Cinder中的加锁机制

## 锁使用场景

**线程锁：**

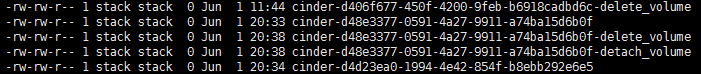
即在同一个进程中，对某段代码或者某个变量加锁，保证同一时间只有一个线程在操作他！但是，由于Python本身多线程支持比较弱，且有GIL的存在，因此在Cinder代码中出现的不多。

**进程锁：**

在同一个操作系统下运行的多个进程间如何协调访问共享资源。考虑有以下两个进程：process1和process2，process1正在从volume1拷贝数据，process2此时想要删除volume1。那么在该种情形下需要对volume1加锁，保证process1拷贝完成后，process2在进行删除。

由于不同的进程由于不共享内存空间，因此不能通过设置变量来实现。进程锁可以通过创建一个文件作为锁标记。进程只需要检查文件是否存在来判断是否有锁。

在Cinder-volume节点的$state-path路径下可以看到大量0字节的文件，这些文件就是用来加锁的。

**分布式锁：**

分布式锁的使用场景和进程锁相同，但是process1和process2运行在不同的主机上。参照进程锁的解决方案，显然需要有一个后端，process1和process2可以从这个后端获取锁信息。数据库、共享文件系统、Zookeeper都可以成为分布式锁的后端。

## 在Cinder中如何加锁

在Cinder中有现成的加锁工具，可以实现对资源或者代码块进行加锁。

**工具一：**基于oslo\_concurrency模块的工具lockutils，该工具不能提供分布式锁。

|  |
| --- |
| **python文件：**  cinder/utils.py  **工作原理：**  cinder/utils.py文件已经导入了oslo\_concurrency模块的装饰器synchronized。在该文件中可以看到synchronized对象，该对象oslo\_concurrency模块的装饰器。  **使用例子：**  **#导入cinder.utils模块**  **from cinder import utils**  **#使用utils.synchronized对需要装饰的方法加锁，下面的例子中'huawei\_cinder\_call'时锁的名字，可以替换成某种资源的ip**  **@utils.synchronized('huawei\_cinder\_call')**  **def call(self, url, data=None, method=None, log\_filter\_flag=False):**  **"""pass"""**  **装饰器synchronized参数说明：**  **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，默认使用CONF. oslo\_concurrency.lock\_path  **semaphores** ：锁内部使用的信号量，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。  **相关配置文件信息：**  [oslo\_concurrency]  #文件锁路径  lock\_path = /opt/stack/data/cinder  #是否使用进程锁  disable\_process\_locking = False |

工具二：基于Tooz模块进行加锁

cinder/coordination.py实现了OpenStack的Lock装饰器synchronized，该方案基于Tooz模块，可以提供分布式锁。但是默认情形下提供的是进程锁。

|  |
| --- |
| python文件：  cinder/coordination.py  使用例子：  1.通过函数装饰器  **from cinder import coordination**  **@coordination.synchronized('{volume.id}-{f\_name}')**  **def foo(volume):**  **pass**  其中volume.id和f\_name在装饰器中会自动被解析成入参volume的id属性，而f\_name被解析成调用的方法名，整个'{volume.id}-{f\_name}'时锁的名称。  2.通过with语句，创建一个Lock对象，这样可以保证with语句体独享整个锁。  **from cinder import coordination**  **with coordination.Lock("LIN"):**  **foo()**  **相关配置文件：**  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。通过backend\_url配置项，可以对接不同后端，使他成为一个分布式锁。  **默认配置：**  **[coordination]**  **backend\_url =file://$state\_path**  **如果对接mysql：**  **[coordination]**  **backend\_url = mysql://<user\_name>:<passwd>@<ip\_addr>/<db\_name>?charset=utf8** |

## Tooz对接Mysql时的工作原理

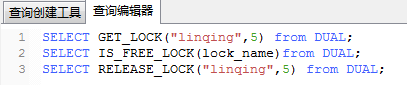
当backend为MySql时，coordination使用以下数据库方法获取/释放锁：

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。

上述三个命令都可以在SQL中直接进行：



参考：

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

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

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