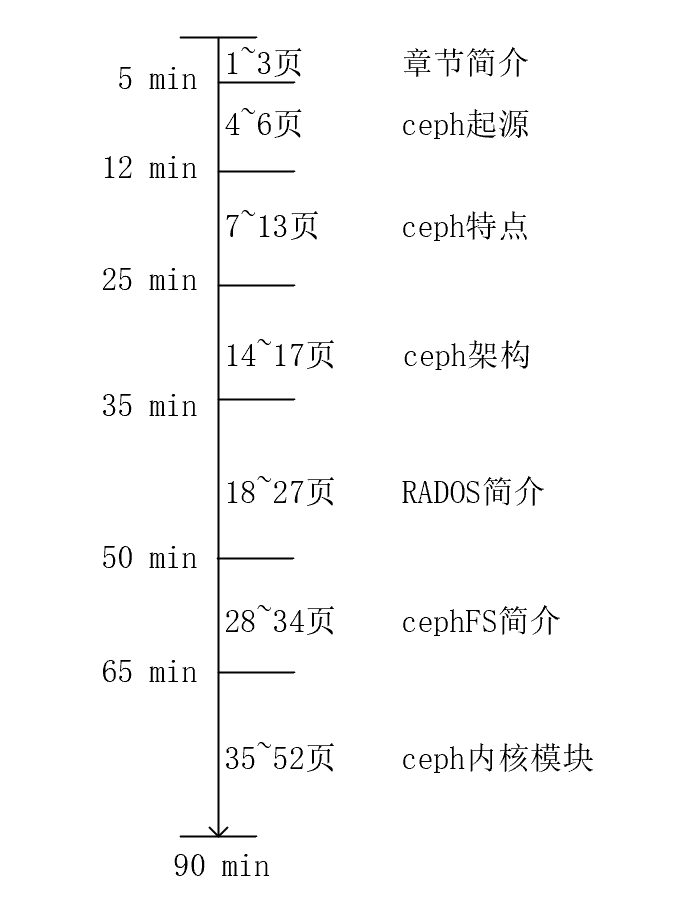
软件所制

第九章 第5讲 分布式文件系统Ceph解析

**学时：**2学时

**教学目的：**系统学习Ceph，深入了解Ceph原理

**课程时间线：**



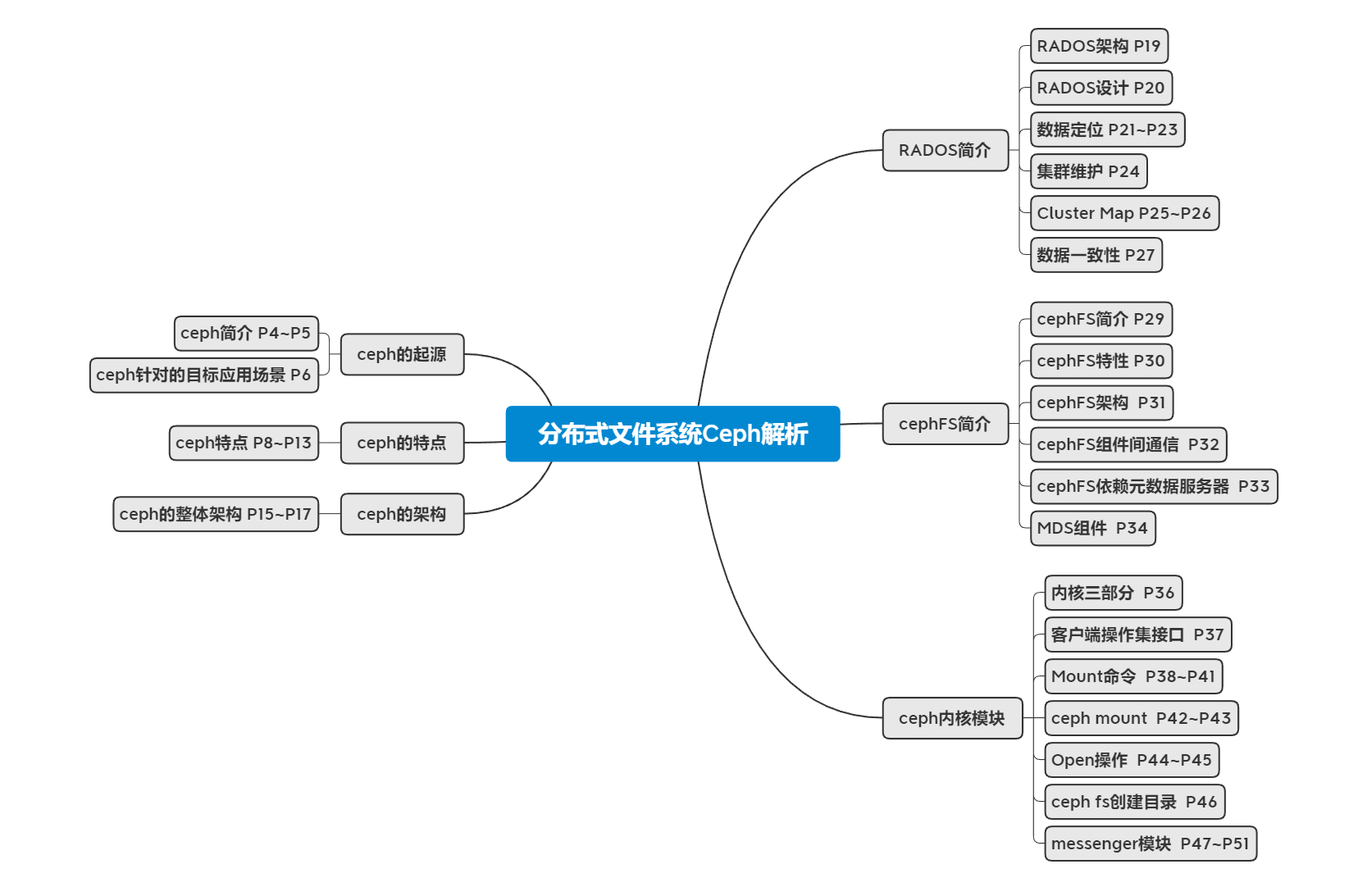
**课外参考读物：**

<https://www.cnblogs.com/kevingrace/p/8387999.html>

<https://www.jianshu.com/p/1e2898d71db8>

Ceph: A Scalable, High-Performance Distributed File System

**知识框图：**



**PPT讲稿：**

1. 我们继续第九章文件系统的学习。
2. 这一节我们主要学习分布式文件系统Ceph解析。
3. 这一讲将从ceph的起源说起。
4. Ceph是一个统一的分布式存储系统，设计初衷是提供比较好的性能、可靠性和可扩展性。Ceph存储集群专为标准硬件设计，使用crush算法来确保数据能能够平均分布在集群中，并且保障所有的集群节点能够快速检索，而没有集中瓶颈。Ceph存储提供以下三种存储服务：1.对象存储，是key/value存储，一个对象可以看成一个文件，只能全写全读，通常以大文件为主，要求足够的io带宽，兼容亚马逊的S3和OpenStack的Swift。2.块存储，它的IO特点与传统的硬盘是一致的，作为块设备像硬盘一样直接挂载。3.文件系统，如同网络文件系统一样挂载，兼容POSIX接口。
5. Ceph项目的起源来源于其创始人Sage Weil在加州大学Santa Cruz分校攻读博士期间的研究课题，项目从2004年开始并于2006年的OSDI学术会议上发表ceph论文，由此ceph开始广为人知。
6. 理解Ceph的设计思想，首先还是要了解Sage设计Ceph时所针对的目标应用场景，换言之，“做这东西的目的是啥？”。事实上，Ceph最初针对的目标应用场景，就是大规模的、分布式的存储系统。所谓“大规模”和“分布式”，是指至少能够承载PB级别的数据，并且由成千上万的存储节点组成。在Sage的思想中，对于这样一个大规模的存储系统，是不能以静态的眼光来看待的。对于其动态特性，可以概括为如下三个“变化”： 存储系统规模的变化，这样大规模的存储系统，往往不是在建设的第一天就能预料到其最终的规模，甚至是根本就不存在最终规模这个概念的。只能是随着业务的不断开展，业务规模的不断扩大，让系统承载越来越大的数据容量。这也就意味系统的规模自然随之变化，越来越大；存储系统中设备的变化，对于一个由成千上万个节点构成的系统，其节点的故障与替换必然是时常出现的情况。而系统一方面要足够可靠，不能使业务受到这种频繁出现的硬件及底层软件问题的影响，同时还应该尽可能智能化，降低相关维护操作的代价；存储系统中数据的变化，对于一个大规模的，通常被应用于互联网应用中的存储系统，其中存储的数据的变化也很可能是高度频繁的。新的数据不断写入，已有数据被更新、移动乃至删除。这种场景需求也是设计时必须予以考虑的。上述三个“变化”就是Ceph目标应用场景的关键特征。Ceph所具备的各种主要特性，也都是针对这些场景特征所提出的。
7. 接下来是ceph的特点。
8. Ceph具有高可靠性，所谓“高可靠”，首先是针对存储在系统中的数据而言，也即，尽可能保证数据不会丢失。其次，也包括数据写入过程中的可靠性，也即，在用户将数据写入Ceph存储系统的过程中，不会因为意外情况的出现造成数据丢失。高性能，为了实现高性能，ceph摒弃了传统的集中式存储元数据寻址的方案，采用了crush算法，这样数据可以分布更加均衡，并行度高；同时也考虑了容灾域的隔离，能够实现各类负载的副本放置规则，例如跨机房、机架感知等；能够支持上千个存储节点的规模，支持tb到pb级的数据。
9. 高度自动化，具体包括了数据的自动replication，自动re-balancing，自动failure detection和自动failure recovery。总体而言，这些自动化特性一方面保证了系统的高度可靠，一方面也保障了在系统规模扩大之后，其运维难度仍能保持在一个相对较低的水平。
10. 高可扩展性，这这里的“可扩展”概念比较广义，既包括了系统规模和存储容量的可扩展，也包括了随着系统节点数增加的聚合数据访问带宽的线性扩展，还包括了基于功能丰富强大的底层API提供多种功能、支持多种应用的功能性可扩展。
11. 充分发挥存储设备自身的计算能力，采用具有计算能力的设备作为存储节点，，这些节点不仅能够存储树，而且具有很强的计算能力；然而多数分布式系统都只是将这些节点作为功能简单的存储节点，如果能够更加充分的发挥这些节点的计算能力，则可以进一步提高系统的性能。
12. 去除所有的中心点，去中心化是实现高可扩展性的一个必要条件。多数分布式解决方案都是采用中心控制节点+集群子节点的架构。当系统中有中心节点时，会引入单点故障点，即当这个中心节点宕机或出现故障时整个系统便会无法提供服务更严重的会死机。如果中心点出现在数据访问的关键路径上，事实上也必然导致数据访问的延迟正大。另外中心节点会制约整个系统的可扩展性，单个节点的所拥有的资源毕竟是有限的，当系统规模扩大时中心节点的性能将不足以支撑系统的运行。
13. 总结一下，无论是“去中心化”还是“依靠存储设备自身的计算能力”，对客户端而言，最直观的感受就是“无需查表，算算就好”。无需查表就是无需像传统存储系统一样拥有一个专门的服务器用于维护元数据和数据的对应关系表，然后需要到这样的中心服务器中去查找数据。算算就好就是ceph 客户端利用本地的少量元数据，根据数据对象的ID信息简单计算即可得到数据存储位置。
14. Ceph的架构。
15. Ceph的架构氛围三层，最底层也是最核心的部分是RADOS对象存储系统，第二层是librados库层。最上层对应着不同形式的存储接口的实现。
16. RADOS自身是一个完整的分布式对象存储系统，它具有可靠、智能、分布式等特性，Ceph的高可靠、高可拓展、高性能、高自动化都是由这一层来提供的，用户数据的存储最终也都是通过这一层来进行存储的，RADOS可以说就是Ceph的核心。基于RADOS层的上一层是LIBRADOS，LIBRADOS是一个库，它允许应用程序通过访问该库来与RADOS系统进行交互，支持多种编程语言，比如C、C++、Python等。
17. 最上层面向应用提供3种不同的存储接口，分别对应三种存储：块存储接口：通过librbd库提供了块存储访问接口。RBD通过Linux内核客户端和QEMU/KVM驱动来提供一个分布式的块设备。它可以为虚拟机提供虚拟磁盘，或者通过内核映射为物理主机提供磁盘空间；对象存储接口：RADOSGW 目前提供了两种类型的API，一种是和AWS的S3接口兼容的API，另一种是和OpenStack的Swift对象接口兼容的API；文件系统接口：目前提供两种接口，一种是标准的posix接口，另一种通过libcephfs库提供文件系统访问接口。文件系统的元数据服务器MDS用于提供元数据访问。数据直接通过librados库访问。
18. RADOS简介。
19. RADOS是ceph软件栈中最关键的组件，其本身是一个支持海量存储对象的分布式对象存储系统，其结构由三类节点构成：osd全称object storage device，是负责数据的存储和维护。一个ceph集群一般都有很多个osd。Monitor负责系统的状态监测，维护cluster map，该数据结构并不是元数据，而是记录当前系统的状态，比如在线的节点、ip、crush算法的配置参数。Monitor通常是由数个节点组成的小集群。他们通过paxos同步数据，用来保存osd元数据。Client，RADOS中有众多的客户端节点，可通过计算得到对象的位置，进而直接与osd通信访问对象。
20. 下图就是RADOS的架构图，一个RADOS系统中包含大量的OSD和少量的用与管理osd集群成员的monitor。而monitor都是一些独立的进程，以及少量的本地存储，monitor之间通过一致性算法保证数据的一致性。RADOS中还有众多的客户端节点，可通过计算得到对象的位置，进而直接与osd通信访问对象。
21. RADOS中基本的存储单位是对象，一般为2MB或4MB，当一个文件要存入RADOS时，首先会被切分成大小固定的对象（最后一个对象大小可能不同），然后将对象分配到一个PG（Placement Group）中，然后PG会复制几份，伪随机地派给不同的存储节点。当新的存储节点被加入集群，会在已有数据中随机抽取一部分数据迁移到新节点。这种概率平衡的分布方式可以保证设备在潜在的高负载下正常工作。更重要的是，数据的分布过程仅需要做几次随机映射，不需要大型的集中式分配表。由图中可知，RADOS中从对象到存储节点采用的是两级映射的方式，先将对象映射到PG中，采用的是简单哈希；PG到OSD列表中采用的crush伪随机算法。
22. 第一层映射将存储对象映射到一个PG(Placement Group)， PG是一组存储对象集合， 是复制和负载均衡的最基本单位，映射方式为pgid= hash(oid) &mask，其中mask是位掩码，用于控制 PG 数量。具体来看，首先是从file->object，object的最大size是由RADOS配置的，当用户要存储一个file，需要将file切分成几个object。object -> PG：每个object都会被映射到一个PG中，然后以PG为单位进行备份以及进一步映射到具体的OSD上。这是一个哈希的过程，我们举个例子：假定这里我们共有256个PG，给每个PG编个号分别叫做0x0, 0x1, ...0xF, 0x10, 0x11... 0xFE, 0xFF。对于对象名分别为bar和foo的两个对象，对他们的对象名进行计算即HASH(‘bar’) = 0x3E0A4162和HASH(‘foo’) = 0x7FE391A0。用随机数除以PG的总数256，得到的余数一定会落在[0x0, 0xFF]之间，也就是这256个PG中的某一个：0x3E0A4162 % 0xFF ===> 0x62和0x7FE391A0 % 0xFF ===> 0xA0，于是乎，对象bar保存到编号为0x62的PG中，对象foo保存到编号为0xA0的PG中。这就是这里哈希的过程。
23. 第二层映射是将PG映射到多个OSD。映射是基于CRUSH，一个可扩展哈希的、受控的复制，CRUSH 是一个伪随机数据分布函数，能够高效地将 PG 映射到一个存储对象副本的 OSD 列表，crush的输入有cluster map描述的是集群中所有osd信息状态和其逻辑关系；placement rule限定了副本个数，以及副本放置规则例如所有副本必须放在不同机架；还有pgid表示PG的编号。Crush算法与普通哈希不同，CRUSH较为稳定， 增加或者减少OSD， 不会导致大量数据迁移， 此外CRUSH允许根据OSD存储能力，设定权重。
24. 集群维护由若干个monitor共同负责整个rados集群中所有osd状态的发现与记录，并且共同形成cluster map的master版本，然后扩散至全体OSD以及client，cluster map扩散的原因是osd需要使用cluster map进行数据的维护而client需要使用cluster map进行数据的寻址。这里因为有众多osd，所以一般的monitor不会主动轮询所有osd的当前状态，而需要osd在刚加入到集群中或者发现自身抑或是其他osd发生异常后主动向monitor上报状态信息，在收到这些上报信息后，monitor将更新cluster map信息并加以扩散。
25. 上面一直在说的cluster map有多重要呢？在PB级的存储规模下，存储系统都是动态的，即系统会随着新设备的部署和旧设备的淘汰而增长或收缩，系统内的设备会持续地崩溃和恢复，大量的数据被创建或者删除，而要实现这些都需要cluster map。cluster map会被复制到集群中的所有部分（存储节点、控制节点，甚至是客户端），并且通过怠惰地传播小增量更新而更新。cluster map中存储了整个集群的数据的分布以及成员。Cluster map不会经常变动，只有当因为一些原因，比如设备崩溃、数据迁移等，cluster map才会更新，因为频率不高所以客户端和OSD都会从monitor获取cluser map并缓存在本地。由于monitor、ODS、client都会保存cluster map，那么如何同步？当cluster map的内容需要改变时，cluster map的版本号被增加，map的版本号可以使通信的双方确认自己的map是否是最新的，版本旧的一方会先将map更新成对方的map，然后才会进行后续操作。一般的monitor中的cluster map的版本号必定最大，所以都以monitor中的cluster map为主，
26. cluster map中实际存储的大约有以下几个参数。Epoch版本号，就是刚才我们介绍的单调递增序列，epoch越大表示cluster map版本越新。各个OSD的网络地址。各个OSD的状态，这里的状态分为两个维度：up和down，in和out。Up和down是用来表示osd是否在正常工作，而in和out表示的是osd是否至少在一个PG中。然后两两组合可以组成四种可能结果，表示的状态也是组合而成。最后还有一些Crush算法的配置参数，表明了Ceph集群的物理层级关系（cluster hierarchy），位置映射规则（placement rules）。
27. 为了保证数据的安全性，一个pg会复制到多个OSD上，因此，需要在修改数据时保证副本的一致性。RADOS提供3种同步机制。在primary-copy方式下，读写操作都发往primary，primary确定写操作的顺序并转发到所有其他副本，等到其他副本写操作完成之后，primary写本地对象，然后返回ack到客户端。在chain方式下，类似Google文件系统，OSD构成一个链，写请求发往第一个OSD，该OSD指定写操作之后转发请求到后一个OSD，最后一个OSD完成写操作之后发送ack到客户端。读请求发往最后一个OSD，保证读取到的数据都有充足的副本。splay方式则更为复杂，是chain和primary-copy的结合体。其中，primary-copy是最常用的复制方式。
28. cephFS简介。
29. ceph文件系统是个POSIX兼容的文件系统，它使用ceph存储集群来存储数据，并且被继承到Linux kernel中。Cephfs相当于RADOS集群的客户端，最早的时候ceph本身就是一个分布式文件系统，但后来从ceph中剥离出了RADOS后作为一个通用的分布式对象存储系统，原先的ceph被重构成了现在的cephfs，并且进一步的在RADOS之上又构建了RBD和RADOS GW。
30. CephFS具有可扩展性，他的client是直接读写osds的，所以osd的扩展性也体现在了cephfs上。支持共享文件系统，因为cephfs本身就是一个共享文件系统，支持多个clent同时读写文件系统的一个file。具有高可用的特性，支持配置元数据服务器集群，也可以配置为activate-standby的主从服务器，提高可用性。支持任意文件/目录布局配置，如果不单独配置会默认继承父目录的布局属性。支持POSIX ACL是和Quotas。
31. Cephfs的架构最底层还是基础的osds和monitors，又添加了mds，上层是支持客户端的cephfs kernel object，cephfs fuse和cephfs library等。
32. 如图所示CephFS各个组件间通信如下：client与mds之间交互的是元数据操作和capalities；client与osd之间交互的是数据io；client与monitor之间交互的是认证与集群map等信息；mds与monitor之间交互的是心跳和集群map等信息；mds与osd之间交互的是元数据io；monitor与osd之间交互的是心跳和集群map等信息。这里的心跳是ceph的一种心跳机制，用于节点间检测对方是否发生故障，以便及时发现故障节点进入相应的故障处理流程。
33. Cephfs各组件通信时，client会首先访问元数据服务器，获得file对应的dentry、inode等信息，并得到file id到object id的映射。通过这个objcet id 可以计算得到object所在的osd便可以直接与osd通信读写文件。
34. Ceph MDS设计的比较强大，作为一个能存储PB级数据的文件系统，它充分考虑了对元数据服务器的要求，设计了MDS集群。另外也引入了MDS的动态子树迁移，MDS的热度负载均衡。但也正是这么超前的设计，使得MDS集群很难做到稳定，所以目前Jewel版本里默认还是单MDS实例，用户可配置主从MDS实例，提高可用性。 但在未来，MDS集群的这些属性都将稳定下来，为我们提供超强的元数据管理性能。上图描述了CephFS的Dynamic subtree partition功能，它支持目录分片在多个MDS之间服务，并支持基于MDS负载均衡的动态迁移。Client跟MDS通信后，会缓存对应的“目录-MDS”映射关系，这样Client任何时候都知道从哪个MDS上获取对应的元数据信息。
35. Ceph的内核模块。
36. Ceph在Linux内核中共有三个部分：cephfs，实现cephfs作为rados客户端位于fs/ceph文件中；ceph rbd实现rados block device的Linux驱动作为rados的客户端位于drivers/block/rbd；ceph messenger实现底层RPC机制messenger、monitor client、osd client、crush算法和认证加密任务位于net/ceph。
37. 首先来看cephfs。Kernel中ceph的客户端源码在fs/ceph中，文档操作接口由ceph\_file\_fops定义。
38. Mount命令是ceph的挂载命令，ceph支持两种挂载方式，一种是使用Linux内核支持的mount命令进行挂载，另一种是使用用户空间文件系统fuse进行网络磁盘挂载方式。两种挂载方式的本质差别是，前者须要有Linux内核的支持。而后者仅仅是工作在Linux上的一个应用软件。这里主要介绍的是基于Linux内核的mount命令。Mount系统调用的五个参数中的四个指针参数所对应的内容还处在用户层，并别对应了dev\_name, dirname, type, flags 和data，内容没有做任何更改，从字面上也很容易理解。接着直接将参数传递给ksys\_mount函数继续处理。ksys\_mount函数很简单，copy\_mount\_string将用户层的内容拷贝到内核层，然后就是调用do\_mount做下面的工作。
39. do\_mount代码主要工作是先将dir\_name解析为path格式到内核，一路解析flags位表，将flags拆分为mnt\_flags和sb\_flags，然后根据flags中的标记决定下面做哪一个mount操作，默认情况下为do\_new\_mount。
40. 注意到了do\_new\_mount这里原来的五个参数变成了六个参数，flags被分成了两部分，dir\_name也变成了path结构，其他还保持原格式被存入内核page中。do\_new\_mount中主要由vfs\_kern\_mount完成主要的挂载工作，其任务是填充一个vfsmount，由do\_add\_mount将上一步得到的vfsmount结构加入到全局目录树中。
41. Ceph的定义是在ceph\_fs\_type其中ceph\_mount函数的主要过程包括创建cephfs客户端，构建超级块，连接ceph cluster，最后设置根目录到cache。
42. 具体看代码，create\_fs\_client用于创建client，其中包括ceph client、mon client和osd client。通过ceph\_set\_super函数获取共享的superblock。通过ceph\_real\_mount完成ceph cluster的连接。
43. 同样的open操作通过open的系统调用，然后调用do\_sys\_open函数，在完成一系列检查过后，这里有几个关键步骤，先是将用户空间的路径名复制到内核空间，再者通过标记位获取一个未使用过的fd文件描述符，最后通过调用do\_file\_open函数完成对轮径的搜索和文件的打开。
44. Ceph目录的操作集定义在ceph\_dir\_iops中，其中包括ceph\_create。在ceph\_create直接调用ceph\_mknod。
45. Ceph\_mknod函数分为三步，第一步是调用ceph\_pre\_init\_acls()函数得到mode和struct ceph\_acls\_info数据结构内容；第二步是调用ceph\_mdsc\_create\_request(CEPH\_MDS\_OP\_MKMOD)函数创建一个mds请求；第三步是调用ceph\_mdsc\_do\_request()函数将请求同步发送给mds进程。
46. messenger模块(信使)是libceph中相对比较独立的部分，旨在为上层各种网络客户端提供稳定可靠、有序的网络服务。messenger构建在网络TCP协议之上，每当上层通过ceph\_con\_init创建一个ceph\_connection对象(对底层TCP连接的封装，含有自身的状态变化)并调用ceph\_con\_open打开该连接后(连接状态变为PREOPEN)，就会在内核工作队列中添加一项新的任务，并对该任务进行一次调度，其入口函数为con\_work。Con\_work分为两个阶段：try\_read和try\_write。Try\_read在socket连接初始化阶段不进行任何操作，因此首次连接会直接跳过try\_read运行try\_write。并在try\_wirte中建立socke连接并设定socket回调函数。
47. conection worker首次被调度时进入socket连接阶段，完成的主要工作是建立TCP socket连接，并将ceph\_connection的状态置为CONNECTING。这里有一个prepare\_wirte\_banner函数，是将客户端的banner内容放入发送缓冲区，banner是用于协商阶段，socket连接阶段最后，中间会有一个协商阶段，向服务端发送了客户端的banner，并等待服务端回复banner。当收到服务端回复banner后，connection worker再次被唤醒，此时主要完成客户端与服务端的信息协商，收到服务端的协商消息后，进行服务端协商消息的处理并进入正常打开阶段可收发消息。就可以正常调用try\_read和try\_write了。
48. 客户端与服务端进行正常消息收发时，总是先在网络连接上发送一个字节的tag，再发送实际的消息。原因是两者可以通过tag来明确消息的具体类别和解析格式。网络流上的数据大体如下图所示。
49. 我们先来看看消息的接收。当标志位为READY代表准备好接收具体的tag内容。会先从网络中接收第一个字节的tag，随后再根据tag的内容准备接受后续的消息。Tag中包括CEPH\_MSGR\_TAG\_MSG、CEPH\_MSGR\_TAG\_ACK、CEPH\_MSGR\_TAG\_CLOSE。如果tag为MSG时，表示后续为一个实际的message消息，开始接收消息内容并调用回调函数；如果tag为ACK或SEQ，则表示后续为一个服务端返回的确认号，可以释放已确认的发送消息。
50. 再来看看发送信息。Try\_write会将发送缓冲区中的内容通过网络进行发送，然后判断是否当前选定的发送消息out\_msg已经发完了，如果发完了，则检查发送队列中是否有待发送的，如果有则从发送队列中取出一个放入out\_msg并跳入上面的代码继续发送。