RabbitMQ Backing Queue结构

**X**

Amqqueue

Backing queue

binding

MQ

in

out

Rabbitmq大体上可以分为两部分（Exchange和MQ），所有发送给RabbitMQ的消息都会先交给Exchange， Exchange的功能类似于路由器，它会根据自身类型（fanout、direct、topic）以及binding信息决定一个消息该被放到哪一个MQ， 而MQ的功能在于暂时存储消息，并将MQ中的消息以订阅或者poll的方式交给接收方。

MQ内部大致又可以分为两部分:amqueue和backing queue, amqqueue负责实现amqp协议规定的mq的基本逻辑，backing queue则实现消息的存储，它会尽量为durable=true的消息做持久化的存储，而在内存不足时将一部分消息放入DISK换取更多的内存空间。Backing queu内部又细分为5各小Q，

Q1

Q2

delta

Q3

Q4

消息在这些Q中传递的“一般”过程q1->q2->delta->q3->q4，这几个Q实现的是

“RAM ->DISK->RAM”这一过程中对消息的分类管理。大多数情况下，一个消息并非需要走完每个小Q，通常大部分都可以略过。与这5各Q对应，在backing queue中消息的生命周期可以分为四个状态：

Alpha：该消息的位置信息和消息本身都在RAM中，这类消息排列在Q1和Q4。

Beta：消息的位置保存在RAM中，消息本身保存在DISK中，这类消息排列在Q2或Q3中。

Gamma: 消息的位置保存RAM和DISK中，消息本身保存在DISK中，这类消息排列在Q2或Q3中。

Delta：消息的位置和消息本身都保存在DISK中，这类消息排列在delta中。

从Q1->Q2->delta这一个过程是将消息逐步从RAM移动到DISK的过程，而delta->Q3->Q4是从DISK逐步移动到RAM的过程。

通常在负载正常时，一个消息不会经历每种状态，如果消息被消费的速度不小于接收新消息的速度，对于不需要保证可靠不丢的消息极可能只会有Alpha状态。对于durable=true的消息，它一定会进入gamma，若开启publish confirm，只有到了这个阶段才会确认该消息已经被接收，若消息消费的速度足够快，内存也充足，这些消息也不会继续走到下一状态。

从上述backing queue对消息的处理过程可以看出，消息若能尽早被消费掉即在不要走完这5个队列，尽量在q1或q2中就被消费掉，就能减少系统的开销。若走的“太深”则会有内存的换入换出增加系统开销。这样就存在一个问题：

通常在系统负载较高时，已接收到的消息若不能很快的被消费掉，这些消息就会进入到很深的队列中去，增加处理每个消息的平均开销。因为要花更多的时间和资源处理“积压”的消息，所以用于处理新来的消息的能力就会降低，使得后来的消息又被积压进入很深的队列，继续加大处理每个消息的平均开销，这样情况就会越来越恶化，使得系统的处理能力大大降低。

根据官方博客，应对这一问题，有三个措施：

1. 进行流量控制。
2. 增加prefetch的值，即一次发送多个消息给接收者，加快消息被消费掉的速度。
3. 才有mutli ack，降低处理ack带来的开销。

目前RabbitMQ已经有了很好的流量控制机制，通过前面多次的测试，在测试过程中观察到的现象是，MQ中堆积的消息数一直都很少（低于5个）。需要使用者做的就是2，3两点。