# Cinder 业务代码里如何控制配额

1. QUOTAS.reserve生成配额
2. QUOTAS.commit 提交配额更新使用量
3. 业务过程异常，QUOTAS.rollback 回退配额

范例.cinder创建备份的时候，修改配额使用量：

***cinder.backup.api.API#create：***

from cinder import quota

from cinder import quota\_utils

def **create**(**self**, **context**, **name**, description, volume\_id,

**container**, incremental=False, availability\_zone=**None**,

**force**=False, snapshot\_id=**None**):

"""Make the RPC call to create a volume backup."""

<!*--省略业务代码-->*

# Reserve a **quota** **before** setting volume **status** **and** **backup** **status**

try:

# 创建配额预留

reserve\_opts = {'backups': 1,

'backup\_gigabytes': volume['size']}

reservations = QUOTAS.reserve(**context**, \*\*reserve\_opts)

**except** **exception**.OverQuota **as** **e**:

quota\_utils.process\_reserve\_over\_quota(

**context**, **e**,

**resource**='backups',

**size**=volume.**size**)

try:

<!*--省略业务代码-->*

# 提交配额预留，修改quota\_usage里的使用量

QUOTAS.**commit**(**context**, reservations)

**except** **Exception**:

# 延迟抛出异常，先把创建了一半的**backup**先销毁了，并回退配额，然后再抛出异常

**with** excutils.save\_and\_reraise\_exception():

try:

**if** **backup** **and** 'id' **in** **backup**:

**backup**.destroy()

finally:

# 回退预留

QUOTAS.**rollback**(**context**, reservations)

# TODO(DuncanT): **In** future, **when** we have a generic **local** attach,

# this can **go** via the scheduler, which enables

# better **load** balancing **and** **isolation** **of** services

**self**.backup\_rpcapi.create\_backup(**context**, **backup**)

**return** **backup**

## 补充知识：

### save\_and\_reraise\_exception介绍

有的时候突然抛出异常导致我们有些希望执行的关键程序没能执行，我们希望让异常延迟一些时候执行，好让我们关键程序跑完。oslo\_utils工具包提供了save\_and\_reraise\_exception这个工具类，很好的实现了这个功能。

***oslo\_utils.excutils.save\_and\_reraise\_exception***源码:

**class** **save\_and\_reraise\_exception**(object):

"""Save current exception, run some code and then re-raise.

In some cases the exception context can be cleared, resulting in None

being attempted to be re-raised after an exception handler is run. This

can happen when eventlet switches greenthreads or when running an

exception handler, code raises and catches an exception. In both

cases the exception context will be cleared.

To work around this, we save the exception state, run handler code, and

then re-raise the original exception. If another exception occurs, the

saved exception is logged and the new exception is re-raised.

In some cases the caller may not want to re-raise the exception, and

for those circumstances this context provides a reraise flag that

can be used to suppress the exception. For example::

except Exception:

with save\_and\_reraise\_exception() as ctxt:

decide\_if\_need\_reraise()

if not should\_be\_reraised:

ctxt.reraise = False

If another exception occurs and reraise flag is False,

the saved exception will not be logged.

If the caller wants to raise new exception during exception handling

he/she sets reraise to False initially with an ability to set it back to

True if needed::

except Exception:

with save\_and\_reraise\_exception(reraise=False) as ctxt:

[if statements to determine whether to raise a new exception]

# Not raising a new exception, so reraise

ctxt.reraise = True

.. versionchanged:: 1.4阿

Added \*logger\* optional parameter.

"""

**def** **\_\_init\_\_**(self, reraise=True, logger=None):

self.reraise = reraise

**if** logger **is** **None**:

logger = logging.getLogger()

self.logger = logger

self.type\_, self.value, self.tb = (**None**, **None**, **None**)

**def** **force\_reraise**(self):

*# 重新抛出异常*

**if** self.type\_ **is** **None** **and** self.value **is** **None**:

**raise** RuntimeError("There is no (currently) captured exception"

" to force the reraising of")

six.reraise(self.type\_, self.value, self.tb)

**def** **capture**(self, check=True):

*# 抓取异常，并暂存信息*

(type\_, value, tb) = sys.exc\_info()

**if** check **and** type\_ **is** **None** **and** value **is** **None**:

**raise** RuntimeError("There is no active exception to capture")

self.type\_, self.value, self.tb = (type\_, value, tb)

**return** self

**def** **\_\_enter\_\_**(self):

*# with 语句的入口函数*

*# TODO(harlowja): perhaps someday in the future turn check here*

*# to true, because that is likely the desired intention, and doing*

*# so ensures that people are actually using this correctly.*

**return** self.capture(check=**False**)

**def** **\_\_exit\_\_**(self, exc\_type, exc\_val, exc\_tb):

*# with 语句的出口函数， 这里调用force\_reraise做重新抛出异常*

**if** exc\_type **is** **not** **None**:

**if** self.reraise:

self.logger.error(\_LE('Original exception being dropped: %s'),

traceback.format\_exception(self.type\_,

self.value,

self.tb))

**return** **False**

**if** self.reraise:

self.force\_reraise()

### 测试demo：

from oslo\_utils import excutils

def **test1**():

a = [1,2,3,4]

try:

print '11111'

print a[10]

print '22222'

except:

with excutils.**save\_and\_reraise\_exception**():

print '3333'

**if** \_\_name\_\_ == '\_\_main\_\_':

**test1**()

C:\Python27\python.exe D:/wangyueWorkspace/mytest/olsoutils/test1.py

Traceback (most recent call last):

File "D:/wangyueWorkspace/mytest/olsoutils/test1.py", line 18, **in** <module>

**test1**()

File "D:/wangyueWorkspace/mytest/olsoutils/test1.py", line 13, **in** test1

print a

File "C:\Python27\lib\site-packages\oslo\_utils\excutils.py", line 220, **in** \_\_exit\_\_

self.**force\_reraise**()

File "C:\Python27\lib\site-packages\oslo\_utils\excutils.py", line 196, **in** force\_reraise

six.**reraise**(self.type\_, self.value, self.tb)

File "D:/wangyueWorkspace/mytest/olsoutils/test1.py", line 7, **in** test1

print a[10]

IndexError: list index out of range

11111

3333

Process finished with exit code 1

说明： 我定义a 数组有4个元素，所以在执行到print a[10]的时候会抛出下标越界的异常IndexError。捕获到这个异常后excutils.save\_and\_reraise\_exception() 会把这个异常的一些信息，包括异常名字、异常消息、堆栈信息先暂存下来，然后运行print '3333'，当with下的程序都执行完了，excutils.save\_and\_reraise\_exception()才把之前暂存的错误重新抛出。

我们看看cinder里是怎么利用这个工具类的。cinder backup-create的代码里，创建过程中如果出错，就用save\_and\_reraise\_exception做延迟抛出：

***cinder.backup.api.API#create：***

**try**:

<!*--创建buckup的业务代码-->*

except Exception:

with excutils.save\_and\_reraise\_exception():

**try**:

*# 销毁掉已经创建一半的backup*

**if** backup and 'id' in backup:

backup.destroy()

**finally**:

*# 回退掉配额占用*

QUOTAS.rollback(context, reservations)

所以excutils.save\_and\_reraise\_exception()很适用于事务处理。