**什么叫消息队列？**

消息（Message）是指在应用间传送的数据。消息可以非常简单，比如只包含文本字符串，也可以更复杂，可能包含嵌入对象。消息队列（Message Queue）是一种应用间的通信方式，消息发送后可以立即返回，由消息系统来确保消息的可靠传递。消息发布者只管把消息发布到 MQ 中而不用管谁来取，消息使用者只管从 MQ 中取消息而不管是谁发布的。这样发布者和使用者都不用知道对方的存在。

**为何用消息队列？**

消息队列是一种应用间的异步协作机制，那什么时候需要使用消息队列呢？像用户下单之后、生成订单、结算，定时给系统注册用户推送活动消息，一些常见的流程类的业务都会用到消息队列服务。

**一、RabbitMQ简介**

RabbitMQ是一个消息的代理器，用于接收和发送消息，你可以这样想，他就是一个邮局，当您把需要寄送的邮件投递到邮筒之时，你可以确定的是邮递员先生肯定会把邮件发送到需要接收邮件的人的手里，不会送错的。在这个比喻中，RabbitMQ就是一个邮箱，也可以理解为邮局和邮递员，他们负责把消息发送出去和用于接收信息。  
RabbitMQ和邮局这两者之间的主要区别是它不会处理纸质邮件，取而代之的是接收、存储和发送二进制数据块，也就是我们通常所说的消息。

**二 、RabbitMQ基本概念**

下图是RabbitMQ服务的内部结构

* *1）Message*  
  消息，它由消息头和消息体两部分组成。消息体是不透明的，但消息头是由一些属性组成的，其中包括：routing-key（路由键）、priority（优先权）、delivery-mode（持久存储）。
* *2）Publisher*  
  生产者，也是消息的生产者，它是向交换器发布消息的应用程序
* *3）Exchange*  
  交换器，用来接收生产者传递过来的消息，然后将这些消息路由至服务器中的队列
* *4）Binding*  
  绑定，用于消息队列与交换器之间的沟通。也是消息路由的规则，相当于一个路由表。
* *5）Queue*  
  消息队列，用来保存消息直到发送给消费者。一个消息可以进入一个或多个队列，除消费者取走消息，否则它一直在消息队列里。
* *6）Connection*  
  网络连接，如：一个TCP连接
* *7）Channel*  
  信道，多路复用连接中一个独立的双向数据传输通道。无论是发布消息、订阅队列、接收消息都是通过信道来完成。复用信道是为了降低系统资源的消耗。
* *8）Consumer*  
  消费者，也就是接收生产者发来的消息的客户端应用。
* *9）Virtual Host*  
  虚拟主机，交换器、消息队列相关的对象。一个VHOST其实可以看成一个rabbitmp服务器，它拥有自己的队列、交换器、绑定与权限机制等。Rabbitmq默认vhost是/。

**三、RabbitMQ 特点**

RabbitMQ 是一个由 Erlang 语言开发的 AMQP 的开源实现。AMQP ：Advanced Message Queue，高级消息队列协议。它是应用层协议的一个开放标准，为面向消息的中间件设计，基于此协议的客户端与消息中间件可传递消息，并不受产品、开发语言等条件的限制。  
RabbitMQ 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。

**具体特点包括：**

**1）可靠性（Reliability**  
RabbitMQ 使用一些机制来保证可靠性，如持久化、传输确认、发布确认。  
**2）灵活的路由（Flexible Routing）**  
在消息进入队列之前，通过Exchange 来路由消息的。对于典型的路由功能，RabbitMQ 已经提供了一些内置的Exchange 来实现。针对更复杂的路由功能，可以将多个Exchange绑定在一起，也通过插件机制实现自己的 Exchange 。  
**3）消息集群（Clustering）**  
多个 RabbitMQ 服务器可以组成一个集群，形成一个逻辑 Broker 。  
**4）高可用（Highly Available Queues）**  
队列可以在集群中的机器上进行镜像，使得在部分节点出问题的情况下队列仍然可用。  
**5）多种协议（Multi-protocol）**  
RabbitMQ 支持多种消息队列协议，比如 STOMP、MQTT 等等。  
**6）多语言客户端（Many Clients）**  
RabbitMQ 几乎支持所有常用语言，比如 Java、.NET、Ruby 等等。  
**7）管理界面（Management UI）**  
RabbitMQ 提供了一个易用的用户界面，使得用户可以监控和管理消息 Broker 的许多方面。  
**8）跟踪机制（Tracing）**  
如果消息异常，RabbitMQ 提供了消息跟踪机制，使用者可以找出发生了什么。  
**9）插件机制（Plugin System）**  
RabbitMQ 提供了许多插件，来从多方面进行扩展，也可以编写自己的插件。

***参考文章 http://www.rabbitmq.com/***

**四、Rabbitmq的工作过程**

* 1）客户端连接到消息队列服务器，开启一个channel
* 2）客户端声明一个exchange、queue，并配置相关属性
* 3）客户端使用routing key，在exchange与queue之间建立好绑定关系
* 4）客户端传递消息到交换器
* 5）交换器接收到消息后，根据预定的KEY与绑定关系，对消息进行路由至消息队列

**五、Rabbitmq服务安装布署**

***1）首先需要安装Erlang环境***

https://bintray.com/rabbitmq/rpm/erlang  
[root@master ~]# rpm -ivh erlang-20.3.8-1.el7.centos.x86\_64.rpm  
准备中...                         ################################# [100%]  
正在升级/安装...  
   1:erlang-20.3.8-1.el7.centos   ################################# [100%]  
[root@master ~]# erl  
Erlang/OTP 20 [erts-9.3.3] [source] [64-bit] [smp:1:1] [ds:1:1:10] [async-threads:10] [hipe] [kernel-poll:false]  
  
Eshell V9.3.3  (abort with ^G)  
1>

***2）下载安装Rabbitmq***

https://github.com/rabbitmq/rabbitmq-server/releases/download/v3.7.6/rabbitmq-server-3.7.6-1.el7.noarch.rpm  
[root@master ~]# ll rabbitmq-server-3.7.6-1.el7.noarch.rpm  
-rw-r--r-- 1 root root 9511623 6月  27 14:00 rabbitmq-server-3.7.6-1.el7.noarch.rpm  
[root@master ~]# rpm --import https://dl.bintray.com/rabbitmq/Keys/rabbitmq-release-signing-key.asc  
[root@master ~]# yum install rabbitmq-server-3.7.6-1.el7.noarch.rpm  
[root@master ~]# chkconfig rabbitmq-server on  
注意：正在将请求转发到“systemctl enable rabbitmq-server.service”。  
Created symlink from /etc/systemd/system/multi-user.target.wants/rabbitmq-server.service to /usr/lib/systemd/system/rabbitmq-server.service.  
[root@master ~]# /sbin/service rabbitmq-server start  
Redirecting to /bin/systemctl start rabbitmq-server.service  
[root@master ~]# ps -ef|grep rabbitmq  
rabbitmq  5609     1 23 14:22 ?        00:00:02 /usr/lib64/erlang/erts-9.3.3/bin/beam.smp -W w -A 64 -MBas ageffcbf -MHas ageffcbf -MBlmbcs 512 -MHlmbcs 512 -MMmcs 30 -P 1048576 -t 5000000 -stbt db -zdbbl 1280000 -K true -- -root /usr/lib64/erlang -progname erl -- -home /var/lib/rabbitmq -- -pa /usr/lib/rabbitmq/lib/rabbitmq\_server-3.7.6/ebin -noshell -noinput -s rabbit boot -sname rabbit@master -boot start\_sasl -kernel inet\_default\_connect\_options [{nodelay,true}] -sasl errlog\_type error -sasl sasl\_error\_logger false -rabbit lager\_log\_root "/var/log/rabbitmq" -rabbit lager\_default\_file "/var/log/rabbitmq/rabbit@master.log" -rabbit lager\_upgrade\_file "/var/log/rabbitmq/rabbit@master\_upgrade.log" -rabbit enabled\_plugins\_file "/etc/rabbitmq/enabled\_plugins" -rabbit plugins\_dir "/usr/lib/rabbitmq/plugins:/usr/lib/rabbitmq/lib/rabbitmq\_server-3.7.6/plugins" -rabbit plugins\_expand\_dir "/var/lib/rabbitmq/mnesia/rabbit@master-plugins-expand" -os\_mon start\_cpu\_sup false -os\_mon start\_disksup false -os\_mon start\_memsup false -mnesia dir "/var/lib/rabbitmq/mnesia/rabbit@master" -kernel inet\_dist\_listen\_min 25672 -kernel inet\_dist\_listen\_max 25672  
rabbitmq  5752     1  0 14:22 ?        00:00:00 /usr/lib64/erlang/erts-9.3.3/bin/epmd -daemon  
rabbitmq  5921  5609  0 14:22 ?        00:00:00 erl\_child\_setup 1024  
rabbitmq  5937  5921  0 14:22 ?        00:00:00 inet\_gethost 4  
rabbitmq  5938  5937  0 14:22 ?        00:00:00 inet\_gethost 4  
root      5951  5300  0 14:23 pts/0    00:00:00 grep --color=auto rabbitmq  
[root@master ~]# netstat -lntup |grep 5672  
-bash: netstat: 未找到命令  
[root@master ~]# netstat -lntup |grep 5672  
tcp        0      0 0.0.0.0:25672           0.0.0.0:\*               LISTEN      5609/beam.smp         
tcp6       0      0 :::5672                 :::\*                    LISTEN      5609/beam.smp

***3）查看状态信息***

[root@master ~]# rabbitmqctl status  
Status of node rabbit@master ...  
[{pid,5609},  
 {running\_applications,  
     [{rabbit,"RabbitMQ","3.7.6"},  
      {mnesia,"MNESIA  CXC 138 12","4.15.3"},  
      {rabbit\_common,  
          "Modules shared by rabbitmq-server and rabbitmq-erlang-client",  
          "3.7.6"},  
      {ranch\_proxy\_protocol,"Ranch Proxy Protocol Transport","1.5.0"},  
      {ranch,"Socket acceptor pool for TCP protocols.","1.5.0"},  
      {ssl,"Erlang/OTP SSL application","8.2.6"},  
      {public\_key,"Public key infrastructure","1.5.2"},  
      {asn1,"The Erlang ASN1 compiler version 5.0.5","5.0.5"},  
      {inets,"INETS  CXC 138 49","6.5.2"},  
      {jsx,"a streaming, evented json parsing toolkit","2.8.2"},  
      {os\_mon,"CPO  CXC 138 46","2.4.4"},  
      {xmerl,"XML parser","1.3.16"},  
      {crypto,"CRYPTO","4.2.2"},  
      {recon,"Diagnostic tools for production use","2.3.2"},  
      {lager,"Erlang logging framework","3.5.1"},  
      {goldrush,"Erlang event stream processor","0.1.9"},  
      {compiler,"ERTS  CXC 138 10","7.1.5"},  
      {syntax\_tools,"Syntax tools","2.1.4"},  
      {syslog,"An RFC 3164 and RFC 5424 compliant logging framework.","3.4.2"},  
      {sasl,"SASL  CXC 138 11","3.1.2"},  
      {stdlib,"ERTS  CXC 138 10","3.4.5"},  
      {kernel,"ERTS  CXC 138 10","5.4.3"}]},  
 {os,{unix,linux}},  
 {erlang\_version,  
     "Erlang/OTP 20 [erts-9.3.3] [source] [64-bit] [smp:1:1] [ds:1:1:10] [async-threads:64] [hipe] [kernel-poll:true]\n"},  
 {memory,  
     [{connection\_readers,0},  
      {connection\_writers,0},  
      {connection\_channels,0},  
      {connection\_other,0},  
      {queue\_procs,0},  
      {queue\_slave\_procs,0},  
      {plugins,5936},  
      {other\_proc,20754544},  
      {metrics,184432},  
      {mgmt\_db,0},  
      {mnesia,72912},  
      {other\_ets,1873688},  
      {binary,55376},  
      {msg\_index,28720},  
      {code,25082003},  
      {atom,1041593},  
      {other\_system,9173572},  
      {allocated\_unused,8315896},  
      {reserved\_unallocated,3670016},  
      {strategy,rss},  
      {total,[{erlang,58272776},{rss,70258688},{allocated,66588672}]}]},  
 {alarms,[]},  
 {listeners,[{clustering,25672,"::"},{amqp,5672,"::"}]},  
 {vm\_memory\_calculation\_strategy,rss},  
 {vm\_memory\_high\_watermark,0.4},  
 {vm\_memory\_limit,3281294131},  
 {disk\_free\_limit,50000000},  
 {disk\_free,122394349568},  
 {file\_descriptors,  
     [{total\_limit,924},{total\_used,2},{sockets\_limit,829},{sockets\_used,0}]},  
 {processes,[{limit,1048576},{used,204}]},  
 {run\_queue,0},  
 {uptime,111},  
 {kernel,{net\_ticktime,60}}]

**六、Rabbitmq CLI介绍**

**自带多个命令行工具**

***1）rabbitmqctl  #管理与操作命令***

* 停止节点
* 访问节点状态，有效配置，运行状况检查
* 虚拟主机管理
* 用户和权限管理
* 政策管理
* 列出队列，连接，渠道，交流，消费者
* 集群成员管理

[root@master ~]# rabbitmqctl list\_bindings    #查看绑定信息  
Listing bindings for vhost /...  
[root@master ~]# rabbitmqctl list\_exchanges   #查看交换器  
Listing exchanges for vhost / ...  
amq.rabbitmq.trace    topic  
amq.match    headers  
    direct  
amq.headers    headers  
amq.direct    direct  
amq.rabbitmq.event    topic  
amq.topic    topic  
amq.fanout    fanout  
[root@master ~]# rabbitmqctl list\_queues   #查看队列  
Timeout: 60.0 seconds ...  
Listing queues for vhost / ...  
更多操作内容请参考：http://www.rabbitmq.com/rabbitmqctl.8.html

***2）rabbitmq-plugins #是一个管理插件的工具***

[root@master ~]# rabbitmq-plugins --help  
Usage:  
rabbitmq-plugins [-n <node>] [-l] [-q] <command> [<command options>]  
  
General options:  
    -n node  
    -q quiet  
    -l longnames  
  
Default node is "rabbit@server", where `server` is the local hostname. On a host  
named "server.example.com", the node name of the RabbitMQ Erlang node will  
usually be rabbit@server (unless RABBITMQ\_NODENAME has been set to some  
non-default value at broker startup time). The output of hostname -s is usually  
the correct suffix to use after the "@" sign. See rabbitmq-server(1) for  
details of configuring the RabbitMQ broker.  
  
Quiet output mode is selected with the "-q" flag. Informational messages are  
suppressed when quiet mode is in effect.  
  
If RabbitMQ broker uses long node names for erlang distribution, "longnames"  
option should be specified.  
  
Some commands accept an optional virtual host parameter for which  
to display results. The default value is "/".  
  
Commands:  
    disable <plugin>|--all [--offline] [--online]  
    enable <plugin>|--all [--offline] [--online]  
    help <command>  
    list [pattern] [--verbose] [--minimal] [--enabled] [--implicitly-enabled]  
    set [<plugin>] [--offline] [--online]  
  
<timeout> - operation timeout in seconds. Default is "infinity".  
参考文章：http://www.rabbitmq.com/rabbitmq-plugins.8.html

***3)rabbitmqadmin #它可以执行一些与WEB界面相同的操作，rabbitmqadmin只是一个专门的HTTP客户端。***

**安装管理插件后，http：//{主机名}：15672 /cli/rabbitmqadmin 进行下载**

[root@master ~]# wget https://raw.githubusercontent.com/rabbitmq/rabbitmq-management/v3.7.6/bin/rabbitmqadmin  
[root@master ~]# cp rabbitmqadmin /usr/local/bin/  
[root@master ~]# chmod +x /usr/local/bin/rabbitmqadmin

**七、RabbitMQ集群**

RabbitMQ 是用 erlang 开发的，集群非常方便，因为 erlang 天生就是一门分布式语言，但其本身并不支持负载均衡。

***RabbitMQ 的集群节点包括：***

**1）内存节点**  
内存节点就是将所有数据放在内存，只保存状态到内存（例外的情况：持久的queue内容将被保存到 disk）  
**2）磁盘节点。**  
磁盘节点将数据放在磁盘，保存状态到内存和磁盘，内存节点虽然不写入磁盘，但是它执行比磁盘节点要好，集群中，只需要一个磁盘节点来保存状态 就足够了，如果集群中只有内存节点，那么不能停止它们，否则所有的状态，消息等都会丢失。

**不过，如前文所述，如果在投递消息时，打开了消息的持久化，那即使是内存节点，数据还是安全的放在磁盘。一个 RabbitMQ 集群中可以共享 user、vhost、queue、exchange 等，所有的数据和状态都是必须在所有节点上复制的，一个例外是那些当前只属于创建它的节点的消息队列，尽管它们可见且可被所有节点读取。 RabbitMQ 节点可以动态地加入到集群中，一个节点它可以加入到集群中，也可以从集群环集群进行一个基本的负载均衡。**

***Rabbit 模式大概分为以下三种：***

**1）单一模式**  
**2）普通模式**  
**3）镜像模式**

**单一模式：最简单的情况，非集群模式。**

**普通模式：默认的集群模式。**

对于 Queue 来说，消息实体只存在于其中一个节点，A、B 两个节点仅有相同的元数据，即队列结构。当消息进入 A 节点的 Queue 中后，consumer 从 B 节点拉取时，RabbitMQ 会临时在 A、B 间进行消息传输，把 A 中的消息实体取出并经过 B 发送给 consumer. 所以 consumer 应尽量连接每一个节点，从中取消息。即对于同一个逻辑队列，要在多个节点建立物理 Queue。否则无论 consumer 连 A 或 B，出口总在 A，会产生瓶颈。该模式存在一个问题就是当 A 节点故障后，B 节点无法取到 A 节点中还未消费的消息实体。如果做了消息持久化，那么得等 A 节点恢复，然后才可被消费。

**镜像模式：**

把需要的队列做成镜像队列，存在于多个节点，属于 RabbitMQ 的 HA 方案。  
该模式解决了上述问题，其实质和普通模式不同之处在于，消息实体会主动在镜像节点间同步，而不是在 consumer 取数据时临时拉取。该模式带来的副作用也很明显，除了降低系统性能外，如果镜像队列数量过多，加之大量的消息进入，集群内部的网络带宽将会被这种同步通讯大大消耗掉,所，在对可靠性要求较高的场合中适用。