

# OpenStack Neutron 代码分析

---

最新版:

[https://github.com/yeasy/tech\\_writing/tree/master/OpenStack/OpenStack\\_HeatClient\\_代码分析.pdf](https://github.com/yeasy/tech_writing/tree/master/OpenStack/OpenStack_HeatClient_代码分析.pdf)

更新历史:

V0.1: 2014-08-07

完成代码基本结构。

## 目录

第 1 章 整体结构.....	1
1.1 doc.....	1
1.2 heatclient.....	1
1.3 tools.....	1
第 2 章 doc.....	2
第 3 章 tools.....	3
第 4 章 heatclient.....	4
4.1 common.....	4
4.1.1 environment_format.py.....	4
4.1.2 http.py.....	4
4.1.3 template_format.py.....	4
4.1.4 template_utils.py.....	4
4.1.5 utils.py.....	4
4.2 openstack.....	4
4.2.1 common.....	4
4.2.1.1 apiclient.....	4
4.2.1.2 cliutils.py.....	5
4.2.1.3 gettextutils.py.....	5
4.2.1.4 importutils.py.....	5
4.2.1.5 jsonutils.py.....	5

4.2.1.6 strutils.py.....	5
4.2.1.7 timeutils.py.....	5
4.2.1.8 uuidutils.py.....	5
4.3 tests.....	5
4.4 v1.....	5
4.4.1 client.py.....	5
4.4.2 shell.py.....	6
4.4.3 actions.py.....	6
4.4.4 build_info.py.....	6
4.4.5 events.py.....	6
4.4.6 resource_types.py.....	7
4.4.7 resources.py.....	7
4.4.8 software_configs.py.....	7
4.4.9 software_deployments.py.....	7
4.4.10 stacks.py.....	7
4.5 client.py.....	7
4.6 exc.py.....	7
4.7 shell.py.....	7

## 第 1 章 整体结构

源代码包 python-heatclient 主要分为 3 个目录和若干文件：

doc, heatclient 和 tools。

除了这 3 个目录外，还包括一些说明文档、安装需求说明文件和测试脚本等。

### 1.1 doc

包括文档生成的相关源码。

### 1.2 heatclient

核心的代码实现都在这个目录下。

可以通过下面的命令来统计主要实现的核心代码量。

```
find heatclient -name "*.py" | xargs cat | wc -l
```

目前版本，约为 10.7k 行。

### 1.3 tools

一些相关的环境安装和命令补全的脚本。

## 第 2 章 doc

可以利用 sphinx 工具来生成文档。

source 子目录：文档相关的代码。

Makefile：用户执行 make 命令。

用户在该目录下通过执行 make html 可以生成 html 格式的说明文档。

## 第 3 章 **tools**

heat.bash\_completion 是 bash 命令补全。

install\_venv.py 和 install\_venv\_common.py 用来创建虚拟 python 开发环境。

requirements\_style\_check.sh 检查输入文件中的依赖按照字母顺序进行排序。

with\_venv.sh 调用 tox 在给定的虚拟环境中执行命令。

## 第 4 章 heatclient

heatclient 主要把通过 heat 输入的命令转化为对 heat-engine 的 REST API 调用，起到一个翻译的作用，因此代码架构十分简单。命令执行的入口在 shell.py 文件。

### 4.1 common

#### 4.1.1 environment\_format.py

提供对字符串的格式检查等辅助操作。

#### 4.1.2 http.py

封装一个完整的 HTTPClient 类，支持 http 的 crud 操作。

#### 4.1.3 template\_format.py

定义对模板中字符串进行处理的方法。

#### 4.1.4 template\_utils.py

对模板进行操作的若干辅助方法，包括从文件中获取模板内容等。

#### 4.1.5 utils.py

一些辅助方法，包括查找资源、导入模块等。

### 4.2 openstack

#### 4.2.1 common

##### 4.2.1.1 apiclient

##### 4.2.1.1.1 auth.py

跟认证相关的方法。

##### 4.2.1.1.2 base.py

定义了若干基础类，包括 Resource 代表 OpenStack 中的资源类型（项目、用户等）、BaseManager 代表进行资源操作的基础类、Extension 代表扩展等。

##### 4.2.1.1.3 client.py

包括两个类 BaseClient 和 HTTPClient。

前者利用自身属性 http\_client 来发出各种 http 资源请求。

后者实现向 OpenStack 的服务发出 HTTP 请求。

#### 4.2.1.1.4 exceptions.py

定义了各种异常类型。

#### 4.2.1.1.5 fake\_client.py

仿造一个假的 server 能回复各种 client 的请求。

#### 4.2.1.2 cliutils.py

命令行辅助函数，包括在给定 manager 中查找资源，获取输入的密码等。

#### 4.2.1.3 gettextutils.py

国际化字符串相关方法。

#### 4.2.1.4 importutils.py

跟导入相关的方法，包括 import\_class、import\_module、import\_object、import\_object\_ns、import\_versioned\_module 和 try\_import。

这些方法会根据传入的参数和名称来导入指定的类、模块、对象等。

#### 4.2.1.5 jsonutils.py

对 json 进行操作，封装 dump 和 load 等。

#### 4.2.1.6 strutils.py

对字符串进行操作，包括转码等。

#### 4.2.1.7 timeutils.py

对时间进行比较，提取和格式化等。

#### 4.2.1.8 uuidutils.py

生成 uuid 和进行校验等。

### 4.3 tests

跟测试相关的文件，包括 fakes.py、test\_build\_info.py、test\_common\_http.py、test\_environment\_format.py、test\_events.py、test\_resource\_types.py、test\_resources.py、test\_shell.py、test\_software\_configs.py、test\_software\_deployments.py、test\_stacks.py、test\_template\_format.py、test\_template\_utils.py、test\_utils.py 等。

## 4.4 v1

版本 1 的包。其他模块可以通过不同版本号来选用某个版本下的实现。

#### 4.4.1 client.py

主要定义了 Client 类。代表跟 OpenStack 进行交互的客户端。

其中 httpclient 是发出 http 请求的代理，而其他的资源操作都是通过 httpclient 来进行的。

例如对 stack 资源进行操作的话，通过 self.stacks 进行。

```
class Client(object):
```



```

"""Client for the Heat v1 API.

:param string endpoint: A user-supplied endpoint URL for the heat
                        service.
:param string token: Token for authentication.
:param integer timeout: Allows customization of the timeout for client
                        http requests. (optional)
"""

def __init__(self, *args, **kwargs):
    """Initialize a new client for the Heat v1 API."""
    self.http_client = http.HTTPClient(*args, **kwargs)
    self.stacks = stacks.StackManager(self.http_client)
    self.resources = resources.ResourceManager(self.http_client)
    self.resource_types = resource_types.ResourceTypeManager(
        self.http_client)
    self.events = events.EventManager(self.http_client)
    self.actions = actions.ActionManager(self.http_client)
    self.build_info = build_info.BuildInfoManager(self.http_client)
    self.software_deployments = \
        software_deployments.SoftwareDeploymentManager(
            self.http_client)
    self.software_configs = software_configs.SoftwareConfigManager(
        self.http_client)

```

#### 4.4.2 shell.py

定义了一系列的 `do_xxx()` 方法，实现 heat 的各种命令操作，包括 `do_action_resume()`、`do_action_suspend()`、`do_build_info()`、`do_create()`、`do_delete()`、`do_describe()`、`do_event()`、`do_event_list()`、`do_event_show()`、`do_gettemplate()`、`do_list()`、`do_output_list()`、`do_output_show()`、`do_resource()`、`do_resource_list()`、`do_resource_metadata()`、`do_resource_show()`、`do_resource_signal()`、`do_resource_template()`、`do_resource_type_list()`、`do_resource_type_show()`、`do_resource_type_template()`、`do_stack_abandon()`、`do_stack_adopt()`、`do_stack_create()`、`do_stack_delete()`、`do_stack_list()`、`do_stack_preview()`、`do_stack_show()`、`do_stack_update()`、`do_template_show()`、`do_template_validate()`、`do_update()`、`do_validate()`等。

这些方法实际上进一步调用 Client 类中资源进行对应操作。

#### 4.4.3 actions.py

定义类 Action 和 ActionManager。

前者继承自 `base.Resource`，作为一种资源，支持更新、删除和获取数据操作。

后者继承自 `stacks.StackChildManager`，支持对 stack 进行暂停和继续操作。

#### 4.4.4 build\_info.py

定义类 `ActiBuildInfo` 和 `BuildInfoManager`。

前者继承自 `base.Resource`，作为一种资源。

后者继承自 `base.BaseManager`，支持获取 `buildinfo`。

#### 4.4.5 events.py

定义类 Event 和 EventManager。

前者继承自 base.Resource，作为一种资源，支持更新、删除和获取数据操作。

后者继承自 stacks.StackChildManager，支持对 event 进行列出和获取等操作。

#### 4.4.6 resource\_types.py

定义类 ResourceType 和 ResourceTypeManager。

前者继承自 base.Resource，作为一种资源。

后者继承自 base.BaseManager，支持获取某个特定资源类型的信息和生成模板操作。

#### 4.4.7 resources.py

定义类 Resource 和 ResourceManager。

前者继承自 base.Resource，支持更新、删除和获取数据操作。

后者继承自 stacks.StackChildManager，支持对资源进行列出、获取和发出信号等操作。

#### 4.4.8 software\_configs.py

定义类 SoftwareConfig 和 SoftwareConfigManager。

前者继承自 base.Resource，作为一种资源，支持删除和获取数据操作。

后者继承自 base.BaseManager，支持获取某个特定软件配置的信息和创建、删除等操作。

#### 4.4.9 software\_deployments.py

定义类 SoftwareDeployment 和 SoftwareDeploymentManager。

前者继承自 base.Resource，作为一种资源，支持更新和获取数据操作。

后者继承自 base.BaseManager，支持对软件部署的 CRUD 和获取 metadata 等操作。

#### 4.4.10 stacks.py

主要实现三个类：Stack、StackChildManager 和 StackManager。

其中 Stack 类继承自基础资源类 base.Resource，支持调用自身的 manager 来进行 CRUD 等操作。

StackManager 继承自 base.BaseManager 类，实现对 stack 资源进行操作，调用其中的 httpclient 来发出对应的 http request。包括对 stack 进行 CRUD 等操作。

### 4.5 client.py

定义 Client 类，作为对外的统一封装。

实际上会找到指定版本下的 client 模块中的 Client 类。

以 v1 为例，会调用 heatclient.v1.client 模块中的 Client 类。

### 4.6 exc.py

定义一些异常和错误类型。包括 HTTPBadGateway、HTTPBadRequest、HTTPConflict、HTTPException 等。

### 4.7 shell.py

当执行 heat 命令的时候，实际上就是导入了本文件。

查看/usr/bin/heat 命令的实现代码

```
import sys

from heatclient.shell import main

if __name__ == "__main__":
    sys.exit(main())
```

而在本文件中，主要是实现了 HeatShell 类，通过 main 函数调用了 HeatShell 类中的 main 函数。

该 main 函数代码主要包括三部分。

第一步是注册各种参数和选项。通过下面的代码，先注册基本的命令项，获取通过命令行传入的参数，简单解析一些已知参数，之后通过调用 get\_subcommand\_parser()方法来注册预置选项和命令。

```
parser = self.get_base_parser()
(options, args) = parser.parse_known_args(argv)
self._setup_logging(options.debug)
self._setup_verbose(options.verbose)

# build available subcommands based on version
api_version = options.heat_api_version
subcommand_parser = self.get_subcommand_parser(api_version)
self.parser = subcommand_parser
```

查看 get\_subcommand\_parser()方法的代码

```
def get_subcommand_parser(self, version):
    parser = self.get_base_parser()

    self.subcommands = {}
    subparsers = parser.add_subparsers(metavar='<subcommand>')
    submodule = utils.import_versioned_module(version, 'shell')
    self._find_actions(subparsers, submodule)
    self._find_actions(subparsers, self)
    self._add_bash_completion_subparser(subparsers)

    return parser
```

其中，submodule 实际上指向给定 version 包下的 shell 模块，如果为 v1，则完整路径则为 heatclient.v1.shell 模块。查看该模块，其中定义了大量的 do\_xxx 方法，例如 do\_create(), do\_stack\_create(), do\_stack\_delete()等等。这些方法实际上就对应了相应 heat 命令的调用。在 self 即本模块中也分别定义了 do\_bash\_completion 和 do\_help 两个方法。

\_find\_actions()方法在给定的模块中寻找类似 do\_stack\_create()格式的方法，并将“do\_”前缀去掉，下划线改为中划线，即完成 do\_stack\_create()-->stack-create 命令的转换。最终在 subparsers 中注册上类似 stack-create 的各项命令，以及他们对应的参数、描述和默认的处理函数（例如 do\_stack\_create()）。\_find\_actions()方法代码为：

```
def _find_actions(self, subparsers, actions_module):
    for attr in (a for a in dir(actions_module) if a.startswith('do_')):
        # I prefer to be hyphen-separated instead of underscores.
```

```

command = attr[3:].replace('_', '-')
callback = getattr(actions_module, attr)
desc = callback.__doc__ or ""
help = desc.strip().split('\n')[0]
arguments = getattr(callback, 'arguments', [])

subparser = subparsers.add_parser(command,
                                   help=help,
                                   description=desc,
                                   add_help=False,
                                   formatter_class=HelpFormatter)
subparser.add_argument('-h', '--help',
                      action='help',
                      help=argparse.SUPPRESS)
self.subcommands[command] = subparser
for (args, kwargs) in arguments:
    subparser.add_argument(*args, **kwargs)
subparser.set_defaults(func=callback)

```

第二步是对输入参数进行处理，首先调用

```
args = subcommand_parser.parse_args(argv)
```

利用前面注册的子命令来进行判断和解析。之后根据解析的结果，判断是否是进行补全操作；并检查相应的用户名、口令等信息是否存在，并将这些信息都打包到一起。

最后一步，是创建一个 `client`，并将前面打包好的参数和命令一起执行。

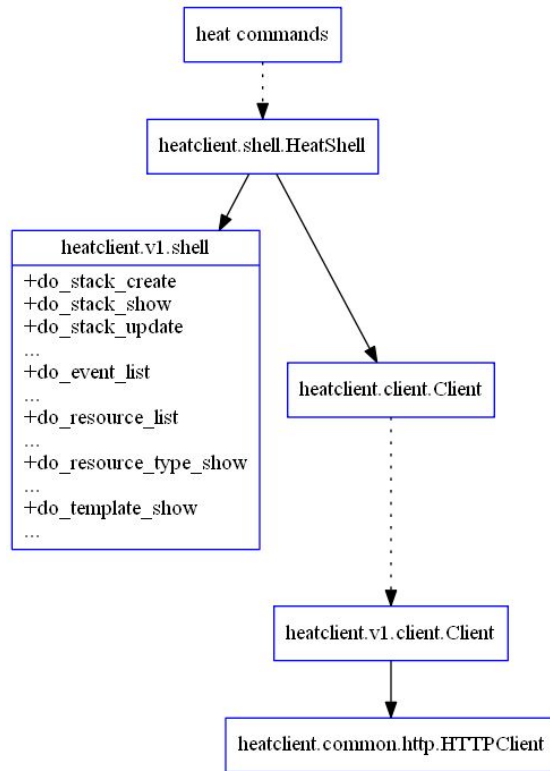
```

client = heat_client.Client(api_version, endpoint, **kwargs)
args.func(client, args)

```

其中 `client` 实际上调用对应版本包下的 `client` 模块中的 `Client` 类。以 `v1` 为例，则为 `heatclient.v1.client.Client`。该客户端中包括能跟 `openstack` 进行交互的相关句柄信息。

整体的调用过程如图表 1 所示。



图表 1 整体调用过程