目 录

1 nova 中的 RPC 机制 **2**
1.1 基础部分 . . . 2
1.1.1 rpc.py . . . 2
1.1.2 olso.messaging . . . 2
1.1.3 Manager 类 . . . 5
1.2 nova-compute 部分 . . . 6
1.2.1 computeAPI 类 . . . 6
1.2.2 ComputeManager 类 . . . 6
1.2.3 nova-compute 部分总结 . . . 6
1.3 nova-conductor 部分 . . . 7
1.3.1 ComputeTaskAPI 类 . . . 7
1.3.2 ComputeTaskManager 类 . . . 7
1.3.3 nova-conductor 部分总结 . . . 7
1.4 nova-scheduler 部分 . . . 7
1.4.1 SchedulerAPI 类 . . . 7
1.4.2 SchedulerManager 类 . . . 8
1.4.3 nova-scheduler 部分总结 . . . 8

1 nova 中的 RPC 机制

1.1 基础部分

1.1.1 rpc.py

```
    def get_client(target, version_cap=None, serializer=None):
        assert TRANSPORT is not None
        serializer = RequestContextSerializer(serializer)
        return messaging.RPCClient(TRANSPORT,
                                   target,
                                   version_cap=version_cap,
                                   serializer=serializer)


    def get_server(target, endpoints, serializer=None):
        assert TRANSPORT is not None
        serializer = RequestContextSerializer(serializer)
        return messaging.get_rpc_server(TRANSPORT,
                                        target,
                                        endpoints,
                                        executor='eventlet',
                                        serializer=serializer)
```

1.1.2 olso.messaging

```
class Target(object):

    """Identifies the destination of messages.

    A Target encapsulates all the information to identify where a message
    should be sent or what messages a server is listening for.

    Different subsets of the information encapsulated in a Target object is
    relevant to various aspects of the API:

      creating a server:
        topic and server is required; exchange is optional
      an endpoint's target:
        namespace and version is optional
      client sending a message:
        topic is required, all other attributes optional

    Its attributes are:

    :param exchange: A scope for topics. Leave unspecified to default to the
      control_exchange configuration option.
    :type exchange: str
    :param topic: A name which identifies the set of interfaces exposed by a
      server. Multiple servers may listen on a topic and messages will be
```

```
      dispatched to one of the servers in a round-robin fashion.
    :type topic: str
    :param namespace: Identifies a particular interface (i.e. set of methods)
      exposed by a server. The default interface has no namespace identifier
      and is referred to as the null namespace.
    :type namespace: str
    :param version: Interfaces have a major.minor version number associated
      with them. A minor number increment indicates a backwards compatible
      change and an incompatible change is indicated by a major number bump.
      Servers may implement multiple major versions and clients may require
      indicate that their message requires a particular minimum minor version.
    :type version: str
    :param server: Clients can request that a message be directed to a specific
      server, rather than just one of a pool of servers listening on the topic.
    :type server: str
    :param fanout: Clients may request that a message be directed to all
      servers listening on a topic by setting fanout to ``True``, rather than
      just one of them.
    :type fanout: bool
    """

    def __init__(self, exchange=None, topic=None, namespace=None,
                 version=None, server=None, fanout=None):
        self.exchange = exchange
        self.topic = topic
        self.namespace = namespace
        self.version = version
        self.server = server
        self.fanout = fanout

class RPCClient(object):
    def prepare(self, exchange=_marker, topic=_marker, namespace=_marker,
                version=_marker, server=_marker, fanout=_marker,
                timeout=_marker, version_cap=_marker, retry=_marker):
        """Prepare a method invocation context.

        Use this method to override client properties for an individual method
        invocation. For example::

            def test(self, ctxt, arg):
                cctxt = self.prepare(version='2.5')
                return cctxt.call(ctxt, 'test', arg=arg)

        :param exchange: see Target.exchange
        :type exchange: str
        :param topic: see Target.topic
        :type topic: str
        :param namespace: see Target.namespace
        :type namespace: str
        :param version: requirement the server must support, see Target.version
        :type version: str
        :param server: send to a specific server, see Target.server
        :type server: str
        :param fanout: send to all servers on topic, see Target.fanout
        :type fanout: bool
        :param timeout: an optional default timeout (in seconds) for call()s
```

```
        :type timeout: int or float
        :param version_cap: raise a RPCVersionCapError version exceeds this cap
        :type version_cap: str
        :param retry: an optional connection retries configuration
                      None or -1 means to retry forever
                      0 means no retry
                      N means N retries
        :type retry: int
        """
        return _CallContext._prepare(self,
                                     exchange, topic, namespace,
                                     version, server, fanout,
                                     timeout, version_cap, retry)

class _CallContext(object):

    _marker = object()

    def __init__(self, transport, target, serializer,
                 timeout=None, version_cap=None, retry=None):
        self.conf = transport.conf

        self.transport = transport
        self.target = target
        self.serializer = serializer
        self.timeout = timeout
        self.retry = retry
        self.version_cap = version_cap

        super(_CallContext, self).__init__()

    def cast(self, ctxt, method, **kwargs):
        """Invoke a method and return immediately. See RPCClient.cast()."""
        msg = self._make_message(ctxt, method, kwargs)
        ctxt = self.serializer.serialize_context(ctxt)

        if self.version_cap:
            self._check_version_cap(msg.get('version'))
        try:
            self.transport._send(self.target, ctxt, msg, retry=self.retry)
        except driver_base.TransportDriverError as ex:
            raise ClientSendError(self.target, ex)

    def call(self, ctxt, method, **kwargs):
        """Invoke a method and wait for a reply. See RPCClient.call()."""
        msg = self._make_message(ctxt, method, kwargs)
        msg_ctxt = self.serializer.serialize_context(ctxt)

        timeout = self.timeout
        if self.timeout is None:
            timeout = self.conf.rpc_response_timeout

        if self.version_cap:
            self._check_version_cap(msg.get('version'))

        try:
```

```
            result = self.transport._send(self.target, msg_ctxt, msg,
                                          wait_for_reply=True, timeout=timeout,
                                          retry=self.retry)
        except driver_base.TransportDriverError as ex:
            raise ClientSendError(self.target, ex)
        return self.serializer.deserialize_entity(ctxt, result)

    @classmethod
    def _prepare(cls, base,
                 exchange=_marker, topic=_marker, namespace=_marker,
                 version=_marker, server=_marker, fanout=_marker,
                 timeout=_marker, version_cap=_marker, retry=_marker):
        """Prepare a method invocation context. See RPCClient.prepare()."""
        kwargs = dict(
            exchange=exchange,
            topic=topic,
            namespace=namespace,
            version=version,
            server=server,
            fanout=fanout)
        kwargs = dict([(k, v) for k, v in kwargs.items()
                       if v is not cls._marker])
        target = base.target(**kwargs)

        if timeout is cls._marker:
            timeout = base.timeout
        if retry is cls._marker:
            retry = base.retry
        if version_cap is cls._marker:
            version_cap = base.version_cap

        return _CallContext(base.transport, target,
                            base.serializer,
                            timeout, version_cap, retry)

    def prepare(self, exchange=_marker, topic=_marker, namespace=_marker,
                version=_marker, server=_marker, fanout=_marker,
                timeout=_marker, version_cap=_marker, retry=_marker):
        """Prepare a method invocation context. See RPCClient.prepare()."""
        return self._prepare(self,
                             exchange, topic, namespace,
                             version, server, fanout,
                             timeout, version_cap, retry)
```

1.1.3 Manager 类

```
class Manager(base.Base, periodic_task.PeriodicTasks):

    def __init__(self, host=None, db_driver=None, service_name='undefined'):
        if not host:
            host = CONF.host
        self.host = host
        self.service_name = service_name
```

```
        self.notifier = rpc.get_notifier(self.service_name, self.host)
        ...
```

1.2 nova-compute 部分

1.2.1 computeAPI 类

```
class ComputeAPI(object):
    def __init__(self):
        super(ComputeAPI, self).__init__()
        target = messaging.Target(topic=CONF.compute_topic, version='3.0')
        ...
        self.client = self.get_client(target, version_cap, serializer)

    def get_client(self, target, version_cap, serializer):
        return rpc.get_client(target,
                              version_cap=version_cap,
                              serializer=serializer)
```

举个例子:

```
    def attach_interface(self, ctxt, instance, network_id, port_id,
                         requested_ip):
        version = '3.17'
        cctxt = self.client.prepare(server=_compute_host(None, instance),
                version=version)
        return cctxt.call(ctxt, 'attach_interface',
                          instance=instance, network_id=network_id,
                          port_id=port_id, requested_ip=requested_ip)
```

1.2.2 ComputeManager 类

```
class ComputeManager(manager.Manager):

    def __init__(self, compute_driver=None, *args, **kwargs):
        ...
        super(ComputeManager, self).__init__(service_name="compute",
                                             *args, **kwargs)
        ...
```

1.2.3 nova-compute 部分总结

ComputeManager 类接受 ComputeAPI 类的 RPC 请求，主机信息由 RPC.cast() 或 RPC.call() 中的 server 决定。

1.3 nova-conductor 部分

1.3.1 ComputeTaskAPI 类

```
class ComputeTaskAPI(object):
    def __init__(self):
        super(ComputeTaskAPI, self).__init__()
        target = messaging.Target(topic=CONF.conductor.topic,
                                  namespace='compute_task',
                                  version='1.0')
        serializer = objects_base.NovaObjectSerializer()
        self.client = rpc.get_client(target, serializer=serializer)
```

1.3.2 ComputeTaskManager 类

```
class ComputeTaskManager(base.Base):
    target = messaging.Target(namespace='compute_task', version='1.9')

    def __init__(self):
        super(ComputeTaskManager, self).__init__()
        self.compute_rpcapi = compute_rpcapi.ComputeAPI()
        self.image_api = image.API()
        self.scheduler_client = scheduler_client.SchedulerClient()
```

1.3.3 nova-conductor 部分总结

ComputeTaskManager 类接受 ComputeTaskAPI 类的 RPC 请求，主机信息由 RPC.cast() 或 RPC.call() 中的 server 决定。

1.4 nova-scheduler 部分

1.4.1 SchedulerAPI 类

```
class SchedulerAPI(object):

    def __init__(self):
        super(SchedulerAPI, self).__init__()
        target = messaging.Target(topic=CONF.scheduler_topic, version='3.0')
        version_cap = self.VERSION_ALIASES.get(CONF.upgrade_levels.scheduler,
                                               CONF.upgrade_levels.scheduler)
        serializer = objects_base.NovaObjectSerializer()
        self.client = rpc.get_client(target, version_cap=version_cap,
                                     serializer=serializer)

    def select_destinations(self, ctxt, request_spec, filter_properties):
        cctxt = self.client.prepare()
```

```
        return cctxt.call(ctxt, 'select_destinations',
            request_spec=request_spec, filter_properties=filter_properties)
```

1.4.2 SchedulerManager 类

```
class SchedulerManager(manager.Manager):
    """Chooses a host to run instances on."""

    target = messaging.Target(version='3.0')

    def __init__(self, scheduler_driver=None, *args, **kwargs):
        if not scheduler_driver:
            scheduler_driver = CONF.scheduler_driver
        self.driver = importutils.import_object(scheduler_driver)
        self.compute_rpcapi = compute_rpcapi.ComputeAPI()
        super(SchedulerManager, self).__init__(service_name='scheduler',
                                               *args, **kwargs)
```

1.4.3 nova-scheduler 部分总结

ComputeManager 类接受 ComputeAPI 类的 RPC 请求，主机信息由 RPC.cast() 或 RPC.call() 中的 server 决定。