OpenStack进行单元测试

# 使用tox

## 构建单元测试的环境

进入tox.ini文件所在目录，执行“tox –e py27”。tox会在当前文件下创建一个.tox目录，并在该目录下创建运行单元测试的虚拟环境。

在tox.ini中定义了众多测试环境，其中py27是默认测试环境，对应的配置Section也是默认的Section。

[tox]  
minversion = 2.0  
skipsdist = True  
envlist = py35,py27,compliance,pep8 # Cinder配置的测试环境

#测试环境的配置项的默认Section

[testenv]

setenv = VIRTUAL\_ENV={envdir}

PYTHONHASHSEED=0

PYTHONWARNINGS=default::DeprecationWarning

# usedevelop表示安装virtualenv的时候，本项目自己的代码采用开发模式安装，也就是不会拷贝代码到virtualenv目录中，只是做个链接。

usedevelop = True

# install\_command表示构建环境的时候要执行的命令，一般是使用pip安装。install\_command = pip install -U {opts} {packages}

install\_command = pip install -c{env:UPPER\_CONSTRAINTS\_FILE:https://git.openstack.org/cgit/openstack/requirements/plain/upper-constraints.txt?h=stable/pike} {opts} {packages}

deps = -r{toxinidir}/test-requirements.txt

# 当构建环境完成之后通过，ostestr命令运行单元测试

commands =

find . -type f -name "\*.pyc" -delete

ostestr {posargs}

whitelist\_externals =

bash

find

passenv = \*\_proxy \*\_PROXY

tox –e py27命令包含了以下过程：通过pip构建虚拟环境（对应的配置项install\_command），之后通过ostestr运行所有测试用例（对应的配置项commands）。

# FAQ

## 关于单元测试代码的目录

默认情形下创建的venvd的测试代码路径为：“./cinder/tests/unit”。

想要修改测试代码路径有以下两总方法：

1. 修改测试配置文件：test\_command参数中的${OS\_TEST\_PATH:-./cinder/tests/unit}
2. 创建一个新的测试虚拟环境，指定setenv参数OS\_TEST\_PATH。

[testenv:ruijie]

setenv =

OS\_TEST\_PATH = ./cinder/ruijie\_cinder/tests/unit

注意：可以通过export命令注入环境变量：

export OS\_TEST\_PATH=./cinder/ruijie\_cinder/tests/unit

## mock.patch的参数顺序

mock.patch装饰的方法在test\_XXX中的入参是逆序的，如下：

@mock.patch("cinder.function\_1 ")

@mock.patch("cinder.function\_2 ")

def test\_get\_volume\_type\_by\_host(self, \_mock\_ function\_2,\_mock\_ function\_1):

self.assertEqual(True, True)

## 进入控制台执行指定的单元测试

# 执行

source .tox/<虚拟环境目录> /bin/activate

# 执行单个单元测试用例

python –m testtools.run <test\_units\_modules>

## 连接数据库

* **使用sqlite**

sqlite是进程内存数据库，使用时无需进行相关的配置。

按照OpenStack规范，单元测试中需要查询数据库时，一般遵循“创建-使用”的顺序。即在setUp中先创建数据库记录，然后在test\_XXX中对数据进行修改、创建。

cinder/tests/unit/utils.py提供了一些创建数据库记录的现成方法，可以直接调用。

* **连接环境的MySql数据库**

通过cinder.test.Database可以在单元测试时访问数据库，默认使用的是Sqlite。修改测试代码，可以连接到Cinder环境的MySql数据库。

修改cinder.test.Database类：

class Database(fixtures.Fixture):  
 def \_\_init\_\_(self, db\_api, db\_migrate, sql\_connection):  
 # self.\_DB = "".join(line for line in conn.connection.iterdump())  
 def setUp(self):  
 # conn.connection.executescript(self.\_DB)

在单元测试文件中替换\_DB\_CACHE对象：

from oslo\_config import cfg  
from cinder.db import migration  
from cinder.db.sqlalchemy import api as sqla\_api  
  
  
DB\_URL = r"mysql+pymysql://root:123@127.0.0.1/cinder?charset=utf8"  
CONF = cfg.CONF  
CONF.set\_default('connection', DB\_URL, 'database')  
\_DB\_CACHE = test.Database(sqla\_api, migration, sql\_connection=CONF.database.connection)  
test.\_DB\_CACHE = \_DB\_CACHE

# 附录：单元测试工具

## unittest

unittest是Python的标准库，提供了最基本的单元测试功能，包括单元测试运行器（简称runner）和单元测试框架。

项目的单元测试代码的测试类可以继承unittest.TestCase类，这样这个类就能够被runner发现并且执行。unittest.TestCase中定义了setUp()，tearDown()，setUpClass()和tearDownClass()方法，用来运行单元测试前的设置工作代码和单元测试后的清理工作代码，所以第三方的单元测试库和框架也都遵循这个规范。

unittest库也提供了一个runner，可以使用$ python -m unittest test\_module的命令来执行某个模块的单元测试。另外，在Python中指定要运行的单元测试用例的完整语法是：path.to.your.module:ClassOfYourTest.test\_method。

## mock

mock模块在测试过程中，对于某些不容易构造或者不容易获取的对象，用一个虚拟的对象来创建以便测试的测试方法。

Python mock模块的简单说明：<http://www.cnblogs.com/fnng/p/5648247.html>

## testtools

testtools是个unittest的扩展框架，主要是在unittest的基础上提供了更好的assert功能，使得写单元测试更加方便。

## fixtures

fixture的意思是固定装置，在Python的单元测试中，是指某段可以复用的单元测试setUp和tearDown代码组合。一个fixture一般用来实现某个组件的setUp和tearDown逻辑，比如测试前要先创建好某些数据，测试后要删掉这些数据，这些操作就可以封装到一个fixture中。这样不同的测试用例就不用重复写这些代码，只要使用fixture即可。fixtures模块是一个第三方模块，提供了一种简单的创建fixture类和对象的机制，并且也提供了一些内置的fixture。

## testscenarios

testscenarios模块满足了场景测试的需求。它的基本用法是在测试类中添加一个类属性scenarios，该属性是一个元组，定义了每一种场景下不同的变量的值。比如说你测试一段数据访问代码，你需要测试该代码在使用不同的驱动时，比如MongoDB、SQL、File，是否都能正常工作。

testscenarios模块在OpenStack Ceilometer中被大量使用。

## subunit

subunit是一个用于传输单元测试结果的流协议。一般来说，运行单元测试的时候是把单元测试的结果直接输出到标准输出，但是如果运行大量的测试用例，这些测试结果就很难被分析。因此就可以使用python-subunit模块来运行测试用例，并且把测试用例通过subunit协议输出，这样测试结果就可以被分析工具聚合以及分析。python-subunit模块自带了一些工具用来解析subunit协议，可以这样运行测试用例：$ python -m subunit.run test\_module | subunit2pyunit，subunit2pyunit命令会解析subunit协议，并且输出到标准输出。

## testrepository

OpenStack中使用testrepository模块管理单元测试用例。testrepository使用python-subunit模块来运行测试用例，然后分析subunit的输出并对测试结果进行记录（记录到本地文件）。举例来说，testrepository可以统计：测试用例的运行时间、显示众多用例中的失败用例、重新运行失败用例、

## coverage

coverage是用来计算代码运行时的覆盖率的，也就是统计多少代码被执行了。它可以和testrepository一起使用，用来统计单元测试的覆盖率，在运行完单元测试之后，输出覆盖率报告。

## tox

tox是用来管理和构建虚拟环境(virtualenv)的。对于一个项目，我们需要运行Python 2.7的单元测试，也需要运行Python 3.4的单元测试，还需要运行PEP8的代码检查。这些不同的任务需要依赖不同的库，所以需要使用不同的虚拟环境。使用tox的时候，我们会在tox的配置文件**tox.ini**中指定不同任务的虚拟环境名称，该任务在虚拟环境中需要安装哪些包，以及该任务执行的时候需要运行哪些命令。

## ddt

数据驱动测试，即黑盒测试（Black-box Testing），又称为功能测试。在Cinder原生代码中不少单元测试代码使用了这个框架（可以当做功能测试代码用）。