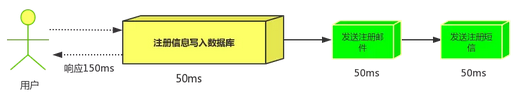
**RabbitMQ**

1.消息队列应用场景

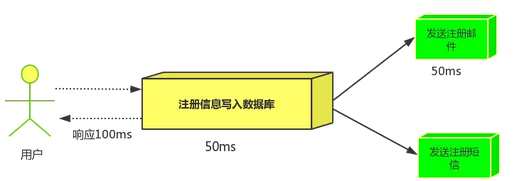
（1）异步处理

例子：用户注册后，需要发邮件和短信

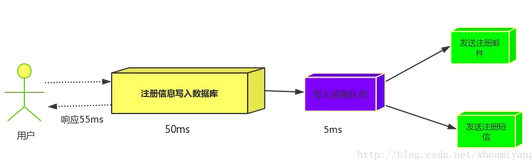
串行方式，先把信息写入数据库，再发送邮件，再发送短信，最后做出响应。



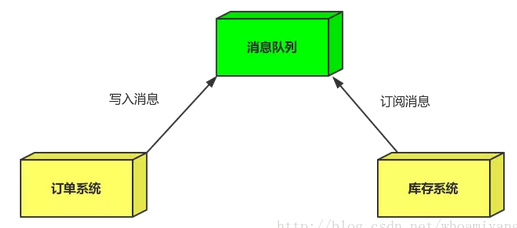
并行方式，先把信息写入数据库，然后并行处理发送邮件和短信，最后做出响应。



使用消息队列，先把信息写入数据库，然后将发送邮件和短信的业务放入消息队列，最后做出响应。



（2）应用解耦



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，做出响应用户订单成功。

当库存系统出现故障，消息队列也能保证消息的可靠投递，不会导致消息的丢失。

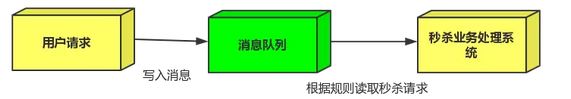
（3）流量削峰

流量削峰一般应用在秒杀活动中

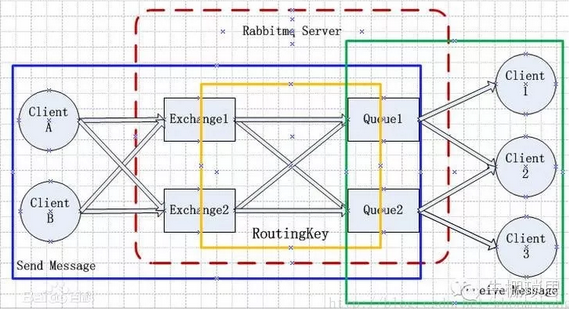
场景：秒杀活动，一般会因为流量过大，导致应用挂掉，为了解决这个问题，一般在应用前 端加入消息队列

作用：1）控制活动人数，超过阀值的订单直接丢弃

2）缓解短时间的高流量压垮应用（应用程序按自己最大处理能力获取订单）



2.系统架构



几个概念说明:   
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.

3.任务分发机制

（1）round-robin dispathching循环分发

RabbbitMQ的分发机制非常适合扩展,而且它是专门为并发程序设计的,如果现在load加重,那么只需要创建更多的Consumer来进行任务处理。

（2）Message acknowledgment消息确认

**4.任务分发机制**

**4.1Round-robin dispathching循环分发**

RabbbitMQ的分发机制非常适合扩展,而且它是专门为并发程序设计的,如果现在load加重,那么只需要创建更多的Consumer来进行任务处理。

**4.2Message acknowledgment消息确认**

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.

**5.Message durability消息持久化**

要持久化队列queue的持久化需要在声明时指定durable=True;   
**这里要注意,队列的名字一定要是Broker中不存在的,不然不能改变此队列的任何属性.**   
队列和交换机有一个创建时候指定的标志durable,durable的唯一含义就是具有这个标志的队列和交换机会在重启之后重新建立,它不表示说在队列中的消息会在重启后恢复   
消息持久化包括3部分   
1. exchange持久化,在声明时指定durable => true

channel.ExchangeDeclare(ExchangeName, "direct", durable: true, autoDelete: false, arguments: null);//声明消息队列，且为可持久化的

* 1

2.queue持久化,在声明时指定durable => true

channel.QueueDeclare(QueueName, durable: true, exclusive: false, autoDelete: false, arguments: null);//声明消息队列，且为可持久化的

* 1

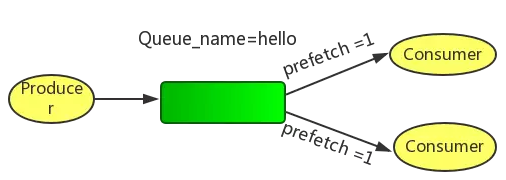
3.消息持久化,在投递时指定delivery\_mode => 2(1是非持久化).

channel.basicPublish("", queueName, MessageProperties.PERSISTENT\_TEXT\_PLAIN, msg.getBytes());

* 1

如果exchange和queue都是持久化的,那么它们之间的binding也是持久化的,如果exchange和queue两者之间有一个持久化，一个非持久化,则不允许建立绑定.   
**注意**：一旦创建了队列和交换机,就不能修改其标志了,例如,创建了一个non-durable的队列,然后想把它改变成durable的,唯一的办法就是删除这个队列然后重现创建。

**6.Fair dispath 公平分发**

你可能也注意到了,分发机制不是那么优雅,默认状态下,RabbitMQ将第n个Message分发给第n个Consumer。n是取余后的,它不管Consumer是否还有unacked Message，只是按照这个默认的机制进行分发.   
那么如果有个Consumer工作比较重,那么就会导致有的Consumer基本没事可做,有的Consumer却毫无休息的机会,那么,Rabbit是如何处理这种问题呢?   
   
通过basic.qos方法设置prefetch\_count=1，这样RabbitMQ就会使得每个Consumer在同一个时间点最多处理一个Message，换句话说,在接收到该Consumer的ack前,它不会将新的Message分发给它

channel.basic\_qos(prefetch\_count=1)

* 1

**注意，这种方法可能会导致queue满。当然，这种情况下你可能需要添加更多的Consumer，或者创建更多的virtualHost来细化你的设计。**

**7.分发到多个Consumer**

**7.1Exchange**

先来温习以下交换机路由的几种类型:   
Direct Exchange:直接匹配,通过Exchange名称+RountingKey来发送与接收消息.   
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路由器才能收到消息,忽略Routing Key.   
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;   
Headers Exchange:消息头订阅,消息发布前,为消息定义一个或多个键值对的消息头,然后消费者接收消息同时需要定义类似的键值对请求头:(如:x-mactch=all或者x\_match=any)，只有请求头与消息头匹配,才能接收消息,忽略RoutingKey.   
默认的exchange:如果用空字符串去声明一个exchange，那么系统就会使用”amq.direct”这个exchange，我们创建一个queue时,默认的都会有一个和新建queue同名的routingKey绑定到这个默认的exchange上去

channel.BasicPublish("", "TaskQueue", properties, bytes);

* 1

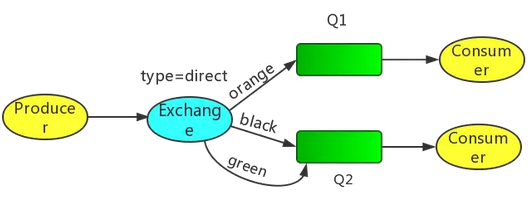
因为在第一个参数选择了默认的exchange，而我们申明的队列叫TaskQueue，所以默认的，它在新建一个也叫TaskQueue的routingKey，并绑定在默认的exchange上，导致了我们可以在第二个参数routingKey中写TaskQueue，这样它就会找到定义的同名的queue，并把消息放进去。   
**如果有两个接收程序都是用了同一个的queue和相同的routingKey去绑定direct exchange的话，分发的行为是负载均衡的，也就是说第一个是程序1收到，第二个是程序2收到，以此类推。**   
**如果有两个接收程序用了各自的queue，但使用相同的routingKey去绑定direct exchange的话，分发的行为是复制的，也就是说每个程序都会收到这个消息的副本。行为相当于fanout类型的exchange。**   
下面详细来说:

**7.2 Bindings 绑定**

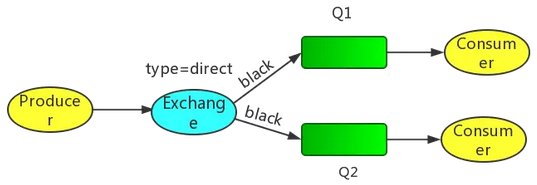
绑定其实就是关联了exchange和queue，或者这么说:queue对exchange的内容感兴趣,exchange要把它的Message deliver到queue。

**7.3Direct exchange**

Driect exchange的路由算法非常简单:通过bindingkey的完全匹配，可以用下图来说明.

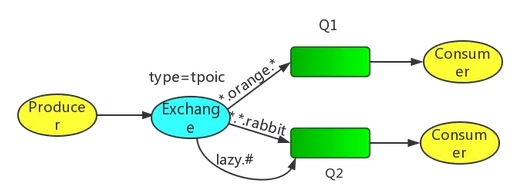
  
   
Exchange和两个队列绑定在一起,Q1的bindingkey是orange，Q2的binding key是black和green.   
当Producer publish key是orange时,exchange会把它放到Q1上,如果是black或green就会到Q2上,其余的Message被丢弃.

**7.4 Multiple bindings**

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.   
 

**7.5 Topic exchange**

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o,第一个key描述了celerity(灵巧),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   
    rounting\_key 为”lazy.orange.rabbit.hujj.ddd”会被投递到Q2中,#匹配0个或多个单词。

**8.消息序列化**

RabbitMQ使用ProtoBuf序列化消息,它可作为RabbitMQ的Message的数据格式进行传输,由于是结构化的数据,这样就极大的方便了Consumer的数据高效处理,当然也可以使用XML，与XML相比,ProtoBuf有以下优势:   
1.简单   
2.size小了3-10倍   
3.速度快了20-100倍   
4.易于编程   
6.减少了语义的歧义.   
，ProtoBuf具有速度和空间的优势，使得它现在应用非常广泛