🡪

RabbitMQ 即一个消息队列，主要是用来实现应用程序的异步和解耦，同时也能起到消息缓冲，消息分发的作用。消息中间件最主要的作用是解耦，中间件最标准的用法是生产者生产消息传送到队列，消费者从队列中拿取消息并处理，生产者不用关心是谁来消费，消费者不用关心谁在生产消息，从而达到解耦的目的。

AMQP，即Advanced Message Queuing Protocol，高级消息队列协议，是应用层协议的一个开放标准，为面向消息的中间件设计。消息中间件主要用于组件之间的解耦，消息的发送者无需知道消息使用者的存在，反之亦然。AMQP的主要特征是面向消息、队列、路由（包括点对点和发布/订阅）、可靠性、安全。

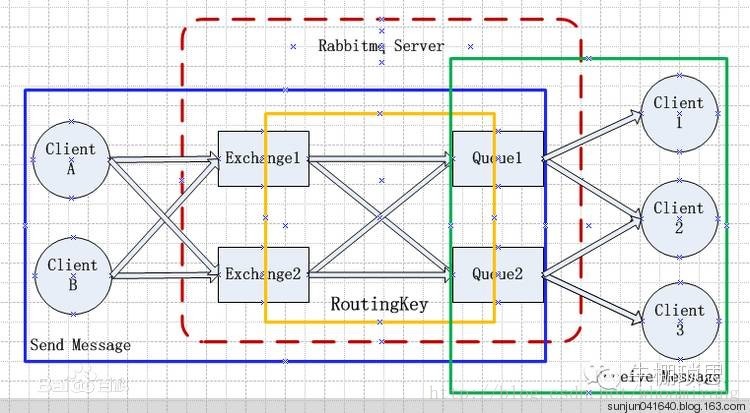
RabbitMQ是实现AMQP（高级消息队列协议）的消息中间件的一种，最初起源于金融系统，用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。RabbitMQ主要是为了实现系统之间的双向解耦而实现的。当生产者大量产生数据时，消费者无法快速消费，那么需要一个中间层。保存这个数据。

RabbitMQ是一个**开源的**AMQP实现，服务器端用Erlang语言编写，支持多种客户端，如：Python、Ruby、.NET、Java、JMS、C、PHP、ActionScript、XMPP、STOMP等，支持AJAX。

建立在这个机制上面，选中RabbitMQ作为消息队列可以选用celery作为broker，但是底层的RabbitMQ还是可以用RabbitMQ自带的调控指令。

🡪

通常我们谈到队列服务, 会有三个概念： 发消息者、队列、收消息者，RabbitMQ 在这个基本概念之上, 多做了一层抽象, 在发消息者和 队列之间, 加入了交换器 (Exchange). 这样发消息者和队列就没有直接联系, 转而变成发消息者把消息给交换器, 交换器根据调度策略再把消息再给队列。



左侧 P 代表 生产者，也就是往 RabbitMQ 发消息的程序。

中间即是 RabbitMQ，其中包括了 交换机 和 队列。

右侧 C 代表 消费者，也就是往 RabbitMQ 取消息的程序。

那么其中概念有：

Broker: 它提供一种传输服务, 它的角色就是维护一条从生产者到消费者的路线，保证数据能按照指定的方式进行传输,   
Exchange：消息交换机, 它指定消息按什么规则, 路由到哪个队列。   
Queue: 消息的载体, 每个消息都会被投到一个或多个队列。   
Binding: 绑定，它的作用就是把exchange和queue按照路由规则绑定起来.   
Routing Key: 路由关键字, exchange根据这个关键字进行消息投递。   
vhost: 虚拟主机, 一个broker里可以有多个vhost，用作不同用户的权限分离。   
Producer: 消息生产者, 就是投递消息的程序.   
Consumer: 消息消费者, 就是接受消息的程序.   
Channel: 消息通道, 在客户端的每个连接里, 可建立多个channel.

🡪

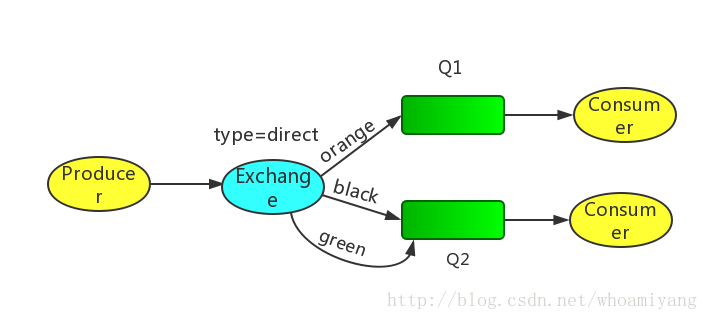
**交换器调度策略**

交换机的功能主要是接收消息并且转发到绑定的队列，交换机不存储消息.

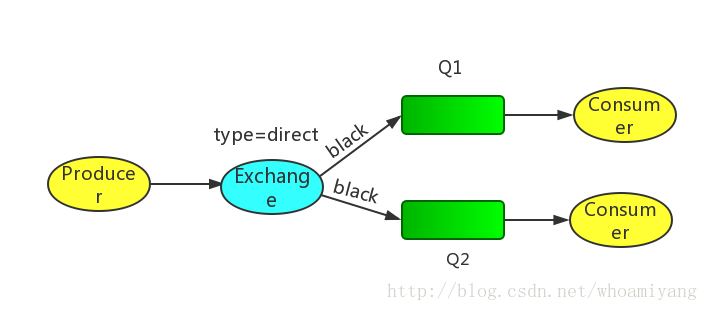
RabbitMQ常用的Exchange Type有三种：fanout、direct、topic。

Fanout exchange: 广播订阅, 向所有的消费者发布消息, 但是只有消费者将队列绑定到该交换机才能收到消息, 忽略Routing Key.

**Direct exchange: 直接匹配**, 通过Exchange名称+RountingKey来发送与接收消息.

   
Exchange和两个队列绑定在一起, Q1的bindingkey是orange，Q2的binding key是black和green.   
当Producer publish key是orange时,exchange会把它放到Q1上,如果是black或green就会到Q2上,其余的Message被丢弃.

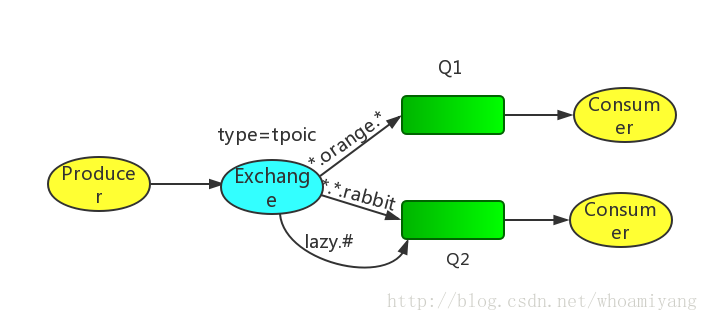
如果是多个queue绑定同一个key也是可以的。

多个queue绑定同一个key也是可以的,对于下图的例子,Q1和Q2都绑定了black,对于routing key是black的Message，会被deliver到Q1和Q2，其余的Message都会被丢弃.   


**Topic exchange:** **主题匹配订阅**, 这里的主题指的是RoutingKey,RoutingKey可以采用通配符,如:\*或#，RoutingKey命名采用.来分隔多个词,只有消息这将队列绑定到该路由器且指定RoutingKey符合匹配规则时才能收到消息;

对于Message的routing\_key是有限制的，不能使任意的。格式是以点号“.”分割的字符表。比如：“stock.usd.nyse”, “nyse.vmw”, “quick.orange.rabbit”。你可以放任意的key在routing\_key中，当然最长不能超过255 bytes。   
对于routing\_key，有两个特殊字符

\*(星号)代表任意一个单词

#(hash)0个或多个单词   
   
Producer发送消息时需要设置routing\_key，routing\_key包含三个单词和连个点号.

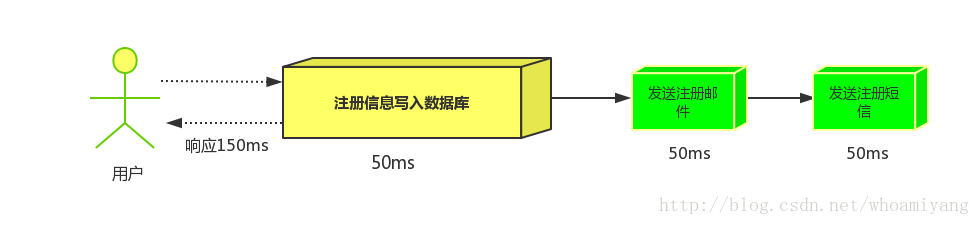
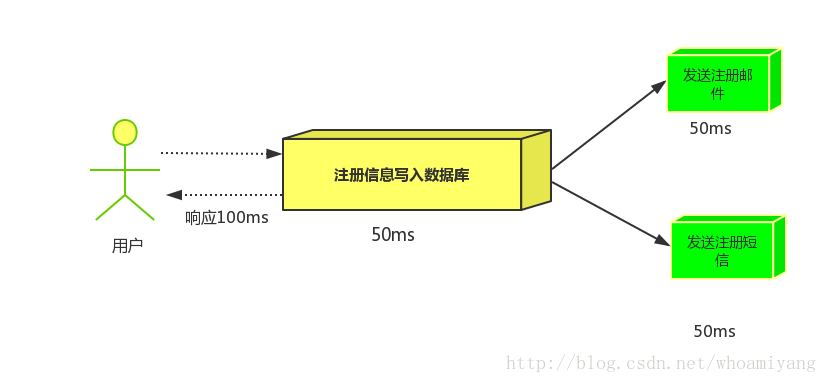
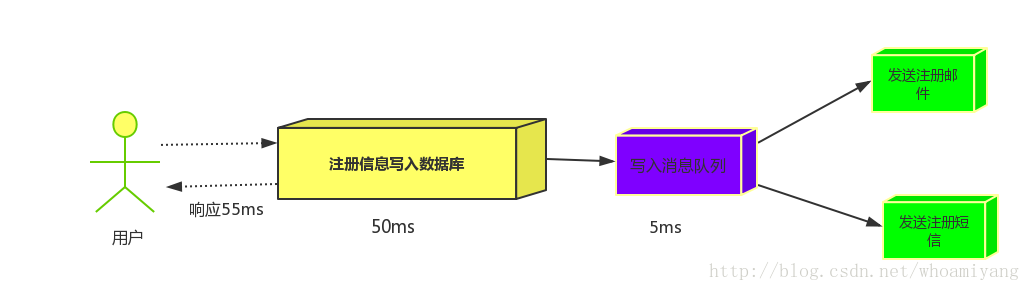
第一个key描述了celerity(灵巧度),第二个是color(色彩),第三个是物种:   
在这里我们创建了两个绑定： Q1 的binding key 是”*.orange.*“； Q2 是 “*.*.rabbit” 和 “lazy.#”：

* + Q1感兴趣所有orange颜色的动物
  + Q2感兴趣所有rabbits和所有的lazy的.   
    例子:rounting\_key 为 “quick.orange.rabbit”将会发送到Q1和Q2中。因为满足\*.orange.\*,以及\*.\*.rabbit匹配。Rabbitmq将向满足匹配的队列都发送消息。  
    rounting\_key 为”lazy.orange.rabbit.hujj.ddd”会被投递到Q2中。不向Q1发送的原因是\*.orange.\*应该是三个词结束。

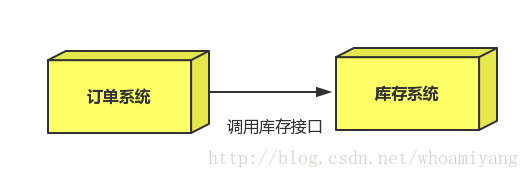
🡪

**应用场景**

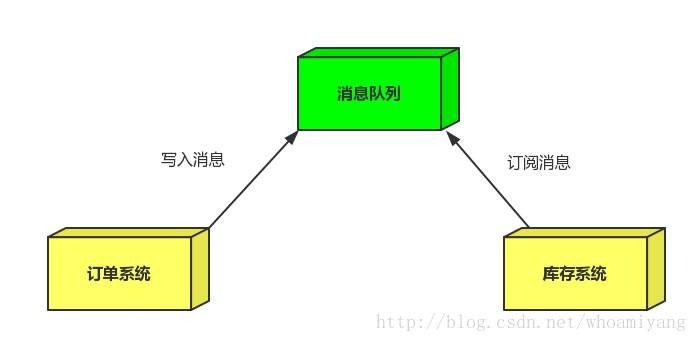
**场景1：**

场景说明：用户注册后，需要发注册邮件和注册短信,传统的做法有两种1.串行的方式;2.并行的方式   
(1) 串行方式: 将注册信息写入数据库后, 发送注册邮件, 再发送注册短信, 以上三个任务全部完成后才返回给客户端。 这有一个问题是,邮件,短信并不是必须的,它只是一个通知,而这种做法让客户端等待没有必要等待的东西.   
  
(2) 并行方式: 将注册信息写入数据库后, 发送邮件的同时, 发送短信, 以上三个任务完成后,返回给客户端,并行的方式能提高处理的时间。   
   
假设三个业务节点分别使用50ms,串行方式使用时间150ms,并行使用时间100ms。虽然并性已经提高的处理时间,但是,前面说过,邮件和短信对我正常的使用网站没有任何影响，客户端没有必要等着其发送完成才显示注册成功,英爱是写入数据库后就返回.   
(3)消息队列   
引入消息队列后，把发送邮件,短信不是必须的业务逻辑异步处理   
   
由此可以看出,引入消息队列后，用户的响应时间就等于写入数据库的时间+写入消息队列的时间(可以忽略不计),引入消息队列后处理后,响应时间是串行的3倍,是并行的2倍。

**场景3：**

场景：双11是购物狂节,用户下单后,订单系统需要通知库存系统,传统的做法就是订单系统调用库存系统的接口.   
   
这种做法有一个缺点:

当库存系统出现故障时,订单就会失败。(这样马云将少赚好多好多钱^ ^)

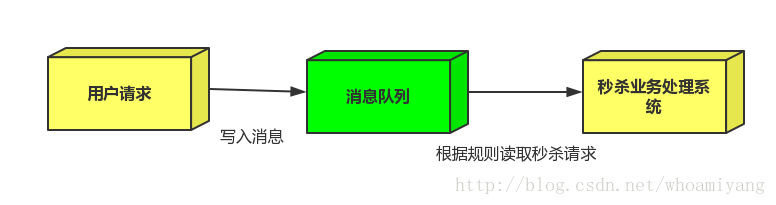
订单系统和库存系统高耦合.   
引入消息队列   


订单系统:用户下单后,订单系统完成持久化处理,将消息写入消息队列,返回用户订单下单成功。

库存系统:订阅下单的消息,获取下单消息,进行库操作。   
就算库存系统出现故障,消息队列也能保证消息的可靠投递,不会导致消息丢失(马云这下高兴了).

**场景4：**

流量削峰

流量削峰一般在秒杀活动中应用广泛   
场景:秒杀活动，一般会因为流量过大，导致应用挂掉,为了解决这个问题，一般在应用前端加入消息队列。   
作用:   
1.可以控制活动人数，超过此一定阀值的订单直接丢弃(我为什么秒杀一次都没有成功过呢^^)   
2. 可以缓解短时间的高流量压垮应用(应用程序按自己的最大处理能力获取订单)   
   
1.用户的请求,服务器收到之后,首先写入消息队列,加入消息队列长度超过最大值,则直接抛弃用户请求或跳转到错误页面.   
2.秒杀业务根据消息队列中的请求信息，再做后续处理.

🡪

**Message acknowledgment消息确认机制**

为了保证数据不被丢失,RabbitMQ支持消息确认机制,为了保证数据能被正确处理而不仅仅是被Consumer收到,那么我们不能采用no-ack，而应该是在处理完数据之后发送ack.   
在处理完数据之后发送ack,就是告诉RabbitMQ数据已经被接收,处理完成,RabbitMQ可以安全的删除它了.   
如果Consumer退出了但是没有发送ack,那么RabbitMQ就会把这个Message发送到下一个Consumer，这样就保证在Consumer异常退出情况下数据也不会丢失.   
RabbitMQ它没有用到超时机制.RabbitMQ仅仅通过Consumer的连接中断来确认该Message并没有正确处理，也就是说RabbitMQ给了Consumer足够长的时间做数据处理。   
如果忘记ack,那么当Consumer退出时,Mesage会重新分发,然后RabbitMQ会占用越来越多的内存.

🡪

**多从绑定**

我们可以为一个exchange绑定多个queue.当一个消息满足调度策略routingKey时候，这个消息将发送给所有满足条件的绑定队列。

我们可以为一个队列绑定多个consumer.当该队列接收到消息时候，绑定该队列的其中一个空闲的consumer都会收到该消息。如果一个consumer正在执行任务，则会由另外空闲的consumer进行接受任务。

🡪

rabbitmqctl status 查看状态

rabbitmqctl add\_user添加用户

rabbitmqctl set\_user\_tags为某用户添加角色，例如管理员，监控员等

rabbitmqctl list\_users展示用户

rabbitmqctl list\_vhosts展示vhosts

rabbitmqctl set\_permission为某用户添加权限，例如可以连接到哪个virtual host等。

🡪

默认情况下rabbitmq会自动创建guest/guest的用户。这个用户只能够在localhost上连接，不能远程连接，如果要在非本机的情况下连接，要进行额外的设置。

🡪

rabbitmqctl\_plugin是rabbitmq自动的plugin管理工具命令。

🡪

在rabbitmqctl的log中会显示当前生效的配置文件，日志文件的所在地等信息。

🡪