目 录

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:

```
    def spawn(self, context, instance, image_meta, injected_files,
              admin_password, network_info=None, block_device_info=None):

        ...

        # 创建镜像
        # disk_info是一个字典，格式如下:
        # {'disk_bus': disk_bus,      the bus for harddisks
        #  'cdrom_bus': cdrom_bus,    the bus for CDROMs
        #  'mapping': mapping}        the disk mapping
        self._create_image(context, instance, disk_info['mapping'],
                           injection_info=injection_info,
                           block_device_info=block_device_info)

        ...

        # 创建xml配置文件
        xml = self._get_guest_xml(context, instance, network_info,
                                  disk_info, image_meta,
                                  block_device_info=block_device_info)

        ...

        # 创建虚拟机和网络
        self._create_domain_and_network(
            context, xml, instance, network_info, disk_info,
            block_device_info=block_device_info,
            post_xml_callback=gen_confdrive,
            destroy_disks_on_failure=True)
```

2.1 创建镜像

相关语句与函数如下:

```
    self._create_image(context, instance, disk_info['mapping'],
                       injection_info=injection_info,
                       block_device_info=block_device_info)

    # _create_image函数如下
    def _create_image(self, context, instance,
                      disk_mapping, suffix='',
                      disk_images=None, network_info=None,
                      block_device_info=None, files=None,
                      admin_pass=None, inject_files=True):
        booted_from_volume = self._is_booted_from_volume(
            instance, disk_mapping)

        def image(fname, image_type=CONF.libvirt.images_type):
            return self.image_backend.image(instance,
                                            fname + suffix, image_type)

        def raw(fname):
            return image(fname, image_type='raw')

        # 创建instance目录，用来存放镜像和libvirt.xml配置文件
        # 调用os.makedirs(path)创建目录
        fileutils.ensure_tree(libvirt_utils.get_instance_path(instance))

        ...

        # 如果disk_images为None，就创建一个disk_images字典
        if not disk_images:
            disk_images = {'image_id': instance['image_ref'],
                           'kernel_id': instance['kernel_id'],
                           'ramdisk_id': instance['ramdisk_id']}

        # 如果kernel、ramdisk存在就跳过
        # 如果不存在将调用nova.image.glance.GlanceImageService.download()函数下载
            image
        if disk_images['kernel_id']:
            fname = imagecache.get_cache_fname(disk_images, 'kernel_id')
            raw('kernel').cache(fetch_func=libvirt_utils.fetch_image,
                                context=context,
                                filename=fname,
                                image_id=disk_images['kernel_id'],
                                user_id=instance['user_id'],
                                project_id=instance['project_id'])
            if disk_images['ramdisk_id']:
                fname = imagecache.get_cache_fname(disk_images, 'ramdisk_id')
                raw('ramdisk').cache(fetch_func=libvirt_utils.fetch_image,
                                     context=context,
                                     filename=fname,
                                     image_id=disk_images['ramdisk_id'],
                                     user_id=instance['user_id'],
                                     project_id=instance['project_id'])
```

```
51
52          inst_type = flavors.extract_flavor(instance)
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
108             specified_fs = eph.get('guest_format')
109             if specified_fs and not self.is_supported_fs_format(specified_fs):
110                 msg = _("%s format is not supported") % specified_fs
111                 raise exception.InvalidBDMFormat(details=msg)
112
113             fn = functools.partial(self._create_ephemeral,
114                                    fs_label='ephemeral%d' % idx,
115                                    os_type=instance["os_type"],
116                                    is_block_dev=disk_image.is_block_dev)
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,
```

```
                    image_meta=None, rescue=None,
                    block_device_info=None, write_to_disk=False):
        ...
        # 根据instance的配置创建一个nova.virt.libvirt.LibvirtConfigObject类
        conf = self._get_guest_config(instance, network_info, image_meta,
                                      disk_info, rescue, block_device_info,
                                      context)
        # 获得虚拟机的xml文件
        xml = conf.to_xml()

        if write_to_disk:
            # 创建libvirt.xml文件
            instance_dir = libvirt_utils.get_instance_path(instance)
            xml_path = os.path.join(instance_dir, 'libvirt.xml')
            libvirt_utils.write_to_file(xml_path, xml)

        ...
        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：

```
    def _create_domain_and_network(self, context, xml, instance, network_info,
                                   disk_info, block_device_info=None,
                                   power_on=True, reboot=False,
                                   vifs_already_plugged=False,
                                   post_xml_callback=None,
                                   destroy_disks_on_failure=False):

        ...

        # 创建虚拟机
        guest = None
        try:
            with self.virtapi.wait_for_instance_event(
                    instance, events, deadline=timeout,
                    error_callback=self._neutron_failed_callback):
                ...
                with self._lxc_disk_handler(instance, instance.image_meta,
                                            block_device_info, disk_info):
                    guest = self._create_domain(
                        xml, pause=pause, power_on=power_on,
                        post_xml_callback=post_xml_callback)
                    ...
        ...

        return guest
```

创建虚拟机的函数代码如下：

```
    def _create_domain(self, xml=None, domain=None,
                       power_on=True, pause=False, post_xml_callback=None):

        # 创建虚拟机
        # 这里的libvirt_guest在driver.py中有引用: from nova.virt.libvirt import
            guest as libvirt_guest
        if xml:
            guest = libvirt_guest.Guest.create(xml, self._host)
        else:
            guest = libvirt_guest.Guest(domain)

        ...

        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：

```
class Guest(object):
    @classmethod
    def create(cls, xml, host):

        try:
            ...
            # 这里使用host创建虚拟机，host是nova/virt/libvirt/host.py中的Host类
            guest = host.write_instance_config(xml)

        ...

        return guest
```

这里的 Host 类的 write_instance_config() 的代码如下所示：

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

        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：

```
    def get_connection(self):

        try:
            # 很清楚地看到conn对象是由_get_connection()创建的
            conn = self._get_connection()

        ...
```

```
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
                详述如何追踪到它的值
4           # _read_only为False
5           # 使用_connect()函数创建wrapped_conn对象
6           # 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：
7           #   conn = virConnectPtr("qemu:///system")
8           #   conn是virConnectPtr对象，python中是virConnect类
9           wrapped_conn = self._connect(self._uri, self._read_only)
10
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 Libvrit 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《resize 虚拟机的流程》这篇文章。

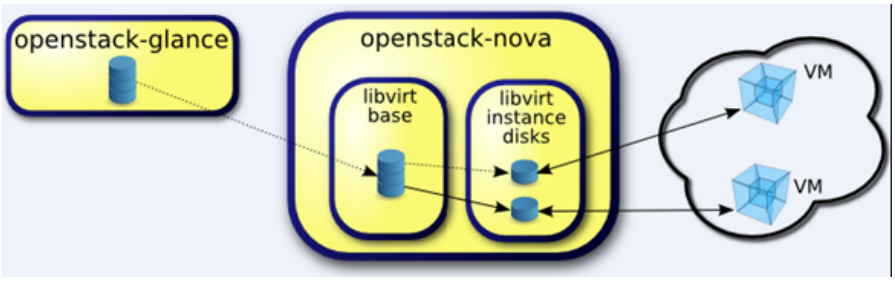
接下来，我只会分析 LibvirtDriver 类中实现以上功能的函数。

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-e017a2fcedb4_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

```
    # nova/virt/libvirt/driver.py LibvirtDriver.migrate_disk_and_power_off()
    def migrate_disk_and_power_off(self, context, instance, dest,
                                   flavor, network_info,
                                   block_device_info=None,
                                   timeout=0, retry_interval=0):
        # block_device_info是一个字典，如下所示:
        # {'swap': swap,
        # 'root_device_name': root_device_name,
        # 'ephemerals': ephemerals,
        # 'block_device_mapping': block_device_mapping}
        #
        # disk_info_text数据类型是一个list，里面有多个字典，字典格式如下
        # {'type': disk_type,
        # 'path': path,
        # 'virt_disk_size': virt_size,
        # 'backing_file': backing_file,
        # 'disk_size': dk_size,
        # 'over_committed_disk_size': over_commit_size}
        # 获取disk配置信息
        disk_info_text = self.get_instance_disk_info(instance['name'],
                block_device_info=block_device_info)

        # 调用json.loads()
        disk_info = jsonutils.loads(disk_info_text)

        # 调用os.path.join(CONF.instances_path, instance['name'])获得inst_base
        # inst_base存放着虚拟机的增量镜像
        # inst_base格式为/var/lib/nova/instances/vm-uuid
        inst_base = libvirt_utils.get_instance_path(instance)
        # 创建resize以后instance增量镜像的存放路径
        inst_base_resize = inst_base + "_resize"

        # 判断是否共用storage
        # 有可能目的主机和源主机本身就是同一台
        # 也有可能目的主机和源主机共享存储
        shared_storage = self._is_storage_shared_with(dest, inst_base)

        # 如果不是共享storage，就需要迁移增量镜像
        # 在目的主机下根据inst_base创建增量镜像的文件夹
        if not shared_storage:
            utils.execute('ssh', dest, 'mkdir', '-p', inst_base)

        # 关闭虚拟机，这个函数稍后分析
        self.power_off(instance, timeout, retry_interval)

        ...

        try:
            # 重命名inst_base为inst_base_resize
            # 也就是将源主机上虚拟机增量文件所在目录名加上“_resize”
            utils.execute('mv', inst_base, inst_base_resize)

            # 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或者是没有后端镜像的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：

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

这个函数如下所示：

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

继续追踪 _destroy() 函数：

```
def _destroy(self, instance):
    try:
        # 根据virConnect.lookupByName()获得virDomain对象
        virt_dom = self._lookup_by_name(instance['name'])
    ...


    old_domid = -1
    if virt_dom is not None:
        try:
            old_domid = virt_dom.ID()
            # 调用virtDomain.destroy()函数关闭虚拟机
            virt_dom.destroy()
        ...

    # 接下来的代码都是用于判断虚拟机是否真正关机
    def _wait_for_destroy(expected_domid):
        try:
            dom_info = self.get_info(instance)
            state = dom_info['state']
            new_domid = dom_info['id']
        except exception.InstanceNotFound:
            LOG.warning(_LW("During wait destroy, instance disappeared."),
                        instance=instance)
            raise loopingcall.LoopingCallDone()

        if state == power_state.SHUTDOWN:
            LOG.info(_LI("Instance destroyed successfully."),
                     instance=instance)
            raise loopingcall.LoopingCallDone()

        if new_domid != expected_domid:
            LOG.info(_LI("Instance may be started again."),
                     instance=instance)
            kwargs['is_running'] = True
            raise loopingcall.LoopingCallDone()

    kwargs = {'is_running': False}
    timer = loopingcall.FixedIntervalLoopingCall(_wait_for_destroy,
                                                 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：

```
    # info是一个字典，如下所示：
    # {'type': disk_type,
    # 'path': path, path的格式为/var/lib/nova/instances/vm_uuid/
    # 'virt_disk_size': virt_size,
    # 'backing_file': backing_file,
    # 'disk_size': dk_size,
    # 'over_committed_disk_size': over_commit_size}
    #
    # img_path格式为/var/lib/nova/instances/vm_uuid/fname
    # from_path格外为/var/lib/nova/instances/vm_uuid_resize/fname
    # tmp_path格式为/var/lib/nova/instances/vm_uuid_resize/fname_rbase

    if info['type'] == 'qcow2' and info['backing_file']:
        tmp_path = from_path + "_rbase"
        # 通过“qemu_img convert”命令将虚拟机的后端镜像和增量镜像合并
        utils.execute('qemu_img', 'convert', '-f', 'qcow2',
                      '-O', 'qcow2', from_path, tmp_path)

        if shared_storage:
            # 如果共享storage，此时只要重命名文件
            # 相当于将文件又移到了/var/lib/nova/instances/vm_uuid/fname
            utils.execute('mv', tmp_path, img_path)
        else:
            # 如果是不同主机，那么需要调用如下函数，稍后分析
            libvirt_utils.copy_image(tmp_path, img_path, host=dest,
                                     on_execute=on_execute,
                                     on_completion=on_completion)
            utils.execute('rm', '-f', tmp_path)

    # 如果类型是raw或者是没有后端镜像的qcow2，就可以直接转移
    # 如果共享storage，这里的host会是None
    else:
        libvirt_utils.copy_image(from_path, img_path, host=dest,
                                 on_execute=on_execute,
                                 on_completion=on_completion)
```

进一步查看 copy_image 函数：

```
    def copy_image(src, dest, host=None, on_execute=None,
                   on_completion=None):
        # 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
        if not host:
            execute('cp', src, dest)
        else:
            dest = "%s:%s" % (host, dest)

            # 通过rsync ssh方式拷贝合并后的镜像到新的host对于instance目录下
            try:
                execute('rsync', '--sparse', '--compress', '--dry-run', src, dest,
                        on_execute=on_execute, on_completion=on_completion)
            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:

```
    def finish_migration(self, context, migration, instance, disk_info,
                         network_info, image_meta, resize_instance,
                         block_device_info=None, power_on=True):

        # resize disks. only "disk" and "disk.local" are necessary.

        # 调用json.loads()
        disk_info = jsonutils.loads(disk_info)

        # disk_info是一个list，其中存放着一个个字典，格式如下:
        # {'type': disk_type,
        # 'path': path, path的格式为/var/lib/nova/instances/vm-uuid/
        # 'virt_disk_size': virt_size,
        # 'backing_file': backing_file,
        # 'disk_size': dk_size,
        # 'over_committed_disk_size': over_commit_size}
        for info in disk_info:
            # 如果是disk，就返回root分区的大小
            # 如果是disk.local，就返回ephemeral分区的大小
            size = self._disk_size_from_instance(instance, info)

            # 对虚拟机进行扩容
            if resize_instance:
                self._disk_resize(info, size)

            # 将磁盘类型转为qcow2
            # 调用命令“qemu-img convert -f raw -O qcow2”
            if info['type'] == 'raw' and CONF.use_cow_images:
                self._disk_raw_to_qcow2(info['path'])

        ...

        # 创建image
        # 这个函数在“nova创建虚拟机时对libvirt的调用”一节中有分析
        self._create_image(context, instance,
                           disk_mapping=disk_info['mapping'],
                           network_info=network_info,
                           block_device_info=None, inject_files=False)

        # 获得虚拟机xml配置文件
        # 这个函数在“nova创建虚拟机时对libvirt的调用”一节中有分析
        xml = self._get_guest_xml(context, instance, network_info, disk_info,
                                  block_device_info=block_device_info,
                                  write_to_disk=True)

        # 创建虚拟机和网络
        # 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 在线迁移扩容时对 libvirt 的调用

4.1 nova 在线迁移扩容的流程介绍

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 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《resize 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 pre_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
             is_block_migration赋值
15       if migrate_data:
16           is_shared_block_storage = migrate_data.get(
17                   'is_shared_block_storage', True)
18           is_shared_instance_path = migrate_data.get(
```

```
                    'is_shared_instance_path', True)
            is_block_migration = migrate_data.get('block_migration', True)
            instance_relative_path = migrate_data.get('instance_relative_path')

        ...

        if not is_shared_instance_path:
            # 为目的主机创建instance dir
            if instance_relative_path:
                instance_dir = os.path.join(CONF.instances_path,
                                            instance_relative_path)
            else:
                instance_dir = libvirt_utils.get_instance_path(instance)

            if os.path.exists(instance_dir):
                raise exception.DestinationDiskExists(path=instance_dir)
            os.mkdir(instance_dir)

            # 如果目的主机和源主机不共享存储
            if not is_shared_block_storage:
                # Ensure images and backing files are present.
                # 如果使用后端镜像，这个函数将创建增量镜像
                # 否则，直接调用“qemu-img create”创建镜像
                # 这个函数稍后分析
                self._create_images_and_backing(context, instance,
                                                instance_dir, disk_info)

        # 如果不是block migration而且不共享instance path
        # 还需要kernel和ramdisk，所以需要调用_fetch_instance_kernel_ramdisk()
        if not (is_block_migration or is_shared_instance_path):

            # Touch the console.log file, required by libvirt.
            console_file = self._get_console_log_path(instance)
            libvirt_utils.file_open(console_file, 'a').close()

            # 下载kernel和ramdisk文件
            self._fetch_instance_kernel_ramdisk(context, instance)

        ...
```

4.2.2 准备后端镜像和增量镜像

```
    def _create_images_and_backing(self, context, instance, instance_dir,
                                   disk_info_json):
        # disk_info数据类型是一个list，里面有多个字典，字典格式如下
        # {'type': disk_type,
        # 'path': path,
        # 'virt_disk_size': virt_size,
        # 'backing_file': backing_file,
        # 'disk_size': dk_size,
        # 'over_committed_disk_size': over_commit_size}
        if not disk_info_json:
            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
                    磁盘镜像
24              if not info['backing_file'] and not os.path.exists(instance_disk):
25                  # 调用“qemu-img create -f info['type'] instance_disk info['
                        virt_disk_size']”创建镜像
26                  libvirt_utils.create_image(info['type'], instance_disk,
27                                             info['virt_disk_size'])
28              elif info['backing_file']:
29                  # Creating backing file follows same way as spawning instances.
30                  cache_name = os.path.basename(info['backing_file'])
31
32                  image = self.image_backend.image(instance,
33                                                   instance_disk,
34                                                   CONF.libvirt.images_type)
35                  # 创建空的Ephemeral disk
36                  if cache_name.startswith('ephemeral'):
37                      image.cache(fetch_func=self._create_ephemeral,
38                                  fs_label=cache_name,
39                                  os_type=instance["os_type"],
40                                  filename=cache_name,
41                                  size=info['virt_disk_size'],
42                                  ephemeral_size=instance['ephemeral_gb'])
43                  # 创建空的Swap disk
44                  elif cache_name.startswith('swap'):
45                      inst_type = flavors.extract_flavor(instance)
46                      swap_mb = inst_type['swap']
47                      image.cache(fetch_func=self._create_swap,
48                                  filename="swap_%s" % swap_mb,
49                                  size=swap_mb * units.Mi,
50                                  swap_mb=swap_mb)
51                  else:
52                      # 从Glance中获取image来创建Root disk
53                      image.cache(fetch_func=libvirt_utils.fetch_image,
54                                  context=context,
55                                  filename=cache_name,
56                                  image_id=instance['image_ref'],
57                                  user_id=instance['user_id'],
58                                  project_id=instance['project_id'],
59                                  size=info['virt_disk_size'])
60
61          # if image has kernel and ramdisk, just download
62          # following normal way.
63          self._fetch_instance_kernel_ramdisk(context, instance)
```

4.3 源主机上的操作

4.3.1 live_migration 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.tempdir(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.tempdir(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：

```
    # virt_dom是libvirt的virDomain对象
    # 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
    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：

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

进一步看 snapshot() 函数：

```
    def snapshot(self, instance, disk_path, image_type=None):
        # self.backend()根据image_type返回相应的类
        # 具体细节在此不详述，有兴趣的可以看Backend类
        backend = self.backend(image_type)
        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。而且现在我们已经通过 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：

```
    snapshot_backend.snapshot_extract(out_path, image_format)

    # snapshot_extract()函数如下
class Qcow2(Image):
    ...
    def snapshot_extract(self, target, out_format):
        libvirt_utils.extract_snapshot(self.path, 'qcow2',
                                       target,
                                       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：

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

```
    ...
    with utils.tempdir(dir=snapshot_directory) as tmpdir:
            try:
                ...
                if live_snapshot:
                    # 动态创建快照
                    self._live_snapshot(virt_dom, disk_path, out_path,
                                        image_format)
                ...
```

进一步看 _live_snapshot() 函数：

```
    def _live_snapshot(self, domain, disk_path, out_path, image_format):
        # domain是虚拟机的virDomain对象

        # 传入VIR_DOMAIN_XML_INACTIVE和VIR_DOMAIN_XML_SECURE参数
        # 获得domain的xml配置文件
        # 这个xml用于之后的undefine()和define()函数
        xml = domain.XMLDesc(
            libvirt.VIR_DOMAIN_XML_INACTIVE |
            libvirt.VIR_DOMAIN_XML_SECURE)

        try:
            # 调用virDomain.blockJobAbort()停止活动块操作
            domain.blockJobAbort(disk_path, 0)
        ...
        # 获得虚拟机backing_file的路径
        src_back_path = libvirt_utils.get_disk_backing_file(disk_path,
```

```
                                                    basename=False)
        disk_delta = out_path + '.delta'
        # 这个函数位于nova/virt/libvirt/utils.py中
        # 函数中调用“qemu-img create -f qcow2 -o src_disk_size,src_back_path
            src_back_path”创建镜像文件
        libvirt_utils.create_cow_image(src_back_path, disk_delta,
                                       src_disk_size)

        try:
            # 因为virDomain.blockRebase()不能对persistent的domain操作
            # 所以调用virDomain.undefine()将domain变为transient
            if domain.isPersistent():
                domain.undefine()

            # 将虚拟机镜像文件的内容拷贝到disk_delta中的镜像文件中
            domain.blockRebase(disk_path, disk_delta, 0,
                               libvirt.VIR_DOMAIN_BLOCK_REBASE_COPY |
                               libvirt.VIR_DOMAIN_BLOCK_REBASE_REUSE_EXT |
                               libvirt.VIR_DOMAIN_BLOCK_REBASE_SHALLOW)

            ...

            # 调用virDomain.blockJobAbort()终止数据拷贝
            domain.blockJobAbort(disk_path, 0)
            ...
        finally:
            # 将domain从transient变为persistent
            self._conn.defineXML(xml)

        # 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
        libvirt_utils.extract_snapshot(disk_delta, 'qcow2',
                                       out_path, image_format)
```

总结一下，openstack **动态创建快照步骤如下：**

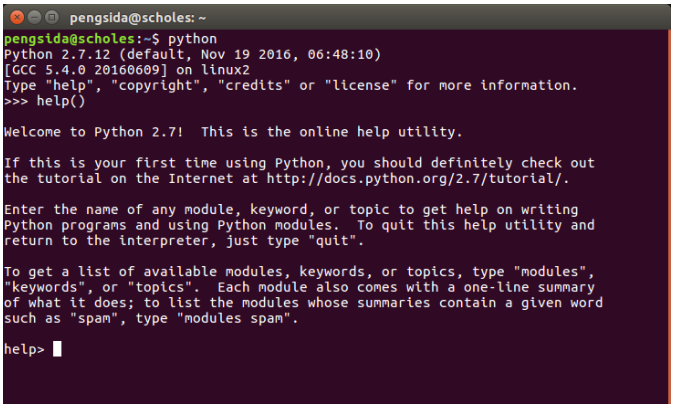
1. **使用“**qemu-img create**”命令创建用于备份的镜像。**
2. **调用** virDomain.blockRebase() **函数将虚拟机的磁盘内容拷贝到后端镜像中。**
3. **使用**”qemu-img convert” **命令将后端镜像转换为一个** qcow2 **文件。**

6 openstack 中对 libvirt 调用的框架

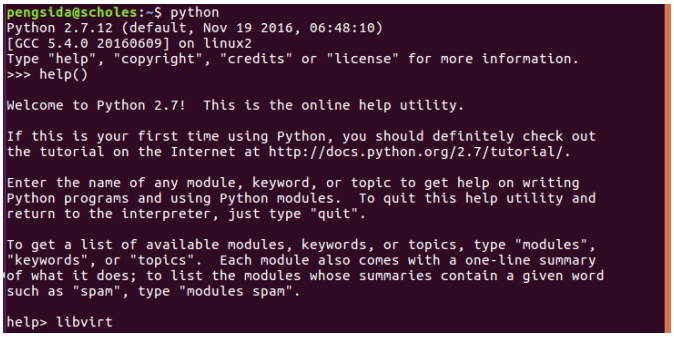
7 查看 libvirt 的 python API

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

1. 首先进入 python 的 help 界面：



2. 查看 libvirt 的帮助文档：



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