介绍

MQ全称为Message Queue，即消息队列， RabbitMQ是由erlang语言开发，基于AMQP（Advanced Message Queue 高级消息队列协议）协议实现的消息队列

开发中消息队列的应用场景：

1、**任务异步处理。**

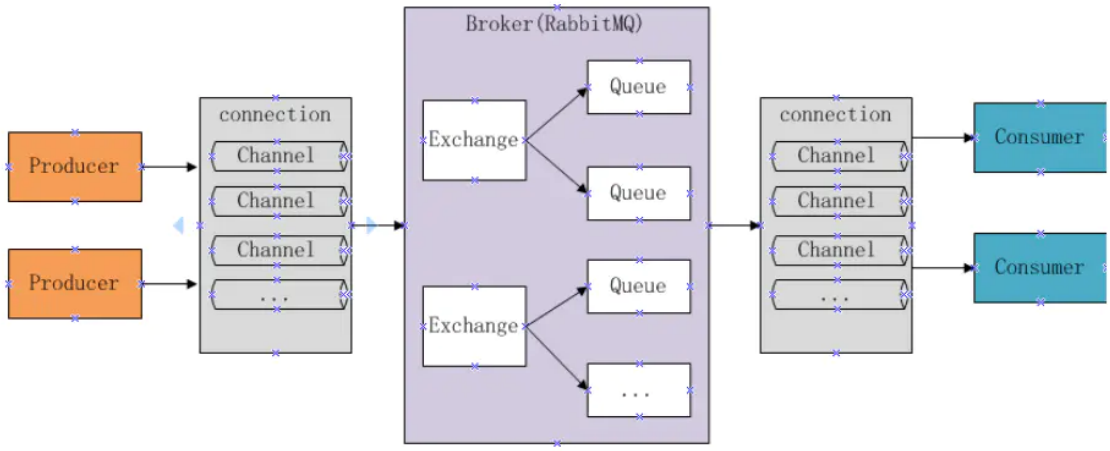
将不需要同步处理的并且耗时长的操作由消息队列通知消息接收方进行异步处理。提高了应用程序的响应时间。

2、**应用程序解耦合**

MQ相当于一个中介，生产方通过MQ与消费方交互，它将应用程序进行解耦合。（无视消息来源传递消息，不受客户端、消息中间件、不同的开发语言环境等条件的限制；）

RabbitMQ的工作原理

下图是RabbitMQ的基本结构：



- Broker(Server)：消息队列服务进程，此进程包括两个部分：Exchange和Queue。
- Exchange：消息队列交换机，按一定的规则将消息路由转发到某个队列，对消息进行过滤。
  - Exchange Type:交换机类型决定了路由消息行为，RabbitMQ中有三种类型Exchange，分别是fanout、direct、topic
- Message Queue：消息队列，存储消息的队列，消息到达队列并转发给指定的消费方。
- Producer：消息生产者，即生产方客户端，生产方客户端将消息发送到MQ。
- Consumer：消息消费者，即消费方客户端，接收MQ转发的消息。

消息发布接收流程：

发送消息

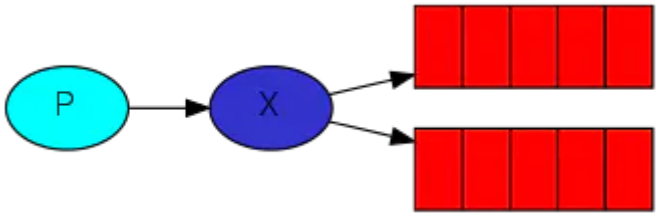
1、生产者和Broker建立TCP连接。

2、生产者和Broker建立通道。

3、生产者通过通道消息发送给Broker，由Exchange将消息进行转发。

Exchange

实际的情况是，生产者将消息发送到Exchange（交换机，下图中的X），由Exchange将消息路由到一个或多个Queue中（或者丢弃）。



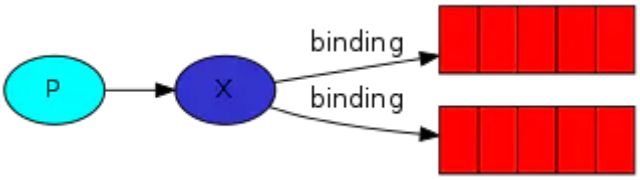
outing key

生产者在将消息发送给Exchange的时候，一般会指定一个routing key（当然也可以不指定），来指定这个消息的路由规则，而这个routing key需要与Exchange Type及binding key联合使用才能最终生效。

在Exchange Type与binding key固定的情况下，我们的生产者就可以在发送消息给Exchange时，通过指定routing key来决定消息流向哪里。RabbitMQ为routing key设定的长度限制为255 bytes。

Binding

RabbitMQ中通过Binding将Exchange与Queue关联起来，这样RabbitMQ就知道如何正确地将消息路由到指定的Queue了。



Binding key

在绑定（Binding）Exchange与Queue的同时，一般会指定一个binding key；消费者将消息发送给Exchange时，一般会指定一个routing key；当binding key与routing key相匹配时，消息将会被路由到对应的Queue中。
在绑定多个Queue到同一个Exchange的时候，这些Binding允许使用相同的binding key。
binding key并不是在所有情况下都生效，它依赖于Exchange Type，比如fanout类型的Exchange就会无视binding key，而是将消息路由到所有绑定到该Exchange的Queue。

Exchange Types

RabbitMQ常用的Exchange Type有fanout、direct、topic、headers这四种（AMQP规范里还提到两种Exchange Type，分别为system与自定义，这里不予以描述），下面分别进行介绍。

4、Exchange将消息转发到指定的Queue（队列）

接收消息

1、消费者和Broker建立TCP连接2、消费者和Broker建立通道

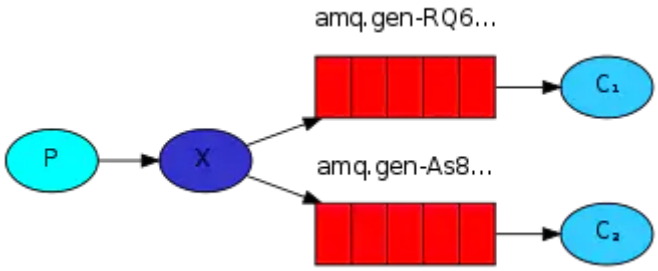
3、消费者监听指定的Queue（队列）

4、当有消息到达Queue时Broker默认将消息推送给消费者。

5、消费者接收到消息。

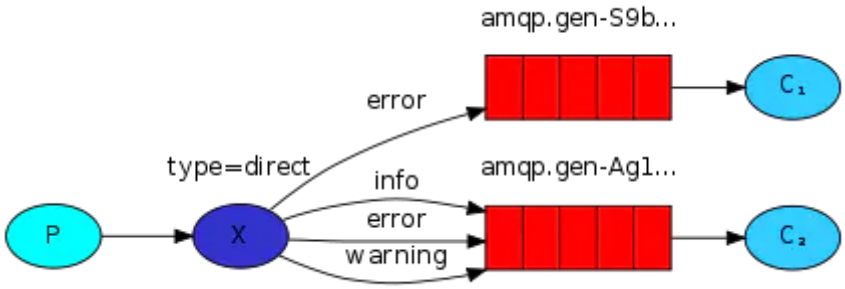
Exchange Types

`fanout`类型的`Exchange`路由规则非常简单，它会把所有发送到该`Exchange`的消息路由到所有与它绑定的`Queue`中。（比较常用的绑定类型)



direct

`direct类型`的`Exchange`路由规则也很简单，它会把消息路由到那些`binding key`与`routing key`完全匹配的`Queue`中。



以上图的配置为例，我们以`routingKey="error"`发送消息到`Exchange`，则消息会路由到`Queue1（amqp.gen-S9b…`，这是由`RabbitMQ`自动生成的`Queue名称`）和`Queue2（amqp.gen-Agl…）`；如果我们以`routingKey="info"`或`routingKey="warning"`来发送消息，则消息只会路由到`Queue2`。如果我们以其他`routingKey`发送消息，则消息不会路由到这两个`Queue`中。

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`。

rabbitMQ的推模式和拉模式

我们知道，消费者有两种方式从消息中间件获取消息：
推模式：消息中间件主动将消息`推送`给消费者
拉模式：消费者主动从消息中间件`拉取`消息

推模式：将消息提前推送给消费者，消费者必须设置一个**缓冲区缓存**这些消息。好处很明显，消费者总是有一堆在内存中待处理的消息，所以效率高。**缺点：**是缓冲区可能会溢出。
实现**推模式**推荐的方式是继承`DefaultConsumer`基类，也可以使用`Spring AMQP`的`SimpleMessageListenerContainer`。

拉模式：在消费者需要时才去消息中间件拉取消息，这段网络开销会明显**增加消息延迟，降低系统吞吐量。**
实现拉模式`RabbitMQ`的`Channel`提供了`basicGet`方法用于拉取消息。

`push`更关注实时性，`pu11`更关注消费者消费能力

推模式是做最常用的，但是某些情况下推模式并不适用：

- 由于某些限制，消费者在某个条件成立时才能消费消息。
- 需要批量拉取消息进行处理。