

# 目 录

<b>1</b>	<b>前言</b>	<b>3</b>
<b>2</b>	<b>nova 创建虚拟机时对 libvirt 的调用</b>	<b>3</b>
2.1	创建镜像 . . . . .	4
2.2	创建 xml 配置文件 . . . . .	6
2.3	创建虚拟机的函数 . . . . .	7
<b>3</b>	<b>nova 冷迁移扩容时对 libvirt 的调用</b>	<b>10</b>
3.1	nova 冷迁移扩容的流程介绍 . . . . .	10
3.2	源主机上的操作 . . . . .	10
3.2.1	预备知识：虚拟机的后端镜像和增量镜像 . . . . .	10
3.2.2	migrate_disk_and_power_off 函数 . . . . .	12
3.2.3	关闭虚拟机 . . . . .	14
3.2.4	迁移镜像文件 . . . . .	16
3.2.5	源主机上冷迁移过程总结 . . . . .	17
3.3	目的主机上的操作 . . . . .	18
3.3.1	finish_migration 函数 . . . . .	18
3.3.2	对虚拟机进行扩容 . . . . .	19
3.3.3	目的主机上冷迁移过程总结 . . . . .	19
<b>4</b>	<b>nova 在线迁移时对 libvirt 的调用</b>	<b>20</b>
4.1	nova 在线迁移的流程介绍 . . . . .	20
4.2	目的主机上的操作 . . . . .	20
4.2.1	pre_live_migration 函数 . . . . .	20
4.2.2	准备相关的镜像文件 . . . . .	21
4.2.3	目的主机上在线迁移总结 . . . . .	23
4.3	源主机上的操作 . . . . .	23
4.3.1	live_migration 函数 . . . . .	23
4.3.2	源主机上在线迁移过程总结 . . . . .	24
<b>5</b>	<b>nova 创建快照时对 libvirt 的调用</b>	<b>25</b>
5.1	获得虚拟机抽象对象 . . . . .	26
5.2	获得虚拟机磁盘路径 . . . . .	26

---

5.3 获得虚拟机磁盘类型 . . . . .	26
5.4 静态创建快照 . . . . .	27
5.5 动态创建快照 . . . . .	29
<b>6 查看 libvirt 的 python API</b>	<b>31</b>

## 1 前言

在 nova/virt/libvirt/driver.py 中有 LibvirtDriver 类，用于调用 libvirt。

## 2 nova 创建虚拟机时对 libvirt 的调用

LibvirtDriver 类中创建虚拟机的函数代码如下：

```
1  def spawn(self, context, instance, image_meta, injected_files,
2      admin_password, network_info=None, block_device_info=None):
3
4      ...
5
6      # 创建镜像
7      # disk_info是一个字典，格式如下：
8      # {'disk_bus': disk_bus,      the bus for harddisks
9      #  'cdrom_bus': cdrom_bus,    the bus for CDROMs
10     #  'mapping': mapping}        the disk mapping
11     self._create_image(context, instance, disk_info['mapping'],
12                        injection_info=injection_info,
13                        block_device_info=block_device_info)
14
15     ...
16
17     # 创建xml配置文件
18     xml = self._get_guest_xml(context, instance, network_info,
19                              disk_info, image_meta,
20                              block_device_info=block_device_info)
21
22     ...
23
24     # 创建虚拟机和网络
25     self._create_domain_and_network(
26         context, xml, instance, network_info, disk_info,
27         block_device_info=block_device_info,
28         post_xml_callback=gen_confdrive,
29         destroy_disks_on_failure=True)
```

## 2.1 创建镜像

相关语句与函数如下：

```

1      self.__create_image(context, instance, disk_info['mapping'],
2                          injection_info=injection_info,
3                          block_device_info=block_device_info)
4
5      # __create_image函数如下
6      def __create_image(self, context, instance,
7                          disk_mapping, suffix='',
8                          disk_images=None, network_info=None,
9                          block_device_info=None, files=None,
10                         admin_pass=None, inject_files=True):
11          booted_from_volume = self._is_booted_from_volume(
12              instance, disk_mapping)
13
14          def image(fname, image_type=CONF.libvirt.images_type):
15              return self.image_backend.image(instance,
16                                              fname + suffix, image_type)
17
18          def raw(fname):
19              return image(fname, image_type='raw')
20
21          # 创建instance目录，用来存放镜像和libvirt.xml配置文件
22          # 调用os.makedirs(path)创建目录
23          fileutils.ensure_tree(libvirt_utils.get_instance_path(instance))
24
25          ...
26
27          # 如果disk_images为None，就创建一个disk_images字典
28          if not disk_images:
29              disk_images = {'image_id': instance['image_ref'],
30                             'kernel_id': instance['kernel_id'],
31                             'ramdisk_id': instance['ramdisk_id']}
32
33          # 如果kernel、ramdisk存在就跳过
34          # 如果不存在将调用nova.image.glance.GlanceImageService.download()函数下载
35          # image
36          if disk_images['kernel_id']:
37              fname = imagecache.get_cache_fname(disk_images, 'kernel_id')
38              raw('kernel').cache(fetch_func=libvirt_utils.fetch_image,
39                                context=context,
40                                filename=fname,
41                                image_id=disk_images['kernel_id'],
42                                user_id=instance['user_id'],
43                                project_id=instance['project_id'])
44          if disk_images['ramdisk_id']:
45              fname = imagecache.get_cache_fname(disk_images, 'ramdisk_id')
46              raw('ramdisk').cache(fetch_func=libvirt_utils.fetch_image,
47                                  context=context,
48                                  filename=fname,
49                                  image_id=disk_images['ramdisk_id'],
50                                  user_id=instance['user_id'],
51                                  project_id=instance['project_id'])

```

```

51 inst_type = flavors.extract_flavor(instance)
52
53
54 # 如果后端镜像存在就跳过
55 # 如果不存在但支持拷贝, 就掉用 Image.clone() 拷贝后端镜像
56 # 否则调用 nova.image.glance.GlanceImageService.download() 函数下载后端镜像
57 if not booted_from_volume:
58     root_fname = imagecache.get_cache_fname(disk_images, 'image_id')
59     size = instance['root_gb'] * units.Gi
60
61     if size == 0 or suffix == '.rescue':
62         size = None
63
64     backend = image('disk')
65     if backend.SUPPORTS_CLONE:
66         def clone_fallback_to_fetch(*args, **kwargs):
67             try:
68                 backend.clone(context, disk_images['image_id'])
69             except exception.ImageUnacceptable:
70                 libvirt_utils.fetch_image(*args, **kwargs)
71         fetch_func = clone_fallback_to_fetch
72     else:
73         fetch_func = libvirt_utils.fetch_image
74
75     backend.cache(fetch_func=fetch_func,
76                  context=context,
77                  filename=root_fname,
78                  size=size,
79                  image_id=disk_images['image_id'],
80                  user_id=instance['user_id'],
81                  project_id=instance['project_id'])
82
83 # Lookup the filesystem type if required
84 os_type_with_default = disk.get_fs_type_for_os_type(
85     instance['os_type'])
86
87 # 如果 ephemeral 分区存在就跳过
88 # 否则调用 nova.utils.mkfs() 创建分区
89 ephemeral_gb = instance['ephemeral_gb']
90 if 'disk.local' in disk_mapping:
91     disk_image = image('disk.local')
92     fn = functools.partial(self._create_ephemeral,
93                            fs_label='ephemeral0',
94                            os_type=instance['os_type'],
95                            is_block_dev=disk_image.is_block_dev)
96     fname = "ephemeral_%s_%s" % (ephemeral_gb, os_type_with_default)
97     size = ephemeral_gb * units.Gi
98     disk_image.cache(fetch_func=fn,
99                     context=context,
100                     filename=fname,
101                     size=size,
102                     ephemeral_size=ephemeral_gb)
103
104 for idx, eph in enumerate(driver.block_device_info_get_ephemerals(
105     block_device_info)):
106     disk_image = image(blockinfo.get_eph_disk(idx))

```

```

107         specified_fs = eph.get('guest_format')
108         if specified_fs and not self.is_supported_fs_format(specified_fs):
109             msg = _("%s format is not supported") % specified_fs
110             raise exception.InvalidBDMFormat(details=msg)
111
112         fn = functools.partial(self._create_ephemeral,
113                                fs_label='ephemeral%d' % idx,
114                                os_type=instance["os_type"],
115                                is_block_dev=disk_image.is_block_dev)
116
117         size = eph['size'] * units.Gi
118         fname = "ephemeral_%s_%s" % (eph['size'], os_type_with_default)
119         disk_image.cache(fetch_func=fn,
120                          context=context,
121                          filename=fname,
122                          size=size,
123                          ephemeral_size=eph['size'],
124                          specified_fs=specified_fs)
125
126         # 如果swap分区存在就跳过
127         # 否则调用nova.utils.mkfs()创建分区
128         if 'disk.swap' in disk_mapping:
129             mapping = disk_mapping['disk.swap']
130             swap_mb = 0
131
132             swap = driver.block_device_info_get_swap(block_device_info)
133             if driver.swap_is_usable(swap):
134                 swap_mb = swap['swap_size']
135             elif (inst_type['swap'] > 0 and
136                   not block_device.volume_in_mapping(
137                       mapping['dev'], block_device_info)):
138                 swap_mb = inst_type['swap']
139
140             if swap_mb > 0:
141                 size = swap_mb * units.Mi
142                 image('disk.swap').cache(fetch_func=self._create_swap,
143                                           context=context,
144                                           filename="swap_%s" % swap_mb,
145                                           size=size,
146                                           swap_mb=swap_mb)
147
148         ...

```

## 2.2 创建 xml 配置文件

相关语句与函数如下：

```

1     xml = self._get_guest_xml(context, instance, network_info,
2                               disk_info, image_meta,
3                               block_device_info=block_device_info)
4
5     # _get_guest_xml() 函数如下
6     def _get_guest_xml(self, context, instance, network_info, disk_info,

```

```

7         image_meta=None, rescue=None,
8         block_device_info=None, write_to_disk=False):
9         ...
10        # 根据 instance 的配置创建一个 nova.virt.libvirt.LibvirtConfigObject 类
11        conf = self._get_guest_config(instance, network_info, image_meta,
12                                     disk_info, rescue, block_device_info,
13                                     context)
14        # 获得虚拟机的 xml 文件
15        xml = conf.to_xml()
16
17        if write_to_disk:
18            # 创建 libvirt.xml 文件
19            instance_dir = libvirt_utils.get_instance_path(instance)
20            xml_path = os.path.join(instance_dir, 'libvirt.xml')
21            libvirt_utils.write_to_file(xml_path, xml)
22
23        ...
24        return xml

```

## 2.3 创建虚拟机的函数

LibvirtDriver 类中调用 libvirt 创建虚拟机和网络的函数代码如下，因为我更关注创建虚拟机的流程，所以把与创建网络有关的代码略去：

```

1    def _create_domain_and_network(self, context, xml, instance, network_info,
2                                   disk_info, block_device_info=None,
3                                   power_on=True, reboot=False,
4                                   vifs_already_plugged=False,
5                                   post_xml_callback=None,
6                                   destroy_disks_on_failure=False):
7
8        ...
9
10       # 创建虚拟机
11       guest = None
12       try:
13           with self.virtapi.wait_for_instance_event(
14               instance, events, deadline=timeout,
15               error_callback=self._neutron_failed_callback):
16               ...
17               with self._lxc_disk_handler(instance, instance.image_meta,
18                                           block_device_info, disk_info):
19                   guest = self._create_domain(
20                       xml, pause=pause, power_on=power_on,
21                       post_xml_callback=post_xml_callback)
22               ...
23       ...
24
25       return guest

```

创建虚拟机的函数代码如下：

```

1  def __create_domain(self, xml=None, domain=None,
2      power_on=True, pause=False, post_xml_callback=None):
3
4      # 创建虚拟机
5      # 这里的 libvirt_guest 在 driver.py 中有引用: from nova.virt.libvirt import
6      #      guest as libvirt_guest
7      if xml:
8          guest = libvirt_guest.Guest.create(xml, self._host)
9      else:
10         guest = libvirt_guest.Guest(domain)
11
12     ...
13
14     return guest

```

从上面的代码可以看出，nova/virt/libvirt/guest.py 中的 Guest 类就是对虚拟机 instance 的抽象。

因为我们这里是新建虚拟机，所以 domain 应该是 None，是调用 Guest.create() 函数来创建虚拟机。下面来看 Guest 类的 create() 函数：

```

1  class Guest(object):
2      @classmethod
3      def create(cls, xml, host):
4
5          try:
6              ...
7              # 这里使用 host 创建虚拟机，host 是 nova/virt/libvirt/host.py 中的 Host 类
8              guest = host.write_instance_config(xml)
9
10         ...
11
12     return guest

```

这里的 Host 类的 write\_instance\_config() 的代码如下所示：

```

1  def write_instance_config(self, xml):
2      # 直接告诉我们，get_connection() 函数返回的是与 libvirt 相关的对象
3      # 下面这行代码可以让我们想起: virsh define demo.xml 这个命令
4      # 实际上 self.get_connection() 返回的是一个 virConnect 对象
5      domain = self.get_connection().defineXML(xml)
6
7      return libvirt_guest.Guest(domain)

```

如果想知道 openstack 怎么创建一个 virConnect 对象，就继续看 get\_connection() 函数：

```

1  def get_connection(self):
2
3      try:
4          # 很清楚地看到 conn 对象是由 __get_connection() 创建的
5          conn = self._get_connection()
6
7      ...

```



```

8
9     return conn

```

进一步看 `_get_connection()` 函数：

```

1     def _get_connection(self):
2         # _wrapped_conn_lock是一个互斥锁
3         with self._wrapped_conn_lock:
4             ...
5             # _wrapped_conn是一个连接到 libvirt 的对象
6             # 当前服务没有调用 libvirt 是, _wrapped_conn是None
7             if self._wrapped_conn is None:
8                 try:
9                     # 连接到 libvirt 并返回一个对象
10                    self._wrapped_conn = self._get_new_connection()
11
12                ...
13
14    return self._wrapped_conn

```

再来看 `_get_new_connection()` 函数：

```

1     def _get_new_connection(self):
2         ...
3         # 这里的 uri 是 'qemu:///system', 根据之前的代码追踪过程可以轻易得知, 在此不再
4         # 详述如何追踪到它的值
5         # _read_only 为 False
6         # 使用 _connect() 函数创建 wrapped_conn 对象
7         # 如果熟悉 libvirt API, 那么看到这条语句会想到下面 C 语言中的这条语句:
8         # conn = virConnectPtr("qemu:///system")
9         # conn 是 virConnectPtr 对象, python 中是 virConnect 类
10        wrapped_conn = self._connect(self._uri, self._read_only)
11
12        ...
13    return wrapped_conn

```

分析到这, 应该算是比较清楚了。我们如果是熟悉 Libvirt API, 那么就能使用 libvirt 的 API 在 openstack 中呼风唤雨了。

## 3 nova 冷迁移扩容时对 libvirt 的调用

### 3.1 nova 冷迁移扩容的流程介绍

流程如下：

1. 首先，目的主机调用 `compute.ComputeManager.prep_resize()` 函数进行扩容前的准备。随后通过 RPC 通信，让源主机调用 `resize_instance()` 函数。因为 `prep_resize()` 函数与 libvirt API 的调用关系较小，在此不做分析。
2. 源主机通过 `compute.ComputeManager.resize_instance()` 函数实现了将虚拟机的增量文件迁移到目的主机上。随后通过 RPC 通信，让目的主机调用 `finish_resize()` 函数。
3. 目的主机通过 `compute.ComputeManager.finish_resize()` 函数实现了一些后续操作。

上面这个流程说得比较简略，而且也略去了 `compute-api`、`compute-conductor` 和 `compute-scheduler` 这三个服务的操作。如果想要详细了解这个过程，可以看《扩容虚拟机的流程》这篇文章。

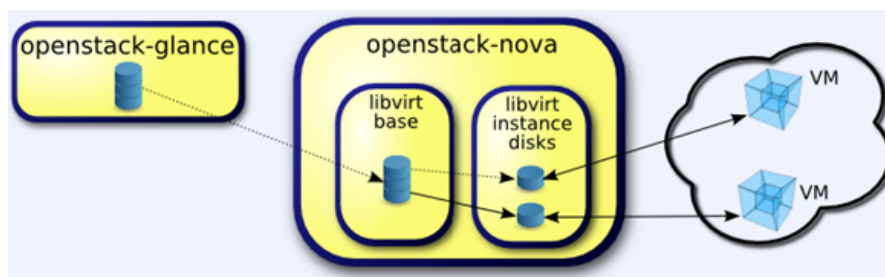
接下来，我只会分析 `LibvirtDriver` 类中实现以上功能的函数。

### 3.2 源主机上的操作

`LibvirtDriver` 类中实现 `resize_instance()` 功能的函数是 `migrate_disk_and_power_off()`。在看这个函数之前，我们需要一些预备知识。

#### 3.2.1 预备知识：虚拟机的后端镜像和增量镜像

在物理机的磁盘上，openstack 使用后端镜像和增量镜像创建虚拟机，如下图所示：



后端镜像 (base 镜像) 是虚拟机的原始镜像文件，可以说是虚拟机创建时产生的镜像文件。它的格式可以是 raw 或 qcow2，在恒天云的环境中是 raw 格式。

增量镜像又叫做 copy-on-write 镜像，由于记录与原始镜像文件的不同内容。增量镜像虽然是一个单独的文件，但它的大部分数据都来自于原始镜像，只有跟原始镜像文件相比有变化的 cluster 才会被记录下来。增量镜像的格式必须是 qcow2。

如果目的主机和源主机的 base 镜像相同，那么只需要迁移增量镜像，就可以在目的主机上使用相同的虚拟机。

如果目的主机上没有相同的 base 镜像，还需要传输 base 镜像。

openstack 中的后端镜像和增量镜像的位置如下图所示：

```
root@pengsida:~# cd /var/lib/nova/instances/
root@pengsida:/var/lib/nova/instances# tree -A
.
├── 2c1c0375-b341-40aa-997f-f41bfa218180_del
│   ├── console.log
│   ├── disk
│   ├── disk.info
│   └── libvirt.xml
├── 3a33e2ea-f39e-4837-8368-5888b9450bed_del
│   ├── console.log
│   ├── disk
│   ├── disk.info
│   └── libvirt.xml
├── 3c483039-cc68-41dc-8faf-8ba460911381_del
│   ├── console.log
│   ├── disk
│   ├── disk.info
│   └── libvirt.xml
├── 900386d4-9831-400f-91ee-7f91816def61
│   ├── console.log
│   ├── disk
│   ├── disk.info
│   └── libvirt.xml
├── 900386d4-9831-400f-91ee-7f91816def61_resize
│   ├── console.log
│   ├── disk
│   ├── disk.info
│   └── libvirt.xml
├── af87d52a-0348-45a3-9ab4-e017a2fcadb4_del
│   ├── console.log
│   ├── disk
│   ├── disk.info
│   └── libvirt.xml
├── base
│   └── ee9eeaf110c260078901a0e5f5e285899539de47
├── compute_nodes
├── locks
│   ├── nova-5318a4338597a167c7f911aefb576392be547e1b
│   ├── nova-ee9eeaf110c260078901a0e5f5e285899539de47
│   └── nova-storage-registry-lock
```

### 3.2.2 migrate\_disk\_and\_power\_off 函数

```

1  # nova/virt/libvirt/driver.py LibvirtDriver.migrate_disk_and_power_off()
2  def migrate_disk_and_power_off(self, context, instance, dest,
3                                flavor, network_info,
4                                block_device_info=None,
5                                timeout=0, retry_interval=0):
6      # block_device_info是一个字典，如下所示：
7      # {'swap': swap,
8      # 'root_device_name': root_device_name,
9      # 'ephemerals': ephemerals,
10     # 'block_device_mapping': block_device_mapping}
11     #
12     # disk_info_text数据类型是一个list，里面有多个字典，字典格式如下
13     # {'type': disk_type,
14     # 'path': path,
15     # 'virt_disk_size': virt_size,
16     # 'backing_file': backing_file,
17     # 'disk_size': dk_size,
18     # 'over_committed_disk_size': over_commit_size}
19     # 获取disk配置信息
20     disk_info_text = self.get_instance_disk_info(instance['name'],
21                                                  block_device_info=block_device_info)
22
23     # 调用json.loads()
24     disk_info = jsonutils.loads(disk_info_text)
25
26     # 调用os.path.join(CONF.instances_path, instance['name'])获得inst_base
27     # inst_base存放着虚拟机的增量镜像
28     # inst_base格式为/var/lib/nova/instances/vm-uuid
29     inst_base = libvirt_utils.get_instance_path(instance)
30     # 创建resize以后instance增量镜像的存放路径
31     inst_base_resize = inst_base + "_resize"
32
33     # 判断是否共用storage
34     # 有可能目的主机和源主机本身就是同一台
35     # 也有可能目的主机和源主机共享存储
36     shared_storage = self._is_storage_shared_with(dest, inst_base)
37
38     # 如果不是共享storage，就需要迁移增量镜像
39     # 在目的主机下根据inst_base创建增量镜像的文件夹
40     if not shared_storage:
41         utils.execute('ssh', dest, 'mkdir', '-p', inst_base)
42
43     # 关闭虚拟机，这个函数稍后分析
44     self.power_off(instance, timeout, retry_interval)
45
46     ...
47
48     try:
49         # 重命名inst_base为inst_base_resize
50         # 也就是将源主机上虚拟机增量文件所在目录名加上"_resize"
51         utils.execute('mv', inst_base, inst_base_resize)
52
53         # 如果共享storage，说明之后还需要将文件拷贝到原来的目录下

```

```

54     # 所以下面需要再创建原目录
55     if shared_storage:
56         dest = None
57         utils.execute('mkdir', '-p', inst_base)
58
59     active_flavor = flavors.extract_flavor(instance)
60
61     # 迁移虚拟机磁盘内容
62     for info in disk_info:
63         # 得到增量文件的路径
64         # img_path格式为/var/lib/nova/instances/vm-uuid/fname
65         img_path = info['path']
66         # 得到增量文件的文件名
67         fname = os.path.basename(img_path)
68         # 创建源主机下增量文件的文件名
69         from_path = os.path.join(inst_base_resize, fname)
70
71         # 如果文件是“disk.swap”，那么就不迁移
72         if (fname == 'disk.swap' and
73             active_flavor.get('swap', 0) != flavor.get('swap', 0)):
74             continue
75
76         # 这两个匿名函数用于之后的rsync ssh拷贝文件
77         on_execute = lambda process: self.job_tracker.add_job(
78             instance, process.pid)
79         on_completion = lambda process: self.job_tracker.remove_job(
80             instance, process.pid)
81
82         # 如果类型是qcow2并且有后端镜像，就把扩容后的后端文件和增量文件合并
83         # 合并后的文件存放在tmp_path中
84         # tmp_path格式为/var/lib/nova/instances/vm-uuid_resize/fname_rbase
85         if info['type'] == 'qcow2' and info['backing_file']:
86             tmp_path = from_path + "_rbase"
87             # 通过“qemu-img convert”命令将虚拟机的后端镜像和增量镜像合并
88             utils.execute('qemu-img', 'convert', '-f', 'qcow2',
89                           '-O', 'qcow2', from_path, tmp_path)
90
91         if shared_storage:
92             # 如果共享storage，此时只要重命名文件
93             # 相当于将文件又移到了/var/lib/nova/instances/vm-uuid/fname
94             utils.execute('mv', tmp_path, img_path)
95         else:
96             # 如果是不同主机，那么需要调用如下函数，稍后分析
97             libvirt_utils.copy_image(tmp_path, img_path, host=dest,
98                                     on_execute=on_execute,
99                                     on_completion=on_completion)
100             utils.execute('rm', '-f', tmp_path)
101
102         # 如果类型是raw或者没有后端镜像的qcow2，就可以直接转移
103         else:
104             libvirt_utils.copy_image(from_path, img_path, host=dest,
105                                     on_execute=on_execute,
106                                     on_completion=on_completion)
107     ...

```

### 3.2.3 关闭虚拟机

相关语句如下：

```
1 self.power_off(instance, timeout, retry_interval)
```

这个函数如下所示：

```
1 def power_off(self, instance, timeout=0, retry_interval=0):
2     if timeout:
3         self._clean_shutdown(instance, timeout, retry_interval)
4         self._destroy(instance)
```

继续追踪 `_destroy()` 函数：

```
1 def _destroy(self, instance):
2     try:
3         # 根据 virConnect.lookupByName() 获得 virDomain 对象
4         virt_dom = self._lookup_by_name(instance['name'])
5         ...
6
7
8     old_domid = -1
9     if virt_dom is not None:
10        try:
11            old_domid = virt_dom.ID()
12            # 调用 virDomain.destroy() 函数关闭虚拟机
13            virt_dom.destroy()
14            ...
15
16        # 接下来的代码都是用于判断虚拟机是否真正关机
17        def _wait_for_destroy(expected_domid):
18            try:
19                dom_info = self.get_info(instance)
20                state = dom_info['state']
21                new_domid = dom_info['id']
22            except exception.InstanceNotFound:
23                LOG.warning(_LW("During wait destroy, instance disappeared."),
24                            instance=instance)
25                raise loopingcall.LoopingCallDone()
26
27            if state == power_state.SHUTDOWN:
28                LOG.info(_LI("Instance destroyed successfully."),
29                          instance=instance)
30                raise loopingcall.LoopingCallDone()
31
32            if new_domid != expected_domid:
33                LOG.info(_LI("Instance may be started again."),
34                          instance=instance)
35                kwargs['is_running'] = True
36                raise loopingcall.LoopingCallDone()
37
38        kwargs = {'is_running': False}
39        timer = loopingcall.FixedIntervalLoopingCall(_wait_for_destroy,
40                                                       old_domid)
```

```
41     timer.start(interval=0.5).wait()
42     if kwargs['is_running']:
43         LOG.info(_LI("Going to destroy instance again."),
44                 instance=instance)
45         self._destroy(instance)
```

### 3.2.4 迁移镜像文件

相关语句如下：

```

1  # info是一个字典，如下所示：
2  # {'type': disk_type,
3  # 'path': path, path的格式为/var/lib/nova/instances/vm-uuid/
4  # 'virt_disk_size': virt_size,
5  # 'backing_file': backing_file,
6  # 'disk_size': dk_size,
7  # 'over_committed_disk_size': over_commit_size}
8  #
9  # img_path格式为/var/lib/nova/instances/vm-uuid/fname
10 # from_path格式为/var/lib/nova/instances/vm-uuid_resize/fname
11 # tmp_path格式为/var/lib/nova/instances/vm-uuid_resize/fname_rbase
12
13 if info['type'] == 'qcow2' and info['backing_file']:
14     tmp_path = from_path + "_rbase"
15     # 通过“qemu-img convert”命令将虚拟机的后端镜像和增量镜像合并
16     utils.execute('qemu-img', 'convert', '-f', 'qcow2',
17                   '-O', 'qcow2', from_path, tmp_path)
18
19     if shared_storage:
20         # 如果共享storage，此时只要重命名文件
21         # 相当于将文件又移到了/var/lib/nova/instances/vm-uuid/fname
22         utils.execute('mv', tmp_path, img_path)
23     else:
24         # 如果是不同主机，那么需要调用如下函数，稍后分析
25         libvirt_utils.copy_image(tmp_path, img_path, host=dest,
26                                  on_execute=on_execute,
27                                  on_completion=on_completion)
28         utils.execute('rm', '-f', tmp_path)
29
30 # 如果类型是raw或者是没有后端镜像的qcow2，就可以直接转移
31 # 如果共享storage，这里的host会是None
32 else:
33     libvirt_utils.copy_image(from_path, img_path, host=dest,
34                              on_execute=on_execute,
35                              on_completion=on_completion)

```

进一步查看 copy\_image 函数：

```

1  def copy_image(src, dest, host=None, on_execute=None,
2                on_completion=None):
3      # 如果共享storage，直接将from_path下的增量文件拷贝到img_path就可以了
4      if not host:
5          execute('cp', src, dest)
6      else:
7          dest = "%s:%s" % (host, dest)
8
9      # 通过rsync ssh方式拷贝合并后的镜像到新的host对于instance目录下
10     try:
11         execute('rsync', '--sparse', '--compress', '--dry-run', src, dest,
12                 on_execute=on_execute, on_completion=on_completion)
13     except processutils.ProcessExecutionError:

```



```
14         execute('scp', src, dest, on_execute=on_execute,  
15                 on_completion=on_completion)  
16     else:  
17         execute('rsync', '--sparse', '--compress', src, dest,  
18                 on_execute=on_execute, on_completion=on_completion)
```

### 3.2.5 源主机上冷迁移过程总结

根据代码总结源主机上冷迁移步骤：

1. 使用 ssh 在目的主机上建立虚机的镜像文件的目录。
2. 关闭虚拟机。
3. 在源宿主主机上将虚拟机增量文件所在的目录名加上“\_\_resize”的后缀。
4. 迁移虚拟机的增量文件。这里有多种可能性，详见上述代码的分析。

## 3.3 目的主机上的操作

### 3.3.1 finish\_migration 函数

LibvirtDriver 类中实现 finish\_resize() 功能的函数是 finish\_migration()。  
函数代码如下：

```

1  def finish_migration(self, context, migration, instance, disk_info,
2                          network_info, image_meta, resize_instance,
3                          block_device_info=None, power_on=True):
4
5      # resize disks. only "disk" and "disk.local" are necessary.
6
7      # 调用 json.loads()
8      disk_info = jsonutils.loads(disk_info)
9
10     # disk_info 是一个 list，其中存放着一个个字典，格式如下：
11     # {'type': disk_type,
12     # 'path': path, path 的格式为 /var/lib/nova/instances/vm-uuid/
13     # 'virt_disk_size': virt_size,
14     # 'backing_file': backing_file,
15     # 'disk_size': dk_size,
16     # 'over_committed_disk_size': over_commit_size}
17     for info in disk_info:
18         # 如果是 disk，就返回 root 分区的大小
19         # 如果是 disk.local，就返回 ephemeral 分区的大小
20         size = self._disk_size_from_instance(instance, info)
21
22         # 对虚拟机进行扩容
23         if resize_instance:
24             self._disk_resize(info, size)
25
26         # 将磁盘类型转为 qcow2
27         # 调用命令 "qemu-img convert -f raw -O qcow2"
28         if info['type'] == 'raw' and CONF.use_cow_images:
29             self._disk_raw_to_qcow2(info['path'])
30
31     ...
32
33     # 创建 image
34     # 这个函数在“nova 创建虚拟机时对 libvirt 的调用”一节中有分析
35     self._create_image(context, instance,
36                        disk_mapping=disk_info['mapping'],
37                        network_info=network_info,
38                        block_device_info=None, inject_files=False)
39
40     # 获得虚拟机 xml 配置文件
41     # 这个函数在“nova 创建虚拟机时对 libvirt 的调用”一节中有分析
42     xml = self._get_guest_xml(context, instance, network_info, disk_info,
43                              block_device_info=block_device_info,
44                              write_to_disk=True)
45
46     # 创建虚拟机和网络
47     # 这个函数在“nova 创建虚拟机时对 libvirt 的调用”一节中有分析

```

```

48         self._create_domain_and_network(context, xml, instance, network_info,
49                                           block_device_info, power_on,
50                                           vifs_already_plugged=True)
51
52         if power_on:
53             # 以下的代码用于检查虚拟机是否运行成功
54             timer = loopingcall.FixedIntervalLoopingCall(
55                                     self._wait_for_running,
56                                     instance)
57             timer.start(interval=0.5).wait()

```

### 3.3.2 对虚拟机进行扩容

相关语句与函数如下：

```

1         if resize_instance:
2             self._disk_resize(info, size)
3
4         # _disk_resize()函数如下
5         def _disk_resize(self, info, size):
6             # 函数功能：将一块磁盘扩容为resize大小
7
8             # If we have a non partitioned image that we can extend
9             # then ensure we're in 'raw' format so we can extend file system.
10            fmt, org = [info['type']] * 2
11            pth = info['path']
12            if (size and fmt == 'qcow2' and
13                disk.can_resize_image(pth, size) and
14                disk.is_image_partitionless(pth, use_cow=True)):
15                # 调用命令 “qemu-img convert -f qcow2 -O raw”
16                self._disk_qcow2_to_raw(pth)
17                fmt = 'raw'
18
19            if size:
20                use_cow = fmt == 'qcow2'
21                # 调用命令 “qemu-img resize pth size”
22                # 如果是raw形式，还会使用resize2fs扩展文件系统
23                # 如果是qcow2形式，就挂载设备
24                disk.extend(pth, size, use_cow=use_cow)
25
26            if fmt != org:
27                # 调用命令 “qemu-img convert -f raw -O qcow2”
28                self._disk_raw_to_qcow2(pth)

```

### 3.3.3 目的主机上冷迁移过程总结

根据代码总结目的主机上冷迁移步骤：

1. 首先对 disk 和 disk.local 进行扩容。
2. 然后创建新的虚拟机，也就是 spawn() 函数中的那几个步骤。

## 4 nova 在线迁移时对 libvirt 的调用

### 4.1 nova 在线迁移的流程介绍

流程如下：

1. 首先,源主机在 `compute.ComputeManager.live_migration()` 函数中通过 `RPC.call()` 让目的主机调用 `compute.ComputeManager.prep_live_migration()`, 并等待这个函数的完成。
2. 目的主机调用 `compute.ComputeManager.prep_live_migration()` 函数进行扩容前的准备。这个函数调用 `virt.libvirt.driver.LibvirtDriver.pre_live_migration()` 进行实际工作。函数结束后返回结果,源主机的 `compute.ComputeManager.live_migration()` 函数继续进行。
3. 源主机在 `compute.ComputeManager.live_migration()` 函数中调用 `virt.libvirt.driver.LibvirtDriver.live_migration()` 完成迁移。

上面这个流程说得比较简略,而且也略去了 `compute-api`、`compute-conductor` 和 `compute-scheduler` 这三个服务的操作。如果想要详细了解这个过程,可以看《在线迁移虚拟机的流程》这篇文章。

接下来,我只会分析 `LibvirtDriver` 类中实现以上功能的函数。

### 4.2 目的主机上的操作

#### 4.2.1 `prep_live_migration` 函数

```
1  # nova/virt/libvirt/driver.py LibvirtDriver.pre_live_migration()
2  def pre_live_migration(self, context, instance, block_device_info,
3                          network_info, disk_info, migrate_data=None):
4      """Preparation live migration."""
5
6      is_shared_block_storage = True
7      is_shared_instance_path = True
8      is_block_migration = True
9      instance_relative_path = None
10
11     # migrate_data: if not None, it is a dict which holds data
12     #                required for live migration without shared
13     #                storage.
14     # 根据 migrate_data 为 is_shared_block_storage、is_shared_instance_path 和
15     # is_block_migration 赋值
16     if migrate_data:
17         is_shared_block_storage = migrate_data.get(
18             'is_shared_block_storage', True)
19         is_shared_instance_path = migrate_data.get(
```

```

19         'is_shared_instance_path', True)
20         is_block_migration = migrate_data.get('block_migration', True)
21         instance_relative_path = migrate_data.get('instance_relative_path')
22
23     ...
24
25     if not is_shared_instance_path:
26         # 为目的主机创建 instance_dir
27         if instance_relative_path:
28             instance_dir = os.path.join(CONF.instances_path,
29                                         instance_relative_path)
30         else:
31             instance_dir = libvirt_utils.get_instance_path(instance)
32
33         if os.path.exists(instance_dir):
34             raise exception.DestinationDiskExists(path=instance_dir)
35         os.mkdir(instance_dir)
36
37         # 如果目的主机和源主机不共享存储
38         if not is_shared_block_storage:
39             # Ensure images and backing files are present.
40             # 如果使用后端镜像，这个函数将创建增量镜像
41             # 否则，直接调用“qemu-img create”创建镜像
42             # 这个函数稍后分析
43             self._create_images_and_backing(context, instance,
44                                             instance_dir, disk_info)
45
46         # 如果不是 block migration 而且不共享 instance path
47         # 还需要 kernel 和 ramdisk，所以需要调用 _fetch_instance_kernel_ramdisk()
48         if not (is_block_migration or is_shared_instance_path):
49
50             # Touch the console.log file, required by libvirt.
51             console_file = self._get_console_log_path(instance)
52             libvirt_utils.file_open(console_file, 'a').close()
53
54             # 下载 kernel 和 ramdisk 文件
55             self._fetch_instance_kernel_ramdisk(context, instance)
56
57     ...

```

#### 4.2.2 准备相关的镜像文件

```

1     def _create_images_and_backing(self, context, instance, instance_dir,
2                                   disk_info_json):
3         # disk_info 数据类型是一个 list，里面有多字典，字典格式如下
4         # {'type': disk_type,
5         #  'path': path,
6         #  'virt_disk_size': virt_size,
7         #  'backing_file': backing_file,
8         #  'disk_size': dk_size,
9         #  'over_committed_disk_size': over_commit_size}
10        if not disk_info_json:
11            disk_info = []

```

```

12         else:
13             # 调用 json.loads() 转为 list 格式
14             disk_info = jsonutils.loads(disk_info_json)
15
16         for info in disk_info:
17             # 获得文件名
18             base = os.path.basename(info['path'])
19             # Get image type and create empty disk image, and
20             # create backing file in case of qcow2.
21             instance_disk = os.path.join(instance_dir, base)
22
23             # 如果不使用 backing file 的话则调用 “qemu-img create” 方法来创建空的
24             # 磁盘镜像
25             if not info['backing_file'] and not os.path.exists(instance_disk):
26                 # 调用 “qemu-img create -f info['type'] instance_disk info['
27                 # virt_disk_size']” 创建镜像
28                 libvirt_utils.create_image(info['type'], instance_disk,
29                                           info['virt_disk_size'])
30             elif info['backing_file']:
31                 # Creating backing file follows same way as spawning instances.
32                 cache_name = os.path.basename(info['backing_file'])
33
34                 image = self.image_backend.image(instance,
35                                                  instance_disk,
36                                                  CONF.libvirt.images_type)
37
38             # 创建空的 Ephemeral disk
39             if cache_name.startswith('ephemeral'):
40                 image.cache(fetch_func=self._create_ephemeral,
41                             fs_label=cache_name,
42                             os_type=instance['os_type'],
43                             filename=cache_name,
44                             size=info['virt_disk_size'],
45                             ephemeral_size=instance['ephemeral_gb'])
46
47             # 创建空的 Swap disk
48             elif cache_name.startswith('swap'):
49                 inst_type = flavors.extract_flavor(instance)
50                 swap_mb = inst_type['swap']
51                 image.cache(fetch_func=self._create_swap,
52                             filename="swap_%s" % swap_mb,
53                             size=swap_mb * units.Mi,
54                             swap_mb=swap_mb)
55
56             else:
57                 # 从 Glance 中获取 image 来创建 Root disk
58                 image.cache(fetch_func=libvirt_utils.fetch_image,
59                             context=context,
60                             filename=cache_name,
61                             image_id=instance['image_ref'],
62                             user_id=instance['user_id'],
63                             project_id=instance['project_id'],
64                             size=info['virt_disk_size'])
65
66             # if image has kernel and ramdisk, just download
67             # following normal way.
68             self._fetch_instance_kernel_ramdisk(context, instance)

```

### 4.2.3 目的主机上在线迁移总结

根据代码总结目的主机上在线迁移的准备过程：

1. 根据 `migrate_data` 获取 `is_shared_block_storage`、`is_shared_instance_path` 和 `is_block_migration` 的信息。
2. 如果不共享 instance path，就创建 instance dir。
3. 如果不共享存储，则需要在目的主机上根据 `disk_info_json` 准备相关的镜像文件。

## 4.3 源主机上的操作

### 4.3.1 live\_migration 函数

```

1  def live_migration(self, context, instance, dest,
2                      post_method, recover_method, block_migration=False,
3                      migrate_data=None):
4      # dest是目标主机
5      # block_migration: if true, do block migration.
6      # migrate_data: implementation specific params
7
8      # 调用_live_migration()函数
9      greenthread.spawn(self._live_migration, context, instance, dest,
10                       post_method, recover_method, block_migration,
11                       migrate_data)

```

`_live_migration()` 函数如下：

```

1  def _live_migration(self, context, instance, dest, post_method,
2                      recover_method, block_migration=False,
3                      migrate_data=None):
4      # dest: destination host
5      # block_migration: if true, do block migration.
6      # migrate_data: implementation specific params
7
8      try:
9          if block_migration:
10             flaglist = CONF.libvirt.block_migration_flag.split(',')
11          else:
12             flaglist = CONF.libvirt.live_migration_flag.split(',')
13             flagvals = [getattr(libvirt, x.strip()) for x in flaglist]
14             logical_sum = reduce(lambda x, y: x | y, flagvals)
15
16             # 调用virConnect.lookupByName()获得virDomain对象
17             dom = self._lookup_by_name(instance["name"])
18
19             # pre_live_migrate_data中的参数在pre_live_migration()函数中有准备
20             # res_data = {'graphics_listen_addrs': {}}
21             # res_data['graphics_listen_addrs']['vnc'] = CONF.vncserver_listen
22             # res_data['graphics_listen_addrs']['spice'] = CONF.spice.server_listen

```

```

23     pre_live_migrate_data = (migrate_data or {}).get(
24         'pre_live_migration_result', {})
25     listen_addrs = pre_live_migrate_data.get('graphics_listen_addrs')
26
27     migratable_flag = getattr(libvirt, 'VIR_DOMAIN_XML_MIGRATABLE',
28                               None)
29
30     # 当 libvirt 带有 VIR_DOMAIN_XML_MIGRATABLE 时, Nova 会调用 libvirt 的
31     # virDomainMigrateToURI2 函数, 否则调用 virDomainMigrateToURI 函数
32     if migratable_flag is None or listen_addrs is None:
33         self._check_graphics_addresses_can_live_migrate(listen_addrs)
34         # 调用 virtDomain.migrateToURI() 函数迁移虚拟机
35         # 函数作用: Migrate the domain object from its current host to the
36         # destination host given by duri.
37         # 这个函数不能指定新的虚拟机的 xml 配置文件
38         dom.migrateToURI(CONF.libvirt.live_migration_uri % dest,
39                         logical_sum,
40                         None,
41                         CONF.libvirt.live_migration_bandwidth)
42     else:
43         old_xml_str = dom.XMLDesc(migratable_flag)
44         new_xml_str = self._correct_listen_addr(old_xml_str,
45                                                  listen_addrs)
46         try:
47             # 这个函数可以指定新的虚拟机的 xml 配置文件
48             dom.migrateToURI2(CONF.libvirt.live_migration_uri % dest,
49                             None,
50                             new_xml_str,
51                             logical_sum,
52                             None,
53                             CONF.libvirt.live_migration_bandwidth)
54         ...

```

### 4.3.2 源主机上在线迁移过程总结

根据代码总结源主机上在线迁移的准备过程:

1. 根据 migrate\_data 获取 listen\_addrs。
2. 根据是否有 VIR\_DOMAIN\_XML\_MIGRATABLE 决定调用 migrateToURI 函数还是调用 migrateToURI2 函数。



## 5 nova 创建快照时对 libvirt 的调用

LibvirtDriver 类中创建虚拟机快照的函数是 snapshot()。

这个函数主要功能的实现都与 libvirt 有关，我留取了一些主要的代码，并在之后逐个分析：

```
1  def snapshot(self, context, instance, image_id, update_task_state):
2
3      try:
4          # 得到虚拟机的抽象对象 virt_dom
5          virt_dom = self._lookup_by_name(instance['name'])
6          ...
7
8          # 得到虚拟机的磁盘路径
9          disk_path = libvirt_utils.find_disk(virt_dom)
10
11         # 得到虚拟机的磁盘类型
12         source_format = libvirt_utils.get_disk_type(disk_path)
13
14         # 如果是静态创建快照，将会执行下面这段代码，稍后会在“静态创建快照”一节分析
15         if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
16             if state == power_state.RUNNING or state == power_state.PAUSED:
17                 ...
18                 virt_dom.managedSave(0)
19
20         # snapshot_backend 也将用于静态创建快照，稍后分析
21         snapshot_backend = self.image_backend.snapshot(instance,
22                                                         disk_path,
23                                                         image_type=source_format)
24
25         with utils.tmpdir(dir=snapshot_directory) as tmpdir:
26             try:
27                 ...
28                 if live_snapshot:
29                     # 动态创建快照
30                     self._live_snapshot(virt_dom, disk_path, out_path,
31                                         image_format)
32             else:
33                 # 静态创建快照
34                 snapshot_backend.snapshot_extract(out_path, image_format)
35             ...
```

## 5.1 获得虚拟机抽象对象

函数中的语句如下:

```
1 # instance是Intance类, Instance类定义在nova/objects/instance.py中
2 # 返回了virDomain对象
3 virt_dom = self._lookup_by_name(instance['name'])
```

\_lookup\_by\_name() 函数如下:

```
1 def _lookup_by_name(self, instance_name):
2     try:
3         # 这里的__conn就是libvirt API中的virConnect对象
4         return self.__conn.lookupByName(instance_name)
5     ...
```

lookupByName 是 libvirt 的一个 API 函数, 可以根据域的名字返回 virDomain 对象。

## 5.2 获得虚拟机磁盘路径

函数中的语句如下:

```
1 # libvirt_utils在driver.py中有引用: from nova.virt.libvirt import utils as
   libvirt_utils
2 disk_path = libvirt_utils.find_disk(virt_dom)
```

find\_disk 函数如下:

```
1 def find_disk(virt_dom):
2     # 在这里, 我们只关心下面这条语句
3     # virt_dom是libvirt的virDomain对象
4     xml_desc = virt_dom.XMLDesc(0)
5     ...
```

这里的 XMLDesc() 是 virDomain 类的一个成员函数, 用于提供域的 XML 配置文件。也就是说, 获得虚拟机磁盘路径的核心函数是 virDomain.XMLDesc()。

虽然我这里省略了 find\_disk() 函数接下来的代码, 但是大家应该也能知道, 接下来就是解析这个 xml 文件, 得到域的磁盘路径。

## 5.3 获得虚拟机磁盘类型

函数中的语句如下:

```
1 # libvirt_utils在driver.py中有引用: from nova.virt.libvirt import utils as
   libvirt_utils
2 # 返回了虚拟机磁盘的类型
```

```
3 source_format = libvirt_utils.get_disk_type(disk_path)
```

get\_disk\_type() 函数如下:

```
1 def get_disk_type(path):
2     ...
3     # 这个函数用于得到磁盘的类型
4     # qemu_img_info() 返回了是 QemuImgInfo 类
5     # QemuImgInfo 类定义于 nova/openstack/common/imageutils.py 中
6     # QemuImgInfo.file_format 存放着磁盘类型的信息
7     return images.qemu_img_info(path).file_format
```

继续追踪 qemu\_img\_info() 函数:

```
1 def qemu_img_info(path):
2     ...
3     # 这条语句相当于执行命令行: qemu-img info disk_path, 返回镜像文件的信息
4     # utils 是 nova/utils.py 文件
5     out, err = utils.execute('env', 'LC_ALL=C', 'LANG=C',
6                             'qemu-img', 'info', path)
7     ...
8     # 返回 QemuImgInfo 类, 这个类定义于 nova/openstack/common/imageutils.py 中
9     return imageutils.QemuImgInfo(out)
```

所以说, openstack 获得虚拟机磁盘信息, 最终是调用了“qemu-img info <filename>”这条命令。

## 5.4 静态创建快照

可以从 snapshot() 函数知道静态创建快照的相关语句如下:

```
1     ...
2     if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
3         if state == power_state.RUNNING or state == power_state.PAUSED:
4             ...
5             #
6             virt_dom.managedSave(0)
7         # 获得与静态创建快照有关的数据结构
8         snapshot_backend = self.image_backend.snapshot(instance,
9                                                         disk_path,
10                                                         image_type=source_format)
11     ...
12     with utils.tmpdir(dir=snapshot_directory) as tmpdir:
13         try:
14             ...
15             else:
16                 # 静态创建快照
17                 snapshot_backend.snapshot_extract(out_path, image_format)
```

我们逐句分析这三句注释了的代码：

1. 第一句注释代码：

```
1      # virt_dom是libvirt的virDomain对象
2      # virDomain.managedSave()函数用于将虚拟机的域挂起，然后将它的内存信息保存
      到虚拟机镜像的一个文件中
3      virt_dom.managedSave(0)
```

其实第一步已经完成了对虚拟机创建快照，只是现在这个快照保存在镜像文件中，还需要将它提取出来。

2. 第二句注释代码：

```
1      # image_backend是nova/libvirt/imagebackend.py文件中定义的Backend类
2      # snapshot()函数根据image_type返回相应的类
3      snapshot_backend = self.image_backend.snapshot(instance,
4      disk_path,
5      image_type=source_format)
```

进一步看 snapshot() 函数：

```
1      def snapshot(self, instance, disk_path, image_type=None):
2          # self.backend()根据image_type返回相应的类
3          # 具体细节在此不详述，有兴趣的可以看Backend类
4          backend = self.backend(image_type)
5          return backend(instance=instance, path=disk_path)
```

3. 在分析第三句注释代码前，我先说一些准备信息。

我分析代码的时候，使用的磁盘类型是 qcow2 类型，所以现在 snapshot\_backend 是 Qcow2 类。而且我们现在已经通过 managedSave() 函数将它的快照保存到虚拟机镜像的一个文件中。所以使用 snapshot\_extract() 函数就很好理解了，就是提取快照文件：

```
1      snapshot_backend.snapshot_extract(out_path, image_format)
2
3      # snapshot_extract()函数如下
4      class Qcow2(Image):
5          ...
6          def snapshot_extract(self, target, out_format):
7              libvirt_utils.extract_snapshot(self.path, 'qcow2',
8              target,
9              out_format)
```

代码到了这一步，已经是即将完成快照的提取。进一步看 `extract_snapshot()` 函数：

```

1  def extract_snapshot(disk_path, source_fmt, out_path, dest_fmt):
2      # 函数相当于执行了 “qemu-img convert -f source_fmt -O dest_fmt
      #      disk_path out_path”
3      # qemu-img convert 就是用来提取快照的命令
4      qemu_img_cmd = ('qemu-img', 'convert', '-f', source_fmt, '-O',
      dest_fmt)
5      ...
6      qemu_img_cmd += (disk_path, out_path)
7      execute(*qemu_img_cmd)

```

总结一下，openstack 静态创建快照步骤如下：

1. 调用 `virDomain.managedSave()` 创建快照。
2. 用 “qemu-img convert” 命令提取快照。

## 5.5 动态创建快照

`snapshot()` 函数中动态创建快照的相关语句如下：

```

1  ...
2  with utils.tmpdir(dir=snapshot_directory) as tmpdir:
3      try:
4          ...
5          if live_snapshot:
6              # 动态创建快照
7              self._live_snapshot(virt_dom, disk_path, out_path,
8                                  image_format)
9          ...

```

进一步看 `_live_snapshot()` 函数：

```

1  def _live_snapshot(self, domain, disk_path, out_path, image_format):
2      # domain是虚拟机的 virDomain对象
3
4      # 传入 VIR_DOMAIN_XML_INACTIVE和 VIR_DOMAIN_XML_SECURE参数
5      # 获得 domain的 xml配置文件
6      # 这个 xml用于之后的 undefine()和 define()函数
7      xml = domain.XMLDesc(
8          libvirt.VIR_DOMAIN_XML_INACTIVE |
9          libvirt.VIR_DOMAIN_XML_SECURE)
10
11     try:
12         # 调用 virDomain.blockJobAbort()停止活动块操作
13         domain.blockJobAbort(disk_path, 0)
14     ...
15     # 获得虚拟机 backing_file的路径
16     src_back_path = libvirt_utils.get_disk_backing_file(disk_path,

```

```

17                                                                 basenane=False)
18     disk_delta = out_path + '.delta'
19     # 这个函数位于nova/virt/libvirt/Utils.py中
20     # 函数中调用 “qemu-img create -f qcow2 -o src_disk_size,src_back_path
21     #     src_back_path” 创建镜像文件
22     libvirt_utils.create_cow_image(src_back_path, disk_delta,
23                                   src_disk_size)
24
25     try:
26         # 因为virDomain.blockRebase()不能对persistent的domain操作
27         # 所以调用virDomain.undefine()将domain变为transient
28         if domain.isPersistent():
29             domain.undefine()
30
31         # 将虚拟机镜像文件的内容拷贝到disk_delta中的镜像文件中
32         domain.blockRebase(disk_path, disk_delta, 0,
33                             libvirt.VIR_DOMAIN_BLOCK_REBASE_COPY |
34                             libvirt.VIR_DOMAIN_BLOCK_REBASE_REUSE_EXT |
35                             libvirt.VIR_DOMAIN_BLOCK_REBASE_SHALLOW)
36
37         ...
38
39         # 调用virDomain.blockJobAbort()终止数据拷贝
40         domain.blockJobAbort(disk_path, 0)
41         ...
42     finally:
43         # 将domain从transient变为persistent
44         self._conn.defineXML(xml)
45
46     # 这个函数调用 “qemu-img convert” 命令将disk_delta中的文件变为一个qcow2文件
47     libvirt_utils.extract_snapshot(disk_delta, 'qcow2',
48                                   out_path, image_format)

```

总结一下，openstack 动态创建快照步骤如下：

1. 使用 “qemu-img create” 命令创建用于备份的镜像。
2. 调用 virDomain.blockRebase() 函数将虚拟机的磁盘内容拷贝到后端镜像中。
3. 使用 “qemu-img convert” 命令将后端镜像转换为一个 qcow2 文件。

## 6 查看 libvirt 的 python API

可以通过 python 的 help() 函数来查看 virConnect 类:

1. 首先进入 python 的 help 界面:

```
pengsida@scholes:~  
pengsida@scholes:~$ python  
Python 2.7.12 (default, Nov 19 2016, 06:48:10)  
[GCC 5.4.0 20160609] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> help()  
  
Welcome to Python 2.7! This is the online help utility.  
  
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.  
  
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".  
  
To get a list of available modules, keywords, or topics, type "modules",  
"keywords", or "topics". Each module also comes with a one-line summary  
of what it does; to list the modules whose summaries contain a given word  
such as "spam", type "modules spam".  
  
help> |
```

2. 查看 libvirt 的帮助文档:

```
pengsida@scholes:~$ python  
Python 2.7.12 (default, Nov 19 2016, 06:48:10)  
[GCC 5.4.0 20160609] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> help()  
  
Welcome to Python 2.7! This is the online help utility.  
  
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.  
  
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".  
  
To get a list of available modules, keywords, or topics, type "modules",  
"keywords", or "topics". Each module also comes with a one-line summary  
of what it does; to list the modules whose summaries contain a given word  
such as "spam", type "modules spam".  
  
help> libvirt
```

3. 通过 “virConnect” 找到对 virConnect 各个函数的介绍:

```
pengsida@scholes:~  
class virConnect(_builtin_.object)  
| Methods defined here:  
|  
| __del__(self)  
|     # virConnect methods from virConnect.py (hand coded)  
|  
| __init__(self, _obj=None)  
|  
| allocPages(self, pages, startCell=0, cellCount=0, flags=0)  
|     Allocate or free some pages in the huge pages pool  
|  
| baselineCPU(self, xmlCPUs, flags=0)  
|     Computes the most feature-rich CPU which is compatible with all give  
n host CPUs.  
|  
| c_pointer(self)  
|     Get C pointer to underlying object  
|  
| changeBegin(self, flags=0)  
|     This function creates a restore point to which one can return  
later by calling virInterfaceChangeRollback(). This function should  
be called before any transaction with interface configuration.  
Once it is known that a new configuration works, it can be committed  
/virConnect
```