## rabbitmq通信原理

### 几个概念

Producter/publisher：消息生产者，就是投递消息的程序。

consumer：消息消费者，就是接受消息的程序。

Broker：简单来说就是消息队列服务器实体。

vhost：虚拟主机，一个broker里可以开设多个vhost，用作不同用户的权限分离。

**Exchange**：消息交换机，它指定消息按什么规则，路由到哪个队列。

**Queue**：消息队列载体，每个消息都会被投入到一个或多个队列。

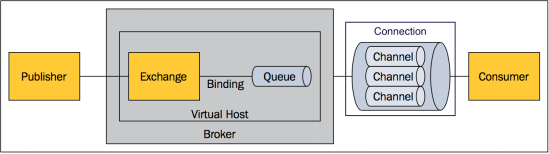
**Binding**：绑定，它的作用就是把exchange和queue按照路由规则绑定起来。

**Routing Key**：路由关键字，exchange根据这个关键字进行消息投递。

channel：消息通道，在客户端的每个连接里，可建立多个channel，每个channel代表一个会话任务。

### rabbitmq通信过程

rabbitmq通信关键过程

RabbitMQ 内部结构图

生产者（publisher）生产一条消息🡪投递到exchange交换器🡪交换器按照消息指定规则(routing key)将消息发送到与routing key规则匹配的消息队列queue🡪消息队列queue通过channel（tcp长连接信道）将消息发送到consumer。

#### 1.2.1 rabbitmq 消息交换机（exchange）

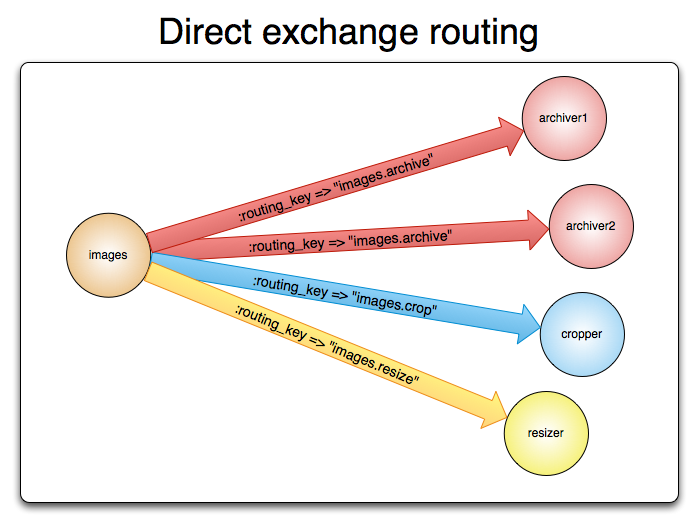
交换机是用来发送消息的AMQP实体。交换机拿到一个消息之后将它路由给一个或零个队列。它使用哪种路由算法是由交换机类型和被称作绑定（bindings）的规则所决定的。AMQP 0-9-1的代理提供了四种交换机：

| **Name（交换机类型）** | **Default pre-declared names（预声明的默认名称）** |
| --- | --- |
| Direct exchange（直连交换机） | (Empty string) and amq.direct |
| Fanout exchange（扇型交换机） | amq.fanout |
| Topic exchange（主题交换机） | amq.topic |
| Headers exchange（头交换机） | amq.match (and amq.headers in RabbitMQ) |

**直连交换机**：直连型交换机（direct exchange）是根据消息携带的路由键（routing key）将消息投递给对应队列的。类似于单播路由。例如：

1）将一个队列绑定到某个交换机上，同时赋予该绑定一个路由键（routing key）

2）当一个携带着路由键为R的消息被发送给直连交换机时，交换机会把它路由给绑定值同样为R的队列。



直连交换机图例

**扇型交换机：**扇型交换机（funout exchange）将消息路由给绑定到它身上的所有队列，而不理会绑定的路由键。如果N个队列绑定到某个扇型交换机上，当有消息发送给此扇型交换机时，交换机会将消息的拷贝分别发送给这所有的N个队列。扇型用来交换机处理消息的广播路由（broadcast routing）。

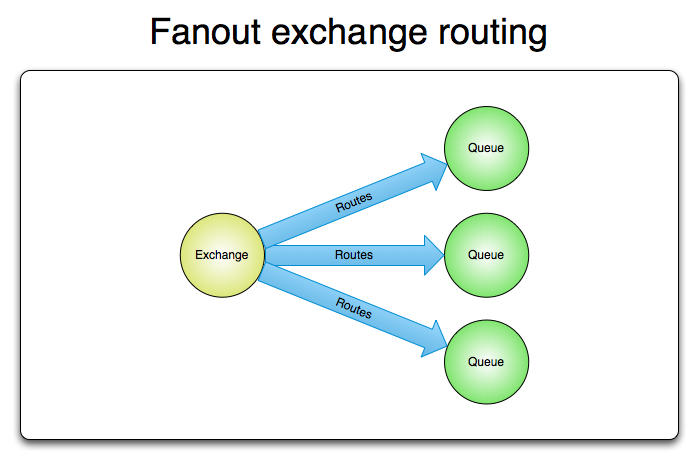
因为扇型交换机投递消息的拷贝到所有绑定到它的队列，所以他的应用案例都极其相似：

1）大规模多用户在线（MMO）游戏可以使用它来处理排行榜更新等全局事件

2）体育新闻网站可以用它来近乎实时地将比分更新分发给移动客户端

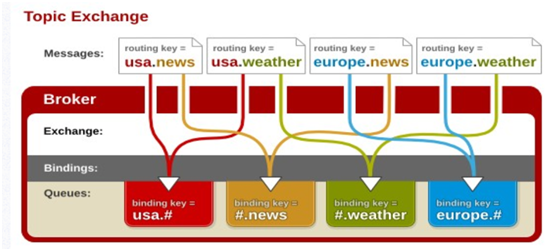
3）分发系统使用它来广播各种状态和配置更新

4）在群聊的时候，它被用来分发消息给参与群聊的用户。（AMQP没有内置presence的概念，因此XMPP可能会是个更好的选择）



扇型交换机图例

**主题交换机：**主题交换机topic通过模式匹配分配消息的路由键属性，将路由键和某个模式进行匹配，此时队列需要绑定到一个模式上。它将路由键和绑定键的字符串切分成单词，这些单词之间用点隔开。它同样也会识别两个通配符：符号“#”和符号“”。#匹配0个或多个单词，匹配不多不少一个单词。

主题交换机图

#### rabbitmq 消息队列载体（queue）

AMQP中的队列（queue）存储着即将被应用消费掉的消息。队列跟交换机共享某些属性，但是队列也有一些另外的属性。

Name

Durable（消息代理重启后，队列依旧存在）

Exclusive（只被一个连接（connection）使用，而且当连接关闭后队列即被删除）

Auto-delete（当最后一个消费者退订后即被删除）

Arguments（一些消息代理用他来完成类似与TTL的某些额外功能）

**队列名称**

队列的名字可以由应用（application）来取，也可以让消息代理（broker）直接生成一个。队列的名字可以是最多255字节的一个utf-8字符串。若希望AMQP消息代理生成队列名，需要给队列的name参数赋值一个空字符串：在同一个通道（channel）的后续的方法（method）中，我们可以使用空字符串来表示之前生成的队列名称。之所以之后的方法可以获取正确的队列名是因为通道可以默默地记住消息代理最后一次生成的队列名称。

以"amq."开始的队列名称被预留做消息代理内部使用。如果试图在队列声明时打破这一规则的话，一个通道级的403 (ACCESS\_REFUSED)错误会被抛出。

**队列持久化**

持久化队列（Durable queues）会被存储在磁盘上，当消息代理（broker）重启的时候，它依旧存在。没有被持久化的队列称作暂存队列（Transient queues）。并不是所有的场景和案例都需要将队列持久化。

持久化的队列并不会使得路由到它的消息也具有持久性。倘若消息代理挂掉了，重新启动，那么在重启的过程中持久化队列会被重新声明，无论怎样，只有经过持久化的消息才能被重新恢复。

**绑定**

绑定（Binding）是交换机（exchange）将消息（message）路由给队列（queue）所需遵循的规则。如果要指示交换机“E”将消息路由给队列“Q”，那么“Q”就需要与“E”进行绑定。绑定操作需要定义一个可选的路由键（routing key）属性给某些类型的交换机。路由键的意义在于从发送给交换机的众多消息中选择出某些消息，将其路由给绑定的队列。

打个比方：

队列（queue）是我们想要去的位于纽约的目的地

交换机（exchange）是JFK机场

绑定（binding）就是JFK机场到目的地的路线。能够到达目的地的路线可以是一条或者多条

拥有了交换机这个中间层，很多由发布者直接到队列难以实现的路由方案能够得以实现，并且避免了应用开发者的许多重复劳动。

如果AMQP的消息无法路由到队列（例如，发送到的交换机没有绑定队列），消息会被就地销毁或者返还给发布者。如何处理取决于发布者设置的消息属性。

**消息确认**

消费者应用（Consumer applications） - 用来接受和处理消息的应用 - 在处理消息的时候偶尔会失败或者有时会直接崩溃掉。而且网络原因也有可能引起各种问题。这就给我们出了个难题，AMQP代理在什么时候删除消息才是正确的？AMQP 0-9-1 规范给我们两种建议：

当消息代理（broker）将消息发送给应用后立即删除。（使用AMQP方法：basic.deliver或basic.get-ok）

待应用（application）发送一个确认回执（acknowledgement）后再删除消息。（使用AMQP方法：basic.ack）

前者被称作自动确认模式（automatic acknowledgement model），后者被称作显式确认模式（explicit acknowledgement model）。在显式模式下，由消费者应用来选择什么时候发送确认回执（acknowledgement）。应用可以在收到消息后立即发送，或将未处理的消息存储后发送，或等到消息被处理完毕后再发送确认回执（例如，成功获取一个网页内容并将其存储之后）。

如果一个消费者在尚未发送确认回执的情况下挂掉了，那AMQP代理会将消息重新投递给另一个消费者。如果当时没有可用的消费者了，消息代理会死等下一个注册到此队列的消费者，然后再次尝试投递。

### Hello World

**send.py的完整代码：**

#!/usr/bin/env python

import pika

#建立rabbitmq连接。

connection =

pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

channel = connection.channel()

#建一个名为"hello"的队列用来将消息投递进去。

channel.queue\_declare(queue='hello')

#RabbitMQ中，消息是不能直接发送到队列中的，这个过程需要通过交换机（exchange）来进行。默认交换机较特别，它允许我们指定消息究竟需要投递到哪个具体的队列中，队列名字需要在routing\_key参数中指定。

channel.basic\_publish(exchange='',routing\_key='hello',body='Hello World!')

print(" [x] Sent 'Hello World!'")

connection.close()

**receive.**py**的完整代码：**

#!/usr/bin/env python

import pika

#建立rabbitmq连接。

connection =

pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

channel = connection.channel()

#可以多次使用queue\_declare命令来创建同一个队列，但是只有一个队列会被真正的创建，发现队列被创建即删除。

channel.queue\_declare(queue='hello')

def callback(ch, method, properties, body):

print(" [x] Received %r" % body)

#告诉rabbitMQ这个回调函数将会从名为"hello"的队列中接收消息。

channel.basic\_consume(callback,queue='hello',no\_ack=True)

#最后，运行一个用来等待消息数据并且在需要的时候运行回调函数的无限循环。

print(' [\*] Waiting for messages. To exit press CTRL+C')

channel.start\_consuming()

在终端中尝试一下我们的程序，首先启动一个消费者，它会持续的运行来等待投递到达。

**python receive.py**

# => [\*] Waiting for messages. To exit press CTRL+C

# => [x] Received 'Hello World!'

然后启动生产者，生产者程序每次执行后都会停止运行。

**python send.py**

# => [x] Sent 'Hello World!'

**成功了！**我们已经通过RabbitMQ发送第一条消息。

## rabbitmq基本的管理功能

### 2.1几个概念

Erlang: Erlang是一门动态类型的函数式编程语言，它也是一门解释型语言，由**Erlang虚拟机**解释执行。RabbitMQ将queue、exchange、bindings等信息存储在Erlang的分布式数据库mnesia中。

下图为安装rabbitmq-server所需要的erlang依赖包：

### 2.2 管理命令

systemctl start/stop rabbitmq-server #启停rabbitmq服务，包括erlang虚拟机及rabbitmq。

rabbitmqctl start/stop #启停在erlang node上运行的rabbitmq，会使rabbitmq停止。

rabbitmq-server –detached # RabbitMQ 以守护程序的方式在后台运行。

rabbitmqctl start\_app/stop\_app #停止erlang node上的rabbitmq的应用，但是erlang node还是会继续运行的。

rabbitmqctl reset #该命令将清除所有的队列。

rabbitmqctl force\_reset #强制设置RabbitMQ节点到它最开始的状态。与reset不同的是，force\_reset会无条件的重置RabbitMQ，不管当前的管理数据库状态和集 群配置。该命令只应该当数据库和集群配置崩溃后，作为最后一次解决办法。

rabbitmqctl list\_queues #查看已声明的队列。

rabbitmqctl list\_exchanges #查看交换器,该命令还可以附加参数，比如列出交换器的名称、类型、是否持久化、是否自动删除：rabbitmqctl list\_exchanges name type durable auto\_delete.

rabbitmqctl list\_bindings #查看绑定。

参考链接：

<http://rabbitmq.mr-ping.com/AMQP/AMQP_0-9-1_Model_Explained.html>

<https://my.oschina.net/guol/blog/186445>

<https://www.jianshu.com/p/79ca08116d57>