[**理解 OpenStack + Ceph （3）：Ceph RBD 接口和工具 [Ceph RBD API and Tools]**](http://www.cnblogs.com/sammyliu/p/4838139.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)

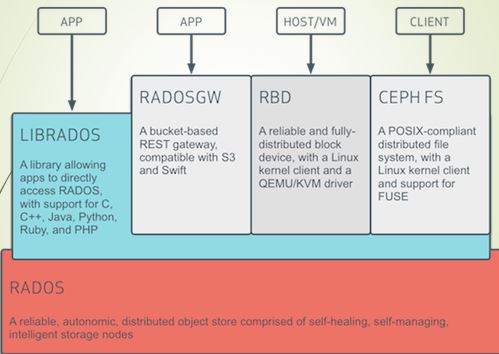
（4）[Ceph 的基础数据结构](http://www.cnblogs.com/sammyliu/p/4843812.html%20)

（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)

  Ceph 作为一个统一的分布式存储，其一大特色是提供了丰富的编程接口。我们来看看下面这张经典的图：



其中，librados 是 Ceph 的基础接口，其它的接口比如 RADOSGW, RBD 和 CephFS 都是基于 librados 实现的。本文试着分析下 Ceph 的各种接口库和常用的工具。

**1 librados**

**1.1 librados 概述**

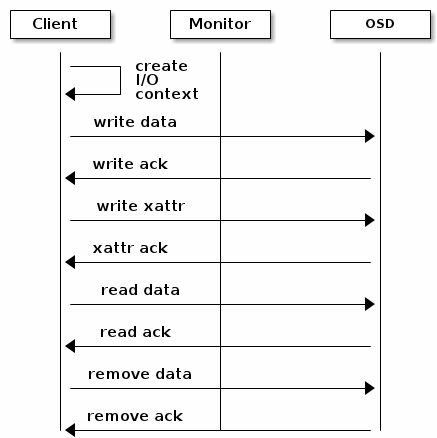
  Ceph 提供一个消息层协议（messaging layer protocol）使得 ceph 客户端可以和 Ceph Monitor 以及 Ceph OSD Daemon 交互。librados 就是一个该协议的编码库形式的实现。所有的 Ceph clients 要么使用 librados 要么使用其封装的更高层 API 与对象存储进行交互。比如，librbd 使用 librados 提供 Ceph 客户端与 RBD 交互 API。



librados 是使用 C++ 实现的。它实现了 Ceph 的一个私有协议，使得客户端可以直接、同步或者异步、并行地和 MON  和 OSD 服务通信，来执行如下操作：

* Pool Operations
* Snapshots
* Read/Write Objects
* Create or Remove
* Entire Object or Byte Range
* Append or Truncate
* Create/Set/Get/Remove XATTRs
* Create/Set/Get/Remove Key/Value Pairs
* Compound operations and dual-ack semantics

librados 于 OSD 交互的示例：

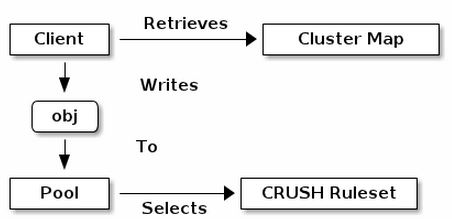


同时也提供 C, python，Java 和 PHD 语言的绑定。使用它之前，你需要安装它：

* 安装 C/C++ 版本的 librados，比如在 Ubuntu 环境中，运行 sudo apt-get install librados-dev，然后你就会在 /usr/include/rados 目录中找到C/C++的头文件。
* 使用 Python 版本的 librados，比如在 Ubuntu 环境中，运行 sudo apt-get install python-rados。具体请见 1.1.2 部分。

一个 Ceph client 通过 librados 存放或者读取数据块到 Ceph 中，需要经过以下步骤：

1. Client 调用 librados 连接到 Ceph monitor，获取 Cluster map。
2. 当 client 要读或者写数据时，它创建一个与 pool 的 I/O Context。该 pool 关联有 ruleset，它定义了数据在 Ceph 存储集群中是怎么存放的。
3. client 通过 I/O Context 提供 object name 给 librados，它使用该 object name 和 cluster map 计算出 PG 和 OSD 来定位到数据的位置。
4. client 直接和 OSD Deamon 交互来读或者写数据。



可见，第一步中，ceph client 需要知道 Ceph Minotor 的访问方法，以及用户的身份信息，而这些信息往往是放在 ceph 配置文件中，比如：

[global]

mon host = 192.168.1.1

keyring = /etc/ceph/ceph.client.admin.keyring

**1.2 librados for phthon**

    Python rados 模块是 librados 的 Python 的很薄的封装，它在 github 上的源文件在 [/src/pybind/rados.py](https://github.com/ceph/ceph/blob/master/src/pybind/rados.py), 安装到 Ubunt 后的源文件在 /usr/lib/python2.7/dist-packages/rados.py。

（1）要使用 rados.py 模块，在你的代码中 import rados

（2）创建一个 cluster handler，你需要提供 ceph.conf 文件和 keystring

import rados, sys

#Create Handle Examples. 你有多种方式来指定 ceph.conf

（1）cluster = rados.Rados(conffile='ceph.conf') #默认路径的 ceph.conf

（2）cluster = rados.Rados(conffile=sys.argv[1]) #指定路径的 ceph.conf

（3）cluster = rados.Rados(conffile = 'ceph.conf', conf = dict (keyring = '/path/to/keyring')) #默认路径的 ceph.conf 和指定的 keystring

（3）连接到 ceph cluster

cluster.connect()

（4）获取 ceph cluster 的信息，以及操作 pool

cluster\_stats = cluster.get\_cluster\_stats()  
pools = cluster.list\_pools()  
cluster.create\_pool('test')  
cluster.delete\_pool('test')

（5）向 ceph 集群读写数据，需要有一个 I/O Context （ioctx）

ioctx = cluster.open\_ioctx('data')

（6）然后就可以读写数据了

ioctx.write\_full("hw", "Hello World!") #向 “hw” object 写入 "Hello World!"

print ioctx.read("hw") #读取 ”hw“ object 的数据

ioctx.remove\_object("hw") #删除 “hw” object

更多的信息，可以参考 [LIBRADOS (PYTHON)](http://docs.ceph.com/docs/master/rados/api/python/)。

**2. librbd：Go 语言实现的访问 Ceph RBD 的接口**

RBD 是  Rados Block Device  的缩写，而 librbd 是访问 RBD 的库。它调用 librados C 语言绑定，以及与Linux 内核中的 rbd 模块通信，来提供 RMD image 的创建、删除、映射和删除映射等操作。其中，创建和删除等操作是调用 librados，而将 RDB Image 映射到主机上则是调用 Linux rbd 内核模块：

// MapDevice maps an image to a device on the host. Returns the device path and

// any errors. On error, the device path will be blank.

func (img \*Image) MapDevice() (string, error)

  librbd 的源代码在 <https://github.com/contiv/librbd>，是使用 Go 语言实现的，调用 librados 的 C 库以及linux 内核模块，在 Ubuntu 14 上它的文件在 /usr/lib/x86\_64-linux-gnu/librbd.so.1。它支持：

1. Open a connection to the rbd pool
2. Creates an image called test (Removing it before if necessary)
3. Maps it to a block device on the local host
4. Creates a snapshot of the image, called test-snap.
5. Lists and prints the snapshots available.

**3. rbd.py：librbd 的 python 封装**

[rbd python 模块](http://docs.ceph.com/docs/master/rbd/librbdpy/) rbd.py 的源代码在 <https://github.com/ceph/ceph/blob/master/src/pybind/rbd.py>，它是 librbd 的一个 python 封装。

[复制代码](javascript:void(0);)

def load\_librbd():

"""

Load the librbd shared library.

"""

librbd\_path = find\_library('rbd')

if librbd\_path:

return CDLL(librbd\_path)

# try harder, find\_library() doesn't search LD\_LIBRARY\_PATH

# in addition, it doesn't seem work on centos 6.4 (see e46d2ca067b5)

try:

return CDLL('librbd.so.1')

except OSError as e:

raise EnvironmentError("Unable to load librbd: %s" % e)

[复制代码](javascript:void(0);)

   它只封装了 librbd 中的同步操作。它主要提供了 RBD、Image 和 SnapIterator 三个类。在 Ubuntu 上，安装 python-rbd 包就可以在你的机器上找到该文件：/usr/lib/python2.7/dist-packages/rbd.py。 提供类似文件访问的方式去访问 ceph RBD image （连接-打开-使用-关闭）。该模块除了提供主要三个类 Rados, Ioctx, 和Image （Rados 对应于 ceph 集群；Ioctx 对应于 pool；Image 对应于 RBD Image）以外，还封装了 librdb 的返回值作为 Error 类。rbd 模块还提供 Error 类的具体的子类，比如 PermissionError 和 IOError。

**3.1 基本流程**

[复制代码](javascript:void(0);)

#使用 rbd 之前，连接到 RADOS，并打开一个 IO context （和特定 pool 相关）

cluster = rados.Rados(conffile='my\_ceph.conf')

cluster.connect()

ioctx = cluster.open\_ioctx('mypool')

#初始化一个 RBD 对象

rbd\_inst = rbd.RBD()

size = 4 \* 1024\*\*3 # 4 GiB

#创建 image

rbd\_inst.create(ioctx, 'myimage', size)

#初始化 image 对象

image = rbd.Image(ioctx, 'myimage')

#准备 600 个字符的数据

data = 'foo' \* 200

#写入数据

image.write(data, 0)

#关闭 image 对象

image.close()

#关闭 IO Context

ioctx.close()

#关闭连接

cluster.shutdown()

[复制代码](javascript:void(0);)

更优化的编码方式：

[复制代码](javascript:void(0);)

with rados.Rados(conffile='my\_ceph.conf') as cluster:

with cluster.open\_ioctx('mypool') as ioctx:

rbd\_inst = rbd.RBD()

size = 4 \* 1024\*\*3 # 4 GiB

rbd\_inst.create(ioctx, 'myimage', size)

with rbd.Image(ioctx, 'myimage') as image:

data = 'foo' \* 200

image.write(data, 0)

[复制代码](javascript:void(0);)

**3.2 *class*rbd.RBD 类**

*class*rbd.RBD 该类封装了 librbd 的 CURD 操作，包括：

* create(*ioctx*, *name*, *size*, *order=None*, *old\_format=True*, *features=0*, *stripe\_unit=0*, *stripe\_count=0*)：创建一个 RBD Image
* list(*ioctx*)  返回 RBD Image 的名称列表
* clone(*p\_ioctx*, *p\_name*, *p\_snapname*, *c\_ioctx*, *c\_name*, *features=0*, *order=None*)：克隆 Image 的snapshot 到一个 COW 克隆
  + **features** (*int*) – bitmask of features to enable; if set, must include layering
  + **order** (*int*) – the image is split into (2\*\*order) byte objects
* remove(*ioctx*, *name*) ：删除一个 RBD image。这可能会需要较长的时间，因为它需要等待组成该 image 的每个对象都被删除。
* rename(*ioctx*, *src*, *dest*)：修改 RBD Image 的名称

**3.3  *class*rbd.Image**

该类的一个实例代表一个 RBD image。它的方法用于对 image 的 I/O 操作和处理其 snapshot。它的主要方法包括：

* *class*rbd.Image(*ioctx*, *name*, *snapshot=None*, *read\_only=False*)
* copy(*dest\_ioctx*, *dest\_name*) Copy the image to another location.
* create\_snap(*name*)  Create a snapshot of the image
* flatten()  Flatten clone image (copy all blocks from parent to child)
* list\_snaps()  Iterate over the snapshots of an image
* protect\_snap(*name*) Mark a snapshot as protected. This means it can’t be deleted until it is unprotected.
* read(*offset*, *length*, *fadvise\_flags=0*)  Read data from the image.
* remove\_snap(*name*)  Delete a snapshot of the image
* resize(*size*)  Change the size of the image
* rollback\_to\_snap(*name*)  Revert the image to its contents at a snapshot.
* set\_snap(*name*)   Set the snapshot to read from. Writes will raise ReadOnlyImage while a snapshot is set. Pass None to unset the snapshot (reads come from the current image) , and allow writing again.
* size()  Get the size of the image  in bytes
* stat()  Get information about the image.
* unprotect\_snap(*name*)   Mark a snapshot unprotected. This allows it to be deleted if it was protected.
* write(*data*, *offset*, *fadvise\_flags=0*)  Write data to the image.
* parent\_info  Get information about a cloned image’s parent (if any)

**4. Ceph RBD 对 Linux 主机和虚机的支持，以及 Linux RBD 内核模块**

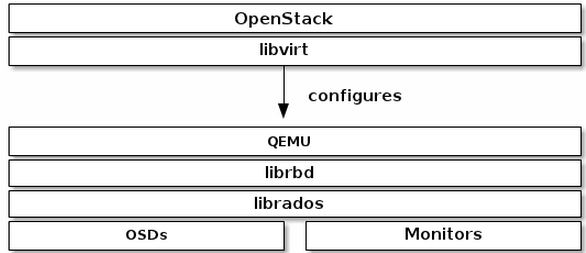
  使用 Ceph 的块存储有两种路径：

* 一种是利用QEMU走librbd路径，主要为虚拟机提供块存储设备
* 另一种是使用 kernel module，走 kernel 的路径，主要为Host提供块设备支持。

  两种途径的接口实现不完全相同。就目前来说，前者是目前更稳定的途径，也是Ceph所有应用场景中最广泛使用的。网上多篇文章认为目前内核模块尚不稳定，建议尽量不要使用。

**4.1 qemu-kvm 使用librbd 访问 RBD**

QEMU 通过 librbd 库访问 RBD，而 librbd 是调用 librados。



QEMU 对 librbd 的调用关系可以使用 ldd 命令查看：

root@compute1:~# ldd /usr/bin/qemu-system-x86\_64 | grep rbd

librbd.so.1 => /usr/lib/x86\_64-linux-gnu/librbd.so.1 (0x00007f76ef9e8000)

qemu-img 命令支持直接创建 rbd 镜像，比如：

qemu-img create -f rbd rbd:vmimages/ubuntu-newdrive 2G

然后可以使用 virsh 将它定义为一个 device，下面是 device.xml 文件：

[复制代码](javascript:void(0);)

<disk type='network' device='disk'>

<driver name='qemu' type='raw'/>

<auth username='vmimages'>

<secret type='ceph' uuid='76e3a541-b997-58ac-f7bd-77dd7d4347cb'/>

</auth>

<source protocol='rbd' name='vmimages/ubuntu-newdrive'>

<host name='192.168.0.100' port='6789'/>

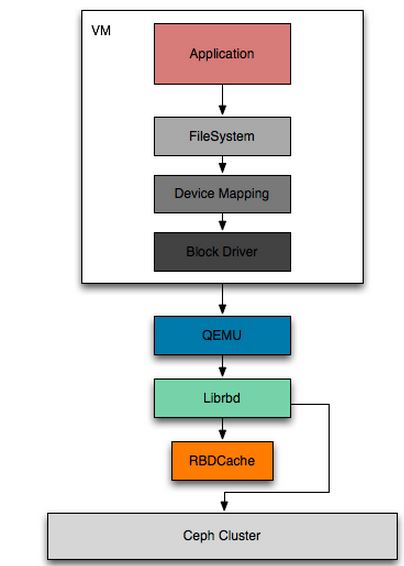
</source>

<target dev='vdc' bus='virtio'/>

</disk>

[复制代码](javascript:void(0);)

然后该 device 添加到虚机中即可。虚机访问 RBD 设备是通过：



   我们知道，Linux Kernel 中的块设备驱动是 virtio\_blk，它会对虚机的块设备各种请求封装成一个消息通过 virtio 框架提供的队列发送到 QEMU 的 IO 线程，QEMU 收到请求后会转给相应的 QEMU Block Driver 来完成请求，当使用 RDB 时 QEMU block driver 会调用 librbd 来访问 Ceph RDB 设备。

**4.2 使用内核模块，将 RBD 设备映射给主机**

    rbd 工具也可以调用 Linux RBD 内核模块来将 RBD Image 挂载到Linux 主机上作为一个新的设备。一开始，该内核模块并没有被加载，但是在执行了 rbd map 操作后，libceph 模块自动被加载：

[复制代码](javascript:void(0);)

root@compute2:~# lsmod | grep rbd                      #此时，rbd 内核模块还没有被加载

root@compute2:~# rbd -n client.cinder map vms/smb01\_d1 #做 rbd map 操作  
/dev/rbd1  
root@compute2:~# lsmod | grep rbd                      #此时， rbd 内核模块已经被自动加载  
rbd 63787 1  
libceph 225461 1 rbd  
root@compute2:~# rmmod rbd                             #这时候，rbd 内核模块也无法被卸载  
rmmod: ERROR: Module rbd is in use

[复制代码](javascript:void(0);)

在看 rbd 的帮助，我们要可以看到 map，unmap，showmapped 命令确实调用的是 rbd 内核模块：

map <image-name> map image to a block device using the kernel

unmap <device> unmap a rbd device that was mapped by the kernel

showmapped show the rbd images mapped by the kernel

过程：

[复制代码](javascript:void(0);)

#创建一个 RDB Image

root@compute1:~# rbd -n client.cinder create vms/smb01\_d2 --size 1000 --image-format 2

#将 Image map 给主机   
root@compute1:~# rbd -n client.cinder map vms/smb01\_d2 /dev/rbd2  
  
#列表已经被映射的 RDB Image  
root@compute1:~# rbd showmapped  
id pool image snap device  
1 vms smb01\_d1 - /dev/rbd1  
2 vms smb01\_d2 - /dev/rbd2  
3 vms smb01\_d3 - /dev/rbd3  
#unmap  
root@compute1:~# rbd unmap /dev/rbd1  
root@compute1:~# rbd unmap /dev/rbd2  
root@compute1:~# rbd unmap /dev/rbd3

[复制代码](javascript:void(0);)

其实，rbd 是和 /sys/bus/rbd 交互来完成map：

[复制代码](javascript:void(0);)

access("/sys/bus/rbd", F\_OK) = 0

socket(PF\_NETLINK, SOCK\_RAW|SOCK\_CLOEXEC|SOCK\_NONBLOCK, 15) = 3

setsockopt(3, SOL\_SOCKET, SO\_ATTACH\_FILTER, "\r\0\0\0\0\0\0\0\3000b\261\375\177\0\0", 16) = 0

bind(3, {sa\_family=AF\_NETLINK, pid=0, groups=00000002}, 12) = 0

getsockname(3, {sa\_family=AF\_NETLINK, pid=6958, groups=00000002}, [12]) = 0

setsockopt(3, SOL\_SOCKET, SO\_PASSCRED, [1], 4) = 0

open("/sys/bus/rbd/add\_single\_major", O\_WRONLY) = -1 ENOENT (No such file or directory)

open("/sys/bus/rbd/add", O\_WRONLY) = 4

write(4, "9.115.251.194:6789,9.115.251.195:6789,9.115.251.218:6789 nam"..., 101) = 101

close(4)

[复制代码](javascript:void(0);)

你甚至可以直接对 /sys/bus/rbd 操作来 map，showmapped 和 unmap RBD Image：

[复制代码](javascript:void(0);)

root@compute1:~#echo "10.20.30.40 name=admin rbd foo" | sudo tee /sys/bus/rbd/add #map

root@compute1:~# ls /sys/bus/rbd/devices #showmapped

1

root@compute1:~# echo 1 | sudo tee /sys/bus/rbd/remove #unmap

1

root@compute1:~# ls /sys/bus/rbd/devices

[复制代码](javascript:void(0);)

相信可以参考 [RBD KERNEL OBJECT OPERATIONS](http://docs.ceph.com/docs/argonaut/rbd/rbd-ko/) 和 [这篇文章](https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-bus-rbd)。

**5. rbd 命令行**

  Ceph 提供 rbd 作为一个操作 ceph rados 块设备（RBD）image 的 工具，根据调用的底层模块，其功能可分为两部分：

* 常规的 create, list, introspect 和 remove block device images，以及 clone images, create snapshots, rollback an image to a snapshot, view a snapshot 等操作，是通过调用 librados 来实现的。其调用层次为 rbd -> librbd -> librados：

root@compute1:~# which rbd

/usr/bin/rbd

root@compute1:~# ldd /usr/bin/rbd | grep rbd

librbd.so.1 => /usr/lib/x86\_64-linux-gnu/librbd.so.1 (0x00007fcf7ae62000)

root@compute1:~# ldd /usr/lib/x86\_64-linux-gnu/librbd.so.1 | grep rados

librados.so.2 => /usr/lib/x86\_64-linux-gnu/librados.so.2 (0x00007f29b543f000)

* 将 RBD image map 给 Linux 主机，以及从 Linux 主机 unmap等操作，是通过和 Linux rbd 内核模块通信来实现的。详情请参考 1.4.2 部分。

  在 Ubuntu 上，你安装了[ceph-common](http://packages.ubuntu.com/wily/amd64/ceph-common/filelist) 就可以得到该工具，其命令行格式为：

**rbd** [ -c *ceph.conf* ] [ -m *monaddr* ] [ -p | –pool *pool* ] [ –size *size* ] [ –order *bits* ] [ *command* ... ]

  部分使用示例：

* 在 pool ‘mypool’ 中创建 100 GB 的 rbd image ‘myimage’:  rbd create mypool/myimage --size 102400
* 在默认的 ‘rbd’ pool 中创建 image：rbd create myimage --size 102400
* 创建对象大小为 8MB 的 image：rbd create mypool/myimage --size 102400 --order 23

这里的 order 参数比较特别，它和 ceph 的对象条带化有关。ceph 客户端的一个数据块，会被条带化成多个小的对象，被保存在 Ceph 分布式对象存储（RADOS） 中。这样的话，对 image 的读和写操作可以被分散到集群的多个节点上，通常来讲这样可以防止某个 image 非常大或者非常忙时单个节点称为性能瓶颈。

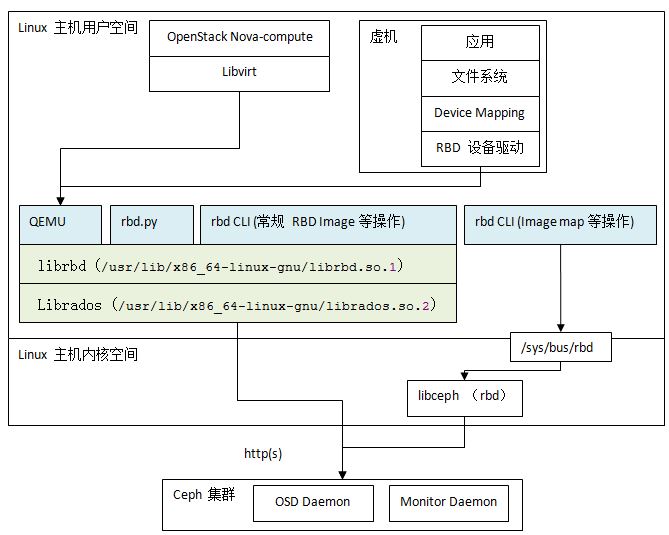
Ceph 的条带化行为受三个参数控制：

* order：被条带化的对象的大小为 2^[*order*] bytes。默认的 oder 为 22，这时候对象大小为4MB。
* stripe\_unit：每个 [stripe\_unit]的连续字节会被相连的保存到同一个对象中，直到去写下一个对象
* stripe\_count：在写入了 [stripe\_unit] 字节到 [stripe\_unit]各对象后，ceph 又重新从第一个对象开始写下一个条带，直到该对象达到了它的最大size。这时候，ceph 转移到下 [stripe\_unit] 个对象。

默认的时候，[stripe\_unit] 等于 object size；stripe\_count 为1. 要设置其他的 [*stripe\_unit*] 值，需要Ceph v0.53 版本及以后版本对 STRIPINGV2 的支持以及使用 format 2 image 格式。

* 删除 image：rbd rm mypool/myimage
* 创建 snapshot：rbd snap create mypool/myimage@mysnap
* 创建 clone：rbd clone mypool/myimage@mysnap otherpool/cloneimage
* 查看snapshot 的所有 clone：rbd children mypool/myimage@mysnap
* 删除 snapshot：rbd snap rm mypool/myimage@mysnap
* 将 image map 到 linux 主机：rbd map mypool/myimage
* 将 image 从主机上删除 （unmap）：rbd unmap /dev/rbd0

**6. 总结一下各模块的关系（以 OpenStack 的计算节点为例）**



**参考链接：**

<http://www.wzxue.com/rbdcache/>

<https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-bus-rbd>

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