对接

配置项	作用	默认值	备注
nfs_shares_config	记录远程磁盘地址的配置文件， 远程磁盘地址如：172.24.3.172:/NFS_Pool_1	/etc/cinder/nfs_shares	配置文件中可以配置多个地址，但是并非一个应一个Pool，而是多个地址对应一个Pool
nfs_sparsed_volumes	创建稀疏磁盘，相当于瘦分配	True	
nfs_qcow2_volumes	创建qcow2格式的磁盘，False时时raw格式	False	
nfs_mount_point_base	远程磁盘的挂载点目录	$state_path/mnt	
nfs_mount_options	执行mount命令时的附加选项		
nfs_mount_attempts	执行mount命令的尝试次数	3	
nfs_snapshot_support	是否支持快照，取决于libvirt版本	False	
reserved_percentage	预留百分比	0	这两个参数都是driver驱动的通用参数
max_over_subscription_ratio	超配比例	20	

驱动

RemoteFSDriver ---> RemoteFSSnapDriverBase ---> RemoteFSSnapDriver ---> NfsDriver

RemoteFSDriver是类似NFS形式的远程驱动实现类，类似的还有WindowsSmbfsDriver、VZStorageDriver

RemoteFSSnapDriverBase、RemoteFSSnapDriver是qcow2快照的实现接口，接口方法：_local_volume_dir(self, volume)

NfsDriver底层通过导入os_brick的相应模块完成底层的mount操作，

```
from os_brick.remotefs import remotefs as remotefs_brick
```

驱动的关键方法

构造函数

读取绝大部分配置项，并且调用os_brick初始化_remotefsclient(RemoteFsClient类)，_remotefsclient的作用相当于商业存储中负责发送ResfulApi的接口类？？

do_setup方法

1.初始化nova_compute的API接口

2.检查mount.nfs工具是否已经安装，这里直接在本机执行mount.nfs命令，利用的是cinder.utils.execute(*cmd,**kwargs)接口，该工具可以进行root权限提权！！

```
try:
    self._execute('mount.nfs', check_exit_code=False,run_as_root=True)
except OSError as exc:
    if exc.errno == errno.ENOENT:
        msg = _('%s is not installed') % package
        raise exception.NfsException(msg)
    else:
        raise
```

2.检查环境是否支持快照

_ensure_shares_mounted方法

_ensure_shares_mounted会调用_remotefsclient.mount(nfs_share, mnt_flags)周期性的检查目录是否挂载，涉及到的linux命令如下：

```
#检查当前路径是否已经挂载
mount
#创建挂载点目录,创建mount_path路径的方式是：mount_base+hashlib.md5(remote_path).hexdigest()
    mkdir -p mount_path
#挂载命令
    mount -t nfs -o <mount-option> remote_path mount_path
```

远程目录何时挂载到*Cinder_Volume?*

***get_volume_stats*方法虽然是*backend*上报*storage*状态的一个定时任务，但是每次环境启动时在*VolumeManager*中会被主动调用，继而调用*_update_volume_stats*方法。**

***_update_volume_stats*会调用*_ensure_shares_mounted*检查挂载状态，如果*nfs_shares_config*文件中的共享目录还未挂载，那么进行挂载。**

Backend状态的统计

***NFS*驱动统计状态时，*nfs_shares_config*中所有的远程目录视为一个*Pool*，而不是每个目录视为一个*Pool*。**

在_update_volume_stats方法中，会累加所有的容量上报到c-sch：

```
global_capacity = 0
global_free = 0
for share in self._mounted_shares:
     capacity, free, used = self._get_capacity_info(share)
     global_capacity += capacity
     global_free += free
```

_get_capacity_info方法中调用以下方法统计单个目录的容量：

```
stat -f -c "%S %b %a" mount_path
```

返回值依次为：块大小（byte）、块数目、可用块数目（统计total_size、total_available）

```
du -sb --apparent-size --exclude '*snapshot*' mount_path   #返回磁盘的实际占用大小
```

由于mount_path/.snapshot路径下存在快照，上面的命令统计快照的大小（该命令统计出的是total_allocated大小？？？why？）

```
drwxr-xr-x 2 root root 4096 Jun 30 10:22 daily.2017-07-01_0010
drwxr-xr-x 2 root root 4096 Jun 30 10:22 daily.2017-07-02_0010
drwxr-xr-x 2 root root 4096 Jun 30 10:22 hourly.2017-07-02_1505
drwxr-xr-x 2 root root 4096 Jun 30 10:22 hourly.2017-07-02_1605
drwxr-xr-x 2 root root 4096 Jun 30 10:22 hourly.2017-07-02_1705
drwxr-xr-x 2 root root 4096 Jun 30 10:22 hourly.2017-07-02_1805
drwxr-xr-x 2 root root 4096 Jun 30 10:22 hourly.2017-07-02_1905
drwxr-xr-x 2 root root 4096 Jun 30 10:22 hourly.2017-07-02_2005
drwxr-xr-x 2 root root 4096 Jun 30 10:22 weekly.2017-07-02_0015
```

创建卷

NfsDriver的create_volume方法实现在RemoteFSDriver中，接口如下：

```
cinder.volume.drivers.remotefs.RemoteFSDriver
def create_volume(self, volume)
```

由于NFS中一个Pool由几个远程盘组成，因此create命令到了驱动时，驱动需要调用_find_share为volume选择一个远程盘（选择结果保存在provider_location中）。

```
def create_volume(self, volume):
```

```
    """Creates a volume.
    :param volume: volume reference
    :returns: provider_location update dict for database
    """
    LOG.debug('Creating volume %(vol)s', {'vol': volume.id})
    self._ensure_shares_mounted()
    volume.provider_location = self._find_share(volume.size)
    LOG.info(_LI('casted to %s'), volume.provider_location)
    self._do_create_volume(volume)
    return {'provider_location': volume.provider_location}
```

_find_share方法会选择一个共享目录，选择的条件如下：
1.卷是否有足够剩余空间，这里会参考max_over_subscription_ratio的值，计算剩余空间，且无论是否配置了nfs_sparsed_volumes
2.选取符合条件1，且已分配容量最小的！注意不是剩余最多的？？
_do_create_volume(self, volume)方法会根据配置文件创建三种不同的磁盘文件,分别对应qcow2、稀疏文件、常规文件，命令如下：

```
qemu-img create -f qcow2 -o preallocation=metadata <file_path> <size_gb * units.Gi>
truncate -s <size_gb> <file_path>  #truncate命令不会占用磁盘大小，但是会改变磁盘的声称大小。
dd if=/dev/zero of=<file_path> bs=1M count=<block_count>  #执行块拷贝，输入是/dev/zero，默认情形下单次拷贝的块大小为1MB
```

```
-rw-r--r-- 1 stack stack     393216 Jul  3 14:17 qcow2_test
-rw-rw-r-- 1 stack stack 1073741824 Jul  3 14:19 regular_test
-rw-rw-r-- 1 stack stack 1073741824 Jul  3 14:18 sparsed_test
```

手工调用命令创建上面三个文件，大小均为1Gb，qcow2、稀疏文件基本立即完成，而常规文件需要将近20s左右时间相应。

创建完成后配置权限为ugo+rw或者660

一个需要注意的地方：

如果*NFS*后端配置的是执行块拷贝的方式创建卷，那么创建过程中可能会出现长时间的状态卡死（卷状态为*“creating”*），并且由于*_find_share*方法计算共享目录剩余容量时考虑了*max_over_subscription_ratio*，因此可能会出现*dd*命令空间不足的情形。

卷快照

接口如下：

```
cinder.volume.drivers.remotefs.RemoteFSSnapDriver
def _create_snapshot(self, snapshot)
```

***NFS*驱动的快照原理：**

在建立磁盘快照时，并不需要复制数据本身。它所作的只是将磁盘的数据文件保存起来，不被覆写。

这个通知动作只需花费极短的时间。接下来的档案修改或任何新增、删除动作，均不会覆写原本数据所在的文件，而是将修改部分写入其它可用的文件中。

首次对*NFS*磁盘进行快照时，*c-vol*会在*nfs_shares_config*目录下创建一个快照映射文件*volume-<uuid>.info*，该文件存储了快照文件的映射顺序，格式如下

```
{
 "active": "volume-b3950e10-929a-4640-81a8-78d7cfd2f3f6.73cdd2fb-80cf-4870-a46d-c3b32564ba34",
 "73cdd2fb-80cf-4870-a46d-c3b32564ba34": "volume-b3950e10-929a-4640-81a8-78d7cfd2f3f6.73cdd2fb-80cf-4870-a46d-c3b32564ba34",
 "1f8766df-639f-4486-a7cf-602043bfde4f": "volume-b3950e10-929a-4640-81a8-78d7cfd2f3f6.1f8766df-639f-4486-a7cf-602043bfde4f",
 "cee7ebe3-e994-4c64-a66a-2dcdeca57bd6": "volume-b3950e10-929a-4640-81a8-78d7cfd2f3f6.cee7ebe3-e994-4c64-a66a-2dcdeca57bd6"
}
```

上面的例子中，对卷*b3950e10-929a-4640-81a8-78d7cfd2f3f6*做了三次快照，每次快照都生成一个格式为*volume-id.snapshot_id*的文件，该文件

是卷的最新活动（*actice*）文件。即*volume-id.snapshot_id*文件表示快照*snapshot_id*创建完成后，对于*volume-id*的所有操作记录。

```
{
"active": "volume-id.snapshot_id_3",(* changed!)
 "snapshot_id_3": "volume-id.snapshot_id_3",
 "snapshot_id_2": "volume-id.snapshot_id_2",(* added!)
 "snapshot_id_1": "volume-id.snapshot_id_1", (* added!)
}
```

cinder允许对in-use状态下的卷进行快照，因此需要nova参与快照创建过程才能保证数据一致性：

***offline*状态下创建快照：**

***1.*读取*volume-<uuid>.info*文件，获取*active*的*image*文件（*volid-snapshot_id-active*）；**

***2.*使用*qemu-img info*命令获取*volid-snapshot_id-active*的镜像信息：**

```
image: volume-b3950e10-929a-4640-81a8-78d7cfd2f3f6.1f8766df-639f-4486-a7cf-602043bfde4f
file format: qcow2
virtual size: 5.0G (5368709120 bytes)
disk size: 200K
cluster_size: 65536
backing file: volume-b3950e10-929a-4640-81a8-78d7cfd2f3f6.cee7ebe3-e994-4c64-a66a-2dcdeca57bd6
backing file format: qcow2
Format specific information:
    compat: 1.1
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
```

***3.*根据上面的信息使用*qemu-img*命令创建快照（快照格式为*qcow2*）**

```
qemu-img create -f qcow2 -o backing_file=<base_file_name>,backing_fmt=<base_file_formate> <snapshot_name> <snapshot_size in GB>
```

4.指定<snapshot_name>文件的后端为当前active的镜像（这里使用非安全模式进行合并）

```
qemu-img rebase -u -b <base_file_name> -F <base_file_formate> <snapshot_name>
```

5.修改<snapshot_name>的文件权限为660

6.如果是安全模式（nas_secure_file_operations=true）修改<snapshot_name>文件的属组。

7.将快照信息写入*volume-<uuid>.info*文件

***offline*状态下删除快照：**

***1.*读取*snapshot_del* 和其*backend_file*的*image*信息；**

***2.*如果当前*active image==snapshot_del*，那么将*snapshot_del*的所有改动提交到他的*backend*，将*volume-<uuid>.info*文件中*active image*设为他的*backend*，并且删除*snapshot_del*文件**

```
qemu-img commit <snapshot_del_path>
```

3.如果*snapshot_del*是快照链中中间某个快照如： *snap_base --> snap_del --> snap_highter*

和步骤2相同，先将*snap_del*的内容提交到*snap_base*，然后修改*snap_highter*的*backend*地址为*snap_base*

```
qemu-img commit <snap_del_path>
qemu-img rebase -u b <snap_base_path> -F qcow2 <snap_highter_path>
```

***in-use*状态下创建快照：**

***1.in-use*状态下创建快照时，需要*VM*修改*XML*文件配置，将当前的*image*文件更新成新的*image-file*；**

整个过程*offline*状态相似，*cinder*通过*qemu-img*命令创建一个空的*snapshot*镜像文件，并将该文件告诉*nova*，*nova*修改他的*XML*文件，将当前虚机读写的镜像文件指向新的快照文件。

***in-use*状态下删除快照：**

和*offline*状态下的过程相似！！

克隆卷

接口如下：

```
cinder.volume.drivers.remotefs.RemoteFSSnapDriverBase
def _create_cloned_volume(self, volume, src_vref)
```

在NFS后端创建卷拷贝时，完成以下动作：

1.对source_volid创建一个快照；

2.使用qemu_img convert命令从快照拷贝数据，到volume对应的镜像；

3.删除创建的source_volid快照；