方案一：基于oslo_concurrency模块的工具lockutils

cinder/utils.py文件已经导入了oslo_concurrency模块的装饰器synchronized：

```
synchronized = lockutils.synchronized_with_prefix('cinder-')
```

使用例子：

```
from cinder import utils
```

```
@utils.synchronized('huawei_cinder_call')
def call(self, url, data=None, method=None, log_filter_flag=False):
    """pass"""
```

装饰器synchronized基于oslo_concurrency模块的加锁工具lockutils。

lockutils实现了线程锁和基于文件的进程锁

```
def synchronized(name, lock_file_prefix=None, external=False, lock_path=None,
                 semaphores=None, delay=0.01):
```

```
lock_file_prefix：lock_name的前缀
```

```
external ： False表示线程锁，True表示进程锁
```

```
lock_path ： 使用进程锁时需要提供锁文件的文件夹路径
```

```
semaphores ： 锁内部使用的信号量，主要用于线程锁，理论上使用线程竞争的是semaphores，因此使用不同的semaphores时，线程间不存在竞争关系。
            lockutils内部实现了一个semaphores。
```

配置文件信息

[oslo_concurrency]

#文件锁路径
lock_path = /opt/stack/data/cinder

#是否使用进程锁
disable_process_locking = False

方案二：基于Tooz模块进行加锁

cinder/coordination.py同样实现了OpenStack的Lock装饰器synchronized，该方案基于Tooz模块

```
def synchronized(lock_name, blocking=True, coordinator=None)
```

```
lock_name: 锁名称
```

```
blocking ： 为True时会一直等待，直到获得锁；False当无法获得锁时立即返回，并抛出异常
```

```
coordinator ： 锁的信号同步变量，默认情况下Tooz内部实现了一个全局coordinate。
```

使用方式：

1.通过函数装饰器

```
from cinder import coordination
@coordination.synchronized('{volume.id}-{f_name}')
```

```
def foo():
    pass
```

其中volume.id和f_name在装饰器中会自动被解析成ID和调用的方法名

2.通过with语句

```
from cinder import coordination
with coordination.Lock("LIN"):
    foo()
```

Tooz相对于第一种方法的优势在于能够通过扩展backend进化成一个分部式锁

默认情形下Tooz的backend对接的是一个文件夹，此时Tooz只能提供进程锁的能力。

```
cfg.StrOpt('backend_url',
           default='file://$state_path',
           help='The backend URL to use for distributed coordination.'),
```

Tooz支持不同的后端，可以通过配置backend_url参数对接不同后端

[coordination]

#对接mysql提供分布式锁

backend_url = mysql://root:stackdb@127.0.0.1/Tooz?charset=utf8

当backend为MySql时，coordination使用以下数据库方法获取/释放锁：

SELECT GET_LOCK(lock_name,TimeOut); #获取指定名称的锁，超时时间为TimeOut，成功获得锁时返回1，否者返回0，发生错误返回NULL。

SELECT RELEASE_LOCK(lock_name); #释放名为Lock_Name的锁，成功返回1，这里MySql提供的是可从入锁，GET_LOCK几次就要释放几次。

SELECT IS_FREE_LOCK(lock_name); 查询锁是否被占用，是返回1，否返回0。

参考：

http://blog.csdn.net/liunian0o0/article/details/54096208

https://docs.openstack.org/developer/tooz/

http://blog.csdn.net/allenson1/article/details/44539709