目 录

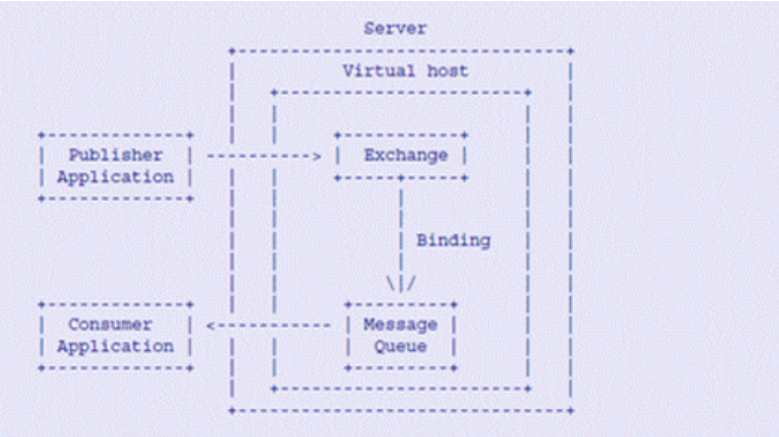
1 openstack 中的 RPC **2**
1.1 AMQP 2
1.2 RPC 中的 client 和 server 3
1.2.1 四种 publisher 3
1.2.2 三种 consumer 3
1.2.3 RPC 发送请求 3
1.3 openstack 中使用 RPC 的例子 5

1 openstack 中的 RPC

openstack 中的 RPC 机制的实现基于 AMQP 作为通讯模型。

1.1 AMQP

AMQP 是用于异步消息通讯的消息中间件协议，它的模型如下所示：



它有四个重要的角色：

1. Exchange，根据 Routing key 转发消息到对应的 Message Queue 中。
2. Routing key，Exchange 根据 Routing key 来决定把消息发送到哪些 Message Queue 中。
3. Publisher，消息发送者，指明消息的 Routing key，然后把消息传给 Exchange。
4. Consumer，消息接受者，从 Message Queue 获取消息。

Exchange 转发消息的原理如下：

```
1 每一个发送的消息上都有一个routing key，而每一个Queue也有一个binding key。
2 Exchange进行消息路由时，会查询每一个Queue，如果某个Queue的binding key与某个消息
      的routing key相匹配，这个消息就会被转发到那个Queue里。
```

AMQP 有三种类型的 Exchange：

1. Direct，binding key 和 routing key 必须完全一致，不支持通配符。
2. Topic，同 Direct 类型，但支持通配符。
3. Fanout，忽略 binding key 和 routing key，消息会被传递到所有绑定的队列上。

1.2 RPC 中的 client 和 server

在 RPC 通信中有两个角色：client 和 server。client 发起 RPC 请求，server 端接受 RPC 请求，然后调用本地的程序执行，最后将执行结果返回给 client。

openstack 中的 RPC 利用了 publisher 和 consumer 实现了 client 和 server 之间的 RPC 通信，其中 publisher 相当于 client，consumer 相当于 server。

1.2.1 四种 publisher

openstack 中四种 publisher 的介绍如下：

1. Direct Publisher：用于点对点的消息通信，在 OpenStack RPC 通信中用于建立 RPC 消息应答通路。创建或声明 Direct Exchange，用于 RPC 的消息返回，Exchange 的名字以消息 id 命名。
2. Topic Publisher：用于 Publish-Subscribe (Pub-Sub) 模式的通信，Topic Publisher 创建 Topic Exchange，用于 RPC 消息发送，并设置消息的 Routing Key 转发消息。
3. Fanout Publisher：创建 Fanout Exchange，用于广播消息的发送，所有绑定到该 Exchange 的 Message Queue 都会接收到消息。
4. Notify Publisher：同 Topic Publisher，在 Openstack 系统中用于发送 Notification 相关的消息。

1.2.2 三种 consumer

openstack 中三种 consumer 的介绍如下：

1. Direct Consumer：接收 Direct Exchange 发送的消息，创建专属的 Message Queue，即 Message Queue 的消息只能由创建 Message Queue 的 Consumer 获取。
2. Topic Consumer：接收 topic Exchange 根据 Routing Key 转发的消息，创建的 Message Queue 是可共享的，这就意味着多个 Consumer 可以订阅同一个 Message Queue，并从其中读取消息。
3. Fanout Consumer：接收 Fanout Exchange 广播的消息，并创建专属的 Message Queue。

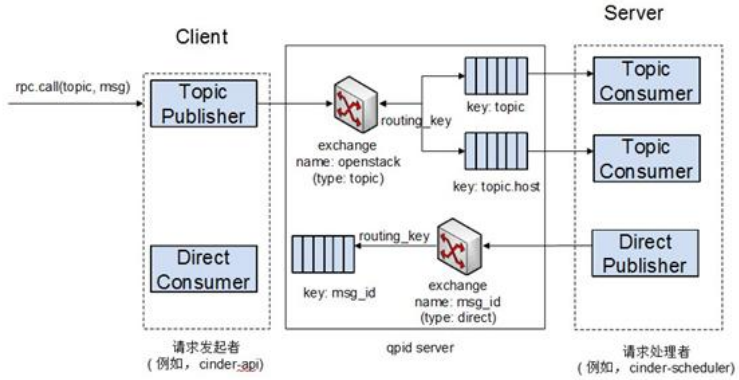
1.2.3 RPC 发送请求

RPC 中的 client 端由 publisher 实现，publisher 发送消息并声明消息地址。consumer 接收消息并进行消息处理，如果需要消息应答则返回处理请求的结果消息。

OpenStack 提供了三种发送消息请求的方式：

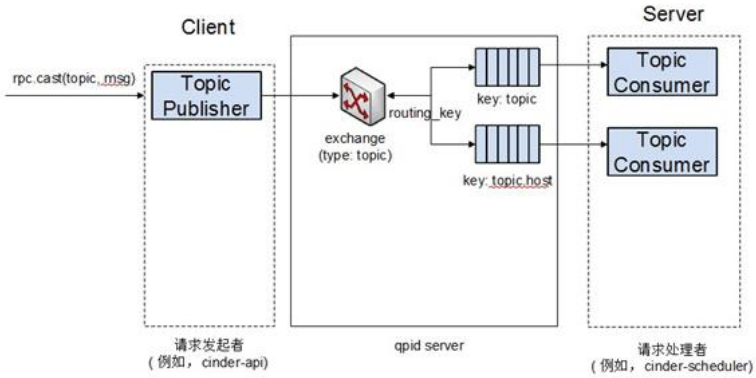
1. rpc.call 发送 RPC 请求并返回请求处理结果，由 Topic Publisher 发送消息，Topic Exchange 根据消息地址进行消息转发至对应的 Message Queue 中，Topic Consumer 监听 Message Queue，发现需要处理的消息则进行消息处理，并由 Direct Publisher 将请求处理结果消息，请求发送方创建 Direct Consumer 监听消息的返回结果。

   如下图所示：

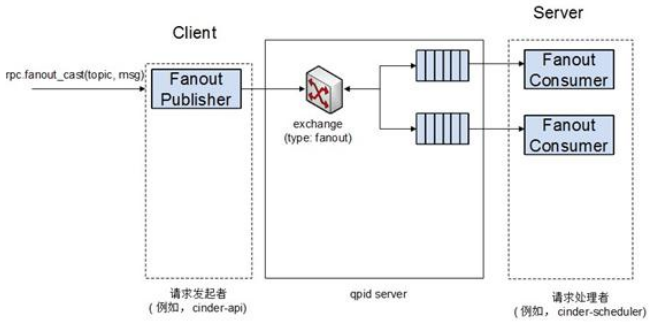


2. rpc.cast 发送 RPC 请求无返回，与 rpc.call 不同之处在于，不需要请求处理结果的返回，因此没有 Direct Publisher 和 Direct Consumer 处理。

   如下图所示：



3. rpc.fanout_cast 用于发送 RPC 广播信息无返回结果，如下图所示：



1.3 openstack 中使用 RPC 的例子

我们分析 cinder 创建 volume 的 RPC 消息的发送过程，从而更加深刻的理解 RPC 消息的处理流程。

Create volume 的过程可以看做是两个阶段的 RPC 请求处理。

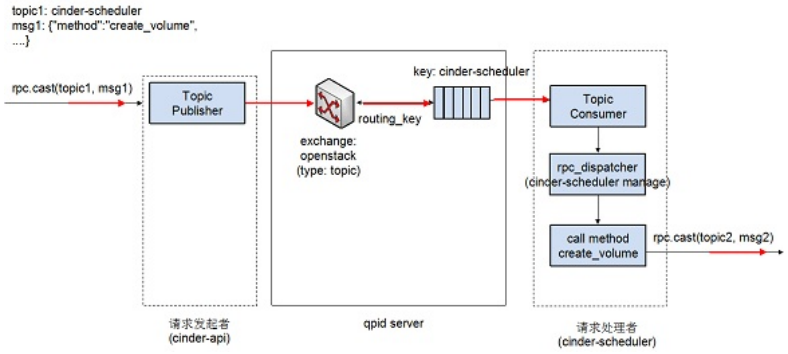
第一阶段：

1. Client 即 cinder-api 调用 RPC.cast 发送消息，RPC.cast 创建 Topic Publisher 对象，topic 为 cinder-scheduler，也就是消息的 Routing Key，Exchange name 为 OpenStack，消息体中标记了消息接收者即 Server 所需要调用的方法“create_volume”。

2. Exchange 接收到消息，根据 Routing Key 将消息发送至 Message Queue cinder-scheduler 上，这个 Message Queue 是由 cinder-scheduler 服务的 Topic Consumer 订阅的，因此，cinder-scheduler 服务将接收到 create_volume 的消息。

3. Consumer 调用注册的 RPC_dispatcher，即 cinder-scheduler manager，然后由 cinder-scheduler manager 调用 RPC 处理方法 create_volume。

第一阶段的流程如下图所示：



第一个阶段的最后是由 cinder-scheduler manager 调用 RPC 处理方法 create_volume，我们第二阶段就是要分析这个过程：

1. Cinder-scheduler 调用 RPC.cast，Exchange name 为 Openstack, routing-key 为 cinder-volume:host。
2. Exchange 接收到消息，根据 Routing Key 将消息发送到队列 cinder-volume:host 上。
3. 在主机 host 上的 Cinder-volume consumer 从 Queue 中接收到 message，回调 RPC_dispatcher 即 cinder-volume manager。
4. Cinder-volume manger 根据消息体中 method 信息，调用 create_volume 方法，创建 volume。

第二阶段的流程如下图所示：