

# 1. AMQ与Rabbit交互

## 1.1 启动会话

客户端	服务器端
协议头	
	connection.start
connection.startOK	

## 1.2 调整正确的信道

在AMQP中，信道使用协商的AMQP连接作为相互传输新的渠道，而且将传输过程与其他正在进行的会话隔离。一个AMQP连接可以有多个信道，允许客户端和服务端之间进行多次会话（多路复用）

# 2. AMQP RPC帧结构

## 2.1 AMQP帧组件

底层AMQP帧由五个不同的组件组成：

1. 帧类型：协议头帧、方法帧、内容头帧、消息体帧、心跳帧
2. 信道编号
3. 以字节为单位的帧大小
4. 帧有效载荷
5. 结束字节标记

帧结构： | 帧头部 | 有效核载 | 结束标记 |

### 2.1.1 帧类型

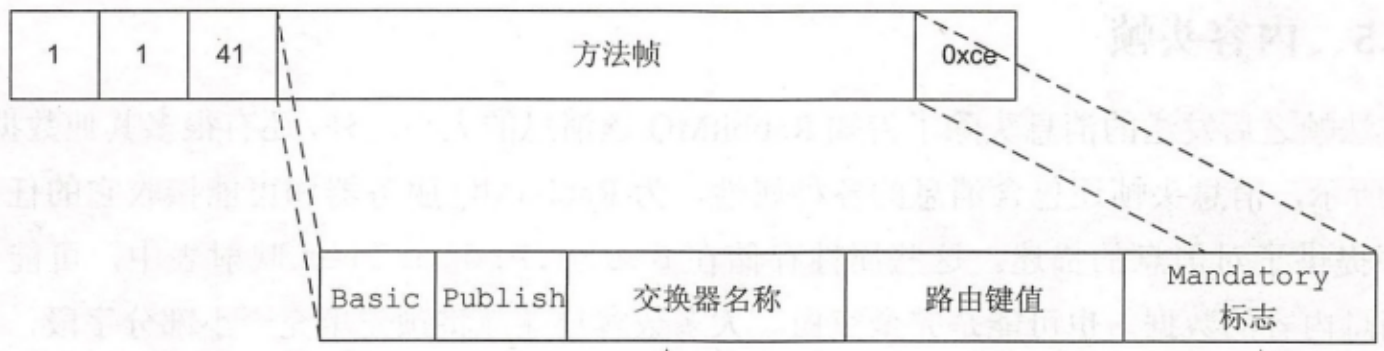
- 协议头帧：用于连接RabbitMQ，仅使用一次
- 方法帧：方法帧携带发送给RabbitMQ或者从RabbitMQ接收到的RPC请求或响应
- 内容头帧：包含一条消息的大小和属性
- 消息体帧：包含消息的内容
- 心跳帧：在客户端和RabbitMQ之间进行传递，作为一种校验机制确保连接的两端可以正常工作

### 2.1.2 将消息编组成帧

向RabbitMQ发送消息时，使用消方法帧、内容头帧和消息体帧，并按照道该顺序发送。如果消息体超过AMQP最大长度限制，则会拆分为多个消息体帧，并按照方法帧、内容头帧和消息体帧的方式发送。

其中方法帧和内容头帧，为了方便传输，有效核载部分为二进制打包数据；而方法体帧为没有进行任何处理的数据

### 2.1.3 方法帧结构

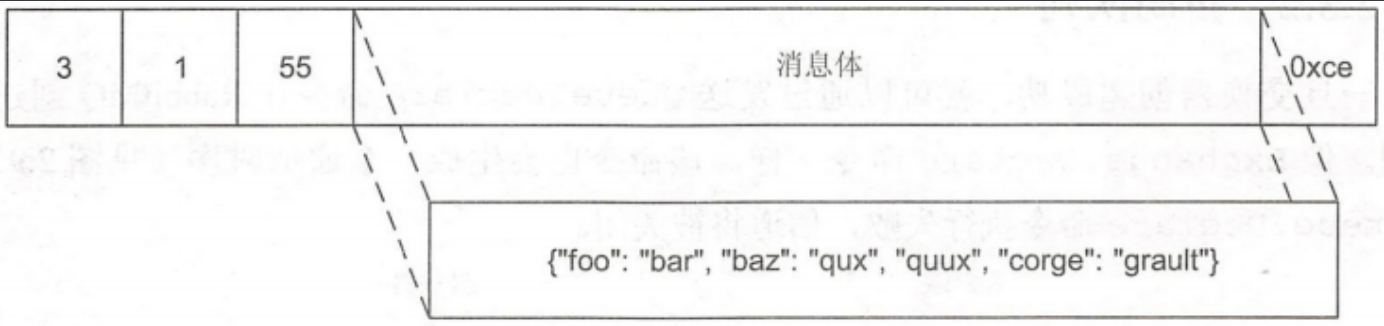


方法帧有效载荷中的每个数据值都是按照数据类型采用特定的格式进行编码，这种编码格式旨在最大限度地减少网络传输中的字节大小，提供数据完整性，并确保数据编组和解组速度尽可能快

### 2.1.4 内容头帧结构



2.1.5 消息体帧结构

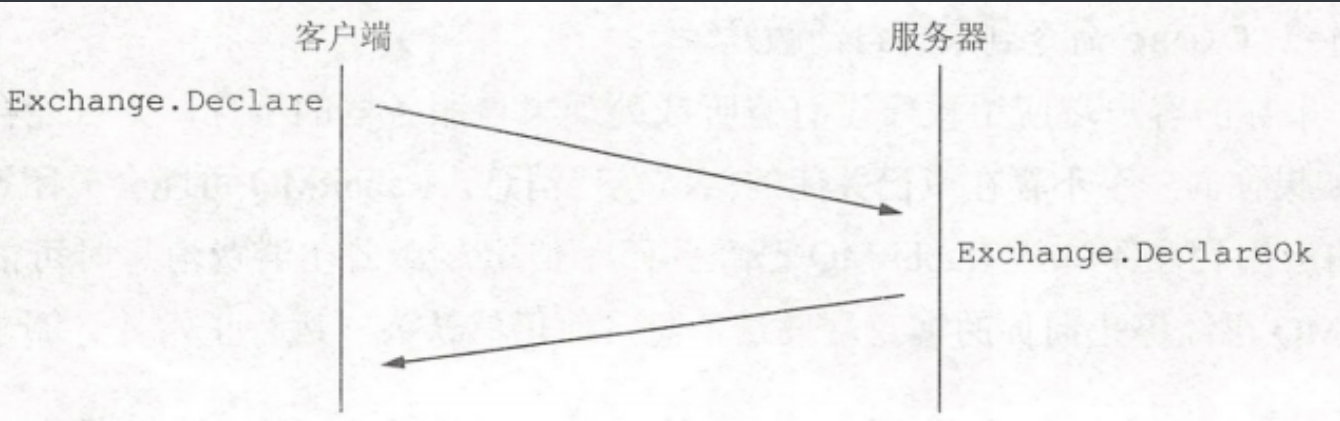


2.2 使用协议

设置交换器和队列，并将交换器和消息队列绑定

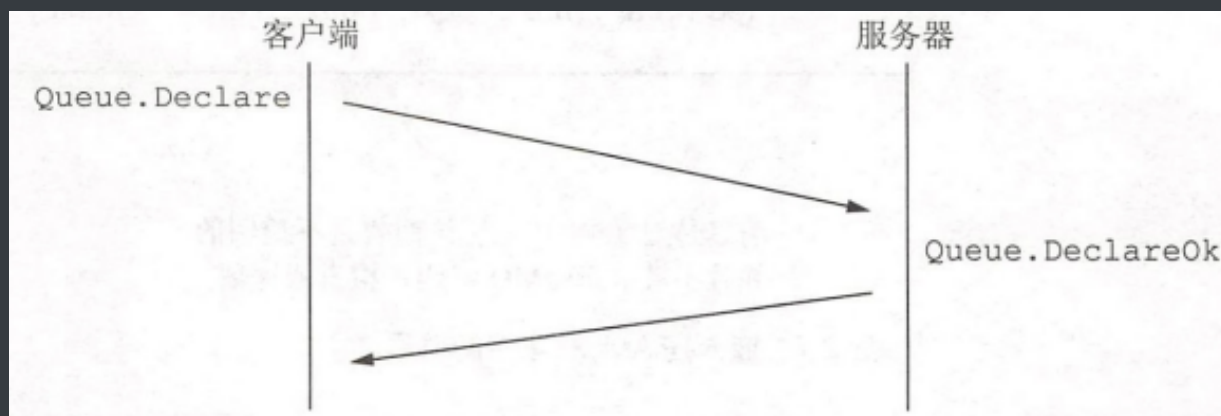
2.2.1 声明交换机exchange

使用Exchange.Declare来建立一个交换器，设置交换器的名称和类型的参数

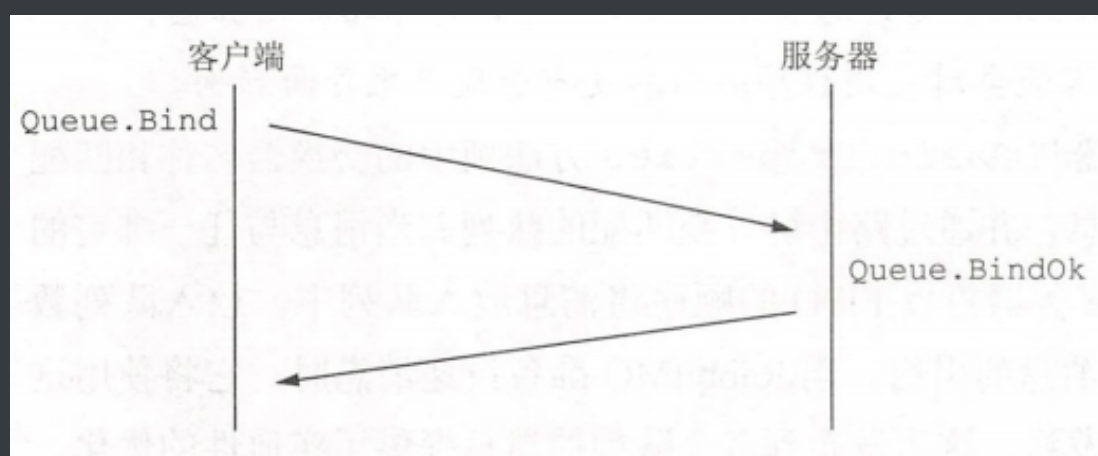


声明失败RabbitMQ使用channel.close命令关闭发送声明交换机的信道。

## 2.2.2 声明一个队列

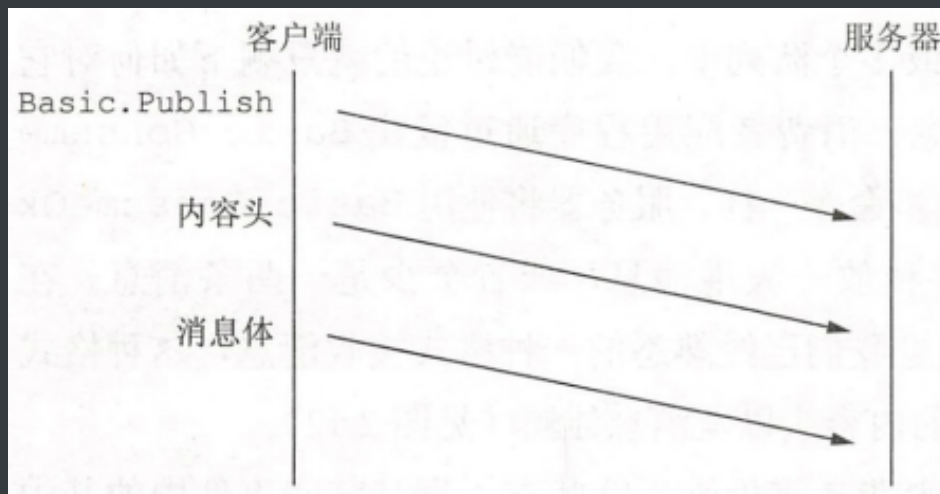


## 2.2.3 绑定队列到交换机



## 2.2.4 发布消息

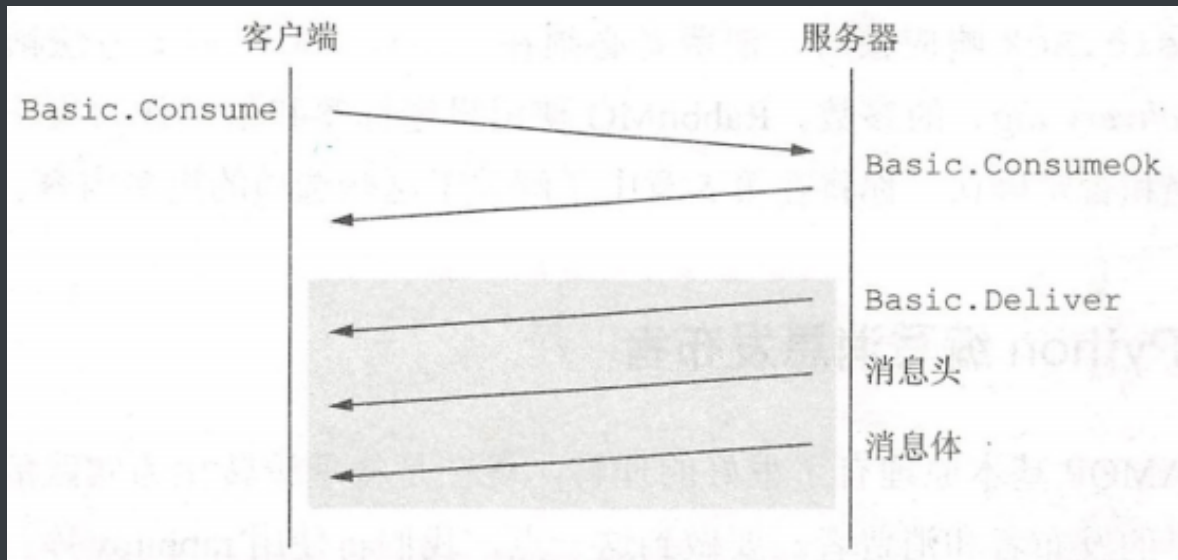
客户端向RabbitMQ发送了一个`basic.publish`的方法头帧、内容头帧和消息体；发布消息至少要发送3个帧：方法头帧，内容头帧和消息体帧。



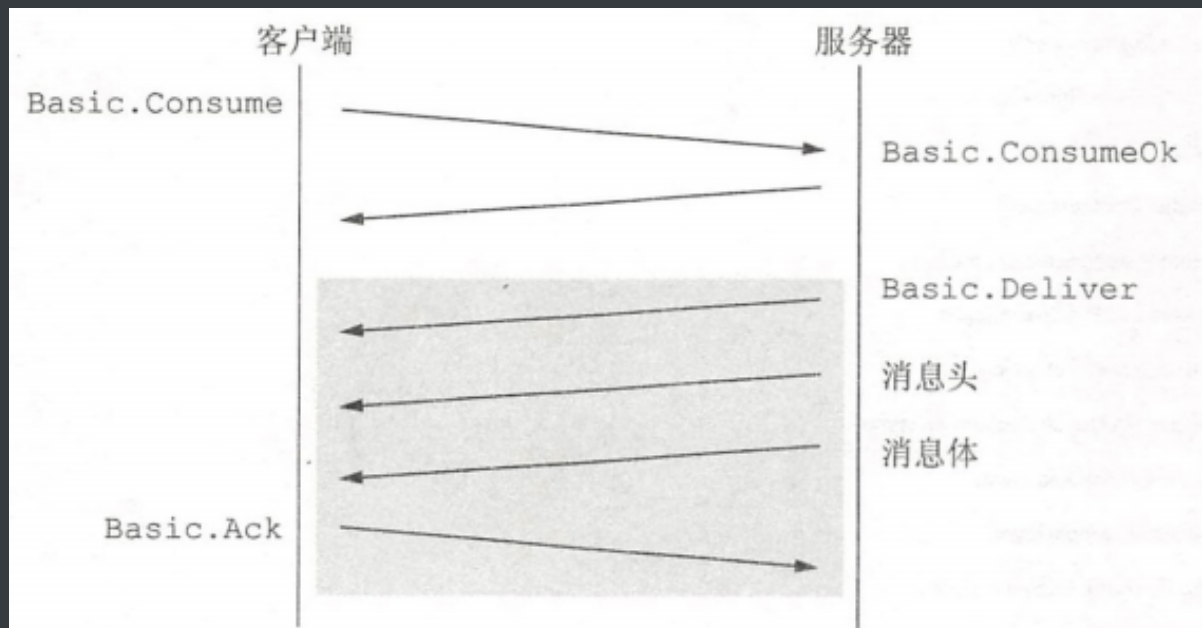
当RabbitMQ接受到一个消息的所有帧并确定下一步操作之前，他将检查方法帧所需要的信息。例如，Basic.Public方法帧携带消息的交换器名称和路由键。RabbitMQ会尝试将Basic.Publish帧中的交换器与配置交换机的数据库进行匹配，如果发现名称相同的交换器，将判断该交换器中的绑定信息，并通过路由键寻找匹配的队列。当消息与任意绑定的队列符合是，按照FIFO的顺序放入队列，放入的不是消息的实体，而是消息的应用。这样只保存一份实体，节省 物理内存。某一个队列中的消息处理方式，无论死被消费、过期还是等待消费，都不会影响该消息在其他队列中的处理方法。当某一个消息的所有副本都被处理，则单个消息数据将被从RabbitMQ的内存中移除。

## 2.2.5 消费消息

消费者发送一个Basic.Consume指令给服务器，而服务器返回一个Basic.ConsumeOK的命令给客户端，表示让客户端做好接收信息的准备。假如消费者想要停止接收消息，可以发送一个Basic.Cancel，这个命令是异步的，表示在接收到服务器响应的Basic.CancelOk之前，消费者还是会继续接收消息。



设置Basic.Consume命令中的no\_ack参数，为true时，RabbitMQ将连续发送消息直到消费者发送一个Basic.Cancel命令或消费者断开连接为止。为false时，消费者必须通过发送Basic.ACK RPC请求来确认收到的每条消息。



# 3. 消息熟悉详解

使用RabbitMQ发布消息时，消息由AMPQ规范中的三个底层帧组成 basic.publish方法帧，内容头帧和消息体帧；

包含在消息体帧（内容头帧）的消息属性时一组预定义的值。

参数	说明
content-type	消息体的MIME类型，如application/json
content-encoding	消息的编码类型，如是否压缩
message-id	消息的唯一性标识，由应用进行设置
correlation-id	一般用做关联消息的message-id，常用于消息的响应
timestamp	消息的创建时刻，整形，精确到秒
expiration	消息的过期时刻，字符串，但是呈现格式为整型，精确到秒
delivery-mode	消息的持久化类型，1为非持久化，2为持久化，性能影响巨大
app-id	应用程序的类型和版本号
user-id	标识已登录用户，极少使用
type	消息类型名称，完全由应用决定如何使用该字段
reply-to	构建回复消息的私有响应队列
headers	键/值对表，用户自定义任意的键和值
priority	指定队列中消息的优先级

1. content-type用于消息体中的数据格式，方便消息发布者和消费者序列化和反序列信息。便于不同语言的应用程序使用。例如JSON、Msgpack、XML等
2. 通过gzip和content-encoding属性压缩消息大小 默认情况AMQP发送的消息不会被压缩。在处理复杂的XML语言时，或者大型的JSON和YAML格式数据时，发布时压缩，接收时解压。
3. 使用messag-id和correlation-id引用消息 他们时应用级别的属性



4. 创建时间：timestamp属性在试图诊断经由RabbitMQ消息流中发生的任何意外行为时非常有用。timestamp来表示消息的创建时间，消费者可以评估消息投递过程的性能。通过该消息可以进一步觉得是否处理消息、丢弃消息，甚至向监控应用程序发布警报消息，以便让其他人知道消息的生存时间已经超过预期值。
5. 消息自动过期 如果消息没有被消费，expiration属性告诉RabbitMQ何时应该丢弃消息。如果把一个已过期的消息发布到服务器，则消息不回路由到任何队列，会被直接丢弃。  
构建延时队列：死信+死信队列
6. delivery-mode平衡速度和安全性：是否在消息被消费之前持久化消息，1表示非持久化信息，2表示持久化信息。
7. type：创建自描述信息
8. headers属性是一个K/V对表，允许用户自定义任意的键和值，RabbitMQ可以根据headers表中填充的值路由消息，而不需要依赖路由键。

## 4 消息发布的性能权衡

### 4.1 消息发布可能遇到的问题

1. 消费发布时保证消息进入队列的重要性有多高？
2. 如果消息无法路由，是否应将消息返回给发布者？
3. 如果消息服务路由，是否应该将其发送到其他地方稍后重新路由
4. 如果RabbitMQ服务器崩溃，可以接受信息丢失吗？
5. RabbitMQ在处理新消息时是否应该确认它已经为发布者执行来所有请求的路由和持久化任务？
6. 消息发布者是否可以批量投递消息，然后从RabbitMQ收到一个确认用于表明所有请求的路由和持久化任务已经批量应用到所有的消息中？
7. 如果要批量发布消息，而这些消息需要去人路由和持久化，那么对每一条消息是否需要为目标队列实现真正意义上的院子提交？
8. 在可靠投递方面是否有可接受的平衡性？
9. 消息发布还有哪些方面会影响消息吞吐量和性能？



## 可靠投递的机制（优化发布者性能）

1. 没有保障：不管消息是否正确投递。
2. 失败返回：设置mandatory，当为true时，表示消息需要强制路由，如果路由失败，会将消息返回给发布者；如果成功，发布者不会收到通知。同时basic.return命令是一个异步命令。
3. 发送确认：采用confirm.select，客户端每次publish一个消息，服务器都会用basic.Ack或者basic.Nack回复。发送确认是事物轻量级的实现。
4. 备用交换机：不可路由、过期的消息成为死信；存储死信息的队列为死信队列。当消息不可被路由时且mandatory为true时，会将消息转发到备用交换机上，通过路由键路由到死信队列中。
5. 高可用队列（HA）：保证消息在队列中不回丢失。HA队列需要RabbitMQ集群环境，它允许多个服务器上拥有冗余副本。HA队列有一个主服务器节点，如果主节点挂掉了，其中一个辅助节点将接管主节点的工作；如果是辅助节点宕机，则其他节点正常运行，共享所有节点上发生的操作状态（类似于redis集群？？）  
  
当使用的是发送确认机制或者事物来发送消息，则只有当HA所有活动节点确定之后，RabbitMQ才会发送成功响应。
6. 事务：事物可以将消息批量发布到RabbitMQ中，然后提交到队列或者回滚。
7. 基于事物的高可用队列
8. 持久化：当delivery-mode=2时，表示需要在消费之前保存在磁盘。RabbitMQ服务器重启之后，消息还存在。delivery-mode=1时表示消息不需要持久化。持久化对浪费大量的IO能力。RabbitMQ将持久化消息写入磁盘，并通过引用追踪他们直到他们不存在与任何队列中为止。当消息的所有引用消失，RabbitMQ将从磁盘中删除消息。


这些可靠机制的速率从1-->8越来越低。

## RabbitMQ回推（限制发布者发送消息如果发送过快的话）

RabbitMQ 2.0之前，对于不合适的发布者，发送Channel.Flow来阻塞发布者，直到发布者接收到另一条Channel.Flow命令。但是这对于那些不重视Channel.Flow命令而滥发消息的发布者，这是无效措施，RabbitMQ最终可能会被拖垮。

RabbitMQ 3.2之前，使用TCP背压机制，它不会要求发送者停止发送消息，而是停止接收TCP套接字的上底层数据。在内部使用信用的概念来管理回推发布者的时机，在建立新的连接时，连接将被分配一个预定数量的可用信用值。RabbitMQ每接收到一个RPC请求就扣除一点信用值，当RPC请求在内部完成，则返还扣除的信用值。当一个连接的信用值不够用时，它将被跳过知道有足够的信用值为止。

在3.2之后，添加了达到连接信用阈值时发送通知的机制，用于通知客户端其连接已经被阻塞。Connection.Blocked和Connection.Unblocked是随时可以发送的异步操作。

 注意：TCP背压和连接阻塞不是经常遇到了，如果遇到了，可以评估扩容。

## 5 消费消息，避免拉取

Q:

1. 为什么要避免拉取消息，而应该倾向于消费消息
2. 如何平衡消息投递的可靠性与性能
3. 如何使用RabbitMQ队列级别的设置来实现自动删除队列、限制消息生存时间等功能

### Basic.Get和Basic.Consum

使用Get获取消息时，每次想要接收消息就必须发送一个新的请求，即使队列中存在多个消息。当发出一个**Basic.Get**时，如果队列中存在一条消息正处于等待处理状态，RabbitMQ就会回应一个Basic.GetOK；没有就会回复Basic.GetEmpty。使用Get请求时，会导致每条消息都会产生与RabbitMQ同步通信的开销。避免使用Basic.get的一个潜在的不太明显的原因时它会影响吞吐量。

Basic.Consum命令来消费信息，与使用Basic.Get创建的同步会话不同，使用Basic.Consum消费消息意味着应用程序会在消息可用时，自动从RabbitMQ接收消息，直到客户端发出Basic.Cancel

在发出Basic.Consum命令时，会创建一个唯一的字符串，用来标识通过已建立的信道与RabbitMQ进行通信的应用程序。该字符串被称为消费者标签。消费者标签在消费者端收到的方法帧中包含目标消费者标签。可以从不同队列接收到的消息执行不同的操作，则可以使用消费者标签来确定处理消息。

## 总结

1. consume是只要队列里面还有消息就一直取。
2. get是只取了队列里面的第一条消息。
3. 因为get开销大，如果需要一个队列取消息的话，首选consume方式，慎用循环get方式。

## 优化消费者性能

当发布消息时，对消息的消费在吞吐量与可靠投递之间存在一种平衡。

**消费者 优化纬度**

1. 基于确认和QoS>1进行消费
2. 基于no-ack模式进行消费：
3. 基于确认进行消费
4. 消费消息和使用事物
5. 获取消息

**消费速度越来越慢**

### 1. 使用no-ack模式实现更快的吞吐量

在amp中是autoack参数（go语言），no-ack=true是让RabbitMQ将消息投递给消费者最快的方式，但这也是发送消息最不可靠的方式。

消息在发送到消费者应用程序中之前有多个数据缓存区接收消息数据。

当RabbitMQ通过打开的连接发送消息时，它使用TCP套接字与客户端通信。如果开启消息确认，那么当RabbitMQ发送消息时，遇到网络问题失败了，通过发送Basic.Ack发送RPC命令，告知RabbitMQ失败了。当关闭了消息确认机制，那么RabbitMQ如果有新的可用消息时，RabbitMQ将会发送消息而不用等待，知道连接的缓存区填满。⚠️：TCP连接在连接的两端都有一个缓存区，用来暂时保存数据。

因为RabbitMQ没有消息确认，如果你的系统架构可以接受消息丢失，这将是RabbitMQ消息消费速度最快的方式。反之，可以选择逐个消息确认或者基于QoS设置的方式。⚠️：QoS设置的方式比逐个消息确认机制吞吐量高。

## 2. 通过QoS设置控制消费者预取

AMQP规定信道要具有服务质量设置，即在确认消息接受之前，消费者可以预先要求接受一定数量的消息。与被禁用确认的消费者不同，如果消费者应用程序在确认消息之前崩溃，则在套接字关闭时，所有预取的消息将返回到队列。

⚠️：如果同时设置QoS和no-ack，则QoS设置的预取值将失效。

**将预取值调整到最佳水平**：过度分配预取量会对消息吞吐量有负面影响。

方法：可以使用单个消费者对一个简单消息进行基准测试。

**一次确认多个消息** 使用QoS设置预取值的好处之一就是不需要每次接受消息使用Basic.Ack回应。将basic.Ack中的multiple设置为true时，RabbitMQ知道你的应用程序想要确认所有以前未确认的消息。

⚠️：虽然一次确认多个消息，可以增加消息的吞吐量，但是当消费者消费完消息，在回复Ack时死亡，则已经消费的消息会被重新放到队列中被其他消费者消费。

## 3. 消费者使用事物

与发布消息到RabbitMQ一样，事物处理允许消费者应用程序提交和回滚批量操作。事物可能会对消息吞吐量产生负面影响，但是有一个例外：如果不使用QoS设置，那么在使用事物来批量确认消息时，实际上可能会看到略微的性能提升。

QoS和事物都是批量确认。

⚠️：事物不使用与已禁用确认的消费者。

## 拒绝消息

消息确认是确保RabbitMQ在丢弃消息之前知道消费者已经接收并处理完消息的一种好方法。但是当消息本身或者消息的处理过程出现问题会发什么？提供两种将消息踢回代理服务器的机制Basic.Reject和Basic.Nack。

### 1. Basic.Reject

用于通知代理服务器无法对所投递的消息进行处理。携带投递标签，用于唯一标识消费者与RabbitMQ进行通信的信道上的消息。当消费者拒绝消息时，可以让RabbitMQ丢弃消息或者使用requeue重新发送消息。当使用requeue时，RabbitMQ将把消息放回队列中并再次处理。

Basic.Reject不能像Ack命令一样一次拒绝多个消息。而Basic.Nack可以实现。

### 2. Basic.Nack

### 3. 死信交换器

队列中的消息可能会变成死信消息(dead-lettered)，进而当以下几个事件任意一个发生时，消息将会被重新发送到一个交换机：

- 消息被消费者使用basic.reject或basic.nack方法并且requeue参数值设置为false的方式进行消息确认(negatively acknowledged)
- 消息由于消息有效期(per-message TTL)过期
- 消息由于队列超过其长度限制而被丢弃

注意，队列的有效期并不会导致其中的消息过期

⚠️：死信交换器与备用交换器不同。过期或者被拒绝的消息通过死信交换器进行投递，而备用交换器则路由那些无法由RabbitMQ路由的信息。

## 路由死信消息(Routing Dead-Lettered Messages)

死信消息将被队列的死信交换机路由到其他队列，在路由时有两种情况：

- 使用在声明队列时指定的死信路由关键字
- 没有设置时，使用消息自身原来的路由关键字

例如，如果你使用foo作为路由关键字发送了一条消息到交换机，当消息成为死信后，它使用foo作为路由关键字被发送到队列的死信交换机。如果队列在声明时指定"x-dead-letter-routing-key"的值为bar，那么消息被发送到死信交换机时将会使用bar作为路由关键字。

注意，如果队列没有设置死信路由关键字，那消息被死信路由时将会使用它自身的原始路由关键字。这包含了CC和BCC头参数设置的路由关键字。

当死信消息被重新发送时，消息确认机制也会在内部被开启，因此，在原始队列删除这条消息之前，消息最终到达的队列—死信队列必须确认该消息。**换句话说，发送队列在接收到死信队列的确认消息之前不会删除原始消息。**注意，如果在特殊情况下服务器宕机，那么同样的消息将会在原始队列和死信队列中同时出现。

消息的死信路由可能会形成一个循环。比如，一个队列的死信的消息没有使用指定的死信路由关键字被发送到默认的交换机时。消息在整个循环(消息到达同一个队列两次)中没有被拒绝，那么消息将被丢弃。

## 死信对消息的影响(Dead-Lettered Effects on Messages)

死信消息修改了它的头部信息：

- 交换机的名称被修改为最后的死信交换机
- 路由关键字可能被改为队列指定的死信路由关键字
- 如果以上发生，名为CC的头参数将被删除
- 名为BCC的头参数将被删除

进行死信路由时，会给每个死信消息的头部增加一个名为x-death的数组。这个数组包含了过于每次死信路由的信息实体，通过一个键值对{queue, reason}区分。每个实体是一张表，包含了一下的字段信息：

- queue：消息称为死信时所在队列的名称
- reason：消息成为死信的原因
- time：消息成为死信的时间，是一个64位的时间戳
- exchange：消息被发送的交换机(当消息多次称为死信消息时，该值为死信交换机)
- routing-keys: 消息被发送时使用的路由关键字，包含了CC关键字但是不包含BCC
- count：在该队列中消息由于这个原因被死信路由的次数
- original-expiration(如果消息时因为消息的TTL称为死信时，有该值)：消息的原始过期时间属性，这个值在消息被死信路由时将被移除，为了避免消息在其他队列中再次过期

新的信息实体将被发送x-death数组的首位，如果该数组中已经存在同样的队列和同样的死信原因的信息实体，那么该实体的count字段将加1并且实体被移到数组的首位。

reason这个属性的值表示了消息变为死信的原因，有以下几种：

- rejected：消息被消息者拒绝并且requeue参数值为false
- expired：消息因为消息的TTL过期
- maxlen：超过了队列允许的最大长度

当进行消息进行首次路由时，将添加三个顶级的头信息：

- x-first-death-reason
- x-first-death-queue
- x-first-death-exchange

它们与消息进行首次死信路由时，设置的reason, queue, exchange字段值相同。一旦添加，它们值将不会再被修改。

注意，这个数组是按照最近发生时间排序，所以最近的死信路由将被记录到第一个实体中。

## 控制队列

1. 临时队列
  - a. 自动删除队列
  - b. 只允许单个消费者
  - c. 自动过期队列
2. 永久队列
  - a. 队列持久性
  - b. 消息自动过期
  - c. 最大长度队列
3. 任意队列



# RabbitMQ任务分发机制

轮询

## 6. 消息路由模式

direct

fanout

topic

headers

## 7. RabbitMQ工作模型

simple

work

pub/sub

routing

topic

## 8. RabbitMQ集群