kvm qemu虚拟化中的快照

快照 snapshots

虚拟机快照是指虚拟机在某个指定时间的时刻的状态，这个状态包括操作系统以及应用程序的运行状态。

快照提供的功能包括：虚拟机在线运行的时候创建快照、连续进行快照、以及回滚到某个快照的功能。

当前kvm、qemu支持在线快照的前提是镜像为qcow2（qemu copy-on-write）格式

qcow2 具有创建一个base-image，以及在base-image（backing file）的基础上创建多个copy-on-write overlays镜像的能力，所以可以使用qcow2来创建瘦虚拟机实例以及进行快照的功能。

|  |
| --- |
| [FedoraBase.img] ----- <- [Fed-guest-1.qcow2] <- [Fed-w-updates.qcow2] <- [Fedora-guest-with-updates-1A]  \  \--- <- [Fed-guest-2.qcow2] <- [Fed-w-updates.qcow2] <- [Fedora-guest-with-updates-2A] |

数据链（RootBase必须是只读的）

如上图所示，使用同一个img进行作为两个虚拟机的backing file文件，启动两个虚拟机可以这样操作

# qemu-img create -b /export/vmimages/RootBase.img -f qcow2 \

/export/vmimages/Fedora-guest-1.qcow2

# qemu-img create -b /export/vmimages/RootBase.img -f qcow2 \

/export/vmimages/Fedora-guest-2.qcow2

附录：

最新版本的qemu-img可以递归查询到整条完整的链:

# qemu-img info --backing-chain /export/vmimages/Fedora-guest-with-updates-1A.qcow2

snapshot

**内置快照（Internal Snapshots）** -- 单个qcow2镜像文件存储了包括数据以及快照的状态信息

内置快照可以进一步细分

内置快照又可以细分一下:-

1. **内置磁盘快照（Internal disk snapshot）**:

快照点的磁盘状态，数据和快照保存在单个qcow2文件中，虚拟机运行状态和关闭状态都可以创建.

* + Libvirt 使用 'qemu-img' 命令创建关机状态的磁盘快照.
  + Libvirt 使用 'savevm' 命令创建运行状态的磁盘快照.

1. **内置系统还原点（Internal system checkpoint）**:

内存状态，设备状态和磁盘状态，可以为运行中的虚拟机创建，所有信息都存储在

同一个qcow2文件中，只有在运行状态才能创建内置系统还原点.

* + Libvirt 使用'savevm' 命令来创建这种快照
* **外置快照（External Snapshots）** -- 当一个快照被创建时，创建时当前的状态保存在当前使用

的磁盘文件中，即成为一个backing file.

此时一个新的overlay被创建出来保存往后的数据.

这个也可以细分一下:-

* 1. **外置磁盘快照（External disk snapshot）**:

磁盘的快照被保存在一个文件中，创建时间点以后的数据被记录到一个新的qcow2文件中.

同样可以在运行和关闭状态创建.

* + - Libvirt 使用 'transaction' 命令来为运行状态创建这种快照.
    - Libvirt 使用'qemu-img' 命令为关闭状态创建这种快照(截止目前功能还在开发中).
  1. **外置系统还原点（External system checkpoint）**:

虚拟机的磁盘状态将被保存到一个文件中，内存和设备的状态将被保存到另外一个新的文件中，

（这个功能也还在开发中）.

* **VM状态（VM state）**:

保存运行状态虚拟机的内存设备状态信息至文件，可以通过此文件恢复到保存时的状态，有点类似系统

的休眠.（注意创建VM状态保存的时候VM磁盘必须是未发生写入改动的）

* 1. Libvirt使用 'migrate' (to file)命令来完成VM状态转储.

QEMU/KVM 快照的定义：

* 磁盘快照：磁盘的内容（可能是虚机的全部磁盘或者部分磁盘）在某个时间点上被保存，然后可以被恢复。
  + 磁盘数据的保存状态：
    - 在一个运行着的系统上，一个磁盘快照很可能只是崩溃一致的（crash-consistent） 而不是完整一致（clean）的，也是说它所保存的磁盘状态可能相当于机器突然掉电时硬盘数据的状态，机器重启后需要通过 fsck 或者别的工具来恢复到完整一致的状态（类似于 Windows 机器在断电后会执行文件检查）。
    - 对一个非运行中的虚机来说，如果上次虚机关闭的时候磁盘是完整一致的，那么其被快照的磁盘快照也将是完整一致的。
  + 磁盘快照有两种：
    - 内部快照 - 使用单个的 qcow2 的文件来保存快照和快照之后的改动。这种快照是 libvirt 的默认行为，现在的支持很完善（创建、回滚和删除），但是只能针对 qcow2 格式的磁盘镜像文件，而且其过程较慢等。
    - 外部快照 - 快照是一个只读文件，快照之后的修改是另一个 qcow2 文件中。外置快照可以针对各种格式的磁盘镜像文件。外置快照的结果是形成一个 qcow2 文件链：original <- snap1 <- snap2 <- snap3。[这里有文章](http://wiki.libvirt.org/page/I_created_an_external_snapshot,_but_libvirt_won't_let_me_delete_or_revert_to_it)详细讨论外置快照。
* 内存状态（或者虚机状态）：只是保持内存和虚机使用的其它资源的状态。如果虚机状态快照在做和恢复之间磁盘没有被修改，那么虚机将保持一个持续的状态；如果被修改了，那么很可能导致数据corruption。
* 系统还原点（system checkpoint）：虚机的所有磁盘的快照和内存状态快照的集合，可用于恢复完整的系统状态（类似于系统休眠）。

关于 崩溃一致（crash-consistent）的附加说明：

* 应该尽量避免在虚机I/O繁忙的时候做快照。这种时候做快照不是可取的办法。
* vmware 的做法是装一个 tools，它是个 PV driver，可以在做快照的时候挂起系统
* 似乎 KVM 也有类似的实现 QEMU Guest Agent，但是还不是很成熟，可参考 <http://wiki.libvirt.org/page/Qemu_guest_agent>

快照还可以分为 live snapshot（热快照）和 Clod snapshot：

* Live snapshot：系统运行状态下做的快照
* Cold snapshot：系统停止状态下的快照

libvit 做 snapshot 的各个 API：

|  |  |  |  |
| --- | --- | --- | --- |
| snapshot | 做快照的 libvirt API | 从快照恢复的 libvirt API | virsh 命令 |
| 磁盘快照 | virDomainSnapshotCreateXML（flags = VIR\_DOMAIN\_SNAPSHOT\_CREATE\_DISK\_ONLY ） | virDomainRevertToSnapshot | virsh snapshot-create/snapshot-revert |
| 内存（状态）快照 | virDomainSave  virDomainSaveFlags  virDomainManagedSave | virDomainRestore  virDomainRestoreFlags  virDomainCreate  virDomainCreateWithFlags | virsh save/restore |
| 系统检查点 | virDomainSnapshotCreateXML | virDomainRevertToSnapshot | virsh snapshot-create/snapshot-revert |

分别来看看这些 API 是如何工作的：

1. virDomainSnapshotCreateXML (virDomainPtr domain, const char \* xmlDesc, unsigned int flags)

作用：根据 xmlDesc 指定的 snapshot xml 和 flags 来创建虚机的快照。

|  |  |  |
| --- | --- | --- |
| flags 包含 | 虚机处于运行状态时快照的做法 | 虚机处于关闭状态时快照的做法 |
| 0 | 创建系统检查点，包括磁盘状态和内存状态比如内存内容 | 保持关机时的磁盘状态 |
| VIR\_DOMAIN\_SNAPSHOT\_CREATE\_LIVE | 做快照期间，虚机将不会被 paused。这会增加内存 dump file 的大小，但是可以减少系统停机时间。部分 Hypervisor 只在做外部的系统检查点时才设置该 flag，这意味着普通快照还是需要暂停虚机。 |  |
| VIR\_DOMAIN\_SNAPSHOT\_CREATE\_DISK\_ONLY | 只做指定磁盘的快照。对应运行着的虚机，磁盘快照可能是不完整的（类似于突然电源被拔了的情形）。 | 只做指定磁盘的快照。 |

其内部实现根据虚机的运行状态有两种情形：

* 对运行着的虚机，API 使用 QEMU Monitor 去做快照，磁盘镜像文件必须是 qcow2 格式，虚机的 CPU 被停止，快照结束后会重新启动。
* 对停止着的虚机，API 调用 qemu-img 方法来操作所有磁盘镜像文件。

[这里](http://libvirt.org/git/?p=libvirt.git;a=blob;f=src/qemu/qemu_driver.c;h=be678f36527fd7918a1aaa69f54a6c939f258714;hb=HEAD)有其实现代码，可见其基本的实现步骤：

[复制代码](javascript:void(0);)

static virDomainSnapshotPtr qemuDomainSnapshotCreateXML

{  
 ....

call qemuDomainSnapshotCreateDiskActive

{

call qemuProcessStopCPUs # 停止 vCPUs

for each disk call qemuDomainSnapshotCreateSingleDiskActive

{

call qemuMonitorDiskSnapshot # 调用 QEMU Monitor 去为每个磁盘做snapshot

}

call qemuProcessStartCPUs # 启动 vCPUs  
 }  
 ....

}

[复制代码](javascript:void(0);)

2. virDomainSave 相关的几个 API

这几个API 功能都比较类似：

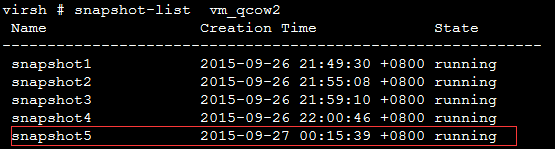
|  |  |
| --- | --- |
| virDomainSave | 该方法会 suspend 一个运行着的虚机，然后保存期内存内容到一个文件中。成功调用以后，domain 将不会处于 running 状态。使用 virDomainRestore 来恢复虚机。 |
| virDomainSaveFlags | 类似于 virDomainSave API，可使用几个  flags。一些 Hypervisor 在调用该方法前需要调用  [virDomainBlockJobAbort](http://libvirt.org/html/libvirt-libvirt-domain.html#virDomainBlockJobAbort)() 方法来停止 block copy 操作。 |
| virDomainManagedSave | 也类似于 virDomainSave API。主要区别是 libvirt 将其内存保存到一个受 libvirt 管理的文件中，因此libvirt 可以一直跟踪 snapshot 的状态；当调用 virDomainCreate/virDomainCreateWithFlags 方法重启该 domain的时候，libvirt 会使用该受管文件，而不是一个空白的文件，这样就可以 restore 该snapshot。 |

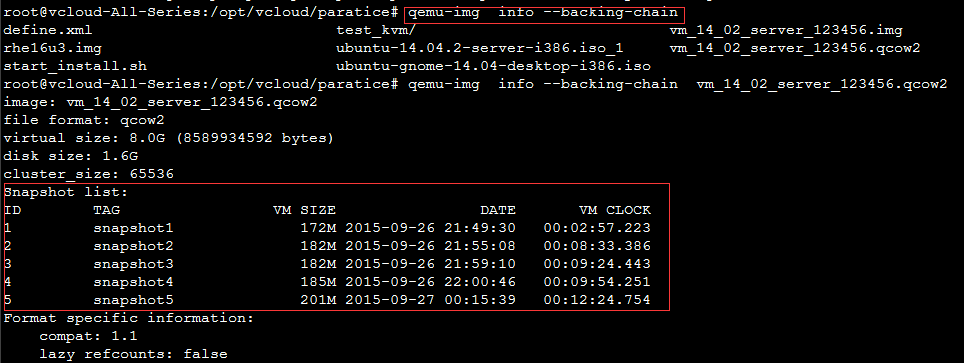
[Features/SnapshotsMultipleDevices](http://wiki.qemu.org/Features/SnapshotsMultipleDevices) 这篇文章讨论同时对多个磁盘做快照的问题。

创建镜像

内置镜像创建

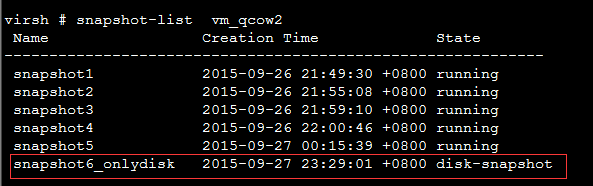


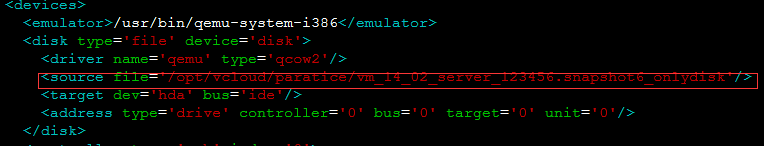


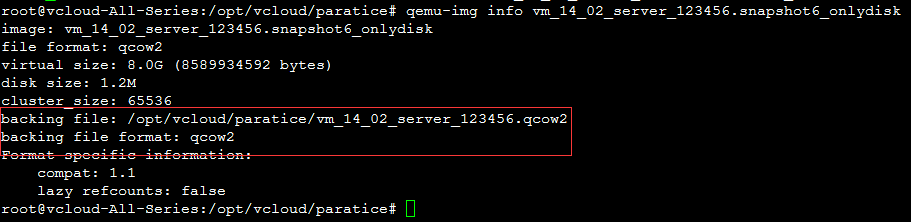


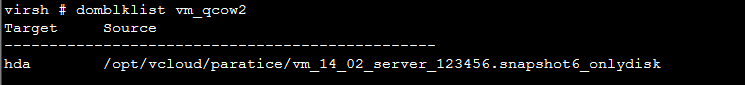
外部快照



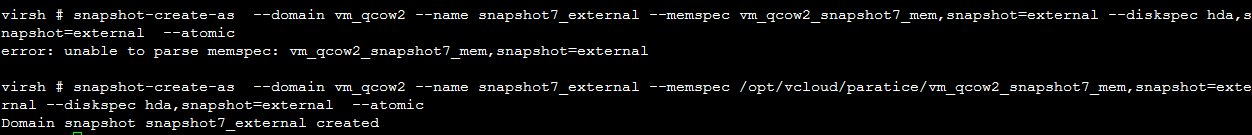


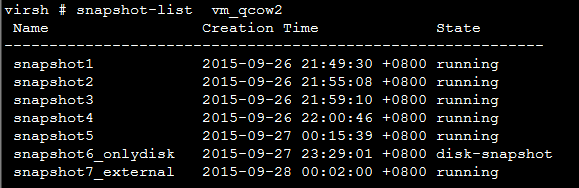


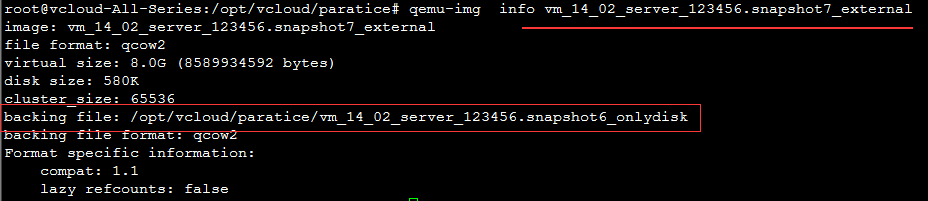




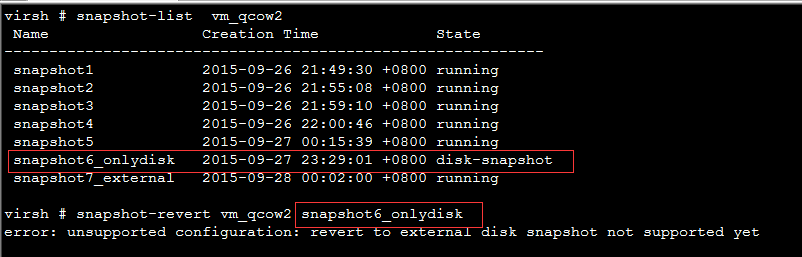
指定mem以及disk进行快照

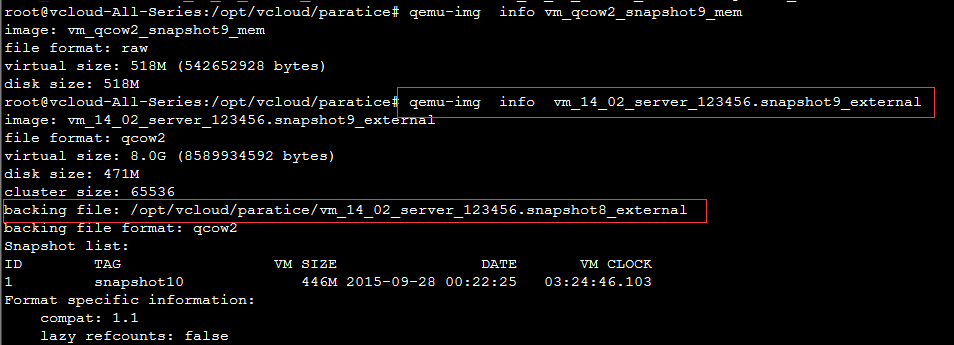




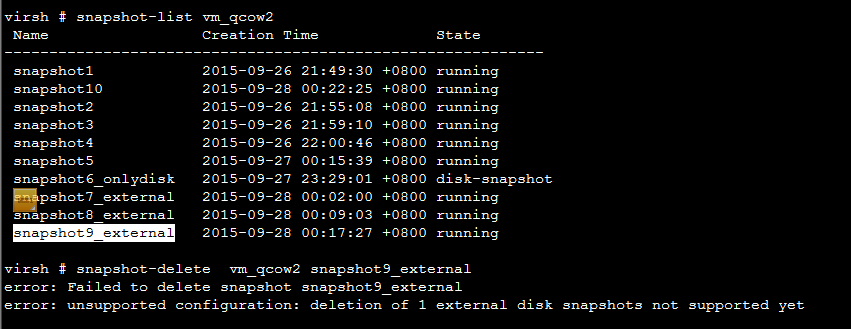


回退操作

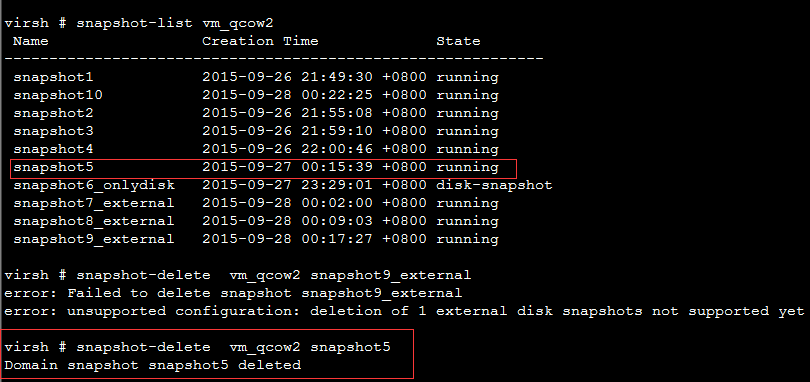




删除虚拟机快照的操作



部署虚拟机环境，出现如下错误提示：



[root@kvm img]# virt-install --virt-type kvm --name centos65 --ram 1024 --cdrom=../iso/CentOS-6.5-x86\_64-bin-DVD1.iso --disk CentOS65.qcow2,format=qcow2 --nonetworks --graphics vnc,listen=0.0.0.0 --noautoconsole --os-type=linux --os-variant=rhel6

Starting install...  
ERROR    internal error Process exited while reading console log output: char device redirected to /dev/pts/2  
qemu-kvm: -drive file=/root/work/img/CentOS65.qcow2,if=none,id=drive-virtio-disk0,format=qcow2,cache=none: could not open disk image /root/work/img/CentOS65.qcow2: Permission denied

Domain installation does not appear to have been successful.  
If it was, you can restart your domain by running:  
virsh --connect qemu:///system start centos65  
otherwise, please restart your installation.

**解决办法：**

# vim /etc/libvirt/qemu.conf  
...  
user = "root"  
group = "root"  
dynamic\_ownership = 0

**#重启libvirtd服**

/etc/init.d/libvirtd  restart

qes1:

|  |
| --- |
| sudo virsh snapshot-create-as test-vm --memspec /var/tmp/test-vm-snapshot  error: internal error: unable to execute QEMU command 'transaction': Could not open '/var/lib/libvirt/images/test-vm.img': Permission denied  解决方法  To get AppArmor (temporarily) out of the picture, I used [aa-complain](http://manpages.ubuntu.com/manpages/jaunty/man8/aa-complain.8.html), which tells AppArmor to not enforce it's policy on some processes, but still make an entry in the log. This required sudo apt-get install apparmor-utils.  First, I used sudo aa-status to see which processes were in "enforced" mode. These relevant processes were libvirtd and the VM. Then I set these to "complain" mode and made the snapshot:  sudo aa-complain /usr/sbin/libvirtd  sudo aa-complain /etc/apparmor.d/libvirt/libvirt-20683be9-691f-42f2-9fd7-7f44ab423c1e  sudo virsh snapshot-create-as test-vm --memspec /var/tmp/test-vm-snapshot  In this way, a file "test-vm-snapshot" was created in /var/tmp.  Afterwards, I used sudo aa-enforce to put these processes back in "enforce" mode.  But why /etc/init.d/apparmor stop didn't help, while the much more subtle aa-complain did, is still beyond me... |

## qcow2 的基本原理

qcow2 镜像格式是 QEMU 模拟器支持的一种磁盘镜像。它也是可以用一个文件的形式来表示一块固定大小的块设备磁盘。与普通的 raw 格式的镜像相比，有以下特性：

1. 更小的空间占用，即使文件系统不支持空洞(holes)；
2. 支持写时拷贝（COW, copy-on-write），镜像文件只反映底层磁盘的变化；
3. 支持快照（snapshot），镜像文件能够包含多个快照的历史；
4. 可选择基于 zlib 的压缩方式
5. 可以选择 AES 加密

[**回页首**](http://www.ibm.com/developerworks/cn/linux/1409_qiaoly_qemuimgages/index.html#ibm-pcon)

## qcow2 镜像文件格式

### 头部信息

每一个 qcow2 文件都以一个大端（big-endian）格式的头开始，结构如下:

##### 清单 1. qcow2 Header

typedef struct QCowHeader {

uint32\_t magic;

uint32\_t version;

uint64\_t backing\_file\_offset;

uint32\_t backing\_file\_size;

uint32\_t cluster\_bits;

uint64\_t size; /\* in bytes \*/

uint32\_t crypt\_method;

uint32\_t l1\_size;

uint64\_t l1\_table\_offset;

uint64\_t refcount\_table\_offset;

uint32\_t refcount\_table\_clusters;

uint32\_t nb\_snapshots;

uint64\_t snapshots\_offset;

} QcowHeader;

下面以一个 10G 的 qcow2 文件为例来分析各字段的含义。

##### 清单 2. qcow2 文件的 16 进制表示

# file 1.cow2

1.cow2: QEMU QCOW Image (v2), 10737418240 bytes

0000000: 5146 49fb 0000 0002 0000 0000 0000 0000 QFI.............

0000010: 0000 0000 0000 0010 0000 0002 8000 0000 ................

0000020: 0000 0000 0000 0014 0000 0000 0003 0000 ................

0000030: 0000 0000 0001 0000 0000 0001 0000 0000 ................

0000040: 0000 0000 0000 0000 0000 0000 0000 0000 ................

0000050: 0000 0000 0000 0000 0000 0000 0000 0000 ................

0000060: 0000 0004 0000 0068 0000 0000 0000 0000 .......h........

0000070: 0000 0000 0000 0000 0000 0000 0000 0000 ................

......

前 4 个比特包含了字符 Q，F，I，然后是 0xfb，实例中的 5146 49fb 是 magic 字段。

接下来的 4 个比特包含了该镜像文件的版本号，实例中的 0000 0002 是 version 字段，代表使用的是 qcow2 版本。

backing\_file\_offset 占用 8 个字节，实例中 0000 0000 0000 0000，给出一个从某个文件开始偏移量。

backing\_file\_size 给出了一个不以 null 结尾的字符串的长度，实例中为 0000 0000。如果这个镜像文件是一个写时拷贝的，那么它是原始文件的路径。

cluster\_bits，32 位（0000 0010），描述了如何映射一个镜像的地址到一个本地文件，它决定了在一个 cluster 中，偏移地址的低位是如何作为索引的。因为 L2 表占用了一个单独的 cluster 并且包含 8 字节的表项（entry），所以 cluster\_bits 只有不足 3 个位，作为 L2 表的索引。

接下来的 size ，8 字节代表了该镜像文件所表示的块设备的大小，实例中为 0000 0002 8000 0000 字节，也就是 10G 的空间。

crypt\_method 如果为 1 代表使用 AES 加密。

l1\_size（0000 0014）和 l1\_table\_offset（0000 0000 0003 0000::）分别给出了 L1 表大小和偏移量。

refcount\_table\_offset 给出 refcount 表的偏移量（0000 0000 0001 0000）而 refcount\_table\_clusters 描述了以 cluster 为单位的 refcount 表的大小（0000 0001）。

nb\_snapshots 给出了该镜像包含的快照数量（0000 0000）， snapshots\_offset 给出每个快照到 QCowSnapshotHeader 的偏移量（0000 0000 0000 0000）。

一个典型的 qcow2 镜像文件包含一下几部分：

1. 上文中提到的头部信息
2. L1 表
3. refcount 表
4. 一个或者多个 refcount 块
5. 快照头
6. L2 表
7. 数据 cluster

### 2 级查找

在 qcow2 中，磁盘的内容是保存在 cluster 中（每个 cluster 包含一些大小为 512 字节的扇区）。为了找到给定地址所在的 cluster，我们需要查找两张表，L1->L2。L1 表保存一组到 L2 表的偏移量，L2 表保存一组到 cluster 的偏移量；

所以一个地址要根据 cluster\_bits（64 位）的设置分成 3 部分，比如说 cluster\_bits=12；

低 12 位是一个 4Kb cluster 的偏移（2 的 12 次方=4Kb）；

接下来 9 位是包含 512 个表项目的 L2 表；

剩下 43 位的代表 L1 表偏移量。

为了获取一个给定地址（64 位）的偏移位置：

1. 从 Head 域中的 l1\_table\_offset 取得 L1 表的地址
2. 用前（64-l2\_bits-cluster\_bits）位地址去索引 L1 表
3. 在 L1 表中的偏移量获得 L2 表的地址
4. 用地址中的接下来的 l2\_bits 去索引 L2 表，获得一个 64 位的表项
5. 用 L2 表中的偏移量获得 cluster 的地址
6. 用地址中剩下的 cluster\_bits 位去索引该 cluster，获得该数据块

如果 L1 表和 L2 表中的偏移量都是空，这块区域就尚未被镜像文件分配。

注意 L1 表和 L2 表中的偏移量的前两位被保留，用做表示'copied' 或'compressed'。

### Copy-on-Write 镜像文件

qcow2 镜像可以用来保存另一个镜像文件的变化，它并不去修改原始镜像文件，只记录与原始镜像文件的不同即可，这种镜像文件就叫做 copy-on-write 镜像。虽然是一个单独的文件，但它的大部分的数据都来自原始镜像，只有跟原始镜像文件相比有变化的 cluster 才会被记录下来。

这很容易去实现，在头部信息中记录原始文件路径即可。当需要从一个 copy-on-write 镜像文件中读取一个 cluster 的时候，首先检查这块区域是否已经在该镜像文件中被分配，如果没有就从原始文件读取。

### 快照

快照有些类似 Copy-On-Write 文件，但区别是快照是一个可写的。快照就是原始文件本身（内部快照）。它既包含做快照之前的原始文件部分，它本身也包含可写的部分。

每一个快照都包含如下的头部结构：

**清单 3. qcow2 快照 Header**

typedef struct QCowSnapshotHeader {

/\* header is 8 byte aligned \*/

uint64\_t l1\_table\_offset;

uint32\_t l1\_size;

uint16\_t id\_str\_size;

uint16\_t name\_size;

uint32\_t date\_sec;

uint32\_t date\_nsec;

uint64\_t vm\_clock\_nsec;

uint32\_t vm\_state\_size;

uint32\_t extra\_data\_size; /\* for extension \*/

/\* extra data follows \*/

/\* id\_str follows \*/

/\* name follows \*/

} QcowSnapshotHeader;

### qcow2 的其他特性

qcow2 支持压缩，它允许每个簇（cluster）单独使用 zlib 压缩。它也支持使用 128 位的 AES 密钥进行加密。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/1409_qiaoly_qemuimgages/index.html#ibm-pcon)

## 创建 qcow2 和 raw 文件以及两种镜像的对比

使用 QEMU 软件包自带的 qemu-img 软件创建 qcow2 文件。

##### 清单 4. 创建 qcow2 和 raw 文件

$ qemu-img create -f qcow2 test.qcow2 10G

Formatting 'test.qcow2', fmt=qcow2 size=10737418240 encryption=off cluster\_size=65536 lazy\_refcounts=off

$ qemu-img create -f raw test.raw 10G

Formatting 'test.raw', fmt=raw size=10737418240

对比两种格式的文件的实际大小以及占用空间大小如下：

##### 清单 5. qcow2 和 raw 文件占用空间情况对比

$ ll -sh test.\*

200K -rw-r--r-- 1 qiaoliyong qiaoliyong 193K 5 月 6 10:29 test.qcow2

0 -rw-r--r-- 1 qiaoliyong qiaoliyong 10G 5 月 6 10:28 test.raw

[qiaoliyong@localhost ]$ stat test.raw

文件："test.raw"

大小：10737418240 块：0 IO 块：4096 普通文件

[qiaoliyong@localhost ]$ stat test.qcow2

文件："test.qcow2"

大小：197120 块：400 IO 块：4096 普通文件

从对比中可以看出 qcow 格式的镜像文件大小位 197120 字节，占用空间为 200K，占用了 200 块磁盘空间。而 raw 格式的文件则没有占用磁盘空间，它是一个空洞文件。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/1409_qiaoly_qemuimgages/index.html#ibm-pcon)

## Raw 格式与 qcow2 转化

QEMU 软件包里面提供的 qemu-img 工具可用于 image 镜像一些常用操作。

将 raw 格式转化为 qcow2 格式的文件命令如下：

qemu-img convert -f raw -O qcow2 test.raw test.raw.qcow2

[qiaoliyong@localhost kimchi]$ ll -sh test.\*

200K -rw-r--r-- 1 qiaoliyong qiaoliyong 193K 5 月 6 10:29 test.qcow2

0 -rw-r--r-- 1 qiaoliyong qiaoliyong 10G 5 月 6 10:28 test.raw

200K -rw-r--r-- 1 qiaoliyong qiaoliyong 193K 5 月 6 10:44 test.raw.qcow2

**两种格式文件的性能比较**

**表 1. 使用 ide 作为虚拟磁盘的驱动的三种镜像格式性能对比**

| **cache =** | **off** | **writethrough** | **writeback** |
| --- | --- | --- | --- |
| Old qcow2 (0.10.5) | 16:52 min | 28:58 min | 6:02 min |
| New qcow2 (0.11.0-rc1) | 5:44 min | 9:18 min | 6:11 min |
| raw | 5:41 min | 7:24 min | 6:03 min |

**表 2. 使用 virtio 作为虚拟磁盘的驱动的三种镜像格式性能对比**

| **cache =** | **off** | **writeback** |
| --- | --- | --- |
| Old qcow2 (0.10.5) | 31:09 min | 8:00 min |
| New qcow2 (0.11.0-rc1) | 18:35 min | 8:41 min |
| raw | 8:48 min | 7:51 min |

[**回页首**](http://www.ibm.com/developerworks/cn/linux/1409_qiaoly_qemuimgages/index.html#ibm-pcon)

## 小结

本文着重介绍了 QEMU 虚拟机使用的镜像文件 qcow2 的格式以及特性，并与 raw 格式镜像做了对比。qcow2 格式的文件虽然在性能上比rRaw 格式的有一些损失（主要体现在对于文件增量上，qcow2 格式的文件为了分配 cluster 多花费了一些时间），但是 qcow2 格式的镜像比 Raw 格式文件更小，只有在虚拟机实际占用了磁盘空间时，其文件才会增长，能方便的减少迁移花费的流量，更适用于云计算系统，同时，它还具有加密，压缩，以及快照等 raw 格式不具有的功能。

<http://www.cnblogs.com/sammyliu/p/4468757.html>

<http://blog.csdn.net/chenyulancn/article/details/11727381>

http://www.ibm.com/developerworks/cn/linux/1409\_qiaoly\_qemuimgages/index.html