# AMQP

#AMQP 0-9-1 简介

介绍RabbitMQ所使用的 AMQP 0-9-1版本。[原始版本](<http://bit.ly/amqp-model-explained>)由[Michael Klishin](<http://twitter.com/michaelklishin>)贡献，[Chris Duncan](<https://twitter.com/celldee>)编辑。

## AMQP 0-9-1 和 AMQP 模型高阶概述

###AMQP是什么

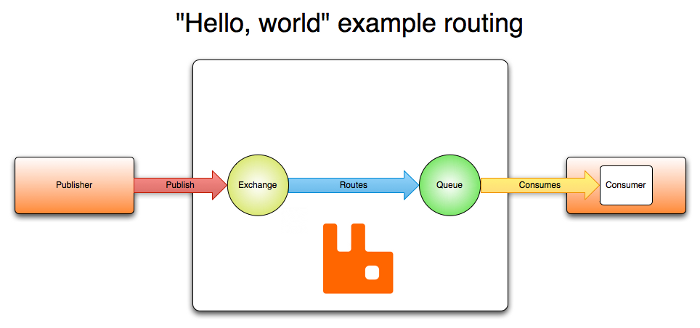
AMQP（高级消息队列协议）是一个网络协议。它支持符合要求的客户端应用（application）和消息中间件代理（messaging middleware broker）之间进行通信。

###消息代理和他们所扮演的角色

消息代理（message brokers）从发布者（publishers）亦称生产者（producers）那儿接收消息，并根据既定的路由规则把接收到的消息发送给处理消息的消费者（consumers）。

由于AMQP是一个网络协议，所以这个过程中的发布者，消费者，消息代理 可以存在于不同的设备上。  
###AMQP 0-9-1 模型简介

AMQP 0-9-1的工作过程如下图：消息（message）被发布者（publisher）发送给交换机（exchange），交换机常常被比喻成邮局或者邮箱。然后交换机将收到的消息根据路由规则分发给绑定的队列（queue）。最后AMQP代理会将消息投递给订阅了此队列的消费者，或者消费者按照需求自行获取。



发布者（publisher）发布消息时可以给消息指定各种消息属性（message meta-data）。有些属性有可能会被消息代理（brokers）使用，然而其他的属性则是完全不透明的，它们只能被接收消息的应用所使用。

从安全角度考虑，网络是不可靠的，接收消息的应用也有可能在处理消息的时候失败。基于此原因，AMQP模块包含了一个消息确认（message acknowledgements）的概念：当一个消息从队列中投递给消费者后（consumer），消费者会通知一下消息代理（broker），这个可以是自动的也可以由处理消息的应用的开发者执行。当“消息确认”被启用的时候，消息代理不会完全将消息从队列中删除，直到它收到来自消费者的确认回执（acknowledgement）。

在某些情况下，例如当一个消息无法被成功路由时，消息或许会被返回给发布者并被丢弃。或者，如果消息代理执行了延期操作，消息会被放入一个所谓的死信队列中。此时，消息发布者可以选择某些参数来处理这些特殊情况。

队列，交换机和绑定统称为AMQP实体（AMQP entities）。

###AMQP是一个可编的程协议

AMQP 0-9-1是一个可编程协议，某种意义上说AMQP的实体和路由规则是由应用本身定义的，而不是由消息代理定义。包括像声明队列和交换机，定义他们之间的绑定，订阅队列等等关于协议本身的操作。

这虽然能让开发人员自由发挥，但也需要他们注意潜在的定义冲突。当然这在实践中很少会发生，如果发生，会以配置错误（misconfiguration）的形式表现出来。

应用程序（Applications）声明AMQP实体，定义需要的路由方案，或者删除不再需要的AMQP实体。

##交换机和交换机类型

交换机是用来发送消息的AMQP实体。交换机拿到一个消息之后将它路由给一个或零个队列。它使用哪种路由算法是由交换机类型和被称作绑定（bindings）的规则所决定的。AMQP 0-9-1的代理提供了四种交换机

| Name（交换机类型） | Default pre-declared names（预声明的默认名称） |  
|--------|--------|  
| Direct exchange（直连交换机） | (Empty string) and amq.direct |  
| Fanout exchange（扇型交换机） | amq.fanout |  
| Topic exchange（主题交换机） | amq.topic |  
| Headers exchange（头交换机） | amq.match (and amq.headers in RabbitMQ) |

除交换机类型外，在声明交换机时还可以附带许多其他的属性，其中最重要的几个分别是：

- Name  
- Durability （消息代理重启后，交换机是否还存在）  
- Auto-delete （当所有与之绑定的消息队列都完成了对此交换机的使用后，删掉它）  
- Arguments（依赖代理本身）

交换机可以有两个状态：持久（durable）、暂存（transient）。持久化的交换机会在消息代理（broker）重启后依旧存在，而暂存的交换机则不会（它们需要在代理再次上线后重新被声明）。然而并不是所有的应用场景都需要持久化的交换机。

###默认交换机

默认交换机（default exchange）实际上是一个由消息代理预先声明好的没有名字（名字为空字符串）的直连交换机（direct exchange）。它有一个特殊的属性使得它对于简单应用特别有用处：那就是每个新建队列（queue）都会自动绑定到默认交换机上，绑定的路由键（routing key）名称与队列名称相同。

举个栗子：当你声明了一个名为"search-indexing-online"的队列，AMQP代理会自动将其绑定到默认交换机上，绑定（binding）的路由键名称也是为"search-indexing-online"。因此，当携带着名为"search-indexing-online"的路由键的消息被发送到默认交换机的时候，此消息会被默认交换机路由至名为"search-indexing-online"的队列中。换句话说，默认交换机看起来貌似能够直接将消息投递给队列，尽管技术上并没有做相关的操作。

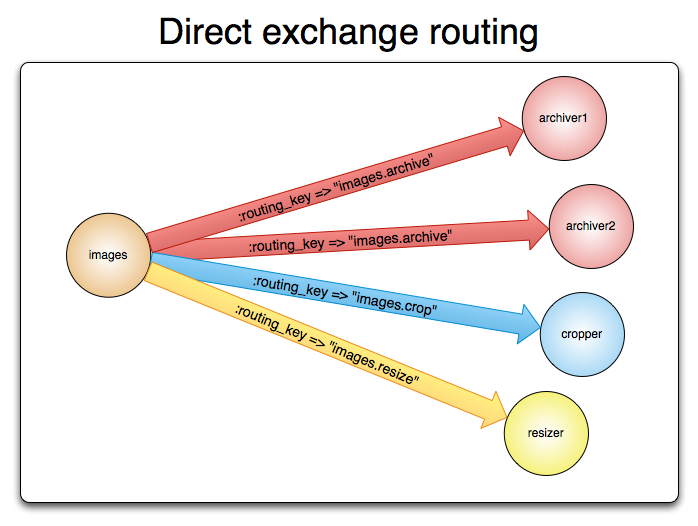
###直连交换机

直连型交换机（direct exchange）是根据消息携带的路由键（routing key）将消息投递给对应队列的。直连交换机用来处理消息的单播路由（unicast routing）（尽管它也可以处理多播路由）。下边介绍它是如何工作的：

- 将一个队列绑定到某个交换机上，同时赋予该绑定一个路由键（routing key）  
- 当一个携带着路由键为`R`的消息被发送给直连交换机时，交换机会把它路由给绑定值同样为`R`的队列。

直连交换机经常用来循环分发任务给多个工作者（workers）。当这样做的时候，我们需要明白一点，在AMQP 0-9-1中，消息的负载均衡是发生在消费者（consumer）之间的，而不是队列（queue）之间。

直连型交换机图例：



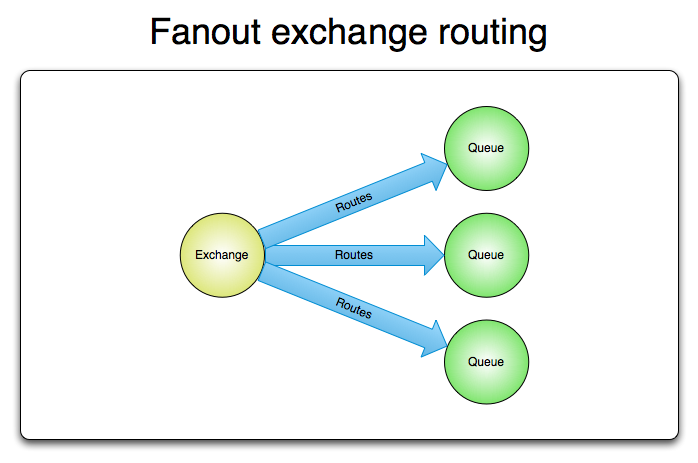
###扇型交换机

扇型交换机（funout exchange）将消息路由给绑定到它身上的所有队列，而不理会绑定的路由键。如果N个队列绑定到某个扇型交换机上，当有消息发送给此扇型交换机时，交换机会将消息的拷贝分别发送给这所有的N个队列。扇型用来交换机处理消息的广播路由（broadcast routing）。

因为扇型交换机投递消息的拷贝到所有绑定到它的队列，所以他的应用案例都极其相似：

- 大规模多用户在线（MMO）游戏可以使用它来处理排行榜更新等全局事件  
- 体育新闻网站可以用它来近乎实时地将比分更新分发给移动客户端  
- 分发系统使用它来广播各种状态和配置更新  
- 在群聊的时候，它被用来分发消息给参与群聊的用户。（AMQP没有内置presence的概念，因此XMPP可能会是个更好的选择）

扇型交换机图例：



###主题交换机

主题交换机（topic exchanges）通过对消息的路由键和队列到交换机的绑定模式之间的匹配，将消息路由给一个或多个队列。主题交换机经常用来实现各种分发/订阅模式及其变种。主题交换机通常用来实现消息的多播路由（multicast routing）。

主题交换机拥有非常广泛的用户案例。无论何时，当一个问题涉及到那些想要有针对性的选择需要接收消息的 多消费者/多应用（multiple consumers/applications） 的时候，主题交换机都可以被列入考虑范围。

使用案例：

- 分发有关于特定地理位置的数据，例如销售点  
- 由多个工作者（workers）完成的后台任务，每个工作者负责处理某些特定的任务  
- 股票价格更新（以及其他类型的金融数据更新）  
- 涉及到分类或者标签的新闻更新（例如，针对特定的运动项目或者队伍）  
- 云端的不同种类服务的协调  
- 分布式架构/基于系统的软件封装，其中每个构建者仅能处理一个特定的架构或者系统。

###头交换机

有时消息的路由操作会涉及到多个属性，此时使用消息头就比用路由键更容易表达，头交换机（headers exchange）就是为此而生的。头交换机使用多个消息属性来代替路由键建立路由规则。通过判断消息头的值能否与指定的绑定相匹配来确立路由规则。

我们可以绑定一个队列到头交换机上，并给他们之间的绑定使用多个用于匹配的头（header）。这个案例中，消息代理得从应用开发者那儿取到更多一段信息，换句话说，它需要考虑某条消息（message）是需要部分匹配还是全部匹配。上边说的“更多一段消息”就是"x-match"参数。当"x-match"设置为“any”时，消息头的任意一个值被匹配就可以满足条件，而当"x-match"设置为“all”的时候，就需要消息头的所有值都匹配成功。

头交换机可以视为直连交换机的另一种表现形式。头交换机能够像直连交换机一样工作，不同之处在于头交换机的路由规则是建立在头属性值之上，而不是路由键。路由键必须是一个字符串，而头属性值则没有这个约束，它们甚至可以是整数或者哈希值（字典）等。

##队列

AMQP中的队列（queue）跟其他消息队列或任务队列中的队列是很相似的：它们存储着即将被应用消费掉的消息。队列跟交换机共享某些属性，但是队列也有一些另外的属性。

- Name  
- Durable（消息代理重启后，队列依旧存在）  
- Exclusive（只被一个连接（connection）使用，而且当连接关闭后队列即被删除）  
- Auto-delete（当最后一个消费者退订后即被删除）  
- Arguments（一些消息代理用他来完成类似与TTL的某些额外功能）

队列在声明（declare）后才能被使用。如果一个队列尚不存在，声明一个队列会创建它。如果声明的队列已经存在，并且属性完全相同，那么此次声明不会对原有队列产生任何影响。如果声明中的属性与已存在队列的属性有差异，那么一个错误代码为406的通道级异常就会被抛出。

###队列名称

队列的名字可以由应用（application）来取，也可以让消息代理（broker）直接生成一个。队列的名字可以是最多255字节的一个utf-8字符串。若希望AMQP消息代理生成队列名，需要给队列的name参数赋值一个空字符串：在同一个通道（channel）的后续的方法（method）中，我们可以使用空字符串来表示之前生成的队列名称。之所以之后的方法可以获取正确的队列名是因为通道可以默默地记住消息代理最后一次生成的队列名称。

以"amq."开始的队列名称被预留做消息代理内部使用。如果试图在队列声明时打破这一规则的话，一个通道级的403 (ACCESS\_REFUSED)错误会被抛出。

###队列持久化

持久化队列（Durable queues）会被存储在磁盘上，当消息代理（broker）重启的时候，它依旧存在。没有被持久化的队列称作暂存队列（Transient queues）。并不是所有的场景和案例都需要将队列持久化。

持久化的队列并不会使得路由到它的消息也具有持久性。倘若消息代理挂掉了，重新启动，那么在重启的过程中持久化队列会被重新声明，无论怎样，只有经过持久化的消息才能被重新恢复。

##绑定

绑定（Binding）是交换机（exchange）将消息（message）路由给队列（queue）所需遵循的规则。如果要指示交换机“E”将消息路由给队列“Q”，那么“Q”就需要与“E”进行绑定。绑定操作需要定义一个可选的路由键（routing key）属性给某些类型的交换机。路由键的意义在于从发送给交换机的众多消息中选择出某些消息，将其路由给绑定的队列。

打个比方：

- 队列（queue）是我们想要去的位于纽约的目的地  
- 交换机（exchange）是JFK机场  
- 绑定（binding）就是JFK机场到目的地的路线。能够到达目的地的路线可以是一条或者多条

拥有了交换机这个中间层，很多由发布者直接到队列难以实现的路由方案能够得以实现，并且避免了应用开发者的许多重复劳动。

如果AMQP的消息无法路由到队列（例如，发送到的交换机没有绑定队列），消息会被就地销毁或者返还给发布者。如何处理取决于发布者设置的消息属性。

##消费者

消息如果只是存储在队列里是没有任何用处的。被应用消费掉，消息的价值才能够体现。在AMQP 0-9-1 模型中，有两种途径可以达到此目的：

- 将消息投递给应用 ("push API")  
- 应用根据需要主动获取消息 ("pull API")

使用push API，应用（application）需要明确表示出它在某个特定队列里所感兴趣的，想要消费的消息。如是，我们可以说应用注册了一个消费者，或者说订阅了一个队列。一个队列可以注册多个消费者，也可以注册一个独享的消费者（当独享消费者存在时，其他消费者即被排除在外）。

每个消费者（订阅者）都有一个叫做消费者标签的标识符。它可以被用来退订消息。消费者标签实际上是一个字符串。

###消息确认

消费者应用（Consumer applications） - 用来接受和处理消息的应用 - 在处理消息的时候偶尔会失败或者有时会直接崩溃掉。而且网络原因也有可能引起各种问题。这就给我们出了个难题，AMQP代理在什么时候删除消息才是正确的？AMQP 0-9-1 规范给我们两种建议：

- 当消息代理（broker）将消息发送给应用后立即删除。（使用AMQP方法：basic.deliver或basic.get-ok）  
- 待应用（application）发送一个确认回执（acknowledgement）后再删除消息。（使用AMQP方法：basic.ack）

前者被称作自动确认模式（automatic acknowledgement model），后者被称作显式确认模式（explicit acknowledgement model）。在显式模式下，由消费者应用来选择什么时候发送确认回执（acknowledgement）。应用可以在收到消息后立即发送，或将未处理的消息存储后发送，或等到消息被处理完毕后再发送确认回执（例如，成功获取一个网页内容并将其存储之后）。

如果一个消费者在尚未发送确认回执的情况下挂掉了，那AMQP代理会将消息重新投递给另一个消费者。如果当时没有可用的消费者了，消息代理会死等下一个注册到此队列的消费者，然后再次尝试投递。

###拒绝消息

当一个消费者接收到某条消息后，处理过程有可能成功，有可能失败。应用可以向消息代理表明，本条消息由于“拒绝消息（Rejecting Messages）”的原因处理失败了（或者未能在此时完成）。当拒绝某条消息时，应用可以告诉消息代理如何处理这条消息——销毁它或者重新放入队列。当此队列只有一个消费者时，请确认不要由于拒绝消息并且选择了重新放入队列的行为而引起消息在同一个消费者身上无限循环的情况发生。

###Negative Acknowledgements

在AMQP中，basic.reject方法用来执行拒绝消息的操作。但basic.reject有个限制：你不能使用它决绝多个带有确认回执（acknowledgements）的消息。但是如果你使用的是RabbitMQ，那么你可以使用被称作negative acknowledgements（也叫nacks）的AMQP 0-9-1扩展来解决这个问题。更多的信息请参考[帮助页面](<https://www.rabbitmq.com/nack.html>)

###预取消息

在多个消费者共享一个队列的案例中，明确指定在收到下一个确认回执前每个消费者一次可以接受多少条消息是非常有用的。这可以在试图批量发布消息的时候起到简单的负载均衡和提高消息吞吐量的作用。For example, if a producing application sends messages every minute because of the nature of the work it is doing.（？？？例如，如果生产应用每分钟才发送一条消息，这说明处理工作尚在运行。）

注意，RabbitMQ只支持通道级的预取计数，而不是连接级的或者基于大小的预取。

##消息属性和有效载荷（消息主体）

AMQP模型中的消息（Message）对象是带有属性（Attributes）的。有些属性及其常见，以至于AMQP 0-9-1 明确的定义了它们，并且应用开发者们无需费心思思考这些属性名字所代表的具体含义。例如：

- Content type（内容类型）  
- Content encoding（内容编码）  
- Routing key（路由键）  
- Delivery mode (persistent or not)   
投递模式（持久化 或 非持久化）  
- Message priority（消息优先权）  
- Message publishing timestamp（消息发布的时间戳）  
- Expiration period（消息有效期）  
- Publisher application id（发布应用的ID）

有些属性是被AMQP代理所使用的，但是大多数是开放给接收它们的应用解释器用的。有些属性是可选的也被称作消息头（headers）。他们跟HTTP协议的X-Headers很相似。消息属性需要在消息被发布的时候定义。

AMQP的消息除属性外，也含有一个有效载荷 - Payload（消息实际携带的数据），它被AMQP代理当作不透明的字节数组来对待。消息代理不会检查或者修改有效载荷。消息可以只包含属性而不携带有效载荷。它通常会使用类似JSON这种序列化的格式数据，为了节省，协议缓冲器和MessagePack将结构化数据序列化，以便以消息的有效载荷的形式发布。AMQP及其同行者们通常使用"content-type" 和 "content-encoding" 这两个字段来与消息沟通进行有效载荷的辨识工作，但这仅仅是基于约定而已。

消息能够以持久化的方式发布，AMQP代理会将此消息存储在磁盘上。如果服务器重启，系统会确认收到的持久化消息未丢失。简单地将消息发送给一个持久化的交换机或者路由给一个持久化的队列，并不会使得此消息具有持久化性质：它完全取决与消息本身的持久模式（persistence mode）。将消息以持久化方式发布时，会对性能造成一定的影响（就像数据库操作一样，健壮性的存在必定造成一些性能牺牲）。

##消息确认

由于网络的不确定性和应用失败的可能性，处理确认回执（acknowledgement）就变的十分重要。有时我们确认消费者收到消息就可以了，有时确认回执意味着消息已被验证并且处理完毕，例如对某些数据已经验证完毕并且进行了数据存储或者索引操作。

这种情形很常见，所以 AMQP 0-9-1 内置了一个功能叫做 消息确认（message acknowledgements），消费者用它来确认消息已经被接收或者处理。如果一个应用崩溃掉（此时连接会断掉，所以AMQP代理亦会得知），而且消息的确认回执功能已经被开启，但是消息代理尚未获得确认回执，那么消息会被从新放入队列（并且在还有还有其他消费者存在于此队列的前提下，立即投递给另外一个消费者）。

协议内置的消息确认功能将帮助开发者建立强大的软件。

##AMQP 0-9-1 方法

AMQP 0-9-1由许多方法（methods）构成。方法即是操作，这跟面向对象编程中的方法没半毛钱关系。AMQP的方法被分组在类（class）中。这里的类仅仅是对AMQP方法的逻辑分组而已。在 [AMQP 0-9-1参考](<https://www.rabbitmq.com/amqp-0-9-1-reference.html>) 中有对AMQP方法的详细介绍。

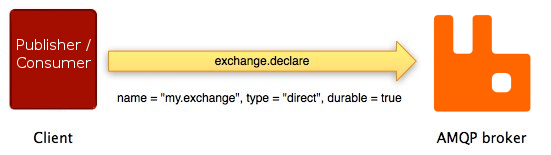
让我们来看看交换机类，有一组方法被关联到了交换机的操作上。这些方法如下所示：

- exchange.declare  
- exchange.declare-ok  
- exchange.delete  
- exchange.delete-ok

（请注意，RabbitMQ网站参考中包含了特用于RabbitMQ的交换机类的扩展，这里我们不对其进行讨论）

以上的操作来自逻辑上的配对：exchange.declare 和 exchange.declare-ok，exchange.delete 和 exchange.delete-ok. 这些操作分为“请求 - requests”（由客户端发送）和“响应 - responses”（由代理发送，用来回应之前提到的“请求”操作）。

如下的例子：客户端要求消息代理使用exchange.declare方法声明一个新的交换机：



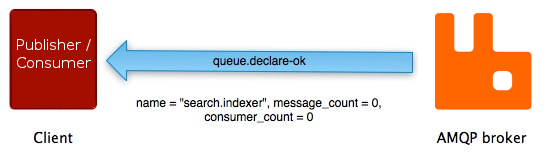
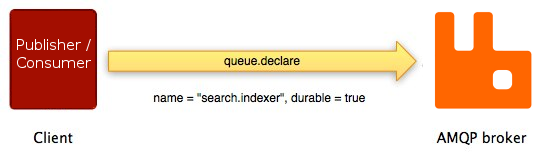
如上图所示，exchange.declare方法携带了好几个参数。这些参数可以允许客户端指定交换机名称、类型、是否持久化等等。

操作成功后，消息代理使用exchange.declare-ok方法进行回应：



exchange.declare-ok方法除了通道号之外没有携带任何其他参数（通道-channel 会在本指南稍后章节进行介绍）。

AMQP队列类的配对方法 - queue.declare方法 和 queue.declare-ok有着与其他配对方法非常相似的一系列事件：



不是所有的AMQP方法都有与其配对的“另一半”。许多（basic.publish是最被广泛使用的）都没有相对应的“响应”方法，另外一些（如basic.get）有着一种以上与之对应的“响应”方法。

##连接

AMQP连接通常是长连接。AMQP是一个使用TCP提供可靠投递的应用层协议。AMQP使用认证机制并且提供TLS（SSL）保护。当一个应用不再需要连接到AMQP代理的时候，需要优雅的释放掉AMQP连接，而不是直接将TCP连接关闭。

##通道

有些应用需要与AMQP代理建立多个连接。无论怎样，同时开启多个TCP连接都是不合适的，因为这样做会消耗掉过多的系统资源并且使得防火墙的配置更加困难。AMQP 0-9-1提供了通道（channels）来处理多连接，可以把通道理解成共享一个TCP连接的多个轻量化连接。

在涉及多线程/进程的应用中，为每个线程/进程开启一个通道（channel）是很常见的，并且这些通道不能被线程/进程共享。

一个特定通道上的通讯与其他通道上的通讯是完全隔离的，因此每个AMQP方法都需要携带一个通道号，这样客户端就可以指定此方法是为哪个通道准备的。

##虚拟主机

为了在一个单独的代理上实现多个隔离的环境（用户、用户组、交换机、队列 等），AMQP提供了一个虚拟主机（virtual hosts - vhosts）的概念。这跟Web servers虚拟主机概念非常相似，这为AMQP实体提供了完全隔离的环境。当连接被建立的时候，AMQP客户端来指定使用哪个虚拟主机。

##AMQP是可扩展的

AMQP 0-9-1 拥有多个扩展点：

- 定制化交换机类型 可以让开发者们实现一些开箱即用的交换机类型尚未很好覆盖的路由方案。例如 geodata-based routing。  
- 交换机和队列的声明中可以包含一些消息代理能够用到的额外属性。例如RabbitMQ中的per-queue message TTL即是使用该方式实现。  
- 特定消息代理的协议扩展。例如RabbitMQ所实现的扩展。  
- 新的 AMQP 0-9-1 方法类可被引入。  
- 消息代理可以被其他的插件扩展，例如RabbitMQ的管理前端 和 已经被插件化的HTTP API。

这些特性使得AMQP 0-9-1模型更加灵活，并且能够适用于解决更加宽泛的问题。

##AMQP 0-9-1 客户端生态系统

AMQP 0-9-1 拥有众多的适用于各种流行语言和框架的客户端。其中一部分严格遵循AMQP规范，提供AMQP方法的实现。另一部分提供了额外的技术，方便使用的方法和抽象。有些客户端是异步的（非阻塞的），有些是同步的（阻塞的），有些将这两者同时实现。有些客户端支持“供应商的特定扩展”（例如RabbitMQ的特定扩展）。

因为AMQP的主要目标之一就是实现交互性，所以对于开发者来讲，了解协议的操作方法而不是只停留在弄懂特定客户端的库就显得十分重要。这样一来，开发者使用不同类型的库与协议进行沟通时就会容易的多。