目 录

1 学习 setup.cfg

1.1 基本术语

模块：python 中可复用的基本代码单元，可由其他代码 import。

纯 python 模块：由 python 编写的模块，包含在单独的 py 文件中。

扩展模块：由实现 python 的底层语言编写的模块，通常包含在单独的动态加载文件中。

包：包是含其他模块的模块，经常由包含 ___init___.py 文件的目录发布。

模块发布：一些 python 模块的集合，这些模块将被一起安装。

纯模块发布：一个只包含纯 python 模块和包的模块发布。

非纯模块发布：至少包含一个扩展模块的模块发布。

1.2 Distutils 简介

Distutils 可以用来在 python 环境中构建和安装额外的模块。

为了使用 Distutils，除了编写源码，还需要：

1. 编写 setup.py。
2. 编写 setup.cfg。
3. 创建一个源码分布。
4. 创建一个或多个二进制发布。

1.3 setup.cfg 格式

setup.cfg 是 Distutils 的配置文件，他的文件格式类似于：

```
1  [command]
2  option=value
3  ...
```

其中，command 是 Distutils 的命令参数，option 是相应的参数选项。

setup.py 文件是和 setup.cfg 配套存在的，可以通过 setup.py 查看 command 都有哪些参数选项：

```
1  python setup.py command --help
```

举个例子：

```
# setup.cfg文件的内容
[build]
build-base=blib
force=1

# 以上文件配置相当于下面的命令
python setup.py build --build-base=blib --force
```

setup.cfg 文件的内容由很多个 section 组成，比如 global、metadata、file 等，提供了软件包的名称、作者等信息。如果我们想去理解代码结构，只需要关注 [entry_points] 这一节。

1.3.1 entry_points

entry_points 提供了一个基于文件系统对象名的注册和 import 机制。

也就是说，entry_points 会把 python 对象和某个 name 关联起来，之后其他的代码只要使用这个 name 就可以找到对应的对象，而不用关心这个对象具体所在的位置。

举一个例子：

```
# 创建一个函数对象
def the_function():
    "function whose name is 'the_function'"
    print "hello from the_function"

# 这个函数对象所在的模块是myns.mypkg.mymodule.py
# entry_points就可以这么写
[entry_points]
# my_ep_group_id相当于一个entry_points组，可以包含多个entry_point
my_ep_group_id =
    my_ep_func = myns.mypkg.mymodule: the_function
```

然后通过“python setup.py install”安装以后，就可以通过“pkg_resources”去调用这些 entry_points：

```
import pkg_resources

named_objects = []
for ep in pkg_resources.iter_entry_points(group='my_ep_group_id'):
    named_objects.append(ep.load())

# 根据entry_points中mu_ep_group_id组的定义，可以知道named_objects[0]是
    the_function
named_objects[0]() # 输出“hello from the_function”
```

1.3.2 openstack 中使用 entry_points 的例子

以 Ceilometer 为例，它 setup.cfg 中一部分内容如下：

```
# ceilometer.compute.virt是一个entry_points组，包含了3个entry_point
ceilometer.compute.virt =
    libvirt = ceilometer.compute.virt.libvirt.inspector: LibvirtInspector
    hyperv = ceilometer.compute.virt.hyperv.inspector: HypervInspector
    vsphere = ceilometer.compute.virt.vmware.inspector: VsphereInspector
```

安装 Ceilometer 以后，其他程序可以利用下面几种方式调用这些 entry_point：

1. 使用 pkg_resources，通过 iter_entry_points 遍历获得这些 entry_point：

```
import pkg_resources
def run_entry_point(data):
    group = 'ceilometer.compute.virt'
    for entrypoint in pkg_resources.iter_entry_points(group=group):
        plugin = entrypoint.load()
        plugin(data) # plugin指向ceilometer.compute.virt中entry_point中注
            册的对象
```

2. 使用 pkg_resources，通过 load_entry_point 函数和 entry_point 的名称来获得注册的对象：

```
from pkg_resources import load_entry_point
fun = load_entry_point('ceilometer', 'ceilometer.compute.virt', 'libvirt')
# fun指向LibvirtInspector这个函数对象
```

3. 使用 stevedore，通过其中的 driver 类来获得注册的对象：

```
from stevedore import driver

def get_hypervisor_inspector():
    try:
        namespace = 'ceilometer.compute.virt'
        # cfg.CONF.hypervisor_inspector是oslo.config的一个配置选项
        mgr = driver.DriverManager(namespace, cfg.CONF.
            hypervisor_inspector, invoke_on_load=True)
        return mgr.driver()
    except ImportError as e:
        LOG.error(_("Unable ro load the hypervisor inspector: %s") % (e))
        return Inspector()
```