介绍RabbitMQ前，有必须先了解一下AMQP协议。AMQP协议是一个高级抽象层消息通信协议，RabbitMQ是AMQP协议的实现。它主要包括以下组件：

1. Server(broker): 接受客户端连接，实现AMQP消息队列和路由功能的进程。

2. Virtual Host:其实是一个虚拟概念，类似于权限控制组，一个Virtual Host里面可以有若干个Exchange和Queue，但是权限控制的最小粒度是Virtual Host

3.Exchange:接受生产者发送的消息，并根据Binding规则将消息路由给服务器中的队列。ExchangeType决定了Exchange路由消息的行为，例如，在RabbitMQ中，ExchangeType有direct、Fanout和Topic三种，不同类型的Exchange路由的行为是不一样的。

4.Message Queue：消息队列，用于存储还未被消费者消费的消息。

5.Message: 由Header和Body组成，Header是由生产者添加的各种属性的集合，包括Message是否被持久化、由哪个Message Queue接受、优先级是多少等。而Body是真正需要传输的APP数据。

6.Binding:Binding联系了Exchange与Message Queue。Exchange在与多个Message Queue发生Binding后会生成一张路由表，路由表中存储着Message Queue所需消息的限制条件即Binding Key。当Exchange收到Message时会解析其Header得到Routing Key，Exchange根据Routing Key与Exchange Type将Message路由到Message Queue。Binding Key由Consumer在Binding Exchange与Message Queue时指定，而Routing Key由Producer发送Message时指定，两者的匹配方式由Exchange Type决定。

7.Connection:连接，对于RabbitMQ而言，其实就是一个位于客户端和Broker之间的TCP连接。

8.Channel:信道，仅仅创建了客户端到Broker之间的连接后，客户端还是不能发送消息的。需要为每一个Connection创建Channel，AMQP协议规定只有通过Channel才能执行AMQP的命令。一个Connection可以包含多个Channel。之所以需要Channel，是因为TCP连接的建立和释放都是十分昂贵的，如果一个客户端每一个线程都需要与Broker交互，如果每一个线程都建立一个TCP连接，暂且不考虑TCP连接是否浪费，就算操作系统也无法承受每秒建立如此多的TCP连接。RabbitMQ建议客户端线程之间不要共用Channel，至少要保证共用Channel的线程发送消息必须是串行的，但是建议尽量共用Connection。

9.Command:AMQP的命令，客户端通过Command完成与AMQP服务器的交互来实现自身的逻辑。例如在RabbitMQ中，客户端可以通过publish命令发送消息，txSelect开启一个事务，txCommit提交一个事务。

在了解了AMQP模型以后，需要简单介绍一下AMQP的协议栈，AMQP协议本身包括三层：

1.       Modle Layer，位于协议最高层，主要定义了一些供客户端调用的命令，客户端可以利用这些命令实现自己的业务逻辑，例如，客户端可以通过queue.declare声明一个队列，利用consume命令获取一个队列中的消息。

2.       Session Layer，主要负责将客户端的命令发送给服务器，在将服务器端的应答返回给客户端，主要为客户端与服务器之间通信提供可靠性、同步机制和错误处理。

3.       Transport Layer，主要传输二进制数据流，提供帧的处理、信道复用、错误检测和数据表示。

从AMQP协议可以看出，MessageQueue、Exchange和Binding构成了AMQP协议的核心，下面我们就围绕这三个主要组件    从应用使用的角度全面的介绍如何利用Rabbit MQ构建消息队列以及使用过程中的注意事项。

1. 声明MessageQueue

      在Rabbit MQ中，无论是生产者发送消息还是消费者接受消息，都首先需要声明一个MessageQueue。这就存在一个问题，是生产者声明还是消费者声明呢。要解决这个问题，首先需要明确：

a)消费者是无法订阅或者获取不存在的MessageQueue中信息。

b)消息被Exchange接受以后，如果没有匹配的Queue，则会被丢弃。

在明白了上述两点以后，就容易理解如果是消费者去声明Queue，就有可能会出现在声明Queue之前，生产者已发送的消息被丢弃的隐患。如果应用能够通过消息重发的机制允许消息丢失，则使用此方案没有任何问题。但是如果不能接受该方案，这就需要无论是生产者还是消费者，在发送或者接受消息前，都需要去尝试建立消息队列。这里有一点需要明确，如果客户端尝试建立一个已经存在的消息队列，Rabbit MQ不会做任何事情，并返回客户端建立成功的。

       如果一个消费者在一个信道中正在监听某一个队列的消息，Rabbit MQ是不允许该消费者在同一个channel去声明其他队列的。Rabbit MQ中，可以通过queue.declare命令声明一个队列，可以设置该队列以下属性:

a) Exclusive：排他队列，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：其一，排他队列是基于连接可见的，同一连接的不同信道是可以同时访问同一个连接创建的排他队列的。其二，“首次”，如果一个连接已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，这个与普通队列不同。其三，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的。这种队列适用于只限于一个客户端发送读取消息的应用场景。

b)   Auto-delete:自动删除，如果该队列没有任何订阅的消费者的话，该队列会被自动删除。这种队列适用于临时队列。

 c)   Durable:持久化，这个会在后面作为专门一个章节讨论。

d)  其他选项，例如如果用户仅仅想查询某一个队列是否已存在，如果不存在，不想建立该队列，仍然可以调用queue.declare，只不过需要将参数passive设为true，传给queue.declare，如果该队列已存在，则会返回true；如果不存在，则会返回Error，但是不会创建新的队列。

2. 生产者发送消息

        在AMQP模型中，Exchange是接受生产者消息并将消息路由到消息队列的关键组件。ExchangeType和Binding决定了消息的路由规则。所以生产者想要发送消息，首先必须要声明一个Exchange和该Exchange对应的Binding。可以通过 ExchangeDeclare和BindingDeclare完成。在Rabbit MQ中，声明一个Exchange需要三个参数：ExchangeName，ExchangeType和Durable。ExchangeName是该Exchange的名字，该属性在创建Binding和生产者通过publish推送消息时需要指定。ExchangeType，指Exchange的类型，在RabbitMQ中，有三种类型的Exchange：direct ，fanout和topic，不同的Exchange会表现出不同路由行为。Durable是该Exchange的持久化属性，这个会在消息持久化章节讨论。声明一个Binding需要提供一个QueueName，ExchangeName和BindingKey。下面我们就分析一下不同的ExchangeType表现出的不同路由规则。

        生产者在发送消息时，都需要指定一个RoutingKey和Exchange，Exchange在接到该RoutingKey以后，会判断该ExchangeType:

                         a) 如果是Direct类型，则会将消息中的RoutingKey与该Exchange关联的所有Binding中的BindingKey进行比较，如果相等，则发送到该Binding对应的Queue中。     
                  b)  如果是 Fanout 类型，则会将消息发送给所有与该 Exchange 定义过 Binding 的所有 Queues 中去，其实是一种广播行为。

        c)如果是Topic类型，则会按照正则表达式，对RoutingKey与BindingKey进行匹配，如果匹配成功，则发送到对应的Queue中。

3. 消费者订阅消息

    在RabbitMQ中消费者有2种方式获取队列中的消息:

       a)  一种是通过basic.consume命令，订阅某一个队列中的消息,channel会自动在处理完上一条消息之后，接收下一条消息。（同一个channel消息处理是串行的）。除非关闭channel或者取消订阅，否则客户端将会一直接收队列的消息。

             b)  另外一种方式是通过basic.get命令主动获取队列中的消息，但是绝对不可以通过循环调用basic.get来代替basic.consume，这是因为basic.get RabbitMQ在实际执行的时候，是首先consume某一个队列，然后检索第一条消息，然后再取消订阅。如果是高吞吐率的消费者，最好还是建议使用basic.consume。

      如果有多个消费者同时订阅同一个队列的话，RabbitMQ是采用循环的方式分发消息的，每一条消息只能被一个订阅者接收。例如，有队列Queue，其中ClientA和ClientB都Consume了该队列，MessageA到达队列后，被分派到ClientA，ClientA回复服务器收到响应，服务器删除MessageA；再有一条消息MessageB抵达队列，服务器根据“循环推送”原则，将消息会发给ClientB，然后收到ClientB的确认后，删除MessageB；等到再下一条消息时，服务器会再将消息发送给ClientA。

       这里我们可以看出，消费者再接到消息以后，都需要给服务器发送一条确认命令，这个即可以在handleDelivery里显示的调用basic.ack实现，也可以在Consume某个队列的时候，设置autoACK属性为true实现。这个ACK仅仅是通知服务器可以安全的删除该消息，而不是通知生产者，与RPC不同。 如果消费者在接到消息以后还没来得及返回ACK就断开了连接，消息服务器会重传该消息给下一个订阅者，如果没有订阅者就会存储该消息。

        既然RabbitMQ提供了ACK某一个消息的命令，当然也提供了Reject某一个消息的命令。当客户端发生错误，调用basic.reject命令拒绝某一个消息时，可以设置一个requeue的属性，如果为true，则消息服务器会重传该消息给下一个订阅者；如果为false，则会直接删除该消息。当然，也可以通过ack，让消息服务器直接删除该消息并且不会重传。

4. 持久化：

        Rabbit MQ默认是不持久队列、Exchange、Binding以及队列中的消息的，这意味着一旦消息服务器重启，所有已声明的队列，Exchange，Binding以及队列中的消息都会丢失。通过设置Exchange和MessageQueue的durable属性为true，可以使得队列和Exchange持久化，但是这还不能使得队列中的消息持久化，这需要生产者在发送消息的时候，将delivery mode设置为2，只有这3个全部设置完成后，才能保证服务器重启不会对现有的队列造成影响。这里需要注意的是，只有durable为true的Exchange和durable为ture的Queues才能绑定，否则在绑定时，RabbitMQ都会抛错的。持久化会对RabbitMQ的性能造成比较大的影响，可能会下降10倍不止。

5. 事务：

     对事务的支持是AMQP协议的一个重要特性。假设当生产者将一个持久化消息发送给服务器时，因为consume命令本身没有任何Response返回，所以即使服务器崩溃，没有持久化该消息，生产者也无法获知该消息已经丢失。如果此时使用事务，即通过txSelect()开启一个事务，然后发送消息给服务器，然后通过txCommit()提交该事务，即可以保证，如果txCommit()提交了，则该消息一定会持久化，如果txCommit()还未提交即服务器崩溃，则该消息不会服务器就收。当然Rabbit MQ也提供了txRollback()命令用于回滚某一个事务。

6. Confirm机制：

      使用事务固然可以保证只有提交的事务，才会被服务器执行。但是这样同时也将客户端与消息服务器同步起来，这背离了消息队列解耦的本质。Rabbit MQ提供了一个更加轻量级的机制来保证生产者可以感知服务器消息是否已被路由到正确的队列中——Confirm。如果设置channel为confirm状态，则通过该channel发送的消息都会被分配一个唯一的ID，然后一旦该消息被正确的路由到匹配的队列中后，服务器会返回给生产者一个Confirm，该Confirm包含该消息的ID，这样生产者就会知道该消息已被正确分发。对于持久化消息，只有该消息被持久化后，才会返回Confirm。Confirm机制的最大优点在于异步，生产者在发送消息以后，即可继续执行其他任务。而服务器返回Confirm后，会触发生产者的回调函数，生产者在回调函数中处理Confirm信息。如果消息服务器发生异常，导致该消息丢失，会返回给生产者一个nack，表示消息已经丢失，这样生产者就可以通过重发消息，保证消息不丢失。Confirm机制在性能上要比事务优越很多。但是Confirm机制，无法进行回滚，就是一旦服务器崩溃，生产者无法得到Confirm信息，生产者其实本身也不知道该消息吃否已经被持久化，只有继续重发来保证消息不丢失，但是如果原先已经持久化的消息，并不会被回滚，这样队列中就会存在两条相同的消息，系统需要支持去重。

消息持久化是 RabbitMQ 最为人津津乐道的特性之一， RabbitMQ 能够在付出最小的性能代价的基础上实现消息的持久化，最大的奥秘就在于 RabbitMQ 多层消息队列的设计上。下面，本文就从 MessageQueue 的设计和消息在 MessageQueue 的生命周期两个方面全面介绍  RabbitMQ 的消息队列。

   RabbitMQ完全实现了AMQP协议，类似于一个邮箱服务。Exchange负责根据ExchangeType和RoutingKey将消息投递到对应的消息队列中，消息队列负责在消费者获取消息前暂存消息。在RabbitMQ中，MessageQueue主要由两部分组成，一个为AMQQueue，主要负责实现AMQP协议的逻辑功能。另外一个是用来存储消息的BackingQueue，本文重点关注的是BackingQueue的设计。

    在RabbitMQ中BackingQueue又由5个子队列组成：Q1、Q2、Delta、Q3和Q4。RabbitMQ中的消息一旦进入队列，不是固定不变的，它会随着系统的负载在队列中不断流动，消息的状态不断发生变化。RabbitMQ中的消息一共有5种状态：

   a)Alpha：消息的内容和消息索引都保存在内存中；

   b)Beta：消息内容保存在磁盘上，消息索引保存在内存中；

   c)Gamma：消息内容保存在磁盘上，消息索引在磁盘和内存都有；

   d)Delta：消息内容和索引都在磁盘上；

   注意：对于持久化的消息，消息内容和消息索引都必须先保存到磁盘上，才会处于上述状态中的一种，而Gamma状态的消息只有持久化的消息才会有该状态。

      BackingQueue 中的 5 个子队列中的消息状态， Q1 和 Q4 对应的是 Alpha 状态， Q2 和 Q3 是 Beta 状态， Delta 对应的是 Delta 状态。上述就是 RabbitMQ 的多层队列结构的设计，我们可以看出从 Q1 到 Q4 ，基本经历的是由 RAM 到 DISK，再到 RAM 的设计。这样的设计的好处就是当队列负载很高的情况下，能够通过将一部分消息由磁盘保存来节省内存空间，当负载降低的时候，这部分消息又渐渐回到内存，被消费者获取，使得整个队列有很好的弹性。下面我们就来看一下，整个消息队列的工作流程。      引起消息流动主要有两方面的因素：其一是消费者获取消息；其二是由于内存不足，引起消息的换出到磁盘上（ Q1-.>Q2 、 Q2->Delta 、 Q3->Delta 、 Q4->Q3 ）。 RabbitMQ 在系统运行时会根据消息传输的速度[计算](https://www.aliyun.com/)一个当前内存中能够保存的最大消息数量（ Target\_RAM\_Count ），当内存中的消息数量大于该值时，就会引起消息的流动。进入队列的消息，一般会按着 Q1->Q2->Delta->Q3->Q4 的顺序进行流动，但是并不是每条消息都一定会经历所有的状态，这个取决于当时系统的负载状况。        当消费者获取消息时，首先会从 Q4 队列中获取消息，如果 Q4 获取成功，则返回，如果 Q4 为空，则尝试从 Q3 获取消息；首先，系统会判断 Q3 队列是否为空，如果为空，则直接返回队列为空，即此时队列中无消息（后续会论证）。如果不为空，则取出 Q3 的消息，然后判断此时 Q3 和 Delta 队列的长度，如果都为空，则可认为 Q2 、 Delta 、 Q3 和 Q4 全部为空 (后续说明 ) ，此时将 Q1 中消息直接转移到 Q4 中，下次直接从 Q4 中获取消息。如果 Q3 为空， Delta 不空，则将 Delta 中的消息转移到 Q3 中；如果 Q3 非空，则直接下次从 Q3 中获取消息。在将 Delta 转移到 Q3 的过程中， RabbitMQ 是按照索引分段读取的，首先读取某一段，直到读到的消息非空为止，然后判断读取的消息个数与 Delta 中的消息个数是否相等，如果相等，则断定此时 Delta 中已无消息，则直接将 Q2 和刚读到的消息一并放入 Q3 中。如果不相等，则仅将此次读到的消息转移到Q3 中。这就是消费者引起的消息流动过程。

     下面我们分析一下由于内存不足引起的消息换出。消息换出的条件是内存中保存的消息数量 + 等待 ACK 的消息的数量 >Target\_RAM\_Count 。当条件触发时，系统首先会判断如果当前进入等待 ACK 的消息的速度大于进入队列的消息的速度时，会先处理等待 ACK 的消息。步骤基本上 Q1->Q2 或者 Q3 移动，取决于 Delta 队列是否为空。 Q4->Q3 移动， Q2 和Q3 向 Delta 移动。     最后，我们来分析一下前面遗留的两个问题，一个是为什么 Q3 队列为空即可认定整个队列为空。试想如果 Q3 为空，Delta 不空，则在 Q3 取出最后一条消息时， Delta 上的消息就会被转移到 Q3 上，与 Q3 空矛盾。如果 Q2 不空，则在 Q3 取出最后一条消息，如果 Delta 为空时，会将 Q2 的消息并入 Q3 ，与 Q3 为空矛盾。如果 Q1 不空，则在 Q3 取出最后一条消息，如果 Delta 和 Q3 均为空时，则将 Q1 的消息转移到 Q4 中，与 Q4 为空矛盾。这也解释了另外一个问题，即为什么 Q3 和Delta 为空， Q2 就为空。     上述就是整个消息在 RabbitMQ 队列中流动过程。从上述流程可以看出，消息如果能够被尽早消费掉，就不需要经历持久化的过程，因为这样会加系统的开销。如果消息被消费的速度过慢， RabbitMQ 通过换出内存的方式，防止内存溢出。