目 录

1 分析 resize 流程前的必要知识

1.1 nova 中的 RPC 机制

1.2 重要的数据类型

1.2.1 req

1.2.2 context

```
# 根据req创建环境上下文context
# context是nova/context.py中的RequestContext类
context = req.environ["nova.context"]
```

1.2.3 instance

```
# 根据req和instance_id创建instance
# instance是nova/context/instance.py中的Instance类
instance = self._get_server(context, req, instance_id)


instance_type
flavor_id
deltas
quotas
vm_state
```

2 nova-api 阶段

入口函数为:

```
# 这个函数在nova/api/openstack/compute/servers.py
def _resize(self, req, instance_id, flavor_id, **kwargs):
    ...
    try:
        # compute_api是nova/compute/api.py中的API类
        self.compute_api.resize(context, instance, flavor_id, **kwargs)
    ...
```

进一步看 API.resize() 函数:

```
# nova/compute/api.py API.resize()
def resize(self, context, instance, flavor_id=None,
           **extra_instance_updates):
```

```
        ...

        # filter_properties与选择本地扩容或选择异地扩容有关
        filter_properties = {'ignore_hosts': []}

        if not CONF.allow_resize_to_same_host:
            filter_properties['ignore_hosts'].append(instance['host'])

        if (not flavor_id and not CONF.allow_migrate_to_same_host):
            filter_properties['ignore_hosts'].append(instance['host'])

        ...

        # scheduler_hint挺重要的，是nova-scheduler的参数
        scheduler_hint = {'filter_properties': filter_properties}
        self.compute_task_api.resize_instance(context, instance,
                extra_instance_updates, scheduler_hint=scheduler_hint,
                flavor=new_instance_type,
                reservations=quotas.reservations or [])
```

```
    # nova/conductor/api.py ComputeTaskAPI.resize_instance()
    def resize_instance(self, context, instance, extra_instance_updates,
                        scheduler_hint, flavor, reservations):
        self.conductor_compute_rpcapi.migrate_server(
            context, instance, scheduler_hint, False, False, flavor,
            None, None, reservations)
```

```
    # nova/conductor/rpcapi.py ComputeTaskAPI.migrate_server()
    def migrate_server(self, context, instance, scheduler_hint, live, rebuild,
                flavor, block_migration, disk_over_commit,
                reservations=None):
        ...
        cctxt = self.client.prepare(version=version)
        return cctxt.call(context, 'migrate_server',
                          instance=instance, scheduler_hint=scheduler_hint,
                          live=live, rebuild=rebuild, flavor=flavor_p,
                          block_migration=block_migration,
                          disk_over_commit=disk_over_commit,
                          reservations=reservations)
```

3 nova-conductor 部分

```
    # nova/conductor/manager.py ComputeTaskManager.migrate_server()
    def migrate_server(self, context, instance, scheduler_hint, live, rebuild,
            flavor, block_migration, disk_over_commit, reservations=None):
        ...
        if live and not rebuild and not flavor:
            self._live_migrate(context, instance, scheduler_hint,
                               block_migration, disk_over_commit)
        elif not live and not rebuild and flavor:
```

```
            ...
            with compute_utils.EventReporter(context, 'cold_migrate',
                                             instance_uuid):
                self._cold_migrate(context, instance, flavor,
                                   scheduler_hint['filter_properties'],
                                   reservations)
        ...
```

4 冷迁移

4.1 冷迁移中的 nova-conductor 部分

```
    # nova/conductor/manager.py ComputeTaskManager._cold_migrate()
    def _cold_migrate(self, context, instance, flavor, filter_properties,
                      reservations):
        ...
        try:
            ...
            # 选择目的主机
            hosts = self.scheduler_client.select_destinations(
                    context, request_spec, filter_properties)
            host_state = hosts[0]
        ...

        try:
            ...
            (host, node) = (host_state['host'], host_state['nodename'])
            self.compute_rpcapi.prep_resize(
                context, image, instance,
                flavor, host,
                reservations, request_spec=request_spec,
                filter_properties=filter_properties, node=node)
        ...
```

```
    # nova/compute/rpcapi.py ComputeAPI.prep_resize()
    def prep_resize(self, ctxt, image, instance, instance_type, host,
                    reservations=None, request_spec=None,
                    filter_properties=None, node=None):
        ...
        cctxt = self.client.prepare(server=host, version=version)
        cctxt.cast(ctxt, 'prep_resize',
                   instance=instance,
                   instance_type=instance_type_p,
                   image=image_p, reservations=reservations,
                   request_spec=request_spec,
                   filter_properties=filter_properties,
                   node=node)
```

4.2 冷迁移中的 nova-compute 部分

4.2.1 目的主机上的操作：prep_resize

```
# nova/compute/manager.py ComputeManager.prep_resize()
def prep_resize(self, context, image, instance, instance_type,
                reservations, request_spec, filter_properties, node):
    ...
    with self._error_out_instance_on_exception(context, instance,
                                               quotas=quotas):
        ...
        try:
            self._prep_resize(context, image, instance,
                              instance_type, quotas,
                              request_spec, filter_properties,
                              node)
        ...
```

```
# nova/compute/manager.py ComputeManager._prep_resize()
def _prep_resize(self, context, image, instance, instance_type,
        quotas, request_spec, filter_properties, node):

    ...
    with rt.resize_claim(context, instance, instance_type,
                         image_meta=image, limits=limits) as claim:
        ...
        self.compute_rpcapi.resize_instance(
                context, instance, claim.migration, image,
                instance_type, quotas.reservations)
```

```
# nova/compute/rpcapi.py ComputeAPI.resize_instance()
def resize_instance(self, ctxt, instance, migration, image, instance_type,
                    reservations=None):
    ...
    cctxt = self.client.prepare(server=_compute_host(None, instance),
            version=version)
    cctxt.cast(ctxt, 'resize_instance',
               instance=instance, migration=migration,
               image=image, reservations=reservations,
               instance_type=instance_type_p)
```

4.2.2 源主机的操作：resize_instance

```
# nova/compute/manager.py ComputeManager.resize_instance()
def resize_instance(self, context, instance, image,
                    reservations, migration, instance_type,
                    clean_shutdown=True):
    ...
    with self._error_out_instance_on_exception(context, instance,
                                               quotas=quotas):
```

```
            ...
            # 获得虚拟机块设备的信息
            block_device_info = self._get_instance_block_device_info(
                                context, instance, bdms=bdms)

            # 关闭虚拟机并迁移虚拟机的增量文件
            disk_info = self.driver.migrate_disk_and_power_off(
                    context, instance, migration.dest_host,
                    instance_type, network_info,
                    block_device_info,
                    timeout, retry_interval)
            ...
            self.compute_rpcapi.finish_resize(context, instance,
                    migration, image, disk_info,
                    migration.dest_compute, reservations=quotas.reservations)
            ...
```

migrate_disk_and_power_off() 是源主机上将虚拟机迁移给目的主机的实现函数，主要利用了 libvirt API。这个函数的分析在《nova 调用 libvirt》中的“nova 扩容时对 libvirt 的调用”一节。

```
# nova/compute/rpcapi.py(690) ComputeAPI.finish_resize()
def finish_resize(self, ctxt, instance, migration, image, disk_info,
        host, reservations=None):
    ...
    cctxt = self.client.prepare(server=host, version=version)
    cctxt.cast(ctxt, 'finish_resize',
               instance=instance, migration=migration,
               image=image, disk_info=disk_info, reservations=reservations)
```

4.2.3 目的主机上的操作：finish_resize

```
# nova/compute/manager.py ComputeManager.finish_resize()
def finish_resize(self, context, disk_info, image, instance,
                  reservations, migration):
    quotas = quotas_obj.Quotas.from_reservations(context,
                                                 reservations,
                                                 instance=instance)
    try:
        self._finish_resize(context, instance, migration,
                            disk_info, image)
    ...
```

```
# nova/compute/manager.py ComputeManager._finish_resize()
def _finish_resize(self, context, instance, migration, disk_info,
                   image):
    ...
    try:
        self.driver.finish_migration(context, migration, instance,
                                     disk_info,
                                     network_info,
```

```
                                        image, resize_instance,
                                        block_device_info, power_on)
        ...
```

5 在线迁移

5.1 在线迁移中的 nova-conductor 部分

```
    # nova/conductor/manager.py ComputeTaskManager._live_migrate()
    def _live_migrate(self, context, instance, scheduler_hint,
                      block_migration, disk_over_commit):
        destination = scheduler_hint.get("host")
        try:
            live_migrate.execute(context, instance, destination,
                                 block_migration, disk_over_commit)
        ...
```

```
    # nova/conductor/tasks/live_migrate.py execute()
    def execute(context, instance, destination,
            block_migration, disk_over_commit):
    task = LiveMigrationTask(context, instance,
                             destination,
                             block_migration,
                             disk_over_commit)
    # TODO(johngarbutt) create a superclass that contains a safe_execute call
    return task.execute()
```

```
    # nova/conductor/tasks/live_migrate.py LiveMigrationTask.execute()
    def execute(self):
        self._check_instance_is_running()
        self._check_host_is_up(self.source)

        if not self.destination:
            self.destination = self._find_destination()
        else:
            self._check_requested_destination()

        # TODO(johngarbutt) need to move complexity out of compute manager
        # TODO(johngarbutt) disk_over_commit?
        return self.compute_rpcapi.live_migration(self.context,
                host=self.source,
                instance=self.instance,
                dest=self.destination,
                block_migration=self.block_migration,
                migrate_data=self.migrate_data)
```

```
    # nova/compute/rpcapi.py ComputeAPI.live_migration()
    def live_migration(self, ctxt, instance, dest, block_migration, host,
```

```
                    migrate_data=None):
        ...
        cctxt = self.client.prepare(server=host, version=version)
        cctxt.cast(ctxt, 'live_migration', instance=instance,
                   dest=dest, block_migration=block_migration,
                   migrate_data=migrate_data)
```

5.2 在线迁移中的 nova-compute 部分

```
# nova/compute/manager.py ComputeManager.live_migration()
def live_migration(self, context, dest, instance, block_migration,
                   migrate_data):
    """Executing live migration.

    :param context: security context
    :param instance: a nova.objects.instance.Instance object
    :param dest: destination host
    :param block_migration: if true, prepare for block migration
    :param migrate_data: implementation specific params

    """

    # NOTE(danms): since instance is not the first parameter, we can't
    # use @object_compat on this method. Since this is the only example,
    # we do this manually instead of complicating the decorator
    if not isinstance(instance, obj_base.NovaObject):
        expected = ['metadata', 'system_metadata',
                    'security_groups', 'info_cache']
        instance = objects.Instance._from_db_object(
            context, objects.Instance(), instance,
            expected_attrs=expected)

    # Create a local copy since we'll be modifying the dictionary
    migrate_data = dict(migrate_data or {})
    try:
        if block_migration:
            block_device_info = self._get_instance_block_device_info(
                context, instance)
            disk = self.driver.get_instance_disk_info(
                instance.name, block_device_info=block_device_info)
        else:
            disk = None

        pre_migration_data = self.compute_rpcapi.pre_live_migration(
            context, instance,
            block_migration, disk, dest, migrate_data)
        migrate_data['pre_live_migration_result'] = pre_migration_data

    except Exception:
        with excutils.save_and_reraise_exception():
            LOG.exception(_LE('Pre live migration failed at %s'),
                          dest, instance=instance)
            self._rollback_live_migration(context, instance, dest,
```

```
                                                block_migration, migrate_data)

        # Executing live migration
        # live_migration might raises exceptions, but
        # nothing must be recovered in this version.
        self.driver.live_migration(context, instance, dest,
                                   self._post_live_migration,
                                   self._rollback_live_migration,
                                   block_migration, migrate_data)
```

5.2.1 目的主机上的操作：pre_live_migration

```
    # nova/compute/manager.py ComputeManager.pre_live_migration()
    def pre_live_migration(self, context, instance, block_migration, disk,
                           migrate_data):
        """Preparations for live migration at dest host.

        :param context: security context
        :param instance: dict of instance data
        :param block_migration: if true, prepare for block migration
        :param migrate_data: if not None, it is a dict which holds data
                             required for live migration without shared
                             storage.

        """
        block_device_info = self._get_instance_block_device_info(
                            context, instance, refresh_conn_info=True)

        network_info = self._get_instance_nw_info(context, instance)
        self._notify_about_instance_usage(
                     context, instance, "live_migration.pre.start",
                     network_info=network_info)

        pre_live_migration_data = self.driver.pre_live_migration(context,
                                       instance,
                                       block_device_info,
                                       network_info,
                                       disk,
                                       migrate_data)

        # NOTE(tr3buchet): setup networks on destination host
        self.network_api.setup_networks_on_host(context, instance,
                                                self.host)

        # Creating filters to hypervisors and firewalls.
        # An example is that nova-instance-instance-xxx,
        # which is written to libvirt.xml(Check "virsh nwfilter-list")
        # This nwfilter is necessary on the destination host.
        # In addition, this method is creating filtering rule
        # onto destination host.
        self.driver.ensure_filtering_rules_for_instance(instance,
                                            network_info)

```

```
        self._notify_about_instance_usage(
                     context, instance, "live_migration.pre.end",
                     network_info=network_info)

        return pre_live_migration_data
```

```
    # nova/virt/libvirt/driver.py LibvirtDriver.pre_live_migration()
    def pre_live_migration(self, context, instance, block_device_info,
                           network_info, disk_info, migrate_data=None):
        """Preparation live migration."""
        # Steps for volume backed instance live migration w/o shared storage.
        is_shared_block_storage = True
        is_shared_instance_path = True
        is_block_migration = True
        instance_relative_path = None
        if migrate_data:
            is_shared_block_storage = migrate_data.get(
                    'is_shared_block_storage', True)
            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 and is_shared_block_storage):
            # NOTE(dims): Using config drive with iso format does not work
            # because of a bug in libvirt with read only devices. However
            # one can use vfat as config_drive_format which works fine.
            # Please see bug/1246201 for details on the libvirt bug.
            if CONF.config_drive_format != 'vfat':
                if configdrive.required_by(instance):
                    raise exception.NoLiveMigrationForConfigDriveInLibVirt()

        if not is_shared_instance_path:
            # NOTE(mikal): this doesn't use libvirt_utils.get_instance_path
            # because we are ensuring that the same instance directory name
            # is used as was at the source
            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)

            if not is_shared_block_storage:
                # Ensure images and backing files are present.
                self._create_images_and_backing(context, instance,
                                                instance_dir, disk_info)

        if not (is_block_migration or is_shared_instance_path):
            # NOTE(angdraug): when block storage is shared between source and
            # destination and instance path isn't (e.g. volume backed or rbd
            # backed instance), instance path on destination has to be prepared
```

```
50
51              # Touch the console.log file, required by libvirt.
52              console_file = self._get_console_log_path(instance)
53              libvirt_utils.file_open(console_file, 'a').close()
54
55              # if image has kernel and ramdisk, just download
56              # following normal way.
57              self._fetch_instance_kernel_ramdisk(context, instance)
58
59          # Establishing connection to volume server.
60          block_device_mapping = driver.block_device_info_get_mapping(
61              block_device_info)
62          for vol in block_device_mapping:
63              connection_info = vol['connection_info']
64              disk_info = blockinfo.get_info_from_bdm(
65                  CONF.libvirt.virt_type, vol)
66              self._connect_volume(connection_info, disk_info)
67
68          if is_block_migration and len(block_device_mapping):
69              # NOTE(stpierre): if this instance has mapped volumes,
70              # we can't do a block migration, since that will
71              # result in volumes being copied from themselves to
72              # themselves, which is a recipe for disaster.
73              LOG.error(
74                  _LE('Cannot block migrate instance %s with mapped volumes') %
75                  instance.uuid)
76              raise exception.MigrationError(
77                  _('Cannot block migrate instance %s with mapped volumes') %
78                  instance.uuid)
79
80          # We call plug_vifs before the compute manager calls
81          # ensure_filtering_rules_for_instance, to ensure bridge is set up
82          # Retry operation is necessary because continuously request comes,
83          # concurrent request occurs to iptables, then it complains.
84          max_retry = CONF.live_migration_retry_count
85          for cnt in range(max_retry):
86              try:
87                  self.plug_vifs(instance, network_info)
88                  break
89              except processutils.ProcessExecutionError:
90                  if cnt == max_retry - 1:
91                      raise
92                  else:
93                      LOG.warn(_LW('plug_vifs() failed %(cnt)d. Retry up to '
94                                   '%(max_retry)d.'),
95                               {'cnt': cnt,
96                                'max_retry': max_retry},
97                               instance=instance)
98                      greenthread.sleep(1)
99
100         res_data = {'graphics_listen_addrs': {}}
101         res_data['graphics_listen_addrs']['vnc'] = CONF.vncserver_listen
102         res_data['graphics_listen_addrs']['spice'] = CONF.spice.server_listen
103
104         return res_data
```

5.2.2 源主机上的操作：live_migration

```
    # nova/compute/manager.py ComputeManager.live_migration()
    def live_migration(self, context, dest, instance, block_migration,
                       migrate_data):
        """Executing live migration.

        :param context: security context
        :param instance: a nova.objects.instance.Instance object
        :param dest: destination host
        :param block_migration: if true, prepare for block migration
        :param migrate_data: implementation specific params

        """

        # NOTE(danms): since instance is not the first parameter, we can't
        # use @object_compat on this method. Since this is the only example,
        # we do this manually instead of complicating the decorator
        if not isinstance(instance, obj_base.NovaObject):
            expected = ['metadata', 'system_metadata',
                        'security_groups', 'info_cache']
            instance = objects.Instance._from_db_object(
                context, objects.Instance(), instance,
                expected_attrs=expected)

        # Create a local copy since we'll be modifying the dictionary
        migrate_data = dict(migrate_data or {})
        try:
            if block_migration:
                block_device_info = self._get_instance_block_device_info(
                    context, instance)
                disk = self.driver.get_instance_disk_info(
                    instance.name, block_device_info=block_device_info)
            else:
                disk = None

            pre_migration_data = self.compute_rpcapi.pre_live_migration(
                context, instance,
                block_migration, disk, dest, migrate_data)
            migrate_data['pre_live_migration_result'] = pre_migration_data

        except Exception:
            with excutils.save_and_reraise_exception():
                LOG.exception(_LE('Pre live migration failed at %s'),
                              dest, instance=instance)
                self._rollback_live_migration(context, instance, dest,
                                              block_migration, migrate_data)

        # Executing live migration
        # live_migration might raises exceptions, but
        # nothing must be recovered in this version.
        self.driver.live_migration(context, instance, dest,
                                   self._post_live_migration,
                                   self._rollback_live_migration,
                                   block_migration, migrate_data)
```

```
    # nova/virt/libvirt/driver.py LibvirtDriver.live_migration()
    def live_migration(self, context, instance, dest,
                       post_method, recover_method, block_migration=False,
                       migrate_data=None):
        """Spawning live_migration operation for distributing high-load.

        :param context: security context
        :param instance:
            nova.db.sqlalchemy.models.Instance object
            instance object that is migrated.
        :param dest: destination host
        :param post_method:
            post operation method.
            expected nova.compute.manager._post_live_migration.
        :param recover_method:
            recovery method when any exception occurs.
            expected nova.compute.manager._rollback_live_migration.
        :param block_migration: if true, do block migration.
        :param migrate_data: implementation specific params

        """

        # 'dest' will be substituted into 'migration_uri' so ensure
        # it does't contain any characters that could be used to
        # exploit the URI accepted by libivrt
        if not libvirt_utils.is_valid_hostname(dest):
            raise exception.InvalidHostname(hostname=dest)

        greenthread.spawn(self._live_migration, context, instance, dest,
                          post_method, recover_method, block_migration,
                          migrate_data)
```

```
    # nova/virt/libvirt/driver.py LibvirtDriver._live_migration()
    def _live_migration(self, context, instance, dest, post_method,
                        recover_method, block_migration=False,
                        migrate_data=None):
        """Do live migration.

        :param context: security context
        :param instance:
            nova.db.sqlalchemy.models.Instance object
            instance object that is migrated.
        :param dest: destination host
        :param post_method:
            post operation method.
            expected nova.compute.manager._post_live_migration.
        :param recover_method:
            recovery method when any exception occurs.
            expected nova.compute.manager._rollback_live_migration.
        :param block_migration: if true, do block migration.
        :param migrate_data: implementation specific params
        """

        # Do live migration.
        try:
```

```
24              if block_migration:
25                  flaglist = CONF.libvirt.block_migration_flag.split(',')
26              else:
27                  flaglist = CONF.libvirt.live_migration_flag.split(',')
28              flagvals = [getattr(libvirt, x.strip()) for x in flaglist]
29              logical_sum = reduce(lambda x, y: x | y, flagvals)
30
31              dom = self._lookup_by_name(instance["name"])
32
33              pre_live_migrate_data = (migrate_data or {}).get(
34                                          'pre_live_migration_result', {})
35              listen_addrs = pre_live_migrate_data.get('graphics_listen_addrs')
36
37              migratable_flag = getattr(libvirt, 'VIR_DOMAIN_XML_MIGRATABLE',
38                                        None)
39
40              if migratable_flag is None or listen_addrs is None:
41                  self._check_graphics_addresses_can_live_migrate(listen_addrs)
42                  dom.migrateToURI(CONF.libvirt.live_migration_uri % dest,
43                                   logical_sum,
44                                   None,
45                                   CONF.libvirt.live_migration_bandwidth)
46              else:
47                  old_xml_str = dom.XMLDesc(migratable_flag)
48                  new_xml_str = self._correct_listen_addr(old_xml_str,
49                                                          listen_addrs)
50                  try:
51                      dom.migrateToURI2(CONF.libvirt.live_migration_uri % dest,
52                                        None,
53                                        new_xml_str,
54                                        logical_sum,
55                                        None,
56                                        CONF.libvirt.live_migration_bandwidth)
57                  except libvirt.libvirtError as ex:
58                      # NOTE(mriedem): There is a bug in older versions of
59                      # libvirt where the VIR_DOMAIN_XML_MIGRATABLE flag causes
60                      # virDomainDefCheckABIStability to not compare the source
61                      # and target domain xml's correctly for the CPU model.
62                      # We try to handle that error here and attempt the legacy
63                      # migrateToURI path, which could fail if the console
64                      # addresses are not correct, but in that case we have the
65                      # _check_graphics_addresses_can_live_migrate check in place
66                      # to catch it.
67                      # TODO(mriedem): Remove this workaround when
68                      # Red Hat BZ #1141838 is closed.
69                      error_code = ex.get_error_code()
70                      if error_code == libvirt.VIR_ERR_CONFIG_UNSUPPORTED:
71                          LOG.warn(_LW('An error occurred trying to live '
72                                       'migrate. Falling back to legacy live '
73                                       'migrate flow. Error: %s'), ex,
74                                   instance=instance)
75                          self._check_graphics_addresses_can_live_migrate(
76                              listen_addrs)
77                          dom.migrateToURI(
78                              CONF.libvirt.live_migration_uri % dest,
79                              logical_sum,
```

```
80                              None,
81                              CONF.libvirt.live_migration_bandwidth)
82                      else:
83                          raise
84
85          except Exception as e:
86              with excutils.save_and_reraise_exception():
87                  LOG.error(_LE("Live Migration failure: %s"), e,
88                            instance=instance)
89                  recover_method(context, instance, dest, block_migration)
90
91          post_method(context, instance, dest, block_migration,
92                      migrate_data)
```