[**理解 OpenStack + Ceph （4）：Ceph 的基础数据结构 [Pool, Image, Snapshot, Clone]**](http://www.cnblogs.com/sammyliu/p/4843812.html)

 本系列文章会深入研究 Ceph 以及 Ceph 和 OpenStack 的集成：

（1）[安装和部署](http://www.cnblogs.com/sammyliu/p/4804037.html%20)

（2）[Ceph RBD 接口和工具](http://www.cnblogs.com/sammyliu/p/4838139.html%20)

（3）[Ceph 物理和逻辑结构](http://www.cnblogs.com/sammyliu/p/4836014.html%20)

（4）[Ceph 的基础数据结构](http://www.cnblogs.com/sammyliu/p/4843812.html%20)

（5）[Ceph 与 OpenStack 集成的实现](http://www.cnblogs.com/sammyliu/p/4838138.html)

（6）[QEMU-KVM 和 Ceph RBD 的 缓存机制总结](http://www.cnblogs.com/sammyliu/p/5066895.html)

（7）[Ceph 的基本操作和常见故障排除方法](http://www.cnblogs.com/sammyliu/p/5555218.html%20)

**1 Pool（池）**

Pool 的概念前面讲过了，Ceph 支持丰富的对 Pool 的操作，主要的包括：

[复制代码](javascript:void(0);)

列表、创建和删除 pool

ceph osd pool create {pool-name} {pg-num} [{pgp-num}] [replicated] [crush-ruleset-name]

ceph osd pool create {pool-name} {pg-num} {pgp-num} erasure [erasure-code-profile] [crush-ruleset-name]

ceph osd pool delete {pool-name} [{pool-name} --yes-i-really-really-mean-it]

QoS 支持：

ceph osd pool set-quota {pool-name} [max\_objects {obj-count}] [max\_bytes {bytes}]

快照创建和删除：

ceph osd pool mksnap {pool-name} {snap-name}

ceph osd pool rmsnap {pool-name} {snap-name}

元数据修改

ceph osd pool set {pool-name} {key} {value}

设置对象拷贝份数（注意该份数包括对象自身）

ceph osd pool set {poolname} size {num-replicas}

在 degraded 模式下的拷贝份数

ceph osd pool set data min\_size 2

[复制代码](javascript:void(0);)

**2 卷（image）**

**2.1 image 之用户所见**

Image 对应于 LVM 的 Logical Volume，它将被条带化为 N 个子数据块，每个数据块将会被以对象（object）形式保存在 RADOS 对象存储中的简单块设备（simple block devicees）。比如：

[复制代码](javascript:void(0);)

#创建 100 MB 大小的名字为 ‘myimage’ 的 RBD Image，默认情况下，它被条带化为 4MB 大小的 25 个对象 （注意 rdb create 命令的 size 参数的单位为 MB）

rbd create mypool/myimage --size 100

#同样是 100MB 大小的 RBD Image，但是它被被条带化为 8MB 大小的13 个对象

rbd create mypool/myimage --size 100 --order 23  
  
#将 image mount 到linux 主机称为一个 deivce /dev/rbd1  
rbd map mypool/myimage  
  
#向 /dev/rbd1 写入数据  
dd if=/dev/zero of=/dev/rbd1 bs=1047586 count=4

#删除 image

rbd rm mypool/myimage

[复制代码](javascript:void(0);)

**2.2 image 之 ceph 系统所见**

接下来我们来看看 image 的一些内部信息。

（1）创建新的对象

首先在一个空的 pool 中创建一个 100 GB 的 image

root@ceph1:~# rbd create -p pool100 image1 --size 102400 --image-format 2

root@ceph1:~# rbd list pool100

image1

这时候在 pool 中看到多了一些对象：

root@ceph1:~# rados -p pool100 ls

rbd\_directory

rbd\_id.image1

rbd\_header.a89c2ae8944a

从名字也能看出来，这些 object 存放的不是 image 的数据，而是 ID，header 之类的元数据信息。其中，rbd\_directory 中保存了pool内所有image的 ID 和 name 信息：

[复制代码](javascript:void(0);)

root@ceph1:~# rados -p pool100 listomapvals rbd\_directory

id\_a89c2ae8944a

value: (10 bytes) :

0000 : 06 00 00 00 69 6d 61 67 65 31 : ....image1

name\_image1

value: (16 bytes) :

0000 : 0c 00 00 00 61 38 39 63 32 61 65 38 39 34 34 61 : ....a89c2ae8944a

[复制代码](javascript:void(0);)

而 rbd\_header 保存的是一个 RBD 镜像的元数据：

[复制代码](javascript:void(0);)

root@ceph1:~# rados -p pool100 listomapvals rbd\_header.a89c2ae8944a

features

value: (8 bytes) :

0000 : 01 00 00 00 00 00 00 00 : ........

object\_prefix

value: (25 bytes) :

0000 : 15 00 00 00 72 62 64 5f 64 61 74 61 2e 61 38 39 : ....rbd\_data.a89

0010 : 63 32 61 65 38 39 34 34 61 : c2ae8944a

order

value: (1 bytes) :

0000 : 16 : .

size

value: (8 bytes) :

0000 : 00 00 00 00 19 00 00 00 : ........

snap\_seq

value: (8 bytes) :

0000 : 00 00 00 00 00 00 00 00 : ........

[复制代码](javascript:void(0);)

这些信息正是下面命令的信息来源：

[复制代码](javascript:void(0);)

root@ceph1:~# rbd -p pool100 info image1

rbd image 'image1':

size 102400 MB in 25600 objects

order 22 (4096 kB objects)

block\_name\_prefix: rbd\_data.a89c2ae8944a

format: 2

features: layering

[复制代码](javascript:void(0);)

同时还能看出来，该 image 的数据对象的名称前缀是 rbd\_header.a89c2ae8944a。而对于一个新建的 image，因为没有数据对象，其实际占用的存储空间只是元数据对象所占的非常小的空间。

（2）向该对象中写入数据 8MB 的数据（该 pool 中一个 object 是 4MB）

[复制代码](javascript:void(0);)

root@ceph1:~# rbd map pool100/image1

root@ceph1:~# rbd showmapped

id pool image snap device

1 pool100 image1 - /dev/rbd1

root@ceph1:~# dd if=/dev/zero of=/dev/rbd1 bs=1048576 count=8

8+0 records in

8+0 records out

8388608 bytes (8.4 MB) copied, 0.316369 s, 26.5 MB/s

[复制代码](javascript:void(0);)

在来看看 pool 中的对象：

root@ceph1:~# rados -p pool100 ls

rbd\_directory

rbd\_id.image1

rbd\_data.a89c2ae8944a.0000000000000000

rbd\_data.a89c2ae8944a.0000000000000001

rbd\_header.a89c2ae8944a

可以看出来多了 2 个 4MB 的 object。继续看第一个对象所在的 OSD：

root@ceph1:~# ceph osd map pool100 rbd\_data.a89c2ae8944a.0000000000000000

osdmap e81 pool 'pool100' (7) object 'rbd\_data.a89c2ae8944a.0000000000000000' -> pg 7.df059252 (7.52) -> up ([8,6,7], p8) acting ([8,6,7], p8)

PG 的 ID 是 7.52，主 OSD 是 8，从 OSD 是 6 和 7。从 OSD 树中可以获知 OSD 8 所在的节点为 ceph3：

root@ceph3:/data/osd2/current/7.52\_head# ceph osd tree

# id weight type name up/down reweight

-1 0.1399 root default

-4 0.03998 host ceph3

5 0.01999 osd.5 up 1

8 0.01999 osd.8 up 1

登录 ceph3，查看 /var/lib/ceph/osd 目录，能看到 ceph-8 目录：

root@ceph3:/var/lib/ceph/osd# ls -l

total 0

lrwxrwxrwx 1 root root 9 Sep 18 02:59 ceph-5 -> /data/osd

lrwxrwxrwx 1 root root 10 Sep 18 08:22 ceph-8 -> /data/osd2

查看 7.52 开头的目录，可以看到两个数据文件：

root@ceph3:/data/osd2/current# find . -name '\*a89c2ae8944a\*'

./7.5c\_head/rbd\uheader.a89c2ae8944a\_\_head\_36B2DADC\_\_7

./7.52\_head/rbd\udata.a89c2ae8944a.0000000000000001\_\_head\_9C6139D2\_\_7

./7.52\_head/rbd\udata.a89c2ae8944a.0000000000000000\_\_head\_DF059252\_\_7

可见：

（1）RBD image 是简单的块设备，可以直接被 mount 到主机，成为一个 device，用户可以直接写入二进制数据。

（2）image 的数据被保存为若干在 RADOS 对象存储中的对象。

（3）image 的数据空间是 thin provision 的，意味着ceph 不预分配空间，而是等到实际写入数据时按照 object 分配空间。

（4）每个 data object 被保存为多份。

（5）pool 将 RBD 镜像的ID和name等基本信息保存在 rbd\_directory 中，这样，rbd ls 命令就可以快速返回一个pool中所有的 RBD 镜像了。

（6）每个 RBD 镜像的元数据将保存在一个对象中，命名为 rbd\_header.<image id>。

（7）RBD 镜像保存在多个对象中，这些对象的命名为 rbd\_data.<image id>.<顺序编号序列>。

（8）RADOS 对象以 OSD 文件系统上的文件形式被保存，其文件名为 udata<image id>.<顺序编号序列>.<其它字符串>。

**3 快照 （snapshot）**

**3.1 snapshot 之用户所见**

    RBD image 的快照（snapshot）是该 image 在特定时刻的状态的一份只读拷贝 （A snapshot is a read-only copy of the state of an image at a particular point in time.）。需要注意的是，在做 snapshot 之前，需要停止 I/O；如果 image 中包含文件系统，系统确保文件系统是处于连续状态。

   用户可以使用 rbd 工具或者其它 API 操作 snapshot：

[复制代码](javascript:void(0);)

rbd create -p pool101 --size 102400 image1 --format 2 #创建 image  
rbd snap create pool101/image1@snap1 #创建snapshot

rbd snap ls pool101/image1 #列表

rbd snap protect pool101/image1@snap1 #保护

rbd snap unprotect pool101/image1@snap1 #去保护

rbd snap rollback pool101/image1@snap1 #回滚snapshot 到 image。注意这是个耗时操作，rbd 会显示进度条

rbd snap rm pool101/image1@snap1 #删除  
rbd snap purge pool101/image1 #删除 image 的所有snapshot  
rbd clone pool101/image1@snap1 image1snap1clone1 #创建 clone  
rbd children pool101/image1@snap1 #列表它的所有 clone

[复制代码](javascript:void(0);)

**3.2 snapshot 之 Ceph 系统所见**

我们也来看看 snapshot 的内部原理。

（1）创建 image1，写入 4MB 的数据，然后创建一个 snapshot: rbd snap create pool100/image1@snap1

（2）Ceph 在该 pool 中没有创建新的的对象，也就是说这时候并没有分配存储空间来给 snap1 创建 data objects。

[复制代码](javascript:void(0);)

root@ceph1:~# rbd map pool101/image1  
root@ceph1:~# rbd showmapped  
id pool image snap device  
1 pool100 image1 - /dev/rbd1  
2 pool101 image1 - /dev/rbd2

root@ceph1:~# dd if=/dev/sda1 of=/dev/rbd2 bs=1048576 count=4  
4+0 records in  
4+0 records out  
4194304 bytes (4.2 MB) copied, 0.123617 s, 33.9 MB/s  
root@ceph1:~# rados -p pool101 ls  
rbd\_directory  
rb.0.fc9d.238e1f29.000000000000  
image1.rbd

root@ceph1:~# rbd snap create pool101/image1@snap1

root@ceph1:~# rbd snap ls pool101/image1  
SNAPID NAME SIZE  
10 snap1 102400 MB

root@ceph1:~# rados -p pool101 ls  
rbd\_directory  
rb.0.fc9d.238e1f29.000000000000  
image1.rbd

[复制代码](javascript:void(0);)

（3）Ceph 而是将 snapshot 的信息增加到了 rbd\_header.{image\_id} 对象中

[复制代码](javascript:void(0);)

root@ceph1:~# rados -p pool101 listomapvals rbd\_header.a9262ae8944a  
snapshot\_0000000000000006

value: (74 bytes) :  
0000 : 03 01 44 00 00 00 06 00 00 00 00 00 00 00 05 00 : ..D.............  
0010 : 00 00 73 6e 61 70 31 00 00 00 00 19 00 00 00 01 : ..snap1.........  
0020 : 00 00 00 00 00 00 00 01 01 1c 00 00 00 ff ff ff : ................  
0030 : ff ff ff ff ff 00 00 00 00 fe ff ff ff ff ff ff : ................  
0040 : ff 00 00 00 00 00 00 00 00 00 : ..........

[复制代码](javascript:void(0);)

（4）再向 image1 中写入 4MB 数据（其实是覆盖第一个 object 中的数据），发现数据目录中多了一个 4MB 的文件：

root@ceph3:/data/osd2/current/8.2c\_head# ls /data/osd/current/8.3e\_head/ -l  
total 8200  
-rw-r--r-- 1 root root 4194304 Sep 28 03:25 rb.0.fc9d.238e1f29.000000000000\_\_a\_AE14D5BE\_\_8  
-rw-r--r-- 1 root root 4194304 Sep 28 03:25 rb.0.fc9d.238e1f29.000000000000\_\_head\_AE14D5BE\_\_8

可见 Ceph 使用 COW （copy on write）方式实现 snapshot：在写入object 之前，将其拷贝出来，作为 snapshot 的 data object，然后继续修改 object 中的数据。

（5）再执行命令 dd if=/dev/sda1 of=/dev/rdb1 bs=1048576 seek=4 count=4 向 image 写入 [4MB，8MB）的数据。该操作创建了第二个 data object。因为这是在做 snapshot 之后创建的，所有它和 snapshot 没有关系。

（6）再创建一个snapshot，然后修改第二个 data object，这时候第二个 data object 所在的文件夹中多出了 snapshot 的一个 data object 文件：

[复制代码](javascript:void(0);)

root@ceph3:/data/osd2/current/8.2c\_head# ls /data/osd/current/8.a\_head/ -l

total 4100

-rw-r--r-- 1 root root 4194304 Sep 28 03:31 rb.0.fc9d.238e1f29.000000000001\_\_head\_9C84738A\_\_8

root@ceph3:/data/osd2/current/8.2c\_head# ls /data/osd/current/8.a\_head/ -l

total 8200

-rw-r--r-- 1 root root 4194304 Sep 28 03:35 rb.0.fc9d.238e1f29.000000000001\_\_b\_9C84738A\_\_8

-rw-r--r-- 1 root root 4194304 Sep 28 03:35 rb.0.fc9d.238e1f29.000000000001\_\_head\_9C84738A\_\_8

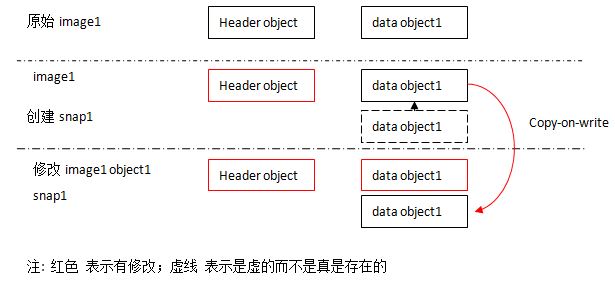
[复制代码](javascript:void(0);)

因此，

（1）snapshot 的 data objects 是和 image 的 data objects 保存在同一个目录中。

（2）snapshot 的粒度不是整个 image，而是RADOS 中的 data object。

（3）当 snapshot 创建时，只是在 image 的元数据对象中增加少量字节的元数据；当 image 的 data objects 被修改（write）时，变修改的 objects 会被拷贝（copy）出来，作为 snapshot 的 data objects。这就是 COW 的含义。



**4 克隆（clone）**

    创建 Clone 是将 image 的某一个 Snapshot 的状态复制变成一个 image。如 imageA 有一个 Snapshot-1，clone 是根据 ImageA 的 Snapshot-1 克隆得到 imageB。imageB 此时的状态与Snapshot-1完全一致，并且拥有 image 的相应能力，其区别在于 ImageB 此时可写。

**4.1 Clone 之 用户所见**

    从用户角度来看，一个 clone 和别的 RBD image 完全一样。你可以对它做 snapshot、读/写、改变大小 等等，总之从用户角度来说没什么限制。同时，创建速度很快，这是因为 Ceph 只允许从 snapshot 创建 clone，而 snapshot 一直是只读的。

[复制代码](javascript:void(0);)

rbd clone pool101/image1@snap1 image1snap1clon3

root@ceph1:~# rbd info image1snap1clon3

rbd image 'image1snap1clon3':

size 102400 MB in 25600 objects

order 22 (4096 kB objects)

block\_name\_prefix: rbd\_data.a8f63d1b58ba

format: 2

features: layering

parent: pool101/image1@snap1

overlap: 102400 MB

[复制代码](javascript:void(0);)

**4.2 Clone 之 Ceph 系统所见**

   从系统角度，clone 也是使用 COW 技术，现在来通过下面的步骤具体了解一下：

（1）创建一个 clone （创建之前，需要 protect snapshot）。你会发现 RADOS 中多了三个object：

[复制代码](javascript:void(0);)

root@ceph1:~# rbd clone pool101/image1@snap1 pool101/image1snap1clone1  
root@ceph1:~# rbd ls -p pool101  
image1  
image1snap1clone1  
root@ceph1:~# rados -p pool101 ls  
rbd\_header.89903d1b58ba  
rbd\_directory  
rbd\_id.image1snap1clone1  
rbd\_id.image1  
rbd\_children  
rbd\_header.a9532ae8944a  
rbd\_data.a9532ae8944a.0000000000000000

[复制代码](javascript:void(0);)

其中，rbd\_children 记录了父子关系：

[复制代码](javascript:void(0);)

root@ceph1:~# rados -p pool101 listomapvals rbd\_children  
key: (32 bytes):  
0000 : 08 00 00 00 00 00 00 00 0c 00 00 00 61 39 35 33 : ............a953  
0010 : 32 61 65 38 39 34 34 61 0e 00 00 00 00 00 00 00 : 2ae8944a........

value: (20 bytes) :  
0000 : 01 00 00 00 0c 00 00 00 38 39 39 30 33 64 31 62 : ........89903d1b  
0010 : 35 38 62 61 : 58ba

[复制代码](javascript:void(0);)

相比 rbd\_header.a9532ae8944a，rbd\_header.89903d1b58ba 只是多了 partent 信息：

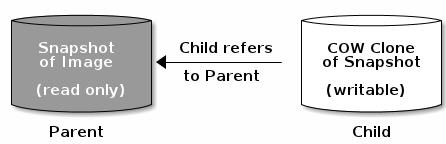
parent  
value: (46 bytes) :  
0000 : 01 01 28 00 00 00 08 00 00 00 00 00 00 00 0c 00 : ..(.............  
0010 : 00 00 61 39 35 33 32 61 65 38 39 34 34 61 0e 00 : ..a9532ae8944a..  
0020 : 00 00 00 00 00 00 00 00 00 00 19 00 00 00 : ..............

这里父子关系和 rbd children 结果的来源：

root@ceph1:~# rbd children pool101/image1@snap1

pool101/image1snap1clone1

可见，Clone 也是对 snapshot 使用 COW 方式实现的。



（2）从 clone 读数据

    从本质上是 clone 的 RBD image 中读数据，对于不是它自己的 data objects，ceph 会从它的 parent snapshot 上读，如果它也没有，继续找它的parent image，直到一个 data object 存在。从这个过程也看得出来，该过程是缺乏效率的。

（3）向 clone 中的 object 写数据

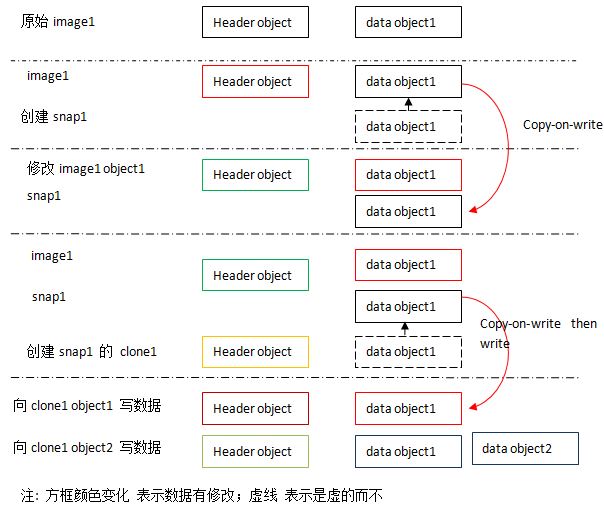
Ceph 会首先检查该 clone image 上的 data object 是否存在。如果不存在，则从 parent snapshot 或者 image 上拷贝该 data object，然后执行数据写入操作。这时候，clone 就有自己的 data object 了。

root@ceph2:/data/osd3/current/8.32\_head# ls -l

total 4100

-rw-r--r-- 1 root root 4194304 Sep 28 05:14 rbd\udata.89903d1b58ba.0000000000000000\_\_head\_CEDDC1B2\_\_8

 这是增加了 clone 后的各对象之间的关系：



**4.3 Flatten clone**

    从上面的分析我们知道，克隆操作本质上复制了一个 metadata object，而 data objects 是不存在的。因此在每次读操作时会先向本卷可能的 data object 访问。在返回对象不存在错误后会向父卷访问对应的对象最终决定这块数据是否存在。因此当存在多个层级的克隆链后，读操作需要更多的损耗去读上级卷的 data objects。只有当本卷的 data object 存在后(也就是写操作后)，才不需要访问上级卷。

    为了防止父子层数过多，Ceph 提供了 flattern 函数将 clone 与 parent snapshot 共享的 data objects 复制到 clone，并删除父子关系。

  rbd 工具的 flatten 方法：

rbd flatten <image-name>： fill clone （image-name） with data of parent (make it independent)  
如果一个 image 是个 clone，从它的 parent snapshot 拷贝所有共享的 blocks （data objects），删除与其 parent 的依赖关系。此时，其 parent snapshot 可以被去保护（unprotected），如果没有其它的 clone 的话，它是允许被删除的。 该功能要求 image 为 format 2 格式。

注意这是一个非常耗时的操作。Flatten 之后，clone 与原来的父 snapshot 之间也不再有关系了，真正成为一个独立的 image：

[复制代码](javascript:void(0);)

root@ceph1:~# rbd flatten image1snap1clon2

Image flatten: 100% complete...done.

root@ceph1:~# rbd info image1snap1clon2

rbd image 'image1snap1clon2':

size 102400 MB in 25600 objects

order 22 (4096 kB objects)

block\_name\_prefix: rbd\_data.fb173d1b58ba

format: 2

features: layering

[复制代码](javascript:void(0);)

令人奇怪的是，该操作在parent image 和 clone 产生的 image 的目录中产生了大量空的文件：

[复制代码](javascript:void(0);)

root@ceph1:/data/osd/current# ls /data/osd/current/8.7f\_head/ -l

total 788

-rw-r--r-- 1 root root 0 Sep 28 05:33 rbd\udata.89903d1b58ba.000000000000014f\_\_head\_17C7607F\_\_8

-rw-r--r-- 1 root root 0 Sep 28 05:33 rbd\udata.89903d1b58ba.0000000000000232\_\_head\_96D143FF\_\_8

-rw-r--r-- 1 root root 0 Sep 28 05:33 rbd\udata.89903d1b58ba.0000000000000399\_\_head\_4D4E557F\_\_8

-rw-r--r-- 1 root root 0 Sep 28 05:33 rbd\udata.89903d1b58ba.00000000000003ae\_\_head\_CE165DFF\_\_8

-rw-r--r-- 1 root root 0 Sep 28 05:33 rbd\udata.89903d1b58ba.00000000000003e1\_\_head\_42EA8A7F\_\_8

-rw-r--r-- 1 root root 0 Sep 28 05:33 rbd\udata.89903d1b58ba.0000000000000445\_\_head\_701607FF\_\_8

[复制代码](javascript:void(0);)

 判断父子关系层数，达到一定数目后 flatten clone，在删除 snapshot 的伪代码：

img = self.rbd.Image(client.ioctx, img\_name) #根据输入的 img\_name 获得其 RBD Image 对象  
\_pool, parent, snap = self.\_get\_clone\_info(img\_name) #当 name 为 img\_name 的 RBD image 是个 clone 时，通过递归，获取它的 parent image 和 parent snapshot

img.flatten() # 将 parent snapshot 的 data 拷贝到该 clone 中

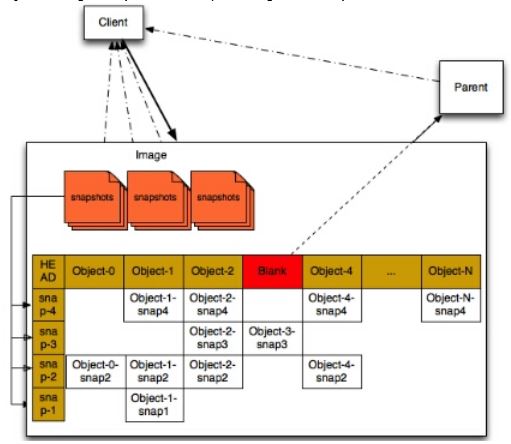
parent\_volume = self.rbd.Image(client.ioctx, parent) #获取 parent image

parent\_volume.unprotect\_snap(snap) #将 snap 去保护

parent\_volume.remove\_snap(snap) #如果snapshot 没有其它的 clone，将其删除

**5. 小结**

RBD image 的 head，object，snapshot 以及 与 client 和 parent 之间的关系：



参考链接：

<http://www.wzxue.com/ceph-librbd-block-library/>

<http://docs.ceph.com/docs/master/architecture/>

<https://www.ustack.com/blog/ceph_infra/>

<https://hustcat.github.io/rbd-image-internal-in-ceph/>

<http://www.wzxue.com/category/ceph-2/>