c-sch创建卷时的TaskFlow：volume_create_scheduler：

```
create_what = {
    'context': context,
    'raw_request_spec': request_spec,
    'filter_properties': filter_properties,#当前环境下是None
    'volume': volume,#指定了ID，availability_zone信息，但是host信息未指定
    'snapshot_id': snapshot_id,#None
    'image_id': image_id,#从image创建时，不是None
}
```

- ExtractSchedulerSpecTask

主要是为了向后兼容，当request_spec为空时创建一个request_spec

```

RequestSpec(
CG_backend=<?>,
cgsnapshot_id=None,
consistencygroup_id=None,
group_backend=<?>,
group_id=None,
image_id=3a1a735f-9ed5-4b94-bf86-ceafa40c6a62,
snapshot_id=None,
source_replicaid=None,
source_volid=None,
volume=Volume(e312e6a6-f7c4-4f6a-9926-a63be273069b),
volume_id=e312e6a6-f7c4-4f6a-9926-a63be273069b,
volume_properties=VolumeProperties,
volume_type=None
)
```

- ScheduleCreateVolumeTask

调用scheduler driver执行调度，并且通过_handle_failure和_notify_failure处理失败！

过滤HOST的操作主要在驱动模块cinder.scheduler.filter_scheduler.FilterScheduler中实现。

同时，可以看到ScheduleCreateVolumeTask会Catch驱动产生的异常，但是catch异常并不是为了重试，重试的环节发生在c-vol

```
def execute(self, context, request_spec, filter_properties, volume):
    try:
        self.driver_api.schedule_create_volume(context, request_spec,
                                               filter_properties)
    except Exception as e:
        # An error happened, notify on the scheduler queue and log that
        # this happened and set the volume to errored out and reraise the
        # error *if* exception caught isn't NoValidBackend. Otherwise *do
        # not* reraise (since what's the point?)
        with excutils.save_and_reraise_exception(
                reraise=not isinstance(e, exception.NoValidBackend)):
            if isinstance(e, exception.NoValidBackend):
                self.message_api.create(
                    context,
                    defined_messages.EventIds.UNABLE_TO_ALLOCATE,
                    context.project_id,
                    resource_type=resource_types.VOLUME,
                    resource_uuid=request_spec['volume_id'])
            try:
                self._handle_failure(context, request_spec, e)
            finally:
```

```
                common.error_out(volume, reason=e)
```

- scheduler driver：cinder.scheduler.filter_scheduler.FilterScheduler

```
def schedule_create_volume(self, context, request_spec, filter_properties):
    """
    1.调用_schedule方法完成HOST过滤，注意这里的返回的HOST级别到POOL为止：
        _schedule方法：
            a:调用weighed_backends = self._get_weighted_candidates(context, request_spec,filter_properties)进行HOST过滤，和排序
            b:根据传入的CG_backend，group_backend 对weighed_backends进一步过滤，这里过滤的级别到backend
                注意：HOST级别的过滤已经在API处完成了
            c.返回一个最优Pool
    2.更新db的信息，这里主要是更新，HOST字段
    3.将HOST信息加入：
                    filter_properties --> retry --> num_attempts : int
                                      --> backends : []
                                      --> hosts : []
    4.调用volume_api

    :param context:
    :param request_spec: 请求的信息，包括CG_backend，image_id，volume_id .....
    :param filter_properties: 过滤器属性，用于记录一些过滤器参数，如volume_type，size，retry 等信息
    :return:
    """
    backend = self._schedule(context, request_spec, filter_properties)
    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)
    self._post_select_populate_filter_properties(filter_properties,
                                                 backend)

    # context is not serializable
    filter_properties.pop('context', None)

    self.volume_rpcapi.create_volume(context, updated_volume, request_spec,
                                     filter_properties,
                                     allow_reschedule=True)
```

_get_weighted_candidates方法代码片段：

```
def _get_weighted_candidates(self, context, request_spec,
                             filter_properties=None):
    """
    1.默认情况下传入的filter_properties是None，此时filter_properties会在该方法中被填充：
        filter_properties --> retry --> num_attempts : int
                          --> backends : []
                          --> hosts : []
        filter_properties --> context
                    --> request_spec
                    --> config_options :来自一个json配置文件，CONF.scheduler_json_config_location
                    --> volume_type -> 创建时传入，nova调用时为None?
                    --> resource_type -> 创建时传入
        filter_properties --> size
                    --> availability_zone
```

```
                --> user_id
                --> metadata
                --> qos_specs
    2.调用筛选后端：cinder.scheduler.host_manager.HostManager
        backends = self.host_manager.get_filtered_backends(backends,filter_properties)
        1.get_filtered_backends根据CONF.scheduler_default_filters的配置从cinder.scheduler.filters下加载class，
            默认时有：
                'AvailabilityZoneFilter'：可用域过滤，其实通过API时已经做了部分工作
                'CapacityFilter'：过滤存储空间不满足的的HOST
                'CapabilitiesFilter'：
                        Volume Provider 有自己的特性（Capabilities），Cinder 允许用户创建 Volume 时通过 Volume Type 指定需要的 Capabi
                        如后端配置中的volume_backend_name参数，可以过滤相应的backend
        2.之后调用cinder.scheduler.base_handler.BackendFilterHandler.get_filtered_objects方法依次进行过滤
                过滤器会依次，调用run_filter_for_index，和filter_all执行过滤
    3.对backends计算权重：
        backends = self.host_manager.get_filtered_backends(backends,filter_properties)
    """
    elevated = context.elevated()

    # Since Cinder is using mixed filters from Oslo and it's own, which
    # takes 'resource_XX' and 'volume_XX' as input respectively, copying
    # 'volume_XX' to 'resource_XX' will make both filters happy.
    volume_type = resource_type = request_spec.get("volume_type")
    #From CONF.scheduler_json_config_location is a json file
    config_options = self._get_configuration_options()

    if filter_properties is None:
        filter_properties = {}
    self._populate_retry(filter_properties,
                         request_spec['volume_properties'])

    request_spec_dict = jsonutils.to_primitive(request_spec)

    filter_properties.update({'context': context,
                              'request_spec': request_spec_dict,
                              'config_options': config_options,
                              'volume_type': volume_type,
                              'resource_type': resource_type})

    self.populate_filter_properties(request_spec,
                                    filter_properties)

    # If multiattach is enabled on a volume, we need to add
    # multiattach to extra specs, so that the capability
    # filtering is enabled.
    multiattach = request_spec['volume_properties'].get('multiattach',
                                                        False)
    if multiattach and 'multiattach' not in resource_type.get(
            'extra_specs', {}):
        if 'extra_specs' not in resource_type:
            resource_type['extra_specs'] = {}

        resource_type['extra_specs'].update(
            multiattach='<is> True')

    # Find our local list of acceptable backends by filtering and
    # weighing our options. we virtually consume resources on
    # it so subsequent selections can adjust accordingly.
```

```
    # Note: remember, we are using an iterator here. So only
    # traverse this list once.


    backends = self.host_manager.get_all_backend_states(elevated)
    # Filter local hosts based on requirements ...
    backends = self.host_manager.get_filtered_backends(backends,
                                                       filter_properties)
    if not backends:
        return []

    LOG.debug("Filtered %s", backends)
    # weighted_backends = WeightedHost() ... the best
    # backend for the job.
    weighted_backends = self.host_manager.get_weighed_backends(
        backends, filter_properties)
    return weighed_backends
```

- 添加自定义Filter

```
# Find our local list of acceptable backends by filtering and
# weighing our options. we virtually consume resources on
# it so subsequent selections can adjust accordingly.

# Note: remember, we are using an iterator here. So only
# traverse this list once.
backends = self.host_manager.get_all_backend_states(elevated)

# Filter local hosts based on requirements ...
backends = self.host_manager.get_filtered_backends(backends,
                                                   filter_properties)
```

1.cinder.scheduler.host_manager.HostManager._choose_backend_filters方法会加载所有配置的Filters并且逐一调用。

2.cinder Filter的配置项 CONF.scheduler_default_filters。默认配置为：AvailabilityZoneFilter,CapacityFilter,CapabilitiesFilter

3.Filter模块的py文件定义在cinder.scheduler.filters包下，并且继承cinder.scheduler.filters.BaseBackendFilter,自定义的Filter需要实现以下

```
def backend_passes(self, backend_state, filter_properties)
```

当前ocata版本已经实现的Filter列表

Filter类名称	功能	备注
IgnoreAttemptedHostsFilter	过滤已经调度过且失败的HOST	注意需要将失败的HOST加入 filter_properties的retry时过滤器才会生
AffinityFilter	亲和性过滤器，有 DifferentBackendFilter和SameBackendFilter两种实现，效果相反。	假设 cinder 对接了两个后端 BackEndA、Ba 通过在创建卷时指定hint,DifferentBacken 会产生如下效果： cinder create --hint different_host=<v cinder create --hint different_host=<v SameBackendFilter和DifferentBackendFil
AvailabilityZoneFilter	过滤cinder卷的可用域（针对c-volume的部署节点过滤）， 实际上这个过滤器的工作在API已经有涉及	相关配置项default_availability_zone、s allow_availability_zone_fallback、clon
		1.假设通过get-pools能够返回是否支持net

CapabilitiesFilter	通过指定Volume_Type来工作的过滤器。默认情况下卷类型为None， 通过在cinder_api节点配置default_volume_type， 可以指定默认的Volume_type	那么可以用netapp_thin_provisioned定义一 2.CapabilitiesFilter有比较多的匹配规则 使用CapabilitiesFilter过滤指定的volume_ 创建一个volume_type 配置volume_backend_name = s!= <backend 其中，s!= <backend_name_a>表示将名字为
CapacityFilter	根据当前卷的剩余空间大小，进行过滤。	CapacityFilter收集的信息有： free_capacity_gb（实际可用的pool空间） reserved_percentage（需要预留的pool空间 provisioned_capacity_gb（实际已经分配的 thin_provisioning_support（是否支持瘦分 按照以下顺序进行判断： 1、如果 free_capacity_gb in ['infinit 2、如果 total_capacity_gb in ['infinit 计算当前剩余空间 free = free_capacity_ 计算考虑瘦分配时的剩余空间 adjusted_fr 3、volume_type是否要求瘦分配（provisio 否者返回free > requested size

DriverFilter	根据filter function对pool进行过滤(一定要指定卷类型否者不能工作)	filter function是一个表达式字符串如：
		filter_function | capabilitie
		DriverFilter 会收集以下信息，作为表达式
		stats = {
		'backend_stats': backend_stats,
		'backend_caps': backend_caps,
		'extra_specs': extra_specs,
		'qos_specs': qos_specs,
		'volume_stats': volume_stats,
		'volume_type': volume_type,
		'filter_function': filter_function,
		}
		filter_function可以在后端配置，如果没有
		疑问：
		1、表达是如何翻译的（规则引擎模块cinde

InstanceLocalityFilter	保证BackEnd的物理机和hypervisor一致	通过在--hint指定local_to_instance = <
JsonFilter	通过--hint指定query事项实现过滤	query是一个Json串，格式为 [<cmd>,$arg_ $arg_in_backend表示要判断的backend参数 --hint query=["not","$volume_backend_n 表示过滤所有名称为NetNetAppIscsiBacken