文章主要是两部分，第一部分介绍 RabbitMQ 的基础信息，网上有很多，如果对 RabbitMQ 比较熟悉可以略过，第二部分主要讲述遇到的问题和影响性能的一些因素以及解决办法。

第一部分

RabbitMQ 基础介绍

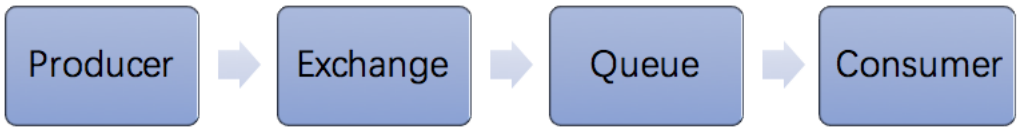
RabbitMQ 是基于 Erlang 语言实现 AMQP（高级消息队列协议）的消息中间件的一种，最初起源于金融系统，用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。

RabbitMQ 主要是为了实现系统之间的双向解耦而实现的，消息的发送者无需知道消息使用者的存在，反之亦然。

一些 RabbitMQ 术语

- Channel
- Producer
- Exchange
- Queue
- Consumer

数据流



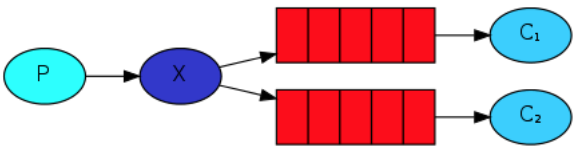
Channel

客户端和服务建立连接，通过 Channel 向 Exchange 发送消息，Exchange 通过 Routing Key 将消息路由到 Queue （Exchange 和 Queue 通过 Routing Key 建立绑定关系，Routing Key 也可以为空）。

Exchange （交换机）

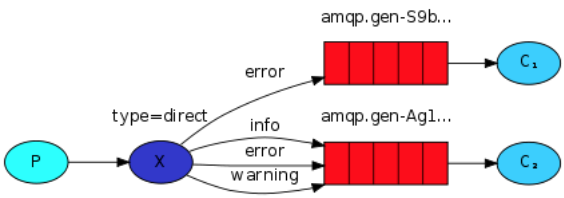
接收消息且转发消息到绑定队列，Producer 只能将消息发给 Exchange，由 Exchange 将消息转发到绑定的 Queue，Exchange 常用的几种类型工作模式，fanout、direct、topic。

fanout 模式即为广播模式，Exchange 会把同一份消息发送给所有的绑定队列，每一个队列收到的消息是相同的。



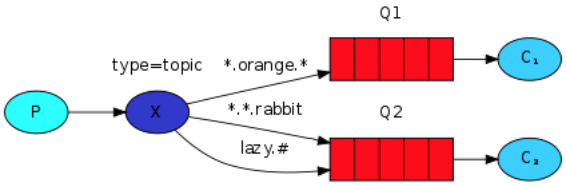
生产者 P 通过 Exchange X 将消息发给两个队列，C1、C2 分别消费对应的队列，他们得到的数据是相同的。

direct 模式通过 RoutingKey 将消息发送给指定的队列



生产者 P 通过 Exchange X 将 error 消息发送到 amqp.gen-S9b 队列，将 info、error、warning 消息发送到 amqp.gen-A1 队列，他们分别对应着消费者 C1、C2。

topic 模式按照规则转发，跟 direct 差不多，只是但是支持模式匹配，通配符等，具有更好的灵活性。



topic 模式下，会对 Routing key 做模式匹配，* 代表匹配一个单词，# 匹配 0 个或多个，*.orange.* 会把指定 Routing key 中间是 orange 的全部路由到 Q1 队列，而 lazy.# 会把是 lazy. 开头的消息分发到 Q2 队列，*.*.rabbit 则代表把前两个是任意词，但是结尾是 .rabbit 的消息分发到 Q2 队列。

交换机的几个属性

- Durabilty 是否持久化
- Auto Delete 是否自动删除

Queue（队列）

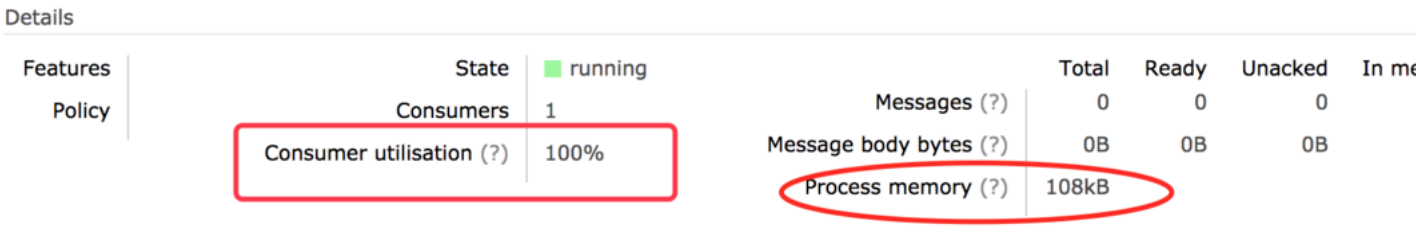
队列是通过 RoutingKey 和 Exchange 绑定在一起的，Queue 的消息都来自 Exchange，Producer 是无法直接像队列发送消息的，一个队列可以和多个 Exchange 绑定在一起。

Queue 的几个属性

- Durabilty
- Auto Delete

第二部分

问题：通过 MQ 监控管理后台发现队列积压严重，内存消耗 6G 多。
使用场景：移动端上报日志，存储到 RabbitMQ，之后服务端消耗 MQ 对日志进行格式化后再次分发。



注意上图中的两个红色框处（当时没有截图），Consumer utilisation 当时在 7% 左右，Process memory 大约 6G 多，生产者每秒大约产生 2000 条消息，Consumer utilisation 代表了消费者的工作效率，一般效率低有几种情况

- 消费者太少
- ACK 回执速度太慢
- 消费者太多

首先我们没有采用 ACK 机制，这样就不存在这个问题，第二 Exchange 和 Queue 在建立的时候，采用了 fanout 模式并且持久化，持久化和非持久化大约有 10 倍的性能差异，如果只是单存的针对同一个队列增加 Consumer，并不会改善效率，而 fanout 的广播模式不利于增加多个 Queue。

解决办法：重建了 Exchange 和 Queue，采用 direct 模式且非持久化的方式，对同一个 Exchange 绑定了 5 个 Queue，生产者随机的将消息分发到某个队列，每个 Queue 会对应一个消费者。因为消息本身是日志，日志有时间的，所以不存在时序的问题，这样就大大的提高了消费者的工作效率，通过这样的改进以后，Consomer utilisation 基本处在 100%，而且也没有出现队列积压。

内存与流量控制参数

- prefetch 是每次从一次性从 broker 里面取的待消费的消息的个数，值太大会增加延迟，太小会导致消息积压。
- vm_memory_high_watermark 内存流量控制，默认 0.4（还可以是绝对值），当占用物理内存的 40% 时，它会引起一个内存报警并且阻塞所有连接。 百分比情况下可使用内存 vm_memory_limit = vm_memory_high_watermark * 物理内存，绝对值情况下 vm_memory_limit = vm_memory_high_watermark。

```
$ rabbitmqctl status | grep vm_memory
 {vm_memory_high_watermark,0.4},
 {vm_memory_limit,40530786713},
```

- vm_memory_high_watermark_paging_ratio 确定了何时执行消息从内存转移到硬盘，默认 0.5，当内存消耗 vm_memory_limit * 0.5 时，开始从内存转移到硬盘。

PS:
使用 RabbitMQ 时，创建 Exchange 和 Queue 要注意自己的使用场景，是否需要持久化，是否需要 ACK 机制，消息分发模式，是否有时序要求等等，千万不要随便建个 fanout 模式放那就用。

参考

https://www.rabbitmq.com/documentation.html
https://emacsist.github.io/tags/#rabbitmq
http://lynnkong.iteye.com/blog/1699684
https://www.gitbook.com/book/geewu/rabbitmq-quick