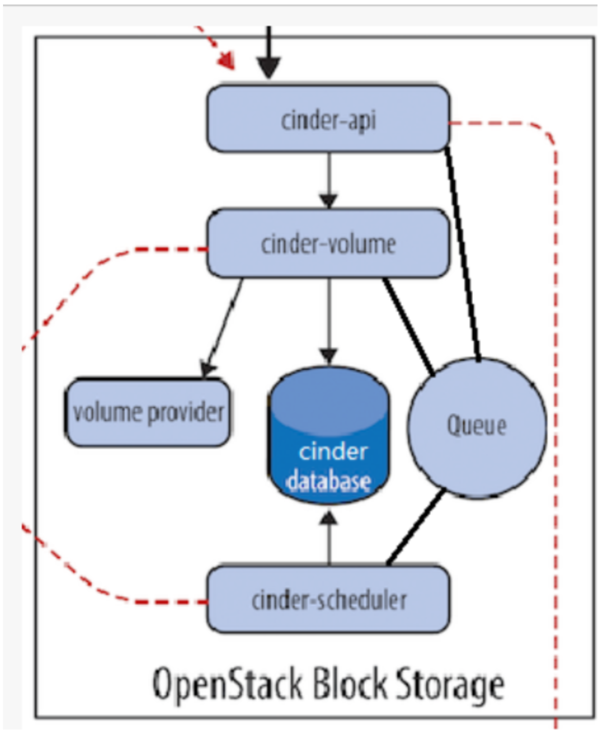
cinder源码分析

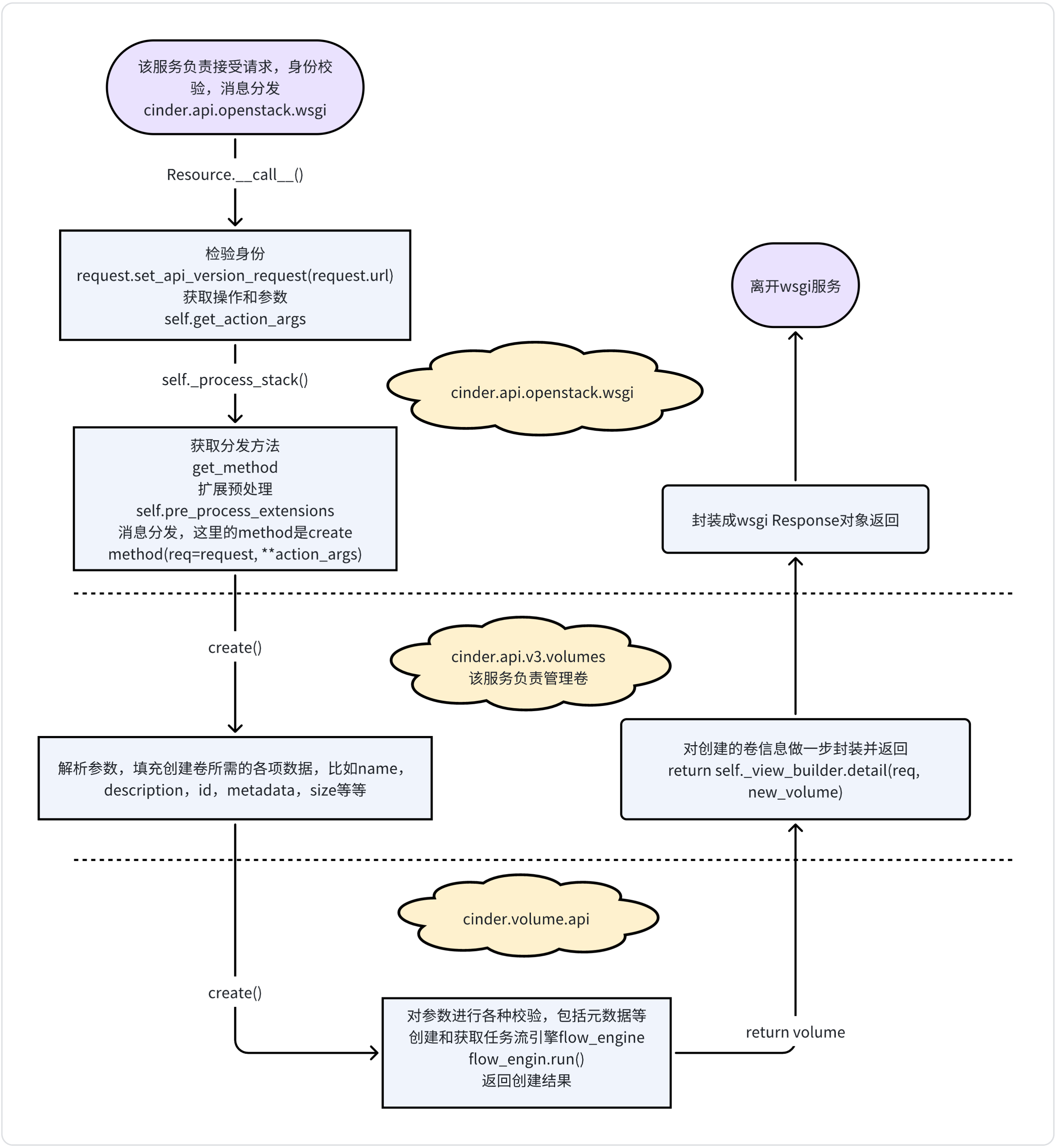


日志规范：时间戳 进程ID 日志级别 服务 [请求ID 项目ID或用户ID] 请求类型 请求URL

源码位置: openstack/cinder

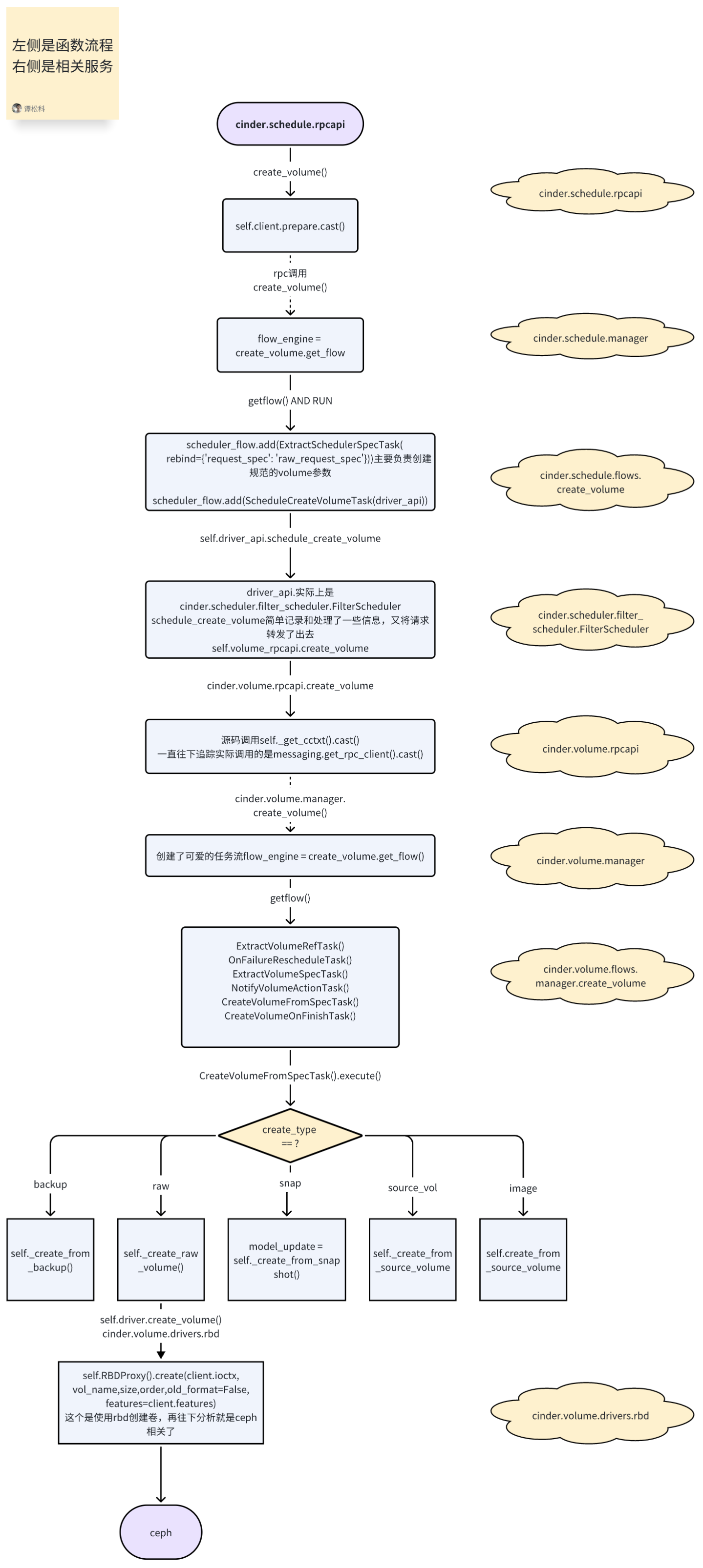
cinder create volume

流程分析



该服务负责接受请求，身份校验，消息分发
cinder.api.openstack.wsgi
Resource.__call__()
检验身份
request.set_api_version_request(request.url)
获取操作和参数
self.get_action_args
self._process_stack()
cinder.api.openstack.wsgi
获取分发方法
get_method
扩展预处理
self.pre_process_extensions
消息分发，这里的method是create
method(req=request, **action_args)
离开wsgi服务
封装成wsgi Response对象返回
create()
cinder.api.v3.volumes
该服务负责管理卷
解析参数，填充创建卷所需的各项数据，比如name，description，id，metadata，size等等
对创建的卷信息做一步封装并返回
return self._view_builder.detail(req, new_volume)
cinder.volume.api
create()
对参数进行各种校验，包括元数据等
创建和获取任务流引擎flow_engine
flow_engin.run()
返回创建结果
return volume

flow_engin中有一个VolumeCastTask，该类中有一个execute函数，它调用了_cast_create_volume函数，它最终调用了scheduler_rpcapi.create_volume


左侧是函数流程
右侧是相关服务
cinder.schedule.rpcapi
create_volume()
self.client.prepare.cast()
rpc调用
create_volume()
cinder.schedule.rpcapi
flow_engine = create_volume.get_flow
cinder.schedule.manager
getflow() AND RUN
scheduler_flow.add(ExtractSchedulerSpecTask(rebind={'request_spec': 'raw_request_spec'}))主要负责创建规范的volume参数
scheduler_flow.add(ScheduleCreateVolumeTask(driver_api))
cinder.schedule.flows. create_volume
self.driver_api.schedule_create_volume
driver_api.实际上是 cinder.scheduler.filter_scheduler.FilterScheduler schedule_create_volume简单记录和处理了一些信息，又将请求转发了出去 self.volume_rpcapi.create_volume
cinder.scheduler.filter_ scheduler.FilterScheduler
cinder.volume.rpcapi.create_volume
源码调用self._get_cctxt().cast() 一直往下追踪实际调用的是messaging.get_rpc_client().cast()
cinder.volume.rpcapi
cinder.volume.manager. create_volume()
创建了可爱的任务流flow_engine = create_volume.get_flow()
cinder.volume.manager
getflow()
ExtractVolumeRefTask()
OnFailureRescheduleTask()
ExtractVolumeSpecTask()
NotifyVolumeActionTask()
CreateVolumeFromSpecTask()
CreateVolumeOnFinishTask()
cinder.volume.flows. manager.create_volume
CreateVolumeFromSpecTask().execute()
create_type == ?
backup
raw
snap
source_vol
image
self._create_from _backup()
self._create_raw _volume()
model_update = self._create_from_snap shot()
self._create_from _source_volume
self.create_from _source_volume
self.driver.create_volume()
cinder.volume.drivers.rbd
self.RBDProxy().create(client.ioctx, vol_name,size,order,old_format=False, features=client.features)
这个是使用rbd创建卷，再往下分析就是ceph相关了
cinder.volume.drivers.rbd
ceph

源码分析

根据上面两张流程图展开源码分析，为了方便描述会对源码删改，有关error日志输出的代码，一般不会分析，只到debug级别。

cinder.api.openstack.wsgi.__call__()

```python
def __call__(self, request):
    """WSGI method that controls (de)serialization and method dispatch."""
    # 第1条api.log.info:
    # cinder.api.openstack.wsgi [None req-75599 c4394 cd799 - - default default]
    # POST http://192.168.163.130:8776/v3/cd799/volumes
    LOG.info("%(method)s %(url)s", {"method": request.method,"url": request.url})
    # 根据header设置API的版本
    request.set_api_version_request(request.url)

    # 识别操作、其参数和请求的内容类型，不用在意是如何get的
    action_args = self.get_action_args(request.environ)
    action = action_args.pop('action', None)
    # 尽早过滤无效的内容类型
    content_type, body = self.get_body(request)
    accept = request.best_match_content_type()

    # 处理请求
    return self._process_stack(request, action, action_args, content_type, body, accept)
```

cinder.api.openstack.wsgi._process_stack()

```python
def _process_stack(self, request, action, action_args, content_type, body, accept):
    """Implement the processing stack."""

    # Get the implementing method
    meth, extensions = self.get_method(request, action, content_type, body)

    if body:
        decoded_body = encodeutils.safe_decode(body, errors='ignore')
        msg = ("Action: '%(action)s', calling method: %(meth)s, body: " "%(body)s") % {'action': action, 'body': decoded_body, 'meth': meth.__name__}
        # api.log.debug: Action: 'create', calling method: create, body: {"volume": {"size": 10, "consistencygroup_id": null, "snapshot_id": null, "name": "testvolume", "description": "My first volume", "volume_type": null, "availability_zone": null, "metadata": {}, "imageRef": null, "source_volid": null, "backup_id": null}} _process_stack /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/api/openstack/wsgi.py:866
        LOG.debug(strutils.mask_password(msg))
    else:
        LOG.debug("Calling method '%(meth)s'", {'meth': meth.__name__})

    # Now, deserialize the request body...
    if content_type:
        contents = self.deserialize(meth, content_type, body)
    else:
        contents = {}

    # Update the action args
    action_args.update(contents)

    project_id = action_args.pop("project_id", None)
    context = request.environ.get('cinder.context')
    if (context and project_id and (project_id != context.project_id)):
        msg = _("Malformed request url")
        return Fault(webob.exc.HTTPBadRequest(explanation=msg))

    # Run pre-processing extensions
    # cinder若接入了扩展，对扩展列表进行预处理
    response, post = self.pre_process_extensions(extensions, request, action_args)

    if not response: # response==None表示上一个函数正常返回了
        try:
            with ResourceExceptionHandler():
                # 将操作发给cinder.api.v3.volumes.create
                action_result = self.dispatch(meth, request, action_args)
        except Fault as ex:
            response = ex

        # 省略一堆解码操作，解码后的结果是resp_obj
        # 处理扩展
        response = self.post_process_extensions(post, resp_obj,
                                                        request, action_args)

        if resp_obj and not response:
            response = resp_obj.serialize(request, accept,
                                          self.default_serializers)

    try:
        msg_dict = dict(url=request.url, status=response.status_int)
        msg = "%(url)s returned with HTTP %(status)s"
    except AttributeError as e:
        msg_dict = dict(url=request.url, e=e)
        msg = "%(url)s returned a fault: %(e)s"
    # 第5条api.log.info:
    # cinder.api.openstack.wsgi [None req-75599 c4394 cd799 - - default default]
    # http://192.168.163.130:8776/v3/cd799/volumes returned with HTTP 202
    LOG.info(msg, msg_dict)

    if hasattr(response, 'headers'):
        # 简单封装响应信息，略

    return response
```

> `pre_process_extensions` 和 `post_process_extensions` 这两个函数在OpenStack Cinder中的作用并不是直接参与卷的创建过程。它们的主要作用是在卷的创建和其他操作的前后提供一个机会来执行额外的逻辑。
>
> 在Cinder中，创建卷的过程通常涉及到一系列复杂的步骤，包括与后端存储交互、分配资源、设置卷属性等。这些步骤是由Cinder的核心逻辑来处理的。而 `pre_process_extensions` 和 `post_process_extensions` 则允许开发者在这个核心逻辑之外插入自定义的行为。
>
> 例如， `pre_process_extensions` 可以用于在实际处理请求之前进行验证、修改请求参数或者记录日志。这些预处理步骤不会改变卷的创建流程，但它们可以准备或调整请求，以确保核心逻辑能够正确执行。
>
> 同样地， `post_process_extensions` 可以用于在请求处理完成后执行清理工作、发送通知或者修改响应数据。这些后处理步骤不会影响卷的创建结果，但它们可以在操作完成后增加额外的步骤。
>
> 这两个函数提供的是一种钩子（hook）机制，允许开发者在不修改核心逻辑的情况下，为Cinder添加额外的功能。这种设计使得Cinder非常灵活，可以适应各种不同的需求和场景。

再往下一层，是action_result = self.dispatch(meth, request, action_args)这行的解析

关于meth是如何绑定到具体函数的，我并不清楚是通过什么机制，只能根据日志判断绑定到了下方这个函数

cinder.api.v3.volumes.create()

```
def create(self, req, body):
    """Creates a new volume.

    :param req: the request
    :param body: the request body
    :returns: dict -- the new volume dictionary
    :raises HTTPNotFound, HTTPBadRequest:
    """
    # api.log.debug: cinder.api.v3.volumes [None req-75599 c4394 cd799 - - default default]
    # Create volume request body: {'volume': {'size': 10, 'consistencygroup_id': None, 'snapshot_id': None, 'name': 'testvolume',
    'description': 'My first volume', 'volume_type': None, 'availability_zone':
    None, 'metadata': {}, 'imageRef': None, 'source_volid': None, 'backup_id':
    None}} create /var/lib/kolla/venv/lib64/python3.9/site-
    packages/cinder/api/v3/volumes.py:297
    LOG.debug('Create volume request body: %s', body)
    context = req.environ['cinder.context']
    req_version = req.api_version_request
    body = scheduler_hints.create(req, body)

    volume = body['volume']
    kwargs = {}
    self.validate_name_and_description(volume, check_length=False)

    # 注意：在REST API中是'name'/'description'，但在数据库层是
    # 'display_name'/'display_description'，所以我们需要做翻译
    # 这里省略一大堆的翻译过程 像这样：
    # kwargs['metadata'] = volume.get('metadata', None)

    size = volume.get('size', None)
    if size is None and kwargs['snapshot'] is not None:
        size = kwargs['snapshot']['volume_size']
    elif size is None and kwargs['source_volume'] is not None:
        size = kwargs['source_volume']['size']
    elif size is None and kwargs.get('backup') is not None:
        size = kwargs['backup']['size']
    # 第2条api.log.info:
    # cinder.api.v3.volumes [None req-75599 c4394 cd799 - - default default]
    # Create volume of 10 GB
    LOG.info("Create volume of %s GB", size)

    # 是否允许被多个虚拟机同时挂载，看描述似乎是已被禁用
    multiattach = utils.get_bool_param('multiattach', volume)
    if multiattach:
        msg = _("multiattach parameter has been removed. The default "
                "behavior is to use multiattach enabled volume types. "
                "Contact your administrator to create a multiattach "
                "enabled volume type and use it to create multiattach "
                "volumes.")
        raise exc.HTTPBadRequest(explanation=msg)
    try:
        # 调用下层的创建卷接口
        new_volume = self.volume_api.create(
            context, size, volume.get('display_name'),
            volume.get('display_description'), **kwargs)
    except exception.VolumeTypeDefaultMisconfiguredError as err:
        raise exc.HTTPInternalServerError(explanation=err.msg)

    retval = self._view_builder.detail(req, new_volume)
    return retval
```

cinder.volume.api.create()

```
def create(self,
           context: context.RequestContext,
           size: Union[str, int],
           name: Optional[str],
           description: Optional[str],
           snapshot: Optional[objects.Snapshot] = None,
           image_id: Optional[str] = None,
           volume_type: Optional[objects.VolumeType] = None,
           metadata: Optional[dict] = None,
           availability_zone: Optional[str] = None,
           source_volume: Optional[objects.Volume] = None,
           scheduler_hints=None,
           source_replica=None,
           consistencygroup: Optional[objects.ConsistencyGroup] = None,
           cgsnapshot: Optional[objects.CGSnapshot] = None,
           source_cg=None,
           group: Optional[objects.Group] = None,
           group_snapshot=None,
           source_group=None,
           backup: Optional[objects.Backup] = None):

    if image_id:
        context.authorize(vol_policy.CREATE_FROM_IMAGE_POLICY)
    else:
        context.authorize(vol_policy.CREATE_POLICY)

    # Check up front for legacy replication parameters to quick fail
    if source_replica:
        msg = _("Creating a volume from a replica source was part of the "
                "replication v1 implementation which is no longer "
                "available.")
        raise exception.InvalidInput(reason=msg)

    # 注意（jdg）：如果我们从快照或卷进行创建，我们可以进行没有大小的创建。
    # 目前，任务流 api 将处理此问题并从源中提取大小。

    # 注意（jdg）：cinderclient 发送大小值的字符串表示形式。 但有人可能会直接调用 API，
    # 这里检查size是否是合法数字
    if size and (not strutils.is_int_like(size) or int(size) <= 0):
        msg = _('Invalid volume size provided for create request: %s '
                '(size argument must be an integer (or string '
                'representation of an integer) and greater '
                'than zero).') % size
        raise exception.InvalidInput(reason=msg)

    # 略，volume_type的规则校验

    # Determine the valid availability zones that the volume could be
    # created in (a task in the flow will/can use this information to
    # ensure that the availability zone requested is valid).
    # 这一段我不懂，大致是为了确保有有效的可用区
    # 此函数包含了日志 第3条api.log.info:
    # cinder.volume.api [None req-75599 c4394 cd799 - - default default]
    # Availability Zones retrieved successfully.
    raw_zones = self.list_availability_zones(enable_cache=True)

    availability_zones = set([az['name'] for az in raw_zones])
    if CONF.storage_availability_zone:
        availability_zones.add(CONF.storage_availability_zone)
    # 检验元数据
    utils.check_metadata_properties(metadata)

    create_what = {
        'context': context,
        'raw_size': size,
        'name': name,
        'description': description,
        'snapshot': snapshot,
        'image_id': image_id,
        'raw_volume_type': volume_type,
        'metadata': metadata or {},
        'raw_availability_zone': availability_zone,
        'source_volume': source_volume,
        'scheduler_hints': scheduler_hints,
        'key_manager': self.key_manager,
        'optional_args': {'is_quota_committed': False},
        'consistencygroup': consistencygroup,
        'cgsnapshot': cgsnapshot,
        'group': group,
        'group_snapshot': group_snapshot,
        'source_group': source_group,
        'backup': backup,
    }
    try:
        # 三元表达式 if (xxx) sched_rpcapi = self.scheduler_rpcapi else
sched_rpcapi = None
        sched_rpcapi = (self.scheduler_rpcapi if (
                        not cgsnapshot and not source_cg and
                        not group_snapshot and not source_group)
                        else None)
        volume_rpcapi = (self.volume_rpcapi if (
                         not cgsnapshot and not source_cg and
                         not group_snapshot and not source_group)
                         else None)
        # 创建任务流引擎，将在下面分析
        flow_engine = create_volume.get_flow(self.db,
                                             self.image_service,
                                             availability_zones,
                                             create_what,
                                             sched_rpcapi,
                                             volume_rpcapi)

    # 绑定一个日志监听，重定向日志输出
    with flow_utils.DynamicLogListener(flow_engine, logger=LOG):
        try:
            # 这一步运行任务流引擎，将在下面分析
            flow_engine.run()
            # 将volume对象信息取出
            vref = flow_engine.storage.fetch('volume')
            # 刷新可用区缓存
            if flow_engine.storage.fetch('refresh_az'):
                self.list_availability_zones(enable_cache=True,
                                             refresh_cache=True)
            # 第4条api.log.info
            # cinder.volume.api [None req-75599 c4394 cd799 - - default default]
            # Create volume request issued successfully.

            LOG.info("Create volume request issued successfully.",
                     resource=vref)
            return vref
        except exception.InvalidAvailabilityZone:
            with excutils.save_and_reraise_exception():
                self.list_availability_zones(enable_cache=True,
                                             refresh_cache=True)
```

cinder.volume.flows.api.create_volume.get_flow()

```
def get_flow(db_api, image_service_api, availability_zones, create_what,
             scheduler_rpcapi=None, volume_rpcapi=None):
    """
        1.为依赖任务注入键和值。
        2.提取并验证输入的键和值。
        3.预留配额（在任何失败时恢复配额）。
        4.创建数据库条目。
        5.提交配额。
        6.转到卷管理器或调度器进行进一步处理。
    """

    flow_name = ACTION.replace(":", "_") + "_api"
    api_flow = linear_flow.Flow(flow_name)

    # 处理API请求并验证输入集，将它们转换为其他Task能识别的值集
    api_flow.add(ExtractVolumeRequestTask(
        image_service_api,
        availability_zones,
        rebind={'size': 'raw_size',
                'availability_zone': 'raw_availability_zone',
                'volume_type': 'raw_volume_type'}))
    # 检查并预留卷创建的配额
    # 在数据库中创建volume条目
    # 将配额预留提交到数据库
    api_flow.add(QuotaReserveTask(),
                 EntryCreateTask(),
                 QuotaCommitTask())
    # 通过提供的RPC API将卷创建任务发送到调度器或卷管理器
    if scheduler_rpcapi and volume_rpcapi:
        api_flow.add(VolumeCastTask(scheduler_rpcapi, volume_rpcapi, db_api))

    # Now load (but do not run) the flow using the provided initial data.
    return taskflow.engines.load(api_flow, store=create_what)
```

查看其中VolumeCastTask类的源码，找到execute函数，任务会被发给_cast_create_volume函数（同一个类）此函数做了些参数的翻译，便直接把任务发送给了scheduler_rpcapi.create_volume()

cinder.scheduler.rpcapi.create_volume()

```
def create_volume(self, ctxt, volume, snapshot_id=None, image_id=None,
                  request_spec=None, filter_properties=None, backup_id=None):
    volume.create_worker()

    # cctxt = self.client.prepare(version=version, **kwargs)
    cctxt = self._get_cctxt()
    msg_args = {'snapshot_id': snapshot_id, 'image_id': image_id,
                'request_spec': request_spec,
                'filter_properties': filter_properties,
                'volume': volume, 'backup_id': backup_id}
    if not self.client.can_send_version('3.10'):
        msg_args.pop('backup_id')
    cctxt.cast(ctxt, 'create_volume', **msg_args)
```

这里发送了一个RPC调用，被调用的是cinder.scheduler.create_volume()方法，关于_get_cctxt()是如何获取到被调方的，这涉及到复杂的初始化流程，本人暂时看不明白。

cinder.scheduler.manager.create_volume()

```
def create_volume(self, context, volume, snapshot_id=None, image_id=None,
                      request_spec=None, filter_properties=None,
                      backup_id=None):
        self._wait_for_scheduler()

        try:
            flow_engine = create_volume.get_flow(context,
                                                     self.driver,
                                                     request_spec,
                                                     filter_properties,
                                                     volume,
                                                     snapshot_id,
                                                     image_id,
                                                     backup_id)
        except Exception:
            msg = _("Failed to create scheduler manager volume flow")
            LOG.exception(msg)
            raise exception.CinderException(msg)

        with flow_utils.DynamicLogListener(flow_engine, logger=LOG):
            flow_engine.run()
```

cinder.scheduler.flows.create_volume.get_flow()

```
def get_flow(context: context.RequestContext,
             driver_api,
             request_spec: Optional[dict] = None,
             filter_properties: Optional[dict] = None,
             volume: Optional[objects.Volume] = None,
             snapshot_id: Optional[str] = None,
             image_id: Optional[str] = None,
             backup_id: Optional[str] = None) -> taskflow.engines.base.Engine:

    """Constructs and returns the scheduler entrypoint flow.

    This flow will do the following:
     1. 为相关任务注入键和值。
     2. 从提供的输入中提取调度程序规范。
     3. 使用提供的调度程序驱动程序选择主机并传递卷创建进一步请求。
    """
    create_what = {
        'context': context,
        'raw_request_spec': request_spec,
        'filter_properties': filter_properties,
        'volume': volume,
        'snapshot_id': snapshot_id,
        'image_id': image_id,
        'backup_id': backup_id,
    }

    flow_name = ACTION.replace(":", "_") + "_scheduler"
    scheduler_flow = linear_flow.Flow(flow_name)

    # This will extract and clean the spec from the starting values.
    # 创建规范的创建卷的参数
    scheduler_flow.add(ExtractSchedulerSpecTask(
        rebind={'request_spec': 'raw_request_spec'}))

    # This will activate the desired scheduler driver (and handle any
    # driver related failures appropriately).
    scheduler_flow.add(ScheduleCreateVolumeTask(driver_api))

    # Now load (but do not run) the flow using the provided initial data.
    return taskflow.engines.load(scheduler_flow, store=create_what)
```

上述两个Task中，前者就是返回一组规范的参数（考虑会有不完整/不规范的请求），后者就调用了一个函数self.driver_api.schedule_create_volume(context, request_spec, filter_properties) 其余的就是错误处理，和主体流程没有关系。

cinder.scheduler.filter_scheduler.FilterScheduler.schedule_create_volume()

```
def schedule_create_volume(self,
                           context: context.RequestContext,
                           request_spec: dict,
                           filter_properties: dict) -> None:
    # 通过选择合适的后端来调度创建存储卷
    backend = self._schedule(context, request_spec, filter_properties)
    # 如果没有找到合适的后端，则抛出异常。
    if not backend:
        raise exception.NoValidBackend(reason=_("No weighed backends "
                                                "available"))
    backend = backend.obj
    volume_id = request_spec['volume_id']

    updated_volume = driver.volume_update_db(
        context, volume_id,
        backend.host,
        backend.cluster_name,
        availability_zone=backend.service['availability_zone'])
    # 填充过滤信息
    self._post_select_populate_filter_properties(filter_properties, backend)
    # context is not serializable
    filter_properties.pop('context', None)
    # 调用卷，实际的创建开始
    self.volume_rpcapi.create_volume(context, updated_volume, request_spec,
                                     filter_properties,
                                     allow_reschedule=True)
```

cinder.colume.rpcapi.create_volume()

```
    def create_volume(self,
                      ctxt: context.RequestContext,
                      volume: 'objects.Volume',
                      request_spec: Optional[dict],
                      filter_properties: Optional[dict],
                      allow_reschedule: bool = True) -> None:
        cctxt = self._get_cctxt(volume.service_topic_queue)
        cctxt.cast(ctxt, 'create_volume',
                   request_spec=request_spec,
                   filter_properties=filter_properties,
                   allow_reschedule=allow_reschedule,
                   volume=volume)
```

和之前cinder.scheduler.rpcapi.create_volume()的套路一样，也是rpc调用，将请求转出去，这次被调方是cinder.volume.manager的create_volume()

cinder.volume.manager.create_volume()

```
def create_volume(self, context, volume, request_spec=None,
                  filter_properties=None,
                  allow_reschedule=True) -> ovo_fields.UUIDField:
    """Creates the volume."""
```

```python
volume_utils.log_unsupported_driver_warning(self.driver)
# 确保数据库中的主机在集群时与我们自己的主机匹配
self._set_resource_host(volume)
#尽早更新我们分配的容量计数器，以最大限度地减少调度程序的竞争条件。
self._update_allocated_capacity(volume)
original_host = volume.host

context_elevated = context.elevated()
if filter_properties is None:
    filter_properties = {}

if request_spec is None:
    request_spec = objects.RequestSpec()

flow_engine = create_volume.get_flow(
    context_elevated,
    self,
    self.db,
    self.driver,
    self.scheduler_rpcapi,
    self.host,
    volume,
    allow_reschedule,
    context,
    request_spec,
    filter_properties,
    image_volume_cache=self.image_volume_cache,
)
snapshot_id = request_spec.get('snapshot_id')
source_volid = request_spec.get('source_volid')
#  如果这个卷来自快照或其它卷，需要保证源不被删除
locked_action: Optional[str]
if snapshot_id is not None:
    # Make sure the snapshot is not deleted until we are done with it.
    locked_action = "%s-%s" % (snapshot_id, 'delete_snapshot')
elif source_volid is not None:
    # Make sure the volume is not deleted until we are done with it.
    locked_action = "%s-%s" % (source_volid, 'delete_volume')
else:
    locked_action = None

def _run_flow() -> None:
    with flow_utils.DynamicLogListener(flow_engine, logger=LOG):
        flow_engine.run()

if locked_action is None:
    _run_flow()
else:
    # 对源进行上锁
    with coordination.COORDINATOR.get_lock(locked_action):
        _run_flow()
# Shared targets is only relevant for some connections.
volume.shared_targets = self._driver_shares_targets()
# TODO(geguileo): service_uuid won''t be enough on Active/Active
# deployments. There can be 2 services handling volumes from the same
# backend.
volume.service_uuid = self.service_uuid
volume.save()
# 第3条volume.log.info
LOG.info("Created volume successfully.", resource=volume)
return volume.id
```

cinder.volume.flows.manager.create_volume.get_flow()

```
def get_flow(context, manager, db, driver, scheduler_rpcapi, host, volume,
             allow_reschedule, reschedule_context, request_spec,
             filter_properties, image_volume_cache=None):

    """Constructs and returns the manager entrypoint flow.
     1.确定是否启用了重新调度（提前）。
     2.为依赖任务注入键和值。
     3.选择2个仅在失败时激活的任务中的1个（一个用于更新数据库状态并通知，另一个用于更新数据库状态并通知并重新调度）。
     4.从提供的输入中提取卷规格。
     5.通知已开始创建卷。
     6.根据提取的卷规格创建卷。
     7.附加一个仅在成功时的任务，该任务通知卷创建已结束并执行进一步的数据库状态更新。
    """

    flow_name = ACTION.replace(":", "_") + "_manager"
    volume_flow = linear_flow.Flow(flow_name)

    create_what = {
        'context': context,
        'filter_properties': filter_properties,
        'request_spec': request_spec,
        'volume': volume,
    }
    # 获取并添加任务
    volume_flow.add(ExtractVolumeRefTask(db, host, set_error=False))

    retry = filter_properties.get('retry', None)

    # 处理恢复时重新安排任务的请求
    do_reschedule = allow_reschedule and request_spec and retry
    volume_flow.add(OnFailureRescheduleTask(reschedule_context, db, manager,
                                            scheduler_rpcapi, do_reschedule))

    LOG.debug("Volume reschedule parameters: %(allow)s "
              "retry: %(retry)s", {'allow': allow_reschedule, 'retry': retry})
    # 提取卷规范，将参数信息等提取出来
    volume_flow.add(ExtractVolumeSpecTask(db))
    # Temporary volumes created during migration should not be notified
    end_notify_suffix = None
    # TODO: (Y release) replace check with: if volume.use_quota:
    if volume.use_quota or not volume.is_migration_target():
        # 通知创建卷已经开始
        volume_flow.add(NotifyVolumeActionTask(db, 'create.start'))
        end_notify_suffix = 'create.end'
    # 创建卷，完成时通知并更新数据库
    # 其中分别包含了第1和第2条volume.log.info
    # Volume 64962: being created as raw with specification: xxx
    # Volume volume-64962 (64962): created successfully
    volume_flow.add(CreateVolumeFromSpecTask(manager,
                                             db,
                                             driver,
                                             image_volume_cache),
                    CreateVolumeOnFinishTask(db, end_notify_suffix))

    # Now load (but do not run) the flow using the provided initial data.
    return taskflow.engines.load(volume_flow, store=create_what)
```

最终在CreateVolumeFromSpecTask._create_raw_volume()中调用driver.create_volume()创建卷，这个driver就是ceph那的接口。

通过上述的源码分析和流程分析可以发现scheduler和volume层的函数调用关系很相似，都是api->manager->flows->下一层，然后folws都有参数的校验/翻译。

工作流分析(flows)

上面分析是针对主要流程的，关于真正干活的flows并没有做深入的了解，现在开始分析三个任务流的具体细节。

1.cinder.volume.flows.api.create_volume.get_flow()

1. 为依赖任务注入键和值。
2. 提取并验证输入的键和值。
3. 预留配额（在任何失败时恢复配额）。

4. 创建数据库条目。
5. 提交配额。
6. 转到卷管理器或调度器进行进一步处理。

```
# 工作流启动的时候会有两条日志提示，分别在启动时和完成时
Flow 'volume_create_api' (6b109) transitioned into state 'RUNNING' from state 'PENDING' _flow_receiver
Flow 'volume_create_api' (6b109) transitioned into state 'SUCCESS' from state 'RUNNING' _flow_receiver
```

ExtractVolumeRequestTask

将 api 请求值处理为一组经过验证的值，将输入转换为有效的集合，返回供其他任务使用

```
def execute(xxx) -> dict[str, Any]:{
    snapshot_id = self._extract_snapshot(snapshot)
    source_volid = self._extract_source_volume(source_volume)

    # 调用服务cinder.volume.flows.api.create_volume
    backup_id = self._extract_backup(backup)

    size = self._extract_size(size, source_volume, snapshot, backup)
    consistencygroup_id = self._extract_consistencygroup(consistencygroup)
    cgsnapshot_id = self._extract_cgsnapshot(cgsnapshot)
    group_id = self._extract_group(group)
    image_meta = self._get_image_metadata(context, image_id, size)
    # 仅展示部分转换
    return {
        'size': size,
        'snapshot_id': snapshot_id,
        'source_volid': source_volid,
        'volume_type': volume_type,
        'volume_type_id': volume_type_id,
        'encryption_key_id': encryption_key_id,
        'qos_specs': specs,
        'consistencygroup_id': consistencygroup_id,
        'cgsnapshot_id': cgsnapshot_id,
        'group_id': group_id,
        'replication_status': replication_status,
        'refresh_az': refresh_az,
        'backup_id': backup_id,
        'multiattach': multiattach,
        'availability_zones': availability_zones
    }
```

```

# 相关日志
cinder.volume.api Flow 'volume_create_api' (6b109) transitioned into state
'RUNNING' from state 'PENDING'
cinder.volume.api Task
'cinder.volume.flows.api.create_volume.ExtractVolumeRequestTask;volume:create'
(5f1fb) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
cinder.volume.flows.api.create_volume Validating volume size '10' using
validate_int _extract_size
cinder.volume.api Task
'cinder.volume.flows.api.create_volume.ExtractVolumeRequestTask;volume:create'
(5f1fb) transitioned into state 'SUCCESS' from state 'RUNNING' with result
'{'size': 10, 'snapshot_id': None, 'source_volid': None, 'volume_type':
VolumeType(created_at=2023-12-
02T14:39:31Z,deleted=False,deleted_at=None,description='Default Volume
Type',extra_specs={},id=456df,is_public=True,name='__DEFAULT__',projects=
[],qos_specs=<?>,qos_specs_id=None,updated_at=2023-12-02T14:39:31Z),
'volume_type_id': '456df', 'encryption_key_id': None, 'qos_specs': None,
'consistencygroup_id': None, 'cgsnapshot_id': None, 'group_id': None,
'replication_status': 'disabled', 'refresh_az': False, 'backup_id': None,
'multiattach': False, 'availability_zones': ['nova']}'

```

QuotaReserveTask

检查并预留卷创建的配额

```
def execute(xxx) -> Optional[dict]:
    values = {'per_volume_gigabytes': size}
    # 检查配额限制
    QUOTAS.limit_check(context, project_id=context.project_id, **values)
    # 预留一个卷或多预留一些字节
    if group_snapshot:
        reserve_opts = {'volumes': 1}
    else:
        reserve_opts = {'volumes': 1, 'gigabytes': size}
    if ('update_size' in optional_args and optional_args['update_size']):
        reserve_opts.pop('volumes', None)
    # 添加卷类型
    QUOTAS.add_volume_type_opts(context, reserve_opts, volume_type_id)
    # 预留资源，启动服务 cinder.quota
    reservations = QUOTAS.reserve(context, **reserve_opts)
    return {
        'reservations': reservations,
    }
    return None
```

```

# 相关日志
cinder.volume.api Task
'cinder.volume.flows.api.create_volume.QuotaReserveTask;volume:create' (074f2)
transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
cinder.quota      Created reservations ['73a2c', '5813f', 'af85f', 'c9e21']
reserve
cinder.volume.api Task
'cinder.volume.flows.api.create_volume.QuotaReserveTask;volume:create' (074f2)
transitioned into state 'SUCCESS' from state 'RUNNING' with result
'{'reservations': ['73a2c', '5813f', 'af085f', 'c9e21']}' _task_receiver
```

EntryCreateTask

在数据库中创建volume条目

```
def execute(xxx) -> dict[str, Any]:
    # 第一步，获取卷id和卷对象
    src_volid = kwargs.get('source_volid')
    src_vol = None
    if src_volid is not None:
        # 从数据库中获取对象
        src_vol = objects.Volume.get_by_id(context, src_volid)
    bootable = False
    if src_vol is not None:
        bootable = src_vol.bootable
    elif kwargs.get('snapshot_id'):
        # 从快照获取对象
        snapshot = objects.Snapshot.get_by_id(context,
kwargs.get('snapshot_id'))
        volume_id = snapshot.volume_id
        snp_vol = objects.Volume.get_by_id(context, volume_id)
        if snp_vol is not None:
            bootable = snp_vol.bootable
    # 提取可用区
    availability_zones = kwargs.pop('availability_zones')
    # 创建卷属性
    volume_properties = {
        'size': kwargs.pop('size'),
        'user_id': context.user_id,
        'project_id': context.project_id,
        'status': 'creating',
        'attach_status': fields.VolumeAttachStatus.DETACHED,
        'encryption_key_id': kwargs.pop('encryption_key_id'),
        # Rename these to the internal name.
        'display_description': kwargs.pop('description'),
        'display_name': kwargs.pop('name'),
        'multiattach': kwargs.pop('multiattach'),
        'bootable': bootable,
    }
    if len(availability_zones) == 1:
        volume_properties['availability_zone'] = availability_zones[0]

    # 合并其他必需参数，这些参数应提供其余的卷属性字段（如果适用
    volume_properties.update(kwargs)
    volume = objects.Volume(context=context, **volume_properties)
    # 创建卷对象，写入数据库
    volume.create()
    # 将字典转换成对象
    volume_properties = objects.VolumeProperties(**volume_properties)

    return {
        'volume_id': volume['id'],
        'volume_properties': volume_properties,
        'volume': volume,
    }

# 相关日志，服务名称：cinder.volume.api
Task 'cinder.volume.flows.api.create_volume.EntryCreateTask;volume:create'
(2ad8e) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task 'cinder.volume.flows.api.create_volume.EntryCreateTask;volume:create'
(2ad8e) transitioned into state 'SUCCESS' from state 'RUNNING' with result
'......' _task_receiver
```

QuotaCommitTask

将配额预留提交到数据库

```
def execute(self, context: context.RequestContext,
            reservations, volume_properties,
            optional_args: dict) -> dict:
    # 提交，底层调用QuotaEngine._driver_class.commit()
    QUOTAS.commit(context, reservations)
    optional_args['is_quota_committed'] = True
    return {'volume_properties': volume_properties}

# 相关日志，服务名称：cinder.volume.api
Task 'cinder.volume.flows.api.create_volume.QuotaCommitTask;volume:create'
(81df8) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task 'cinder.volume.flows.api.create_volume.QuotaCommitTask;volume:create'
(81df8) transitioned into state 'SUCCESS' from state 'RUNNING' with result
'......' _task_receiver
```

VolumeCastTask

通过提供的RPC API将卷创建任务发送到调度器或卷管理器

```
def execute(self, context: context.RequestContext, **kwargs) -> None:
    scheduler_hints = kwargs.pop('scheduler_hints', None)
    db_vt = kwargs.pop('volume_type')
    kwargs['volume_type'] = None
    if db_vt:
        kwargs['volume_type'] = objects.VolumeType()
        objects.VolumeType()._from_db_object(context,
                                             kwargs['volume_type'], db_vt)
    request_spec = objects.RequestSpec(**kwargs)
    filter_properties = {}
    if scheduler_hints:
        filter_properties['scheduler_hints'] = scheduler_hints
    self._cast_create_volume(context, request_spec, filter_properties)

def _cast_create_volume(xxx) -> None:
    # 提取各种参数，从request_spec中，这个字典存储创建卷的需要的所有信息
    source_volid = request_spec['source_volid']
    volume = request_spec['volume']
    snapshot_id = request_spec['snapshot_id']
    image_id = request_spec['image_id']
    cgroup_id = request_spec['consistencygroup_id']
    group_id = request_spec['group_id']
    backup_id = request_spec['backup_id']
    if cgroup_id:
        cgroup = objects.ConsistencyGroup.get_by_id(context, cgroup_id)
        request_spec['resource_backend'] =
volume_utils.extract_host(cgroup.resource_backend)
    elif group_id:
        group = objects.Group.get_by_id(context, group_id)
        request_spec['resource_backend'] =
volume_utils.extract_host(group.resource_backend)
    elif snapshot_id and CONF.snapshot_same_host:
        snapshot = objects.Snapshot.get_by_id(context, snapshot_id)
        request_spec['resource_backend'] = snapshot.volume.resource_backend
    elif source_volid:
        source_volume_ref = objects.Volume.get_by_id(context, source_volid)
        request_spec['resource_backend'] = (source_volume_ref.resource_backend)
    # 将消息发送给scheduler
    self.scheduler_rpcapi.create_volume(
        context,
        volume,
        snapshot_id=snapshot_id,
        image_id=image_id,
        request_spec=request_spec,
        filter_properties=filter_properties,
        backup_id=backup_id)

# 相关日志，服务名称：cinder.volume.api
Task 'cinder.volume.flows.api.create_volume.VolumeCastTask;volume:create' (add9d) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task 'cinder.volume.flows.api.create_volume.VolumeCastTask;volume:create' (add9d) transitioned into state 'SUCCESS' from state 'RUNNING' with result 'None' _task_receiver
```

2.cinder.scheduler.flows.create_volume.get_flow()

1. 为相关任务注入键和值。
2. 从提供的输入中提取调度程序规范。
3. 使用提供的调度程序驱动程序选择主机并传递卷创建进一步请求。

```
# 开始和完成时的日志，服务名称：cinder.scheduler.manager
Flow 'volume_create_scheduler' (efa1f) transitioned into state 'RUNNING' from state 'PENDING' _flow_receiver
Flow 'volume_create_scheduler' (efa1f) transitioned into state 'SUCCESS' from state 'RUNNING' _flow_receiver
```

ExtractSchedulerSpecTask

从不规范的请求中提取规范参数对象

```
def execute(self,
            context: context.RequestContext,
            request_spec: Optional[dict],
            volume: objects.Volume,
            snapshot_id: Optional[str],
            image_id: Optional[str],
            backup_id: Optional[str]) -> dict[str, Any]:
    # For RPC version < 1.2 backward compatibility
    if request_spec is None:
        request_spec = self._populate_request_spec(volume, snapshot_id, image_id, backup_id)
    return {
        'request_spec': request_spec,
    }
```

```
 def _populate_request_spec(xxx) -> dict[str, Any]:
    volume_type_id = volume.volume_type_id
    vol_type = volume.volume_type
    return {
        'volume_id': volume.id,
        'snapshot_id': snapshot_id,
        'image_id': image_id,
        'backup_id': backup_id,
        'volume_properties': {
            'size': utils.as_int(volume.size, quiet=False),
            'availability_zone': volume.availability_zone,
            'volume_type_id': volume_type_id,
        },
        'volume_type': list(dict(vol_type).items()),
    }
# 相关日志，服务名称：cinder.scheduler.manager
Task
'cinder.scheduler.flows.create_volume.ExtractSchedulerSpecTask;volume:create'
(ad0c5) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task
'cinder.scheduler.flows.create_volume.ExtractSchedulerSpecTask;volume:create'
(ad0c5) transitioned into state 'SUCCESS' from state 'RUNNING' with result '<一
组规范的参数>' _task_receiver
```

ScheduleCreateVolumeTask

激活调度程序驱动程序并处理任何后续故障

```
def execute(self,
            context: context.RequestContext,
            request_spec: dict,
            filter_properties: dict,
            volume: objects.Volume) -> None:
    try:
        # 调度
        self.driver_api.schedule_create_volume(context, request_spec,
filter_properties)
    except Exception as e:
        # 创建异常信息
        self.message_api.create(
            context,
            message_field.Action.SCHEDULE_ALLOCATE_VOLUME,
            resource_uuid=request_spec['volume_id'],
            exception=e)
        # 其余错误处理略
 # 相关日志，服务名称：cinder.scheduler.manager
```

```
Task
'cinder.scheduler.flows.create_volume.ScheduleCreateVolumeTask;volume:create'
(efd7a) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task
'cinder.scheduler.flows.create_volume.ScheduleCreateVolumeTask;volume:create'
(efd7a) transitioned into state 'SUCCESS' from state 'RUNNING' with result
'None' _task_receiver
```

然而事情并没有这么简单，在上面的两个日志中间，还有一堆日志信息：

```
oslo_db.sqlalchemy.engines MySQL server mode set to
STRICT_TRANS_TABLES,STRICT_ALL_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DI
VISION_BY_ZERO,TRADITIONAL,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
_check_effective_sql_mode /var/lib/kolla/venv/lib64/python3.9/site-
packages/oslo_db/sqlalchemy/engines.py:342
cinder.scheduler.host_manager Updating capabilities for xhs@rbd-1#rbd-1:
{'vendor_name': 'Open Source', 'driver_version': '1.3.0', 'storage_protocol':
'ceph', 'total_capacity_gb': 3.47, 'free_capacity_gb': 3.47,
'reserved_percentage': 0, 'multiattach': True, 'thin_provisioning_support':
True, 'max_over_subscription_ratio': '20.0', 'location_info':
'ceph:/etc/ceph/ceph.conf:2344f:cinder:volumes', 'backend_state': 'up',
'qos_support': True, 'volume_backend_name': 'rbd-1', 'replication_enabled':
False, 'allocated_capacity_gb': 10, 'filter_function': None,
'goodness_function': None, 'timestamp': datetime.datetime(2023, 12, 4, 2, 16,
19, 472651)} update_from_volume_capability
/var/lib/kolla/venv/lib64/python3.9/site-
packages/cinder/scheduler/host_manager.py:415
cinder.scheduler.base_filter Starting with 1 host(s) get_filtered_objects
/var/lib/kolla/venv/lib64/python3.9/site-
packages/cinder/scheduler/base_filter.py:97
cinder.scheduler.base_filter Filter AvailabilityZoneFilter returned 1 host(s)
get_filtered_objects /var/lib/kolla/venv/lib64/python3.9/site-
packages/cinder/scheduler/base_filter.py:125
cinder.scheduler.filters.capa Checking if host xhs@rbd-1#rbd-1 can create a 10
GB volume (64962) backend_passes /var/lib/kolla/venv/lib64/python3.9/site-
packages/cinder/scheduler/filters/capacity_filter.py:57
cinder.scheduler.filters.capa Storage Capacity factors {'total_capacity':
3.47, 'free_capacity': 3.47, 'reserved_capacity': 0,
'total_reserved_available_capacity': 3.47, 'max_over_subscription_ratio':
20.0, 'total_available_capacity': 69.4, 'provisioned_capacity': 10,
'calculated_free_capacity': 59.4, 'virtual_free_capacity': 59.4,
'free_percent': 85.59, 'provisioned_ratio': 0.14, 'provisioned_type': 'thin'}
backend_passes /var/lib/kolla/venv/lib64/python3.9/site-
packages/cinder/scheduler/filters/capacity_filter.py:137
```

```
cinder.scheduler.filters.capa Checking provisioning for request of 10 GB. Backend: host 'xhs@rbd-1#rbd-1': free_capacity_gb: 3.47, total_capacity_gb: 3.47, allocated_capacity_gb: 10, max_over_subscription_ratio: 20.0, reserved_percentage: 0, provisioned_capacity_gb: 10, thin_provisioning_support: True, thick_provisioning_support: False, pools: {}, updated at: 2023-12-04 02:16:19.472651 backend_passes /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/filters/capacity_filter.py:155
cinder.scheduler.filters.capa Space information for volume creation on host xhs@rbd-1#rbd-1 (requested / avail): 10/59.4 backend_passes /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/filters/capacity_filter.py:186
cinder.scheduler.base_filter Filter CapacityFilter returned 1 host(s) get_filtered_objects /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/base_filter.py:125
cinder.scheduler.base_filter Filter CapabilitiesFilter returned 1 host(s) get_filtered_objects /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/base_filter.py:125
cinder.scheduler.filter_sched Filtered [host 'xhs@rbd-1#rbd-1': free_capacity_gb: 3.47, total_capacity_gb: 3.47, allocated_capacity_gb: 10, max_over_subscription_ratio: 20.0, reserved_percentage: 0, provisioned_capacity_gb: 10, thin_provisioning_support: True, thick_provisioning_support: False, pools: {}, updated at: 2023-12-04 02:16:19.472651] _get_weighted_candidates /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/filter_scheduler.py:358
cinder.scheduler.base_weight Weigher CapacityWeigher returned, weigher value is {max: 59.4, min: 59.4} get_weighed_objects /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/base_weight.py:156
cinder.scheduler.host_manager Weighed [WeighedHost [host: xhs@rbd-1#rbd-1, weight: 0.0]] get_weighed_backends /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/host_manager.py:565
cinder.scheduler.filter_sched Choosing xhs@rbd-1#rbd-1 _choose_top_backend /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/filter_scheduler.py:594
cinder.scheduler.host_manager Consumed 10 GB from backend: host 'xhs@rbd-1#rbd-1': free_capacity_gb: -6.529999999999999, total_capacity_gb: 3.47, allocated_capacity_gb: 20, max_over_subscription_ratio: 20.0, reserved_percentage: 0, provisioned_capacity_gb: 20, thin_provisioning_support: True, thick_provisioning_support: False, pools: {}, updated at: 2023-12-04 02:17:08.153610 consume_from_volume /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/scheduler/host_manager.py:354
```

下方是gpt的解读供参考：

```
MySQL server mode set to                            数据库设置的日志
Updating capabilities for xhs@rbd-1#rbd-1           更新存储后端的存储能力，比如容量预留百分比等
Starting with 1 host(s)                             开始调度过程，表明有1个存储后端参与调度
Filter AvailabilityZoneFilter returned 1 host(s) 有1个存储后端通过了AvailabilityZoneFilter过滤器
Checking if host xhs@rbd-1#rbd-1 can create a 10 GB volume    检查后端是否能创建10GB卷
Storage Capacity factors                            存储容量，总容量、可用容量、预留容量等
Filter CapacityFilter returned 1 host(s)            有1个存储后端通过了CapacityFilter过滤器
Filtered [host ‘xhs@rbd-1#rbd-1’]                   存储后端（xhs@rbd-1#rbd-1）通过了所有的过滤器
Weigher CapacityWeigher returned, weigher value is {max: 59.4, min: 59.4}  表明存储后端（xhs@rbd-1#rbd-1）的权重值是59.4
Choosing xhs@rbd-1#rbd-1 存储后端（xhs@rbd-1#rbd-1） 被选中
Consumed 10 GB from backend: host ‘xhs@rbd-1#rbd-1’  从存储后端（xhs@rbd-1#rbd-1）上消耗了10GB的容量
```

关键的有三个部分：

1.AvailabilityZoneFilter 2.CapacityFilter 3.CapacityWeigher

日后在一大堆日志中找关键字的话，就是'Filte'和'Weigher'

离开日志，回到源码这里：

ScheduleCreateVolumeTask的execute函数就调用了一个api，显然这一堆日志都是api干的好事，这个api实际上是 `filter_scheduler.FilterScheduler`

在 `cinder.scheduler.filter_scheduler.FilterScheduler.schedule_create_volume()` 的源码分析中有一个函数调用：

```
backend = self._schedule(context, request_spec, filter_properties)
```

就是这个函数产生进行的过滤，从这个函数开始继续往下查看：

```
def _schedule(self,
              context: context.RequestContext,
              request_spec: dict,
                  filter_properties: Optional[dict] = None):
    # 经过过滤和权重计算后的候选backend，下方展开
    weighed_backends = self._get_weighted_candidates(context, request_spec,
filter_properties)
    # 清除一些和资源后端不匹配的候选后端
    resource_backend = request_spec.get('resource_backend')
    if weighed_backends and resource_backend:
        resource_backend_has_pool =
bool(volume_utils.extract_host(resource_backend, 'pool'))
        for backend in weighed_backends[::-1]:
            backend_id = (
                backend.obj.backend_id if resource_backend_has_pool
                else volume_utils.extract_host(backend.obj.backend_id)
            )
            if backend_id != resource_backend:
                weighed_backends.remove(backend)
    if not weighed_backends:
        assert filter_properties is not None
        LOG.warning('No weighed backend found for volume '
                    'with properties: %s',
                    filter_properties['request_spec'].get('volume_type'))
        return None
    # 从剩余backend中选出最优的
    return self._choose_top_backend(weighed_backends, request_spec)

# 关于什么是资源后端，什么是存储后端
## 在Cinder中，"backend"通常指的是存储后端，也就是运行cinder-volume服务的节点
### 每个存储后端都有一组特定的配置，例如存储类型，存储容量
## "资源后端"是指请求规范中指定的后端。
### 如果一个卷是从一个快照创建的，那么这个卷的资源后端就是这个快照所在的后端
### 如果一个卷是从一个已经存在的卷创建的，那么这个卷的资源后端就是这个已经存在的卷所在的后端。
```

_get_weighted_candidates

```
def _get_weighted_candidates(xxx) -> list:

    # 略过一堆不知所云的filter_properties的获取工作

    # 获取后端
    backends = self.host_manager.get_all_backend_states(elevated)

    # 过滤后端
    backends = self.host_manager.get_filtered_backends(backends,
filter_properties)
```

```
    if not backends:
        return []

    LOG.debug("Filtered %s", backends)
    # 为后端添加权重
    weighed_backends = self.host_manager.get_weighed_backends(
        backends, filter_properties)
    return weighed_backends
```

get_filtered_backends

```
def get_filtered_backends(self, backends, filter_properties,
                          filter_class_names=None):
    # 先获取过滤器
    if filter_class_names is not None:
        filter_classes = self._choose_backend_filters(filter_class_names)
    else:
        filter_classes = self.enabled_filters
    # 然后返回过滤结果
    return self.filter_handler.get_filtered_objects(filter_classes, backends,
filter_properties)

def _choose_backend_filters(self, filter_cls_names) -> list:
    if not isinstance(filter_cls_names, (list, tuple)):
        filter_cls_names = [filter_cls_names]
    good_filters = []
    bad_filters = []
    for filter_name in filter_cls_names:
        found_class = False
        for cls in self.filter_classes:
            # 匹配的过滤器
            if cls.__name__ == filter_name:
                found_class = True
                good_filters.append(cls)
                break
        # 无效的过滤器
        if not found_class:
            bad_filters.append(filter_name)
    if bad_filters:
        raise exception.SchedulerHostFilterNotFound(
            filter_name=", ".join(bad_filters))
    return good_filters

def get_filtered_objects(self, filter_classes, objs: Iterable,
                         filter_properties: dict, index: int = 0) -> list:
```

```

    list_objs = list(objs)
    # 上方filter的日志中有它
    LOG.debug("Starting with %d host(s)", len(list_objs))

    part_filter_results = []
    full_filter_results = []
    for filter_cls in filter_classes:
        cls_name = filter_cls.__name__
        start_count = len(list_objs)
        filter_class = filter_cls()

        if filter_class.run_filter_for_index(index):
            objs = filter_class.filter_all(list_objs, filter_properties)
            if objs is None:
                LOG.info("Filter %s returned 0 hosts", cls_name)
                full_filter_results.append((cls_name, None))
                list_objs = None
                break

            list_objs = list(objs)
            end_count = len(list_objs)
            part_filter_results.append((cls_name, start_count, end_count))
            remaining = [getattr(obj, "host", obj)
                         for obj in list_objs]
            full_filter_results.append((cls_name, remaining))
            # 上方filter的日志中有它
            LOG.debug("Filter %(cls_name)s returned ""%(obj_len)d host(s)",
                      {'cls_name': cls_name, 'obj_len': len(list_objs)})
    if not list_objs:
        self._log_filtration(full_filter_results,
                             part_filter_results, filter_properties)
    return list_objs
```

get_weighed_backends

```
def get_weighed_backends(self, backends, weight_properties,
                         weigher_class_names=None) -> list:
    """Weigh the backends."""
    # 获取计算器
    weigher_classes = self._choose_backend_weighers(weigher_class_names)
    # 获取带权后端
    weighed_backends = self.weight_handler.get_weighed_objects(
        weigher_classes, backends, weight_properties)

```

```python
    LOG.debug("Weighed %s", weighed_backends)
    return weighed_backends

def _choose_backend_weighers(self,
        weight_cls_names: Optional[list[str]]) -> list:

    if weight_cls_names is None:
        weight_cls_names = CONF.scheduler_default_weighers
    if not isinstance(weight_cls_names, (list, tuple)):
        weight_cls_names = [weight_cls_names]

    good_weighers = []
    bad_weighers = []
    for weigher_name in weight_cls_names:
        found_class = False
        for cls in self.weight_classes:
            if cls.__name__ == weigher_name:
                # 匹配的权重计算器类，将其添加到有效的权重计算器类列表中
                good_weighers.append(cls)
                found_class = True
                break
        if not found_class:
            # 这里还有一个无效的权重计算器类列表
            bad_weighers.append(weigher_name)
    if bad_weighers:
        raise exception.SchedulerHostWeigherNotFound(
            weigher_name=", ".join(bad_weighers))
    return good_weighers


def get_weighed_objects(self, weigher_classes, obj_list,
                        weighing_properties):

    weighed_objs = [wts.WeighedHost(obj, 0.0) for obj in obj_list]
    for weigher_cls in weigher_classes:
        weigher = weigher_cls()
        weights = weigher.weigh_objects(weighed_objs, weighing_properties)
        for i, weight in enumerate(weights):
            obj = weighed_objs[i]
            obj.weight += weigher.weight_multiplier() * weight

    # Avoid processing empty lists
    if not weighed_objs:
        return []

    total_weight = 0.0
    table = []
    for weighed_obj in weighed_objs:
        total_weight += weighed_obj.weight
        max_value = total_weight
        table.append((max_value, weighed_obj))

    # Now draw a random value with the computed range
    winning_value = random.random() * total_weight

    winning_index = 0
    for (i, (max_value, weighed_obj)) in enumerate(table):
        if max_value > winning_value:
            # Return a single element array with the winner.
            winning_index = i
            break
    # 这里将随机的winning_index作为列表中的第一个元素返回，上面这个winning_*是随机算法的结果
    return weighed_objs[winning_index:] + weighed_objs[0:winning_index]
```

3.cinder.volume.flows.manager.create_volume.get_flow()

1. 确定是否启用了重新调度（提前）
2. 为依赖任务注入键和值
3. 选择2个仅在*失败*时激活的任务中的1个（一个用于更新数据库状态并通知，另一个用于更新数据库状态并通知并*重新调度*）
4. 从提供的输入中提取卷规格
5. 通知已开始创建卷
6. 根据提取的卷规格创建卷
7. 附加一个仅在成功时的任务，该任务通知卷创建已结束并执行进一步的数据库状态更新

```
# 开始和完成时的日志。服务名称：cinder.volume.manager
Flow 'volume_create_manager' (6229b) transitioned into state 'RUNNING' from state 'PENDING' _flow_receiver
Flow 'volume_create_manager' (62429b) transitioned into state 'SUCCESS' from state 'RUNNING' _flow_receiver
```

ExtractVolumeRefTask

提取给定卷 ID 的卷引用

```
def execute(self, context, volume):
    # 从数据库中获取卷
    volume.refresh()
    return volume

# 相关日志。服务名称：cinder.volume.manager
Task
'cinder.volume.flows.manager.create_volume.ExtractVolumeRefTask;volume:create'
(40a5f) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task
'cinder.volume.flows.manager.create_volume.ExtractVolumeRefTask;volume:create'
(40a5f) transitioned into state 'SUCCESS' from state 'RUNNING' with result
'Volume(_name_id=None,admin_metadata=
{},attach_status='detached',availability_zone='nova',bootable=False,cluster=<?
>,cluster_name=None,consistencygroup=<?
>,consistencygroup_id=None,created_at=2023-12-
04T02:17:07Z,deleted=False,deleted_at=None,display_description='My first
volume',display_name='testvolume',ec2_id=None,encryption_key_id=None,glance_met
adata=<?>,group=<?>,group_id=None,host='xhs@rbd-1#rbd-1',id=6478980b-4976-44b4-
a777-55ce66fc5962,launched_at=None,metadata=
{},migration_status=None,multiattach=False,previous_status=None,project_id='cd7
99',provider_auth=None,provider_geometry=None,provider_id=None,provider_locatio
n=None,replication_driver_data=None,replication_extended_status=None,replicatio
n_status=None,scheduled_at=2023-12-
04T02:17:08Z,service_uuid=None,shared_targets=True,size=10,snapshot_id=None,sna
pshots=<?
>,source_volid=None,status='creating',terminated_at=None,updated_at=2023-12-
04T02:17:08Z,use_quota=True,user_id='c4394',volume_attachment=VolumeAttachmentL
ist,volume_type=VolumeType(45274d75-f642-41eb-a369-
c99d623566df),volume_type_id=45274d75-f642-41eb-a369-c99d623566df)'
_task_receiver
```

OnFailureRescheduleTask

处理失败时重新安排任务的请求

它的execute函数是空的，但是却提供了revert函数用于处理失败

```
def revert(self, context, result, flow_failures, volume, **kwargs):
    # 如果不需要重新调度，就返回False
    if not self.do_reschedule:
        common.error_out(volume)
        LOG.error("Volume %s: create failed", volume.id)
        return False
    # 失败类型在不需要重新调度的类型列表中
    for failure in flow_failures.values():
        if failure.check(*self.no_reschedule_types):
            common.error_out(volume)
```

```
                LOG.error("Volume %s: create failed", volume.id)
                return False

        # Use a different context when rescheduling.
        if self.reschedule_context:
            cause = list(flow_failures.values())[0]
            context = self.reschedule_context
            try:
                self._pre_reschedule(volume)
                self._reschedule(context, cause, volume=volume, **kwargs)
                self._post_reschedule(volume)
                return True
            except exception.CinderException:
                LOG.exception("Volume %s: rescheduling failed", volume.id)

        return False
```

具体的重新调度细节不阐述了。在正常情况下，它的日志非常简洁啥也没干

```
# 服务名称 cinder.volume.manager
Task 'cinder.volume.flows.manager.create_volume.OnFailureRescheduleTask;volume:create' (8184e) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task 'cinder.volume.flows.manager.create_volume.OnFailureRescheduleTask;volume:create' (8184e) transitioned into state 'SUCCESS' from state 'RUNNING' with result 'None' _task_receiver
```

ExtractVolumeSpecTask

就是做了一个参数翻译

```
def execute(self, context, volume, request_spec):
    get_remote_image_service = glance.get_remote_image_service

    volume_name = volume.name
    volume_size = utils.as_int(volume.size, quiet=False)
    # 创建一个字典，方便在不同类型之间切换
    specs = {
        'status': volume.status,
        'type': 'raw',  # This will have the type of the volume to be
                        # created, which should be one of [raw, snap,
                        # source_vol, image, backup]
        'volume_id': volume.id,
        'volume_name': volume_name,
        'volume_size': volume_size,
    }

    if volume.snapshot_id:
        # We are making a snapshot based volume instead of a raw volume.
        specs.update({
            'type': 'snap',
            'snapshot_id': volume.snapshot_id,
        })
    elif volume.source_volid:
        source_volid = volume.source_volid
        source_volume_ref = objects.Volume.get_by_id(context,
                                                     source_volid)
        specs.update({
            'source_volid': source_volid,
            'source_volstatus': source_volume_ref.status,
            'type': 'source_vol',
        })
    elif request_spec.get('image_id'):
        image_href = request_spec['image_id']
        image_service, image_id = get_remote_image_service(context,
                                                           image_href)
        specs.update({
            'type': 'image',
            'image_id': image_id,
            'image_location': image_service.get_location(context,
                                                         image_id),
            'image_meta': image_service.show(context, image_id),
            'image_service': image_service,
        })
    elif request_spec.get('backup_id'):
        specs.update({
            'type': 'backup',
            'backup_id': request_spec['backup_id'],
            'need_update_volume': True,
        })
    return specs

# 相关日志，服务名称：cinder.volume.manager
Task 'cinder.volume.flows.manager.create_volume.ExtractVolumeSpecTask;volume:create' (e5bbe) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task 'cinder.volume.flows.manager.create_volume.ExtractVolumeSpecTask;volume:create' (e5bbe) transitioned into state 'SUCCESS' from state 'RUNNING' with result
```

```
'{'status': 'creating', 'type': 'raw', 'volume_id': '64962', 'volume_name':
'volume-64962', 'volume_size': 10}' _task_receiver
```

NotifyVolumeActionTask

通知数据库开始创建卷

```
def execute(self, context, volume):
    if not self.event_suffix:
        return

    try:
        # 这一步应该是去通知数据库了，内部通过了奇怪的rpc机制获取了一个通知器，通过通知器转发
        # rpc.get_notifier("volume", host).info(context, 'volume.%s' % event_suffix, usage_info)
        volume_utils.notify_about_volume_usage(context, volume, self.event_suffix,
                                                host=volume.host)
    except exception.CinderException:
        LOG.exception("Failed notifying about the volume"
                      " action %(event)s for volume %(volume_id)s",
                      {'event': self.event_suffix, 'volume_id': volume.id})

# 相关日志，服务名称：cinder.volume.manager
Task 'cinder.volume.flows.manager.create_volume.NotifyVolumeActionTask;volume:create, create.start' (3dffd) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task 'cinder.volume.flows.manager.create_volume.NotifyVolumeActionTask;volume:create, create.start' (3d0b053a-4bff-4296-b129-e9a2b71a1ffd) transitioned into state 'SUCCESS' from state 'RUNNING' with result 'None' _task_receiver
```

CreateVolumeFromSpecTask

根据不同的条件创建卷(看流程图)，这里使用rbd创建创建

```
def execute(self,
            context: cinder_context.RequestContext,
            volume: objects.Volume,
            volume_spec) -> dict:
    volume_spec = dict(volume_spec)
    volume_id = volume_spec.pop('volume_id', None)
    if not volume_id:
        volume_id = volume.id

    # we can't do anything if the driver didn't init
    if not self.driver.initialized:
        driver_name = self.driver.__class__.__name__
        LOG.error("Unable to create volume. "
                  "Volume driver %s not initialized", driver_name)
        raise exception.DriverNotInitialized()

    # For backward compatibilty
    volume.populate_consistencygroup()

    create_type = volume_spec.pop('type', None)
    LOG.info("Volume %(volume_id)s: being created as %(create_type)s "
             "with specification: %(volume_spec)s",
             {'volume_spec': volume_spec, 'volume_id': volume_id,
              'create_type': create_type})

    model_update: dict
    if create_type == 'raw':
        # 调用rbd创建 return self.driver.create_volume(volume)
        model_update = self._create_raw_volume(context, volume, **volume_spec)
    else : xxx

    try:
        if model_update:
            with volume.obj_as_admin():
                volume.update(model_update)
                volume.save()
    except exception.CinderException:
        LOG.exception("Failed updating model of volume %(volume_id)s "
                      "with creation provided model %(model)s",
                      {'volume_id': volume_id, 'model': model_update})
        raise
    return volume_spec


# 相关日志
cinder.volume.manager Task
'cinder.volume.flows.manager.create_volume.CreateVolumeFromSpecTask;volume:crea
te' (7ec951) transitioned into state 'RUNNING' from state 'PENDING'
_task_receiver
cinder.volume.drivers.rbd creating volume 'volume-64962' create_volume
/var/lib/kolla/venv/lib64/python3.9/site-
packages/cinder/volume/drivers/rbd.py:1146
```

```
cinder.volume.drivers.rbd connecting to cinder@ceph (conf=/etc/ceph/ceph.conf, timeout=5). _do_conn /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/volume/drivers/rbd.py:597
cinder.volume.drivers.rbd extra_specs: {} _is_replicated_type /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/volume/drivers/rbd.py:998
cinder.volume.drivers.rbd extra_specs: {} _is_multiattach_type /var/lib/kolla/venv/lib64/python3.9/site-packages/cinder/volume/drivers/rbd.py:1007
cinder.volume.manager Task 'cinder.volume.flows.manager.create_volume.CreateVolumeFromSpecTask;volume:create' (7e951) transitioned into state 'SUCCESS' from state 'RUNNING' with result '{'status': 'creating', 'volume_name': 'volume-64962', 'volume_size': 10}' _task_receiver
```

CreateVolumeOnFinishTask

创建完毕，收尾工作

```
def execute(self, context, volume, volume_spec):
    need_update_volume = volume_spec.pop('need_update_volume', True)
    if not need_update_volume:
        super(CreateVolumeOnFinishTask, self).execute(context, volume)
        return

    new_status = self.status_translation.get(volume_spec.get('status'),
                                             'available')
    update = {
        'status': new_status,
        'launched_at': timeutils.utcnow(),
    }
    try:
        # 更新卷的信息并保存在数据库
        volume.update(update)
        volume.save()
        # 发送通知
        super(CreateVolumeOnFinishTask, self).execute(context, volume)
    except exception.CinderException:
        LOG.exception("Failed updating volume %(volume_id)s with "
                      "%(update)s", {'volume_id': volume.id,
                                     'update': update})
    # 即使更新失败，卷也成功被创建了
    LOG.info("Volume %(volume_name)s (%(volume_id)s): "
             "created successfully",
             {'volume_name': volume_spec['volume_name'],
              'volume_id': volume.id})
```

```

# 相关日志，服务名称: cinder.volume.manager
Task 'cinder.volume.flows.manager.create_volume.CreateVolumeOnFinishTask;volume:create, create.end' (a4da7) transitioned into state 'RUNNING' from state 'PENDING' _task_receiver
Task 'cinder.volume.flows.manager.create_volume.CreateVolumeOnFinishTask;volume:create, create.end' (a4da7) transitioned into state 'SUCCESS' from state 'RUNNING' with result 'None' _task_receiver
```