

OpenStack Heat 代码分析

最新版:

[https://github.com/yeasy/tech_writing/tree/master/OpenStack/OpenStack Heat 代码分析.pdf](https://github.com/yeasy/tech_writing/tree/master/OpenStack/OpenStack%20Heat%E4%BB%A3%E7%A0%B4%E5%8F%86%E6%9E%99.pdf)

更新历史:

V0.2: 2014-08-18

完成对整体代码的分析。

V0.1: 2014-08-09

完成代码基本结构。

目录

第 1 章 整体结构.....	1
1.1 bin.....	1
1.2 contrib.....	1
1.3 doc.....	1
1.4 etc.....	1
1.5 heat.....	1
1.6 tools.....	1
第 2 章 bin.....	2
2.1 heat-api.....	2
2.2 heat-api-cfn.....	2
2.3 heat-api-cloudwatch.....	2
2.4 heat-engine.....	2
2.5 heat-keystone-setup-domain.....	3
2.6 heat-manage.....	3
2.7 cinder-keystone-setup.....	3
2.8 heat-db-setup.....	3
2.9 heat-keystone-setup.....	3
第 3 章 contrib.....	4
3.1 barbican.....	4
3.2 docker.....	4

3.3 extraroute.....	4
3.4 heat_keystoneclient_v2.....	4
3.5 marconi.....	4
3.6 nova_flavor.....	4
3.7 rackspace.....	4
第 4 章 doc.....	5
第 5 章 etc.....	6
5.1 environment.d.....	6
5.2 templates.....	6
5.3 api-paste.ini.....	6
5.4 heat.conf.sample.....	8
5.5 policy.json.....	8
第 6 章 heat.....	9
6.1 api.....	9
6.1.1 aws.....	9
6.1.1.1 ec2token.py.....	9
6.1.1.2 exception.py.....	9
6.1.1.3 utils.py.....	9
6.1.2 cfn.....	9
6.1.2.1 v1.....	9

6.1.2.2 versions.py.....	9
6.1.3 cloudwatch.....	9
6.1.3.1 __init__.py.....	9
6.1.3.2 versions.py.....	9
6.1.3.3 watch.py.....	9
6.1.4 middleware.....	9
6.1.4.1 fault.py.....	10
6.1.4.2 ssl.py.....	10
6.1.4.3 version_negotiation.py.....	10
6.1.5 openstack.....	10
6.1.5.1 v1.....	10
6.1.5.2 versions.py.....	11
6.2 cloudinit.....	11
6.2.1 loguserdata.py.....	11
6.2.2 part_handler.py.....	11
6.2.3 boothook.sh.....	11
6.2.4 config.....	11
6.3 cmd.....	11
6.3.1 manager.py.....	11
6.4 common.....	11

6.4.1 auth_password.py.....	11
6.4.2 auth_url.py.....	11
6.4.3 config.py.....	12
6.4.4 context.py.....	12
6.4.5 crypt.py.....	12
6.4.6 custom_backend_auth.py.....	12
6.4.7 environment_format.py.....	12
6.4.8 exception.py.....	12
6.4.9 heat_keystoneclient.py.....	12
6.4.10 identifier.py.....	12
6.4.11 messaging.py.....	12
6.4.12 param_utils.py.....	12
6.4.13 plugin_loader.py.....	12
6.4.14 policy.py.....	12
6.4.15 serializers.py.....	12
6.4.16 short_id.py.....	12
6.4.17 template_format.py.....	12
6.4.18 timeutils.py.....	13
6.4.19 urlfetch.py.....	13
6.4.20 wsgi.py.....	13

6.5 db.....	13
6.5.1 sqlalchemy.....	13
6.5.2 api.py.....	13
6.5.3 sync.py.....	13
6.5.4 utils.py.....	13
6.6 doc.....	13
6.7 engine.....	13
6.7.1 cfn.....	13
6.7.1.1 functions.py.....	13
6.7.1.2 template.py.....	13
6.7.2 hot.....	13
6.7.2.1 functions.py.....	13
6.7.2.2 parameters.py.....	13
6.7.2.3 template.py.....	14
6.7.3 clients.....	14
6.7.3.1 os.....	14
6.7.3.2 __init__.py.....	14
6.7.3.3 client_plugin.py.....	14
6.7.4 notification.....	14
6.7.4.1 __init__.py.....	14

6.7.4.2 autoscaling.py.....	14
6.7.4.3 stack.py.....	14
6.7.5 resources.....	14
6.7.5.1 __init__.py.....	14
6.7.5.2 ceilometer.....	15
6.7.5.3 neutron.....	15
6.7.5.4 software_config.....	15
6.7.5.5 autoscaling.py.....	15
6.7.5.6 cloud_watch.py.....	15
6.7.5.7 eip.py.....	15
6.7.5.8 glance_image.py.....	15
6.7.5.9 instance.py.....	15
6.7.5.10 internet_gateway.py.....	15
6.7.5.11 loadbalancer.py.....	15
6.7.5.12 network_interface.py.....	15
6.7.5.13 nova_floatingip.py.....	15
6.7.5.14 nova_keypair.py.....	15
6.7.5.15 nova_servergroup.py.....	15
6.7.5.16 os_database.py.....	16
6.7.5.17 random_string.py.....	16

6.7.5.18 resource_group.py.....	16
6.7.5.19 route_table.py.....	16
6.7.5.20 s3.py.....	16
6.7.5.21 security_group.py.....	16
6.7.5.22 server.py.....	16
6.7.5.23 stack.py.....	16
6.7.5.24 subnet.py.....	16
6.7.5.25 swift.py.....	16
6.7.5.26 user.py.....	16
6.7.5.27 volume.py.....	16
6.7.5.28 vpc.py.....	16
6.7.5.29 wait_condition.py.....	16
6.7.5.30 iso_8601.py.....	17
6.7.5.31 nova_utils.py.....	17
6.7.5.32 template_resource.py.....	17
6.7.6 api.py.....	17
6.7.7 attributes.py.....	17
6.7.8 constraints.py.....	17
6.7.9 dependencies.py.....	17
6.7.10 environment.py.....	17

6.7.11 event.py.....	17
6.7.12 function.py.....	17
6.7.13 parameter_groups.py.....	17
6.7.14 parameters.py.....	17
6.7.15 parser.py.....	17
6.7.16 plugin_manager.py.....	17
6.7.17 properties.py.....	18
6.7.18 resource.py.....	18
6.7.19 rsrc_defn.py.....	18
6.7.20 scheduler.py.....	18
6.7.21 service.py.....	18
6.7.22 signal_responder.py.....	19
6.7.23 stack.py.....	19
6.7.24 stack_lock.py.....	19
6.7.25 stack_resource.py.....	19
6.7.26 stack_user.py.....	19
6.7.27 support.py.....	19
6.7.28 template.py.....	19
6.7.29 timestamp.py.....	19
6.7.30 update.py.....	19

6.7.31 watchrule.py.....	19
6.8 locale.....	19
6.9 Openstack.....	19
6.9.1 Common.....	20
6.9.1.1 Config.....	20
6.9.1.2 Crypto.....	20
6.9.1.3 Middleware.....	20
6.9.1.4 Context.py.....	20
6.9.1.5 eventlet_backdoor.py.....	20
6.9.1.6 excutils.py.....	20
6.9.1.7 fileutils.py.....	20
6.9.1.8 gettextutils.py.....	20
6.9.1.9 importutils.py.....	20
6.9.1.10 jsonutils.py.....	20
6.9.1.11 local.py.....	20
6.9.1.12 lockutils.py.....	20
6.9.1.13 log.py.....	20
6.9.1.14 loopingcall.py.....	21
6.9.1.15 network_utils.py.....	21
6.9.1.16 policy.py.....	21

6.9.1.17 processutils.py.....	21
6.9.1.18 service.py.....	21
6.9.1.19 sslutils.py.....	21
6.9.1.20 strutils.py.....	21
6.9.1.21 systemd.py.....	21
6.9.1.22 threadgroup.py.....	21
6.9.1.23 timeutils.py.....	21
6.9.1.24 uuidutils.py.....	21
6.9.1.25 versionutils.py.....	21
6.10 rpc.....	21
6.10.1 api.py.....	21
6.10.2 client.py.....	21
6.11 scaling.....	22
6.11.1 template.py.....	22
6.12 tests.....	22
6.13 __init__.py.....	22
6.14 version.py.....	22
第 7 章 rally-scenarios.....	22
第 8 章 tools.....	23
8.1 config.....	24

8.2 其它.....	24
8.3 清理.....	24
第 9 章 理解代码.....	24
9.1 调用逻辑.....	25

第 1 章 整体结构

源代码主要分为 6 个目录和若干文件：

bin, contrib, doc, etc, neutron 和 tools。

除了这 6 个目录外，还包括一些说明文档、安装需求说明文件等。

1.1 bin

主要包括 heat-api、heat-api-cfn、heat-api-cloudwatch、heat-engine、heat-manage。

这些是相应服务的主文件。

1.2 contrib

包括若干第三方的插件，包括 docker、extraroute、keystoneclient_v2、marconi 和 rackspace 等。

1.3 doc

包括文档生成的相关源码。

1.4 etc

跟服务和配置相关的文件，基本上该目录中内容在安装时会被复制到系统的/etc/目录下。

environment.d/default.yaml 文件。

templates 目录下有两个 yaml 格式的模板文件。

api-paste.ini，启动 heat-api 服务时候的 paste-deploy 配置文件。

1.5 heat

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

目前版本，约为 107k 行。

1.6 tools

一些相关的代码格式化检测、环境安装的脚本。

第 2 章 bin

2.1 heat-api

启动 heat-api 服务的主文件。主要代码为

```
cfg.CONF(project='heat', prog='heat-api')
cfg.CONF.default_log_levels = ['amqpplib=WARN',
                               'qpid.messaging=INFO',
                               'keystone=INFO',
                               'eventlet.wsgi.server=WARN',
                               ]
logging.setup('heat')

app = config.load_paste_app()

port = cfg.CONF.heat_api.bind_port
host = cfg.CONF.heat_api.bind_host
LOG.info('Starting Heat ReST API on %s:%s' % (host, port))
server = wsgi.Server()
server.start(app, cfg.CONF.heat_api, default_port=port)
notify.startup_notify(cfg.CONF.onready)
server.wait()
```

可见，其主要过程为，从 api-paste.ini 文件中读取应用并加载，之后启动一个 wsgi 服务，上面运行配置好的应用，并根据配置文件指定，发出启动完成的通知。

2.2 heat-api-cfn

类似的，该文件启动 heat-api-cfn 服务。

2.3 heat-api-cloudwatch

类似的，该文件启动 heat-api-cloudwatch 服务，为 heat cloudwatch 提供 API 服务。

2.4 heat-engine

启动 heat-engine 服务，其过程如下

```
cfg.CONF(project='heat', prog='heat-engine')
logging.setup('heat')
messaging.setup()

from heat.engine import service as engine

srv = engine.EngineService(cfg.CONF.host, rpc_api.ENGINE_TOPIC)
launcher = service.launch(srv, workers=cfg.CONF.num_engine_workers)
# We create the periodic tasks here, which mean they are created
# only in the parent process when num_engine_workers>1 is specified
srv.create_periodic_tasks()
notify.startup_notify(cfg.CONF.onready)
```

```
launcher.wait()
```

其主要通过调用 `engine.EngineService` 类来实现应用，用 `service` 包来管理服务。

2.5 heat-keystone-setup-domain

配置 `heat domain`。

2.6 heat-manage

调用 `heat.cmd.manage` 包中的 `main` 方法。响应 `heat-manage` 命令。

2.7 cinder-keystone-setup

`cinder` 相关认证信息。

2.8 heat-db-setup

shell 脚本，安装和启动一个 `mysql` 服务器来作为 `heat` 的数据库后端。

2.9 heat-keystone-setup

shell 脚本，配置 `heat` 在 `keystone` 中的用户信息。

第 3 章 contrib

3.1 barbican

3.2 docker

3.3 extraroute

3.4 heat_keystoneclient_v2

3.5 marconi

3.6 nova_flavor

3.7 rackspace

第 4 章 **doc**

可以利用 `sphinx` 工具来生成文档。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。

source 子目录：文档相关的代码。

Makefile：用户执行 `make` 命令。

docbkx 目录：`api` 参考和 `heat` 管理文档。

第 5 章 etc

5.1 environment.d

5.2 templates

5.3 api-paste.ini

定义了 WSGI 应用和路由信息。利用 Paste 来实例化 Neutron 的 APIRouter 类，将资源（端口、网络、子网）映射到 URL 上，以及各个资源的控制器。

在 neutron-server 启动的时候，一般会指定参数 `--config-file neutron.conf --config-file xxx.ini`。看 `neutron/server/__init__.py` 的代码: `main()` 主程序中会调用 `config.parse(sys.argv[1:])` 来读取这些配置文件中的信息。而 `api-paste.ini` 信息中定义了 `neutron`、`neutronapi_v2_0`、若干 `filter` 和两个 `app`。

```
pipeline for custom cloud backends
# i.e. in heat.conf:
# [paste_deploy]
# flavor = custombackend
#
[pipeline:heat-api-custombackend]
pipeline = request_id faultwrap versionnegotiation context custombackendauth apiv1app

# heat-api-cfn pipeline
[pipeline:heat-api-cfn]
pipeline = cfnversionnegotiation ec2authtoken authtoken context apicfnv1app

# heat-api-cfn pipeline for standalone heat
# relies exclusively on authenticating with ec2 signed requests
[pipeline:heat-api-cfn-standalone]
pipeline = cfnversionnegotiation ec2authtoken context apicfnv1app

# heat-api-cloudwatch pipeline
[pipeline:heat-api-cloudwatch]
pipeline = versionnegotiation ec2authtoken authtoken context apicwapp

# heat-api-cloudwatch pipeline for standalone heat
# relies exclusively on authenticating with ec2 signed requests
[pipeline:heat-api-cloudwatch-standalone]
pipeline = versionnegotiation ec2authtoken context apicwapp

[app:apiv1app]
paste.app_factory = heat.common.wsgi:app_factory
heat.app_factory = heat.api.openstack.v1:API

[app:apicfnv1app]
```

```

paste.app_factory = heat.common.wsgi:app_factory
heat.app_factory = heat.api.cfn.v2:API

[app:apicwapp]
paste.app_factory = heat.common.wsgi:app_factory
heat.app_factory = heat.api.cloudwatch:API

[filter:versionnegotiation]
paste.filter_factory = heat.common.wsgi:filter_factory
heat.filter_factory = heat.api.openstack:version_negotiation_filter

[filter:faultwrap]
paste.filter_factory = heat.common.wsgi:filter_factory
heat.filter_factory = heat.api.openstack:faultwrap_filter

[filter:cfnversionnegotiation]
paste.filter_factory = heat.common.wsgi:filter_factory
heat.filter_factory = heat.api.cfn:version_negotiation_filter

[filter:cwversionnegotiation]
paste.filter_factory = heat.common.wsgi:filter_factory
heat.filter_factory = heat.api.cloudwatch:version_negotiation_filter

[filter:context]
paste.filter_factory = heat.common.context:ContextMiddleware_filter_factory

[filter:ec2authtoken]
paste.filter_factory = heat.api.aws.ec2token:EC2Token_filter_factory

[filter:ssl]
paste.filter_factory = heat.common.wsgi:filter_factory
heat.filter_factory = heat.api.openstack:sslmiddleware_filter

# Middleware to set auth_url header appropriately
[filter:authurl]
paste.filter_factory = heat.common.auth_url:filter_factory

# Auth middleware that validates token against keystone
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory

# Auth middleware that validates username/password against keystone
[filter:authpassword]
paste.filter_factory = heat.common.auth_password:filter_factory

# Auth middleware that validates against custom backend
[filter:custombackendauth]
paste.filter_factory = heat.common.custom_backend_auth:filter_factory

```

```
# Middleware to set x-openstack-request-id in http response header
[filter:request_id]
paste.filter_factory = heat.openstack.common.middleware.request_id:RequestIdMiddleware.factory
```

以 heat-api 服务为例，会最终找到：

```
[app:apiv1app]
paste.app_factory = heat.common.wsgi:app_factory
heat.app_factory = heat.api.openstack.v1:API
```

其中，heat.common.wsgi:app_factory 实际上根据传入参数进一步加载应用。

```
def __call__(self, global_conf, **local_conf):
    """The actual paste.app_factory protocol method."""
    factory = self._import_factory(local_conf)
    return factory(self.conf, **local_conf)
```

运行时，global={'__file__': '/usr/share/heat/api-paste-dist.ini', 'here': '/usr/share/heat'}
local={'heat.app_factory': 'heat.api.openstack.v1:API'}，所以实际上加载的还是 heat.api.openstack.v1.API。

而 heat.api.openstack.v1:API 中定义了对各种资源（包括 stack、resource、event、action、info、software config、software deployment 等）进行操作的 URL 所对应的处理方法。经过简单处理，后会发出对应的 rpc 消息给 heat-engine。

5.4 heat.conf.sample

样例配置程序，是 heat 服务的默认主配置文件。

5.5 policy.json

配置策略。

每次进行 API 调用时，会采取对应的检查，policy.json 文件发生更新后会立即生效。

该文件在开头定义

```
"context_is_admin": "role:admin",
"deny_stack_user": "not role:heat_stack_user",
"deny_everybody": "!",
```

目前支持的策略有三种：rule、role 或者 generic。

其中 rule 后面会跟一个文件名，例如

```
"cloudformation.ListStacks": "rule:deny_stack_user",
```

其策略为 rule:deny_stack_user，表明仅限 heat_stack_user 角色使用。

role 策略后面会跟一个 role 名称，表明只有指定 role 才可以执行。

generic 策略则根据参数来进行比较。

第 6 章 heat

该目录下包含了 heat 实现的主要代码。

6.1 api

因为 heat 面向的操作对象是可扩展的，包括 openstack 资源、aws 等等，所以 api 也分为不同的类型，以备后面仅限扩展。

6.1.1 aws

6.1.1.1 ec2token.py

定义 EC2Token 类，利用 keystone 来验证一个 EC2 请求，并将它转化为 token。

6.1.1.2 exception.py

定义处理 api 的异常类。

6.1.1.3 utils.py

一些辅助方法，包括提取参数等。

6.1.2 cfn

6.1.2.1 v1

6.1.2.1.1 signal.py

定义 SignalController 类。

6.1.2.1.2 stacks.py

定义 StackController 类，作为 stack 资源的 WSGI 控制器。

6.1.2.2 versions.py

返回版本信息的控制器类。

6.1.3 cloudwatch

6.1.3.1 __init__.py

定义了继承自 wsgi.Router 的 API 类。

6.1.3.2 versions.py

定义返回版本信息的控制器类。

6.1.3.3 watch.py

定义 WatchController 类，包含有一个 EngineClient 句柄，发送请求给 HeatEngine。

6.1.4 middleware

6.1.4.1 fault.py

定义了 Fault 类和 FaultWrapper 类。

6.1.4.2 ssl.py

定义了 SSLMiddleware 类。

6.1.4.3 version_negotiation.py

定义了 VersionNegotiationFilter 类。

6.1.5 openstack

6.1.5.1 v1

6.1.5.1.1 views

定义了格式化输出的一些方法。

6.1.5.1.2 __init__.py

里面定义了一个很重要的类：API，这个类直接接收 restful 的 api 调用，它里面带有一个 StackController 类句柄，将不同的 url 请求映射为对应资源的方法。例如

```
stacks_resource = stacks.create_resource(conf)
with mapper.submapper(controller=stacks_resource,
                       path_prefix="{tenant_id}") as stack_mapper:
    stack_mapper.connect("template_validate",
                        "/validate",
                        action="validate_template",
                        conditions={'method': 'POST'})
```

会将 url 到/tenant_id/validate 的 POST 请求交给 stacks_resource 对象的 validate_template 方法处理，即调用 stacks.StackController 类的 validate_template 方法。

该类中分别定义了对 Stack 资源、Resource 资源、Event 资源、Action 资源、Info 资源、Software config 资源、Software deployment 资源的相关 http 请求的处理，分别映射到该包中对应模块下的控制器上。

这些控制器根据收到的请求调用自身的对应方法，这些方法调用自身的 engineclient（调用 messaging 中的 rpc client）来发出 rpc 请求调用 engine 中的方法，主题为 api.ENGINE_TOPIC。

6.1.5.1.3 actions.py

定义 ActionController 类，响应对 Action 资源的请求，调用自身的 engineclient 进行 stack 的暂停和继续操作。

6.1.5.1.4 build_info.py

定义 BuildInfoController 类，响应对 BuildInfo 资源的请求，调用自身的 engineclient 进行获取 buildinfo 操作。

6.1.5.1.5 events.py

定义 EventController 类，响应对 Event 资源的请求，调用自身的 engineclient 进行列出和获取操作。

6.1.5.1.6 resources.py

6.1.5.1.7 software_configs.py

6.1.5.1.8 software_deployments.py

6.1.5.1.9 stacks.py

6.1.5.1.10 util.py

6.1.5.2 versions.py

响应对 api 版本请求，返回 http response。

6.2 cloudinit

6.2.1 loguserdata.py

将用户数据写到日志。

6.2.2 part_handler.py

6.2.3 boothook.sh

6.2.4 config

6.3 cmd

6.3.1 manager.py

支持一些用户命令，包括 db_version、db_sync、purge_deleted 等。

6.4 common

6.4.1 auth_password.py

定义 KeystonePasswordAuthProtocol 类，提供通过用户名密码来进行认证的方法。

6.4.2 auth_url.py

定义 AuthUrlFilter 类。

6.4.3 config.py

定义配置文件中使用的配置项关键字和默认值。以及 load_paste_app()等方法。

6.4.4 context.py

定义 RequestContext 类，请求的上下文信息。

6.4.5 crypt.py

加密和解密的相关方法。

6.4.6 custom_backend_auth.py

定义 AuthProtocol 类。

6.4.7 environment_format.py

对环境字符串进行解析。

6.4.8 exception.py

各种异常类。

6.4.9 heat_keystoneclient.py

定义 KeystoneClient 类和 KeystoneClientV3 类。封装 keystoneclient，以供资源使用。

6.4.10 identifier.py

定义 HeatIdentifier 类。

6.4.11 messaging.py

定义一些消息的序列化处理。

6.4.12 param_utils.py

解析参数辅助函数。

6.4.13 plugin_loader.py

定义动态从指定包加载所有模块的方法 load_modules。

6.4.14 policy.py

定义 Enforcer 类，加载和实施规则。

6.4.15 serializers.py

json 和 xml 回复的序列化类。

6.4.16 short_id.py

生成 id 方法。

6.4.17 `template_format.py`

一些模板格式化的方法。

6.4.18 `timeutils.py`

对时间进行处理的方法，包括转化为 ISO 8601 格式等。

6.4.19 `urlfetch.py`

从 URL 中获取数据的方法。

6.4.20 `wsgi.py`

wsgi 相关的封装类，包括请求、路由器、服务器等。

6.5 db

6.5.1 `sqlalchemy`

6.5.2 `api.py`

对数据库进行操作的 API。

6.5.3 `sync.py`

同步数据库。

6.5.4 `utils.py`

数据库辅助方法。

6.6 doc

6.7 engine

6.7.1 `cfn`

6.7.1.1 `functions.py`

一些继承自 `function.Function` 的类，代表模板中的各种方法。

6.7.1.2 `template.py`

定义 `CfnTemplate` 类，代表一个 cfn 的 stack 模板。

6.7.2 `hot`

6.7.2.1 `functions.py`

定义模板中函数相关的解析方法。

6.7.2.2 parameters.py

定义 HOTParameters 类和 HOTParamSchema 类，解析模板中的参数。

6.7.2.3 template.py

定义 HOTemplate，代表一个模板。

6.7.3 clients

6.7.3.1 os

包括对 openstack 中各种服务 client 的插件，例如 NeutronClientPlugin、NovaClientPlugin 等。

6.7.3.2 __init__.py

定义了 OpenStackClients，来创建和使用各种 client。

6.7.3.3 client_plugin.py

定义 ClientPlugin 类，抽象基础元类，包含各种异常类。

6.7.4 notification

6.7.4.1 __init__.py

6.7.4.2 autoscaling.py

定义 send 方法。

6.7.4.3 stack.py

定义 send 方法。

6.7.5 resources

这里面定义了对应 template 中各种资源的具体类，一般都是继承自 resource.Resource 类。这些类在最后会定义一个 resource_mapping 方法，返回 template 的某个关键词和对应类的映射。

例如 instance 模块中定义：

```
def resource_mapping():
    return {
        'AWS::EC2::Instance': Instance,
        'OS::Heat::HARestarter': Restarter,
    }
```

6.7.5.1 __init__.py

在导入包的时候，会创建 plugin_manager 并从所有资源中导入可用的映射信息。其中，定义的 initialise 方法在 heatengine 服务启动的时候会被调用。

initialise 方法同时调用了 `_load_global_environment` 方法，该方法调用 `_load_global_resources` 将所有的插件加载进来，并注册相关的资源。

6.7.5.2 ceilometer

定义对 Alarm 和 CombinationAlarm 两种资源的处理。

6.7.5.3 neutron

定义对防火墙、floatingip、负载均衡、metering、net、gateway、port、router、subnet、vpn 等网络相关的资源的处理。

以 Port 为例，定义了 Port 类，通过 `resource_mapping` 指定 `OS::Neutron::Port` 资源映射到 Port 类上。

Port 类中定义 `properties_schema` 和 `attributes_schema`。并定义对 CUD 操作的对应方法 `handle_create()`、`handle_delete()`、`handle_update()`。

6.7.5.4 software_config

6.7.5.5 autoscaling.py

处理 autoscaling 相关关键字。

6.7.5.6 cloud_watch.py

处理 `OS::Heat::CWLiteAlarm` 关键字。

6.7.5.7 eip.py

处理 AWS 中 EIP 相关关键字。

6.7.5.8 glance_image.py

处理 glance image 关键字。

6.7.5.9 instance.py

处理 `AWS::EC2::Instance` 和 `OS::Heat::HARestarter` 关键字。

6.7.5.10 internet_gateway.py

处理 `AWS::EC2::InternetGateway` 和 `AWS::EC2::VPCGatewayAttachment` 关键字。

6.7.5.11 loadbalancer.py

处理 `AWS::ElasticLoadBalancing::LoadBalancer` 关键字。

6.7.5.12 network_interface.py

处理 `AWS::EC2::NetworkInterface` 关键字。

6.7.5.13 nova_floatingip.py

处理 floatingIP 相关关键字。

6.7.5.14 nova_keypair.py

处理 `OS::Nova::KeyPair` 关键字。

6.7.5.15 nova_servergroup.py

处理 OS::Nova::ServerGroup 关键字。

6.7.5.16 os_database.py

处理 OS::Trove::Instance 关键字。

6.7.5.17 random_string.py

处理 OS::Heat::RandomString 关键字。

6.7.5.18 resource_group.py

处理 OS::Heat::ResourceGroup 关键字。

6.7.5.19 route_table.py

处理路由表相关关键字。

6.7.5.20 s3.py

处理 AWS::S3::Bucket 关键字。

6.7.5.21 security_group.py

处理 AWS::EC2::SecurityGroup 关键字。

6.7.5.22 server.py

处理 OS::Nova::Server 关键字。

6.7.5.23 stack.py

处理 AWS::CloudFormation::Stack 关键字。

6.7.5.24 subnet.py

处理 AWS::EC2::Subnet 关键字。

6.7.5.25 swift.py

处理 OS::Swift::Container 关键字。

6.7.5.26 user.py

处理 AWS::IAM::User、AWS::IAM::AccessKey 和 OS::Heat::AccessPolicy 关键字。

6.7.5.27 volume.py

处理 AWS::EC2::Volume、AWS::EC2::VolumeAttachment、OS::Cinder::Volume 和 OS::Cinder::VolumeAttachment 关键字。

6.7.5.28 vpc.py

处理 AWS::EC2::VPC 关键字。

6.7.5.29 wait_condition.py

6.7.5.30 iso_8601.py

定义 ISO8601Constraint 类，解析 iso8601 标准的时间为普通时间对象。

6.7.5.31 nova_utils.py

一些辅助处理函数，包括更新 meta、刷新等。

6.7.5.32 template_resource.py

定义 TemplateResource 类，代表嵌套 stack 资源。

6.7.6 api.py

定义一些格式化的方法，包括对 stack、资源属性等进行格式化。让它们符合 API 输出格式。

6.7.7 attributes.py

代表资源的 attribute。定义类 Attribute、Attributes 和 Schema。

6.7.8 constraints.py

一些限制类和 schema，例如处理允许的 pattern 等。

6.7.9 dependencies.py

用图模型来处理资源的依赖关系。

6.7.10 environment.py

定义资源信息类 ResourceInfo。

6.7.11 event.py

定义 Event 类，代表资源的状态发生改变。

6.7.12 function.py

定义 Function 类，是模板中函数的抽象基础类。

6.7.13 parameter_groups.py

定义 ParameterGroups 类，处理模板中的参数组。

6.7.14 parameters.py

定义 Parameter、NumberParam、BooleanParam、StringParam 等类，处理模板中的参数。

6.7.15 parser.py

指定 Template 为 template.Template，Stack 为 stack.Stack 类。

6.7.16 plugin_manager.py

对 plugin 进行管理。

定义 `PluginManager` 类和 `PluginMapping` 类。

前者会导入所有的插件。在初始化的时候根据传入参数以及配置文件 `heat.conf` 中 `plugin_dirs` 指定的目录（默认是 `/usr/lib64/heat/`、`/usr/lib/heat`）去进行扫描所有模块并导入。

后者定义方法去扫描 `PluginManager` 中加载的模块，并根据指定感兴趣的 `mapping` 方法名称，例如 `resource_mapping`、`available_resource_mapping`、`constraint_resource_mapping` 等来导入相关的方法。

```
def load_from_module(self, module):
    """Return the mapping specified in the given module.

    If no such mapping is specified, an empty dictionary is returned.
    """
    for mapping_name in self.names:
        mapping_func = getattr(module, mapping_name, None)
        if callable(mapping_func):
            fmt_data = {'mapping_name': mapping_name, 'module': module}
            try:
                mapping_dict = mapping_func(*self.args, **self.kwargs)
            except Exception:
                LOG.error(_('Failed to load %(mapping_name)s '
                           'from %(module)s') % fmt_data)
                raise
            else:
                if isinstance(mapping_dict, collections.Mapping):
                    return mapping_dict
                elif mapping_dict is not None:
                    LOG.error(_('Invalid type for %(mapping_name)s '
                               'from %(module)s') % fmt_data)
    return {}
```

6.7.17 properties.py

定义 `Property` 类，处理模板中的 `property`。

6.7.18 resource.py

定义 `Resource` 基础类，是其他资源类的基类。

资源的属性包括行动、状态等，另外定义 `default_client_name`，用来向 `openstack` 中的服务发起操作指令。

6.7.19 rsrc_defn.py

定义 `ResourceDefinition` 类和 `ResourceDefinitionCore` 类。

6.7.20 scheduler.py

对任务进行管理的一些辅助类。

6.7.21 service.py

定义了若干很重要的服务类，首先是 `EngineListener`，继承自 `service.Service`，监听 `AMQP` 队列。

EngineService 服务类，继承自 service.Service，bin/heat-engine 脚本启动的服务就是这个类。heatclient 发出的命令经过 heatapi，最终会以 rpc 请求消息的方式调用本类中的方法。包括对 stack、software_config 等资源的 create、delete 等。该服务启动后会创建一个 EngineListener 来监听消息。

6.7.22 signal_responder.py

定义 SignalResponder 类，继承自 stack_user.StackUser，响应 create、delete 方法，清除数据等。

6.7.23 stack.py

定义了 Stack 类，继承自 collections.Mapping，代表 heat template 中的一个 stack 对象，支持创建、删除、更新、回滚、暂停、继续、快照等操作。

6.7.24 stack_lock.py

定义 StackLock 类，用于维护一个 stack 资源上的锁。

6.7.25 stack_resource.py

定义了 StackResource 类，继承自 resource.Resource。抽象类，允许父级 stack 像管理一个资源一样来管理一个子 Stack。

6.7.26 stack_user.py

定义了 StackUser 类，继承自 resource.Resource。
创建一个关联 stack 的 user。

6.7.27 support.py

定义 SupportStatus，表示支持状态。

6.7.28 template.py

定义 Template 类和 TemplatePluginManager 类，管理模板。

6.7.29 timestamp.py

定义 Timestamp 类。管理时间戳。

6.7.30 update.py

定义 StackUpdate 类，升级一个存在的 stack 到一个新的模板。

6.7.31 watchrule.py

定义 WatchRule 类。

6.8 locale

多国语言支持相关。

6.9 Openstack

6.9.1 Common

含有 openstack 提供的基础服务类。openstack-common 项目负责维护这些代码，为所有的 openstack 子项目提供方法支持。

6.9.1.1 Config

还有 generator.py 文件，可以根据指定的代码文件来提取配置项，生成样例配置文件。

6.9.1.2 Crypto

跟加密相关的辅助方法。

6.9.1.3 Middleware

实现 WSGI 的 middleware 层。

6.9.1.4 Context.py

定义处理 http 请求上下文的 RequestContext 类。

6.9.1.5 eventlet_backdoor.py

实现 eventlet 的后门，监听。

6.9.1.6 excutils.py

定义处理异常类 save_and_reraise_exception，可以保存当前异常，运行一些代码，然后再次抛出异常。

6.9.1.7 fileutils.py

定义一些对文件进行操作的辅助方法。

6.9.1.8 gettextutils.py

语言国际化。

6.9.1.9 importutils.py

根据输入参数来动态导入对象、类等等。

6.9.1.10 jsonutils.py

对 json 内容进行处理。

6.9.1.11 local.py

弱本地引用类 WeakLocal。

6.9.1.12 lockutils.py

处理锁相关操作。

6.9.1.13 log.py

log 相关操作。

6.9.1.14 loopingcall.py

周期性调用类，包括 DynamicLoopingCall、FixedIntervalLoopingCall、LoopingCallBase。

6.9.1.15 network_utils.py

一些网络配置相关的辅助方法，包括从字符串解析主机名和端口、解析 url 等。

6.9.1.16 policy.py

对策略的处理。

6.9.1.17 processutils.py

管理进程。

6.9.1.18 service.py

实现服务基础类 Service、Services 等。

6.9.1.19 sslutils.py

定义使用 ssl 的辅助方法。

6.9.1.20 strutils.py

对字符串进行解码、编码等操作。

6.9.1.21 systemd.py

对 systemd 进行管理的方法。

6.9.1.22 threadgroup.py

定义 thread 类和 threadgroup 类，对线程进行管理。

6.9.1.23 timeutils.py

对时间相关的字符串进行处理等。

6.9.1.24 uuidutils.py

生成和检查 uuid。

6.9.1.25 versionutils.py

检查版本兼容信息。

6.10 rpc

6.10.1 api.py

定义了一些常量，包括 ENGINE_TOPIC、STACK_KEYS 等。

6.10.2 client.py

定义了 EngineClient 类，该类可以利用 rpc 跟 heatengine 进行通信完成各种资源操作。对资源进行操作的各个控制器实际上就是调用本类中的方法。

支持利用 call 和 cast 方法发出各种消息来调用方法，包括对 stack 的 show、count 等。

6.11 scaling

6.11.1 template.py

定义方法，支持利用已有的资源创建新的模板等。

6.12 tests

测试相关。

6.13 __init__.py

安装_()方法，支持国际化。

6.14 version.py

获取版本信息。

第 7 章 **rally-scenarios**

包括 OpenStack CI 进行 rally benchmark 测试时候的一些场景。

第 8 章 tools

包括跟安装和开发相关的一些工具。

8.1 config

检查配置文件和生成样例配置文件。

8.2 其它

`cfn-json2yaml`

转化 `cfn-json` 格式为 `yaml` 格式。

`install_venv.py`, `neutron` 开发的时候, 可能需要一套虚环境, 该脚本可以安装一个 `python` 虚环境。

`install_venv_common.py`, 为 `install_venv.py` 提供必要的方法。

`with_venv.sh`, 启用虚环境。

`heat-db-drop`, 清除 `heat` 的数据库。

`requirements_style_check.sh`, 检查 `requirement` 的格式。

8.3 清理

`clean.sh`, 清除代码编译中间结果。

第 9 章 理解代码

本部分试图从专题和业务流程的角度来剖析 heat 代码，以便理解如此设计的内涵。

9.1 调用逻辑