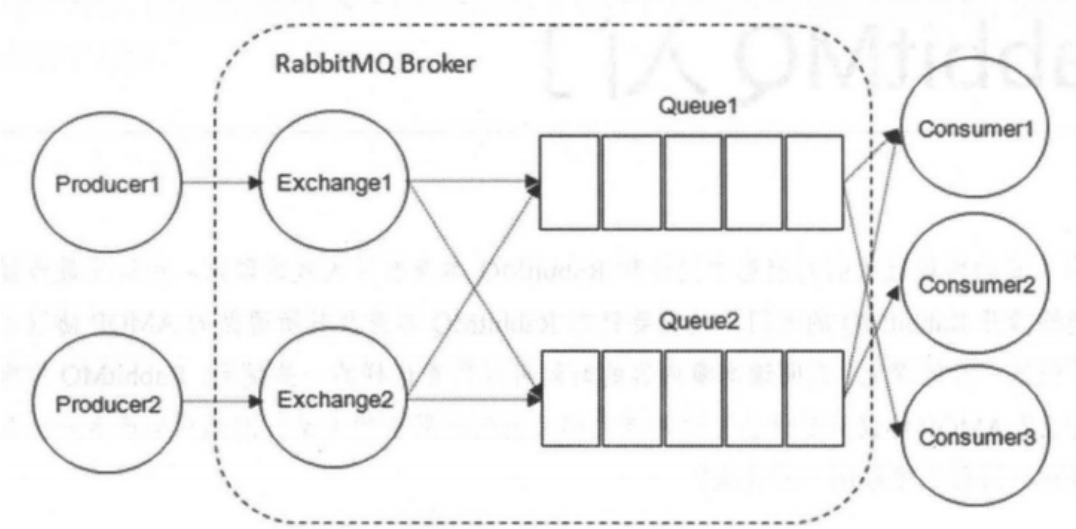
RabbitMQ概念03 - Rabbitmq详解.note

RabbitMq架构模型图：



RabbitMQ Server

也叫Broker Server，它不是运送食物的卡车，而是一种传输服务。它的角色就是维护一条从Producer到Consumer的路线，保证数据能够按照指定的方式进行传输

Client P

消息生产者，就是投递消息的程序。

也叫Producer，数据的发送方。创建一个Message，并且发送给RabbitMQ Server，一个Message有两个部分：payload（有效载荷）和label（标签）。payload顾名思义就是传输的数据。label是exchange的名字或者说是一个tag，它描述了payload，而且RabbitMQ也是通过这个label来决定把这个Message发给哪个Consumer。AMQP仅仅描述了label，而RabbitMQ决定了如何使用这个label的规则

Client C

消息消费者，就是接受消息的程序，也叫Consumer，数据的接收方。

把queue比作是一个有名字的邮箱。当有Message到达某个邮箱后，RabbitMQ把它发送给它的某个订阅者即Consumer。当然可能会把同一个Message发送给很多的Consumer。在这个Message中，只有payload，label已经被删掉了。对于Consumer来说，它是不知道谁发送的这个信息的,就是协议本身不支持。当然了,如果Producer发送的payload包含了Producer的信息就另当别论了

Connection

就是一个TCP的连接。Producer和Consumer都是通过TCP连接到RabbitMQ Server的。以后我们可以看到，程序的起始处就是建立这个TCP连接

Channel

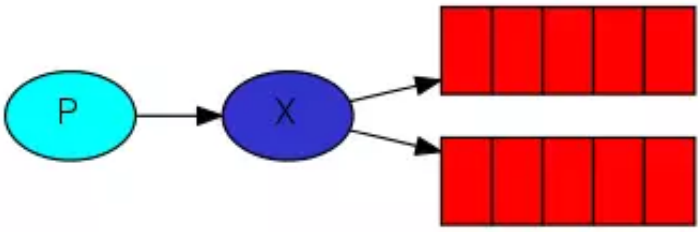
消息通道，在客户端的每个连接里，可建立多个channel，每个channel代表一个会话任务。

那么，为什么使用Channel，而不是直接使用TCP连接?

对于操作系统来说，建立和关闭TCP连接是有代价的，频繁的建立关闭TCP连接对于系统的性能有很大的影响，而且TCP的连接数也有限制，这也限制了系统处理高并发的能力。但是，在TCP连接中建立Channel是没有上述代价的。

Exchange

生产者将消息发送到Exchange（交换器，下图中的X），由Exchange将消息路由到一个或多个Queue中（或者丢弃）



Queue

Queue（队列）是RabbitMQ的内部对象，用于存储消息，RabbitMQ中的消息都只能存储在Queue中，生产者生产消息并最终投递到Queue中，消费者可以从Queue中获取消息并消费

Routing Key、Binding、Binding key（用来处理消息与交换机、队列的绑定，具体使用在交换器类型中详解）

RabbitMQ中通过**Binding**将Exchange与Queue关联起来，会指定一个**Binding key**

生产者在将消息发送给Exchange的时候，会指定一个**Routing Key**

这样RabbitMQ就知道如何正确地将消息路由到指定的Queue了

例如在direct类型的Exchange中：

```
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "rk1");
channel.basicPublish(EXCHANGE_NAME, "rk1", Message类型, Message内容);
```

如上两行代码：绑定指定的Binding key 与 发送消息指定的Routing key一致，才能将消息发送给该队列

但是 Binding key并不是在所有情况下都生效，它依赖于Exchange Type ， 上面只是以direct类型举例

PS： BindingKey 其实也属于路由键中的一种，包括官方文档和 RabbitMQ Java API
中都把 BindingKey 和 RoutingKey 看作 RoutingKey，只不过一个用在绑定阶段，一个用在消息发送阶段

交换器详解

交换器创建参数说明

创建一个交换器有多个重载，参数如下：

- exchange: 交换器名称
- type : 交换器类型 direct、 fanout、 topic、 headers
- durable: 是否持久化,durable设置为true表示持久化,反之是非持久化,持久化的可以将交换器存盘,在服务器重启的时候不会丢失信息
- autoDelete是否自动删除，true表示自动删除。

但是自动删除有一个前提条件：至少有一个队列或者交换器和这个交换器绑定，

然后进行解绑。

例如：有A、B两个队列与这个交换器进行了绑定，该交换器设置了自动删除

true，那么当A、B两个队列都删掉（或者都与这个交换器解绑），这个交

换器会在A、B两个队列全部都跟它解绑完毕的时候自动删除（即使设置

的持久化）

- internal 是否内置,如果设置 为true,则表示是内置的交换器,客户端程序无法直接发送消息到这个交换器中,只能通过交换器路由到交换器的方式
- arguments:其它一些结构化参数比如:alternate-exchange

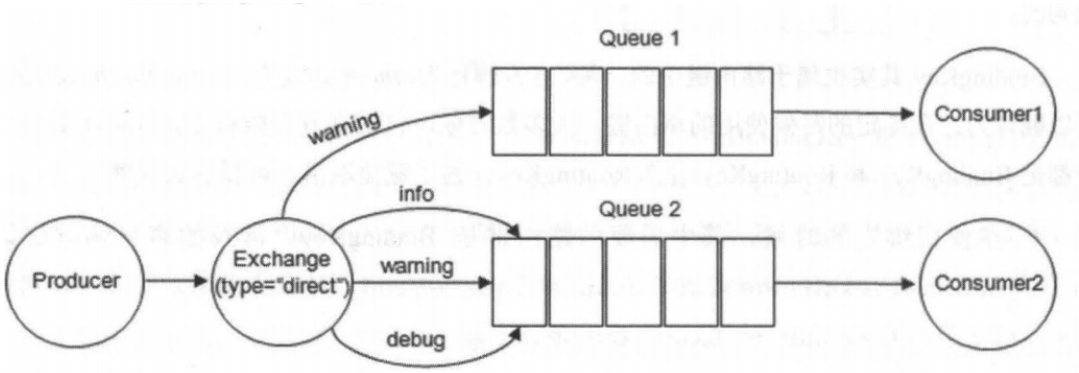
交换器类型与路由详解

- **fanout**

它会把所有发送到该交换器的消息路由到所有与该交换器绑定的队列中

- **direct**

它会把消息路由到那些Binding key与Routing key完全匹配的Queue中，



如上图，交换器与queue1绑定的路由键为 warning，交换器与queue2绑定的路由键有三个info、warning、debug

如果我们发送一条消息，并在发送消息的时候设置路由键为"warning"，则消息会路由

到 Queuel 和 Queue2

如果在发送消息的时候设置路由键为" info" 或者 "debug"，消息只会路由到 Queue2

如果以其他的路由键发送消息，则消息不会路由到这两个队列中

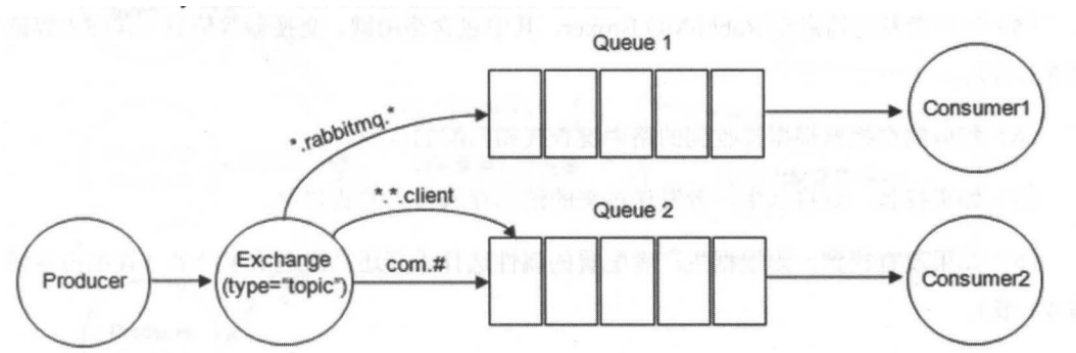
- **topic**

它与direct类型的Exchage相似，也是将消息路由到Binding Key与Routing

Key相匹配的Queue中，但这里的匹配规则有些不同，这里是采取的通配符

形式

其中"*"用于匹配一个单词，"#"用于匹配多个单词（可以是零个）



路由键为" com.rabbitmq.client" 的消息会同时路由到 Queuel 和 Queue2;

路由键为" com.hidden.client" 的消息只会路由到 Queue2 中:

路由键为" com.hidden.demo" 的消息只会路由到 Queue2 中:

路由键为 "java.rabbitmq.demo" 的消息只会路由到 Queuel 中:

路由键为" java.uti l.concurrent" 的消息将会被丢弃或者返回给生产者(需要设置

mandatory 参数)，因为它没有匹配任何路由键。

- **headers**

headers类型的Exchange不依赖于Routing Key与Binding Key的匹配规则来

路由消息，而是根据发送的消息内容中的headers属性进行匹配。

在绑定Queue与Exchange时指定一组键值对；当消息发送到Exchange时，

RabbitMQ会取到该消息的headers（也是一个键值对的形式），对比其中的

键值对是否完全匹配Queue与Exchange绑定时指定的键值

对。如果完全匹配

则消息会路由到该Queue，否则不会路由到该Queue

队列详解

队列创建参数说明

- queue 队列名称
- durable 是否持久化
- exclusive 是否排他

如果一个队列被声明为排他队列，该队列仅对首次声明它的连接可见，并在

连接断开时自动删除。

这里需要注意三点:

1 排他队列是基于连接(Connection) 可见的，同一个连接的不同信道

(Channel)是可以同时访问同一连接创建的排他队列

2 "首次"是指如果一个连接己经声明了 一个排他队列，其他连接是不允

许建立同名的排他队列的

3 即使该队列是持久化的，一旦连接关闭或者客户端退出，该排他队列都

会被自动删除，这种队列适用于一个客户端同时发送和读取消息的应用

场景。

- autoDelete 是否自动删除

自动删除的前提是:至少有一个消费者连接到这个队列，之后所有与

这个队列连接的消费者都断开时，才会自动删除（类似交换器）

- argurnents 设置队列的其他一些参数