用户调用Nova API挂载磁盘

```
os-volume_attachments
{
  "volumeAttachment": {
    "device": "/dev/vdb",
    "serverId": "6edad723-c62d-40d4-a29e-d9ac6a3247ee",
    "id": "3b9fdb6c-200e-494d-aa71-f47c70e87d7e",
    "volumeId": "3b9fdb6c-200e-494d-aa71-f47c70e87d7e"
  }
}
```

```
API接口相应nova volume-attach请求：
nova.api.openstack.compute.volumes.VolumeAttachmentController
def create(self, req, server_id, body)
```

```
nova.compute.api.API
    @check_instance_lock
    @check_instance_state(vm_state=[vm_states.ACTIVE, vm_states.PAUSED,vm_states.STOPPED, vm_states.RESIZED,
                                    vm_states.SOFT_DELETED, vm_states.SHELVED,
                                    vm_states.SHELVED_OFFLOADED])
    def attach_volume(self, context, instance, volume_id, device=None,
                      disk_bus=None, device_type=None):
        """Attach an existing volume to an existing instance."""
        if device and not block_device.match_device(device):
            raise exception.InvalidDevicePath(path=device)

        is_shelved_offloaded = instance.vm_state == vm_states.SHELVED_OFFLOADED
        if is_shelved_offloaded:
            """
            shelved_offloaded状态下的虚机也允许挂载,shelved_offloaded状态下虚机从hypervisor层面清除，
            在_attach_volume_shelved_offloaded方法中虚机并没有实际在KVM层面执行挂载，只是创建block_device_mapping表的记录
            并且通知Cinder修改卷状态为in-use,并且直接创建attachment记录
            """
            return self._attach_volume_shelved_offloaded(context,instance,volume_id,
                                                         device,disk_bus,device_type)

      return self._attach_volume(context, instance, volume_id, device,
                                 disk_bus, device_type)
```

```
nova.compute.api.API
    def _attach_volume(self, context, instance, volume_id, device,
                       disk_bus, device_type):
        """Attach an existing volume to an existing instance.

        This method is separated to make it possible for cells version
        to override it.
```

```
        """
        """
        _create_volume_bdm方法负责在block_device_mapping表中创建一个记录，接口会通过API节点创建，
        或者API节点调用nova_compute节点的RPC接口reserve_block_device_name来创建！
        """
        volume_bdm = self._create_volume_bdm(
            context, instance, device, volume_id, disk_bus=disk_bus,
            device_type=device_type)
        try:
            """
            可用域检查与卷状态置attaching
            """
            self._check_attach_and_reserve_volume(context, volume_id, instance)
         self.compute_rpcapi.attach_volume(context, instance, volume_bdm)
        except Exception:
            with excutils.save_and_reraise_exception():
                volume_bdm.destroy()

        return volume_bdm.device_name

```

```
nova.compute.manager.ComputeManager
    @wrap_exception()
    @wrap_instance_fault
    def attach_volume(self, context, instance, bdm):
        """Attach a volume to an instance."""
        """
        nova.virt.block_device -- > driver_block_device
        convert_volume方法的主要主要作用是将Block_Device_Mapping中的ORM对象装换成
        DriverVolumeBlockDevice、DriverSnapshotBlockDevice、DriverImageBlockDevice、DriverBlankBlockDevice等对象
        """
        driver_bdm = driver_block_device.convert_volume(bdm)

        """
        对下面的代码用uuid加锁
        """
        @utils.synchronized(instance.uuid)
        def do_attach_volume(context, instance, driver_bdm):
            try:
                return self._attach_volume(context, instance, driver_bdm)
            except Exception:
                with excutils.save_and_reraise_exception():
                    bdm.destroy()

        do_attach_volume(context, instance, driver_bdm)
```

```
nova.compute.manager.ComputeManager
    def _attach_volume(self, context, instance, bdm):
        context = context.elevated()
        LOG.info(_LI('Attaching volume %(volume_id)s to %(mountpoint)s'),
```

```
                {'volume_id': bdm.volume_id,
                 'mountpoint': bdm['mount_device']},
                instance=instance)
        try:
            """
            DriverVolumeBlockDevice、DriverSnapshotBlockDevice、DriverImageBlockDevice、DriverBlankBlockDevice都有att
            但是他们的接口稍有不同，除了DriverVolumeBlockDevice，其他三种类型都需要在cinder处创建卷，因此需要传递wait_func参
            换句话说DriverSnapshotBlockDevice、DriverImageBlockDevice、DriverBlankBlockDevice的attach包含的volume crea
            主要用于创建虚机的流程。
            如果虚机创建完成，将一个已有卷挂载到VM，走DriverVolumeBlockDevice的attach接口！
            """
            bdm.attach(context, instance, self.volume_api, self.driver,
                       do_check_attach=False, do_driver_attach=True)
        except Exception:
            with excutils.save_and_reraise_exception():
                LOG.exception(_LE("Failed to attach %(volume_id)s "
                                  "at %(mountpoint)s"),
                              {'volume_id': bdm.volume_id,
                               'mountpoint': bdm['mount_device']},
                              instance=instance)
                self.volume_api.unreserve_volume(context, bdm.volume_id)

        info = {'volume_id': bdm.volume_id}
        self._notify_about_instance_usage(
            context, instance, "volume.attach", extra_usage_info=info)
```

nova.virt.block_device.DriverVolumeBlockDevice中的attach方法：

整个过程如下：

1. 调用os-brick获取本机信息，以及传输协议需要的信息

   ---->nova.virt.block_device.DriverVolumeBlockDevice.attach

2. 通过cinderAPI获取VOLUME的连接信息

   ---->nova.virt.block_device.DriverVolumeBlockDevice.attach

3. 调用协议驱动建立连接，如NFS协议mount关键目录，iscsi协议调用os-brick建立连接等（该过程发生在nova_compute容器中）！

   --->nova.virt.driver.LibvirtDriver.attach_volume

4. nova-compute通知nova-libvirt创建virtio设备驱动，并更新虚机的XML配置(该过程发生在nova_libvirt容器中)！

   --->nova.virt.driver.LibvirtDriver.attach_volume

5. 通知Cinder修改状态，并且完成数据库的最后更新！

   ---->nova.virt.block_device.DriverVolumeBlockDevice.attach

```
nova.virt.block_device.DriverVolumeBlockDevice
    @update_db
    def attach(self, context, instance, volume_api, virt_driver,
               do_check_attach=True, do_driver_attach=False, **kwargs):

        volume = volume_api.get(context, self.volume_id)
        if do_check_attach:
            """
            检查VOLUME的状态，挂载虚机的时候是不检查的，考虑到multi-attach？？
            """
```

```
            volume_api.check_attach(context, volume, instance=instance)

        volume_id = volume['id']
        context = context.elevated()
        """
        libvirt.LibvirtDriver -->virt_driver
        os_brick.initiator.connector,收集本机的信息，以及一些存储协议建立连接时所用的信息。
        """
        connector = virt_driver.get_volume_connector(instance)
        """
        向Cinder请求卷的连接信息 --> os-initialize_connection
        """
        connection_info = volume_api.initialize_connection(context, volume_id,connector)
        if 'serial' not in connection_info:
            connection_info['serial'] = self.volume_id
        self._preserve_multipath_id(connection_info)

        """
        对于active状态下的VM进行attach，do_driver_attach必定为true；
        而创建过程中一些卷的attach，do_driver_attach为false，应为在虚机启动过程中，已经执行attach了
        """

        if do_driver_attach:
            encryption = encryptors.get_encryption_metadata(
                context, volume_api, volume_id, connection_info)

            try:
                """
                Libvirt驱动中的attach方法建立连接!

                """

            virt_driver.attach_volume(context, connection_info, instance,
                                      self['mount_device'], disk_bus=self['disk_bus'],
                                      device_type=self['device_type'], encryption=encryption)


            except Exception:
                with excutils.save_and_reraise_exception():
                    LOG.exception(_LE("Driver failed to attach volume "
                                      "%(volume_id)s at %(mountpoint)s"),
                                  {'volume_id': volume_id,
                                   'mountpoint': self['mount_device']},
                                  instance=instance)
                    volume_api.terminate_connection(context, volume_id,
                                                    connector)
        self['connection_info'] = connection_info
        if self.volume_size is None:
            self.volume_size = volume.get('size')

        mode = 'rw'
        if 'data' in connection_info:
            mode = connection_info['data'].get('access_mode', 'rw')

        if volume['multiattach'] or volume['attach_status'] == "detached":
            # NOTE(mriedem): save our current state so connection_info is in
            # the database before the volume status goes to 'in-use' because
            # after that we can detach and connection_info is required for
```

```
            # detach.
            """
            更新Block_Device_Mapping表中的信息
            """
            self.save()
            try:
                """
                cinder的attach接口主要工作是更新volume的状态，已经attachments表中的记录!
                并且该接口为driver提供了一个回调方法，BaseVD.attach()该方法没有实际操作，可以用来定制!
                """
            volume_api.attach(context, volume_id, instance.uuid,self['mount_device'], mode=mode)
            except Exception:
                with excutils.save_and_reraise_exception():
                    if do_driver_attach:
                        try:
                            virt_driver.detach_volume(connection_info,
                                                      instance,
                                                      self['mount_device'],
                                                      encryption=encryption)
                        except Exception:
                            LOG.warning(_LW("Driver failed to detach volume "
                                            "%(volume_id)s at %(mount_point)s."),
                                        {'volume_id': volume_id,
                                         'mount_point': self['mount_device']},
                                        exc_info=True, instance=instance)
                    volume_api.terminate_connection(context, volume_id,
                                                    connector)

                    # Cinder-volume might have completed volume attach. So
                    # we should detach the volume. If the attach did not
                    # happen, the detach request will be ignored.
                    volume_api.detach(context, volume_id)
```

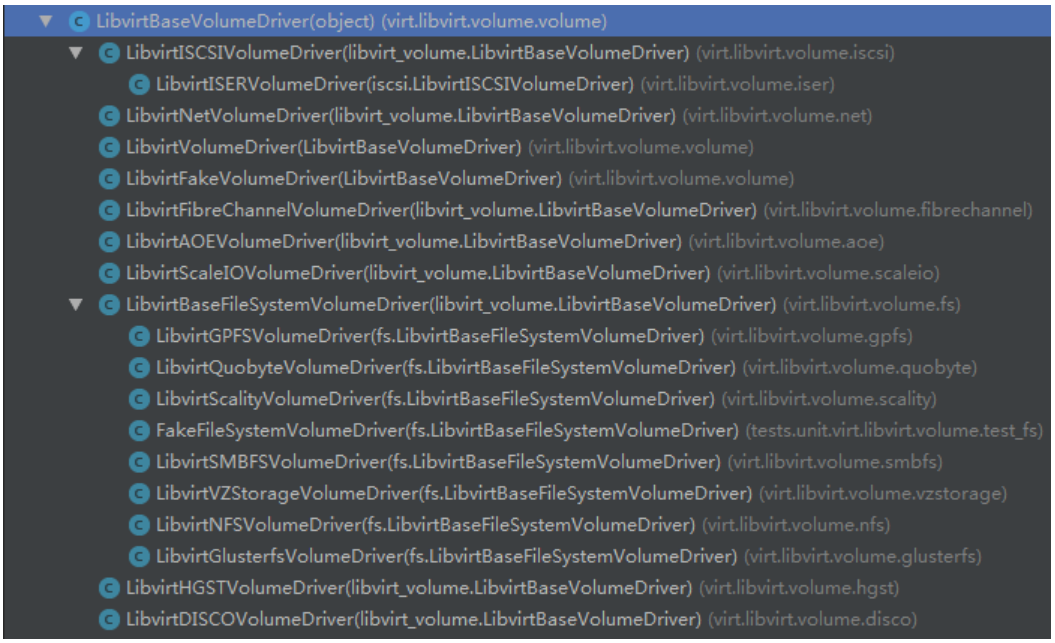
关于Libvirt实现的不同存储协议：

```
#包路径：nova.virt.libvirt.volume
@profiler.trace_cls("volume_api")
class LibvirtBaseVolumeDriver(object):
    """Base class for volume drivers."""
    def __init__(self, host, is_block_dev):
        self.host = host
        self.is_block_dev = is_block_dev

    def get_config(self, connection_info, disk_info):
        """Returns xml for libvirt."""
        """生成该存储在VM中使用的XML"""
        return conf

    def connect_volume(self, connection_info, disk_info):
        """Connect the volume."""
        pass

    def disconnect_volume(self, connection_info, disk_dev):
        """Disconnect the volume."""
        pass
```



cinder的ATTACH接口，这个接口实际上是NOVA完成ATTACH操作以后的一个回调接口，主要用途是修改状态，以及创建attachment记录！

```
cinder.volume.manager.VolumeManager
    @coordination.synchronized('{volume_id}')
    def attach_volume(self, context, volume_id, instance_uuid, host_name,
                      mountpoint, mode, volume=None):
        """Updates db to show volume is attached."""
        # FIXME(lixiaoy1): Remove this in v4.0 of RPC API.
        if volume is None:
            # For older clients, mimic the old behavior and look
            # up the volume by its volume_id.
            volume = objects.Volume.get_by_id(context, volume_id)
        # Get admin_metadata. This needs admin context.
        with volume.obj_as_admin():
            volume_metadata = volume.admin_metadata
        # check the volume status before attaching
        if volume.status == 'attaching':
            if (volume_metadata.get('attached_mode') and volume_metadata.get('attached_mode') != mode):
                raise exception.InvalidVolume(reason=_("being attached by different mode"))

        if (volume.status == 'in-use' and not volume.multiattach and not volume.migration_status):
            raise exception.InvalidVolume(reason=_("volume is already attached"))

        host_name_sanitized = utils.sanitize_hostname(
            host_name) if host_name else None
        if instance_uuid:
            attachments = (VA_LIST.get_all_by_instance_uuid(context, instance_uuid))
        else:
            attachments = (VA_LIST.get_all_by_host(context, host_name_sanitized))
        if attachments:
            # check if volume<->instance mapping is already tracked in DB
            """
            检查attachments表的记录如果记录已经存在，将volume.status改为in-use，并且返回相应attachment!
            """
```

```
            for attachment in attachments:
                if attachment['volume_id'] == volume_id:
                    volume.status = 'in-use'
                    volume.save()
                    return attachment

        self._notify_about_volume_usage(context, volume,"attach.start")
        """
        创建attachment表记录，将attachment状态改为ATTACHING
        """
        attachment = volume.begin_attach(mode)

        if instance_uuid and not uuidutils.is_uuid_like(instance_uuid):
            attachment.attach_status = (fields.VolumeAttachStatus.ERROR_ATTACHING)
            attachment.save()
            raise exception.InvalidUUID(uuid=instance_uuid)

        if volume_metadata.get('readonly') == 'True' and mode != 'ro':
            attachment.attach_status = (fields.VolumeAttachStatus.ERROR_ATTACHING)
            attachment.save()
            self.message_api.create(
                context, defined_messages.EventIds.ATTACH_READONLY_VOLUME,
                context.project_id, resource_type=resource_types.VOLUME,
                resource_uuid=volume.id)
            raise exception.InvalidVolumeAttachMode(mode=mode,volume_id=volume.id)

        try:
            utils.require_driver_initialized(self.driver)

            LOG.info(_LI('Attaching volume %(volume_id)s to instance '
                         '%(instance)s at mountpoint %(mount)s on host '
                         '%(host)s.'),
                     {'volume_id': volume_id, 'instance': instance_uuid,
                      'mount': mountpoint, 'host': host_name_sanitized},
                     resource=volume)

            """
            关于驱动的一个回调函数！可以为一些定制预留接口！
            """
            self.driver.attach_volume(context,
                                      volume,
                                      instance_uuid,
                                      host_name_sanitized,
                                      mountpoint)
        except Exception:
            with excutils.save_and_reraise_exception():
                attachment.attach_status = (
                    fields.VolumeAttachStatus.ERROR_ATTACHING)
                attachment.save()
        """
        完成attach
        """
        volume = attachment.finish_attach(
            instance_uuid,
            host_name_sanitized,
            mountpoint,
            mode)

```

```
91          self._notify_about_volume_usage(context, volume, "attach.end")
92          LOG.info(_LI("Attach volume completed successfully."),
93                   resource=volume)
94          return attachment
```