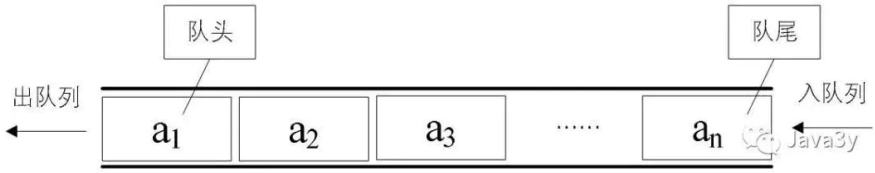
消息队列分享

2020年7月6日

一、什么是消息队列（简述）

消息队列（Message Queue，简称MQ），从字面意思上看，本质是个队列，FIFO先入先出，只不过队列中存放的内容是message而已。

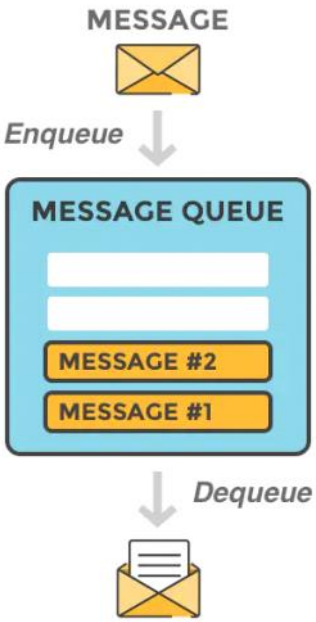
1.队列:



java中的queue功能较简单，没有持久化，可存储队列较少。

2.邮件:

消息队列可以简单理解为类似于邮件



3.简单消息队列

某个消息队列中间件描述的队列是这样子的 ：

P（producer）代表生产者：发送消息到队列
Q（queue）中间红色是队列：存储消息（类似于邮件）
C（consumer）代表消费者：接受消息

4.医院挂号

某医院挂号队列:

病人是生产者;

自己在队列中;

医院或者挂号窗口是消费者。
时代在进步，
《东华标准版数字化医院信息管理系统》
http://58.56.200.231:8888/dthealth/web/csp/dhc.logon.csp
线上预约挂号，从此告别线下排队

5.聚餐

生产者是厨师，
队列是饭菜，
消费者是消费者。

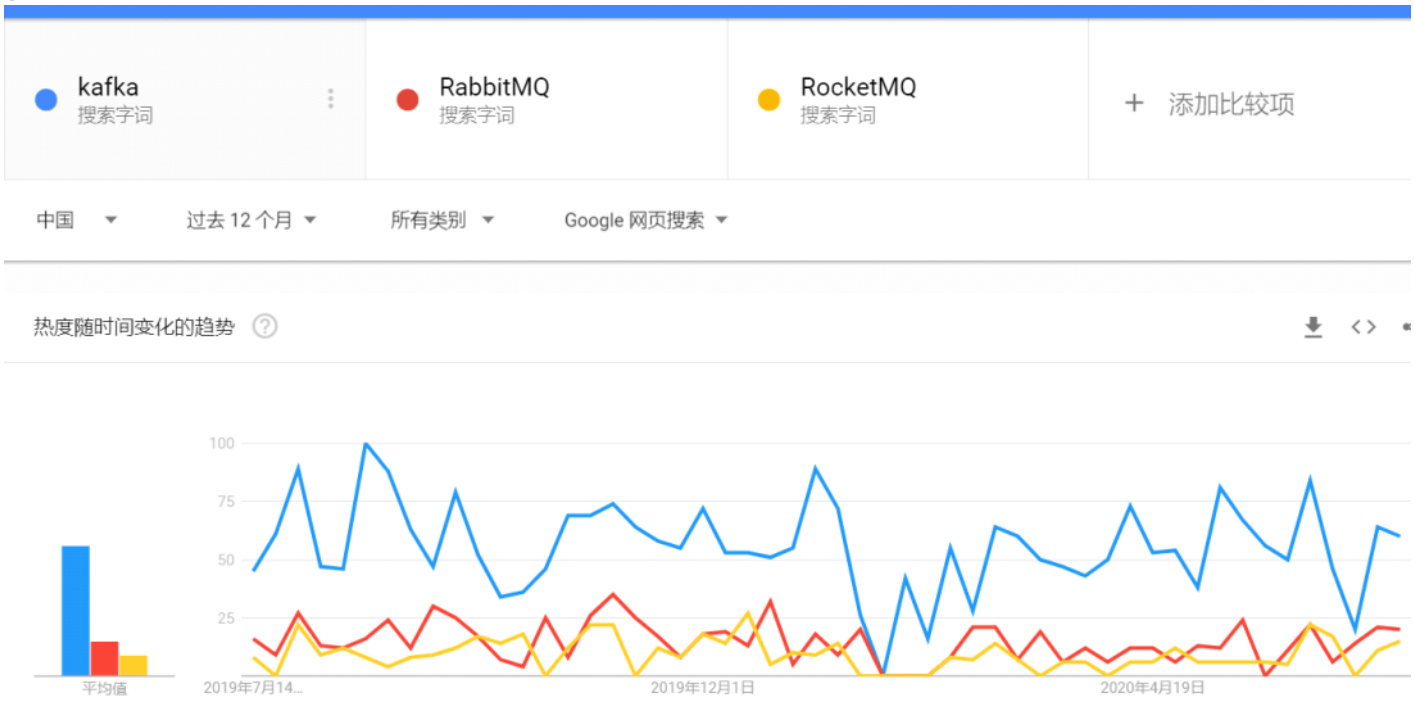
二、各种消息队列的产品的特点，以及我们的选择?

市面上比较流行的消息队列有Kafka，RabbitMQ，RocketMQ
kafka追求高吞吐量，一开始的目的就是用于日志收集和传输；
RabbitMQ 支持对消息的可靠传递，支持事务，开源社区活跃，管理界面良好；
RocketMQ 高吞吐量，高可靠性，有开源版和商业版，国内用户较多，开源版本不够成熟。

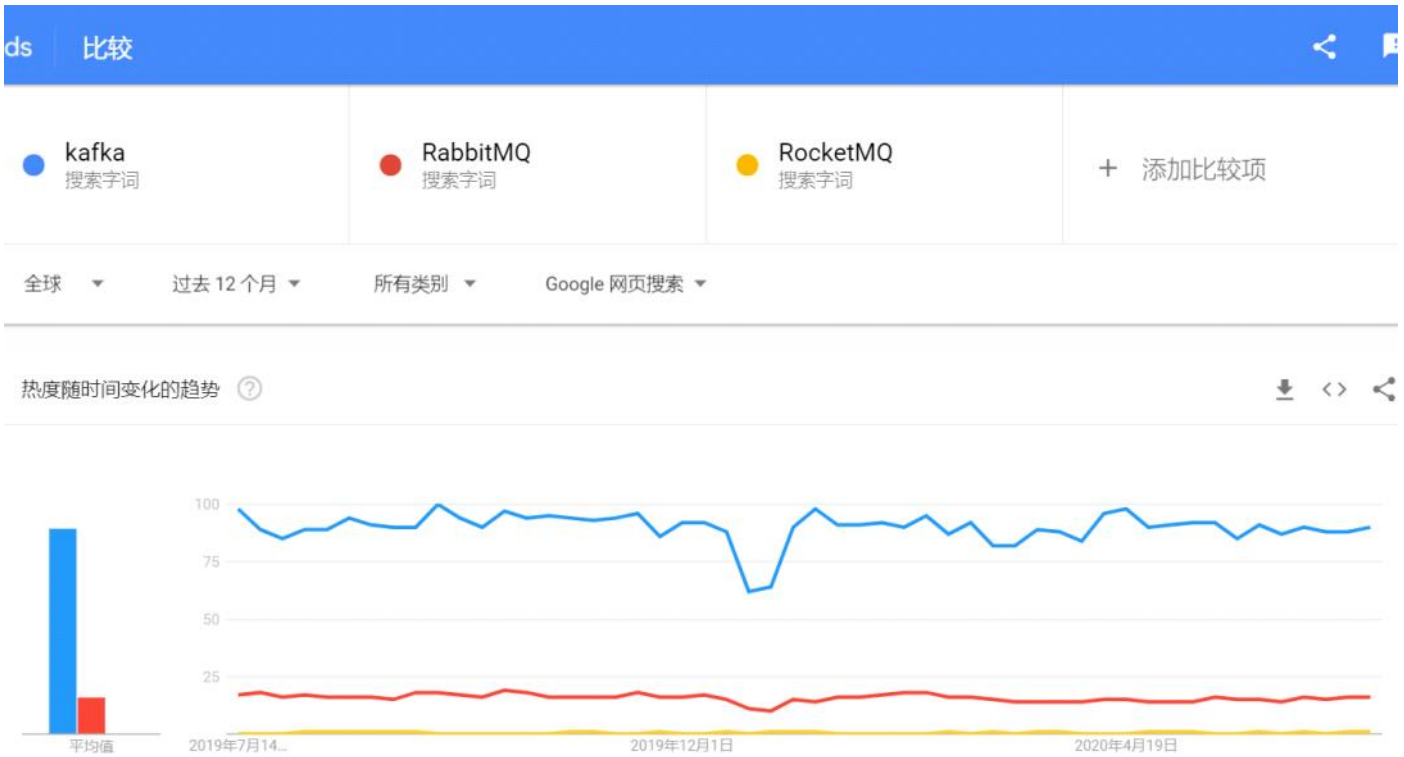
推荐RabbitMQ（恶龙打算内定，可有壮士挑战恶龙），管理界面良好，可靠传递，社区活跃，相同问题容易找到解决方案

先简单说下，跳word文档稍微瞅瞅

谷歌趋势，中国地区https://trends.google.com/trends/explore?geo=CN&q=kafka,RabbitMQ,RocketMQ



谷歌趋势，全球地区

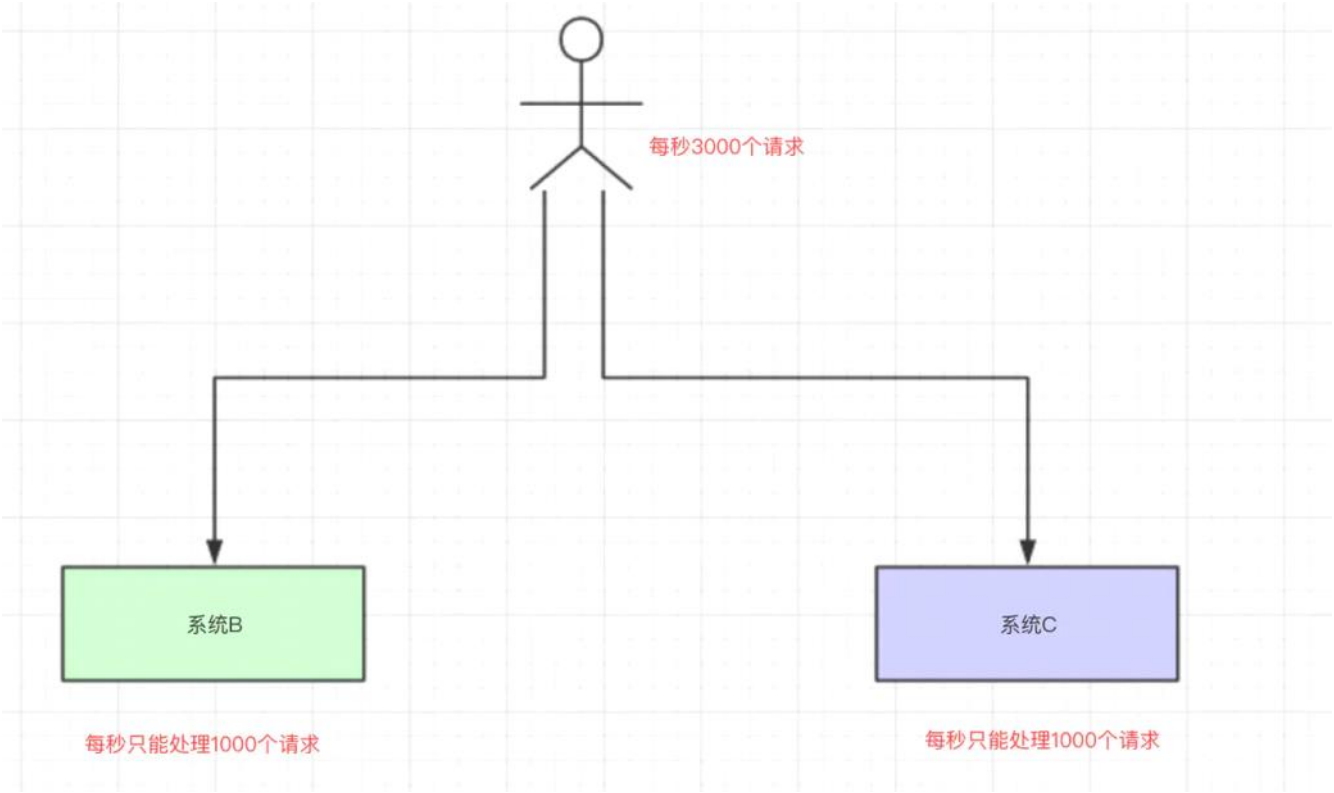


三、消息队列的优点?

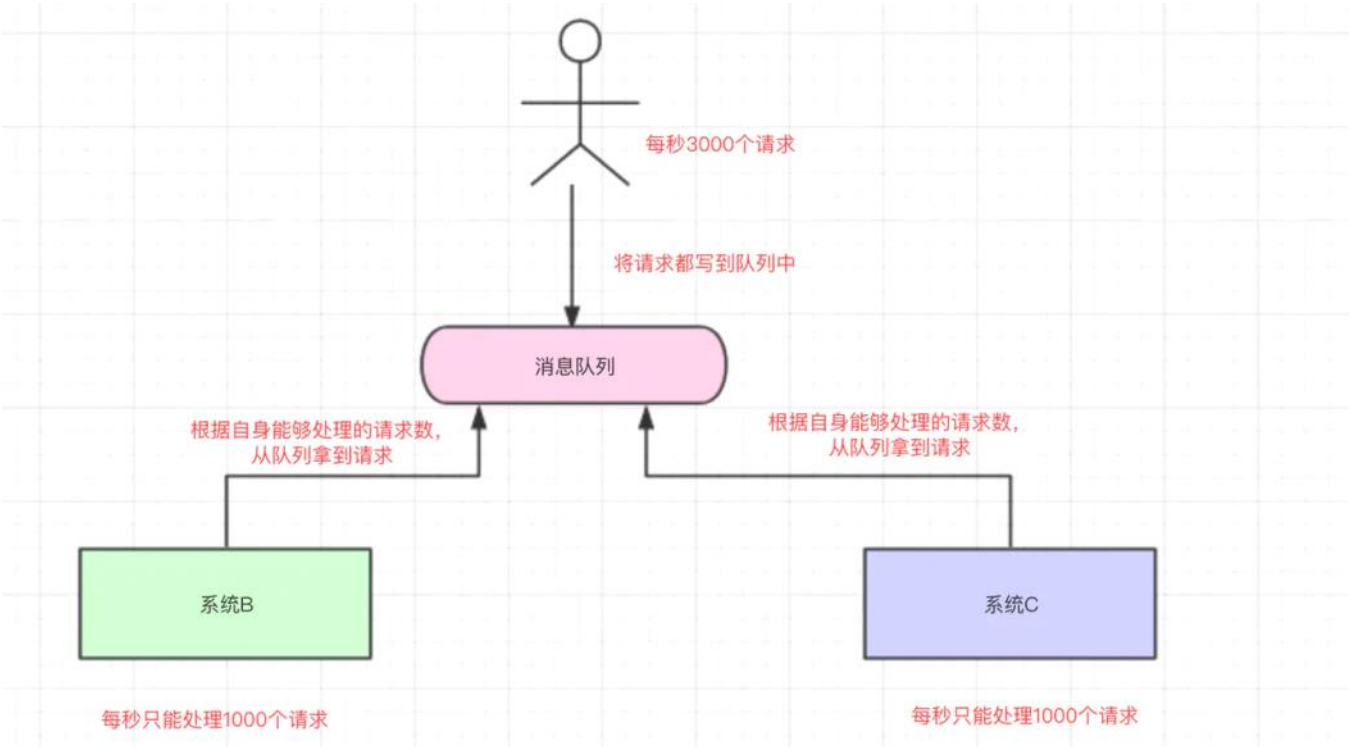
消峰:

我们来一个场景，现在我们每个月要搞一次大促，大促期间的并发可能会很高的，比如每秒

3000个请求。假设我们现在有两台机器处理请求，并且每台机器只能每次处理1000个请求。



那多出来的1000个请求，可能就把我们整个系统给搞崩了...所以，有一种办法，我们可以写到消息队列



系统B和系统C根据自己的能够处理的请求数去消息队列中拿数据，这样即便有每秒有8000个请求，那只是把请求放在消息队列中，去拿消息队列的消息由系统自己去控制，这样就不会把整个系统给搞崩。

最近：

最近某医院挂号，提交订单时，会提示“操作人数过多”，我多次操作，都是同样的提示，过上半小时再操作，告诉我号没了，既然你不给我号，浪费我时间弄啥呢。

解决方案：技术上使用消息队列，用户一次下单，最终挂号成功与否，可以在处理后微信或短信

通知。

异步：

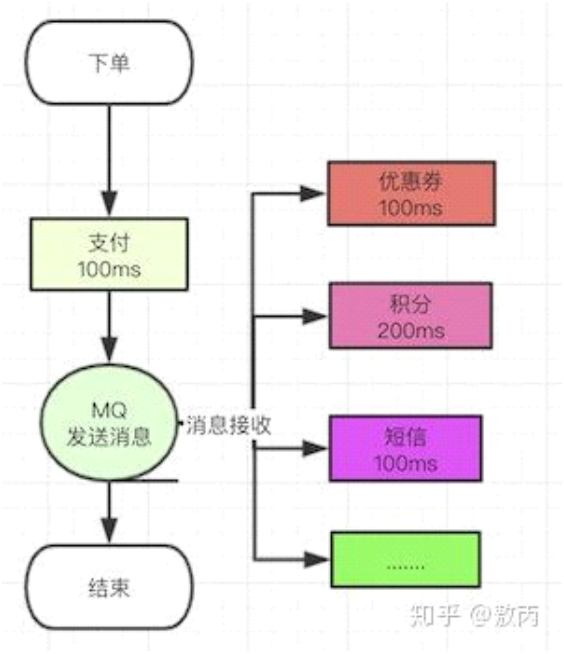
在项目中，将一些无需即时返回且耗时的操作提取出来，进行了异步处理，提高响应。
什么是异步？调用者调用了被调用者之后，并不等待业务返回结果，一般是被调用者通过回调方法来通知调用者。
订单支付的例子：
刚开始简单流程，下单付款就好，后来加了许多其它步骤，如下


```
public class Order{

    public void pay() {
        do 订单支付成功
        do 消耗优惠券
        do  增加积分
        do  发送短信
        增加信用度

        Return "支付成功";
    }
}
```

据知情人士透露，优秀的电商里面，这个中间可能有10个以上流程，如果这么处理可能等待几十秒。
改造后，支付完成的消息发送给发布订阅交换机，发布订阅交换机将相同内容的消息广播给优惠券、积分、短信等。




```
public class Order{

    public void pay() {
        do 订单支付成功
        Send orderId=007 到 订单支付成功的消息队列
        Return "支付成功";
    }
}
```

我们的例子，资质审批：

我们在资质审批的时候，分为两步，资质审批和资质同步，审批本身并不耗时，可以将特别耗时的资质同步提取出来。（稍后会有项目中实际代码举例）

解耦:

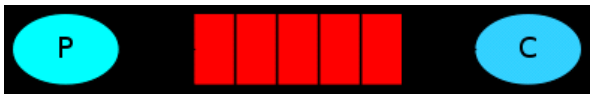
使用了消息队列它的所有优势就都有了，我们用异步的图解释解耦。

这个例子中会有优惠券，积分，短信系统，这些消息，往后可能还有系统A、B、C、D，每次需要修改系统下单逻辑，耦合度太高；而使用消息队列，“发布订阅”模式下，当新的系统增加或减少时候，通过增加或减少消息订阅者来处理，不用修改代码，不用重启。

四、消息队列的几种模式

简单队列模式、工作队列模式、发布订阅者模式（广播）模式、直接模式、主题模式

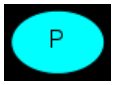
简单队列模式:



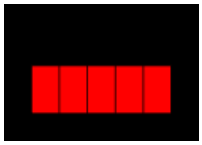
我们从传统的hello world开始

RabbitMQ是一个消息中间件，他接受和转发消息。你可以把它当作邮局：当你把邮件放到邮箱时，你确信邮寄员最终会把你的信件传递给收件人。类似的，RabbitMQ是一个邮箱，邮局和邮寄员。 RabbitMQ 和邮局不同之处在于它不是处理报纸，而是接收，存储和转发二进制数据块-消息。 RabbitMQ，发送消息使用的一些行业术语。

- 生产没有别的意思，只是发送。一个发送消息的程序是一个生产者。



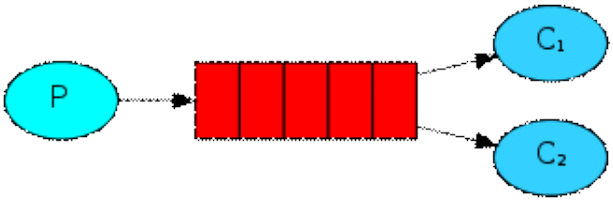
- 队列是一个居住在RabbitMQ中的邮箱，尽管消息流过RabbitMQ和你的应用，它们只能被存储在一个队列中。一个队列只会被主机内存和硬盘限制。它本质上是一个大的消息缓冲器。许多生产者可以发送消息到一个队列，许多消费则可以尝试接收来自一个队列里的数据。 我们是这样表示一个队列的：



- 消费和接收有相似的意思。一个消费者是一个程序大部分时候等待接收消息：

注意生产者，消费者，和队列(原文这里是broker，我猜测应该broker的核心是队列)不需要住在一起；事实上大多数应用中它们也没有。一个应用可以即是生产者又是消费者。 "Hello World" （使用 Spring 先进的消息队列协议（Advanced Message Queuing Protocol (AMQP))) 这部分教程我们会写两个程序使用spring-amqp库;一个生产者发送一个单条消息，消费者接收消息并打印它们。我们会掩饰一些spring amqp api细节，专注于简单的事情来开始。这是一个"Hello World"的消息。 下图。"P"是生产者"C"是消费者。中间的盒子是队列-一个RabbitMQ为消费者保持消息缓冲区。

工作队列模式



在第一个教程中，我们写了程序去发送和接收消息从一个制定队列中。在这个章节我们会创建一个工作队列用来分配耗时的任务(每个任务都很耗时，每个任务分配给一个工作者)在多个工作者之间。 工作队列的主要思想是避免立刻去做和完成一个资源密集型任务（多个资源可能会有多个请求，比较耗时(一下子做不完)，前面已经有人在排队了，并不能立即处理。类似银行办业务，我们生产了复杂任务，它们排成一个队列，多个业务员帮我们处理，这种任务有些时候不能立刻给你执行，也不能立刻给你做完）。而是安排稍后做完成它们。我们把任务封装成消息发送到队列中。一个在后台运行的工作者进程会去除任务并最终执行它。当你运行多个工作者进程时，任务会被它们分享。 这中概念在网站应用中很有用(那些无法在一个网络请求窗口处理的复杂任务)。

这份教程前一部分我们发送了一个消息包含"hello world".现在我们发送的字符代表复杂的任务。我们现实世界的任务。比如改变图片尺寸或者渲染PDF文档。所以我们伪造一个假装我们很忙-通过线程睡觉的方法。我们在字符串中妇孺了大量的...来让它看起来很复杂；每个点会一秒的工作量。比如说一个伪造的任务如Hello... 会消耗3秒。

消息收到确认(也就是ack)

做一个任务可能需要几秒，你或许会好奇当一个消费者开始一个长任务且干到一般儿的时候死掉了会发生什么事情。 官方文档（// TODO 有兴趣了，瞅瞅） Spring AMQP默认会采取保守措施来消息确认(看样子是重新排队)。如果一个监听器抛出一个异常，容器(我猜测是RabbitMQ)调用 channel.basicReject(deliveryTag, requeue) 重新排队时默认设置，除非你明确设置： defaultRequeueRejected=false (此处原文使

用了or，我猜测此处是因果关系，上面设置了false，抛出了后面的异常)监听器抛出异常AmqpRejectAndDontRequeueException。这是典型的行为你想要从你的监听器获得的。再这种模式不用去担心忘记收到消息确认。处理完消息后，监听器会调用channel.basicAck() 消息收到确认必须和消息接收的的通道在同一个通道。尝试使用不同的通道确认会导致通道级别异常。可以参考确认指导去学习更多。Spring AMQP一般注意这点除非我们组合使用代码和客户端组合使用，这是一些需要上心的。

发布订阅者模式

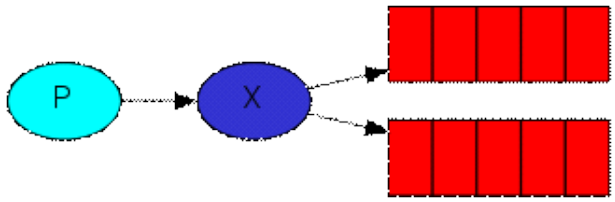
在这章我们会实现扇子模式(什么鬼)来交付一个消息到多个消费者。这种模式也被熟知为“发布订阅者模式”。 本质上，发布消息将会广播到所有接收者。

交换

在之前的章节我们发送和接收消息来自于一个队列。现在是时候介绍在RabbitMQ中完整的消息模型了。 让我们（荡起双桨）回顾下之前的章节我们都学了些什么

- 一个生产者是一个发送消息的用户应用
- 一个队列是一个存储消息的缓冲器
- 一个消费者是一个接收的用户应用

RabbitMQ中消息模型的核心思想是生产者永远不直接发送任何消息给队列。大多数时候生产者甚至不知道一个消息是否被交付给任何队列。 取而代之，生产者只发送消息给交换器。一个交换器是个很简单的东西。它一方面从盛传着接收消息另一方面把它们推到队列里。交换器必须完全的知道它要用自己接收到的消息做什么。是否应该把他们追加到一个特定的队列中？是否应当追加到多个队列中？或者是否丢弃？。规则由交换器类型定义。

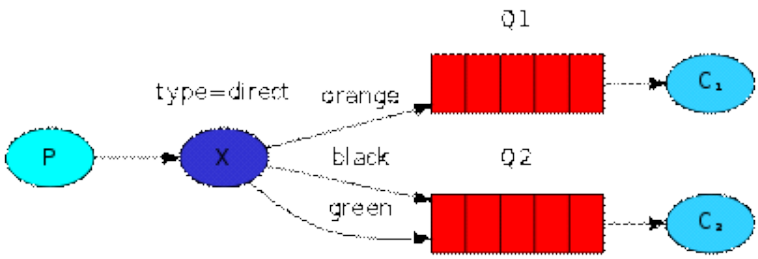


、

直接模式

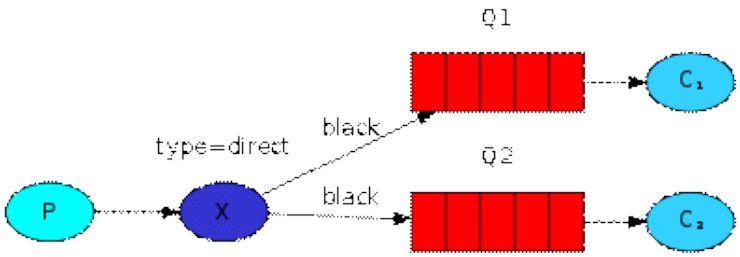
直接交换机 前面教程我们的消息系统广播所有消息到消费者。我们想扩展它通过允许基颜色类型的消息过滤器。举个例子，我们想要一个程序接收到的错误日志消息写到硬盘，而不浪费硬盘空间在警告和信息日志上。 我们使用的生产者消费者（fanout交换机）模式不能给我们提供这样的灵活性-它只能无脑的广播。 我们将会使用直接交换机代替，它背后的路由算法很简单--一个消息路由关键字和它所要去的队列绑定的关键字匹配。

为了说明它，考虑如下设置



在这个设置中，我们看到直接交换机 X又两个队列绑定在它身上。第一个队列的绑定关键字是orange,第二个又两个绑定，一个是black，另一个是green。 在这样设置中一个带有orange关键字的消息发布到交换机将会被路由到Q1队列，带有路由关键字black和green的消息会被发送到队列Q2.其它所有的消息会被丢弃。

多绑定



给多个队列绑定同一个绑定关键字是合法的。在我们的例子中我们可以增加一个绑定在X和Q1使用关键字black.如此，直接交换机的行为像fanout将广播消息到所有匹配的队列中。一个关键字为black的消息将会被传递到Q1和Q2.

发布消息

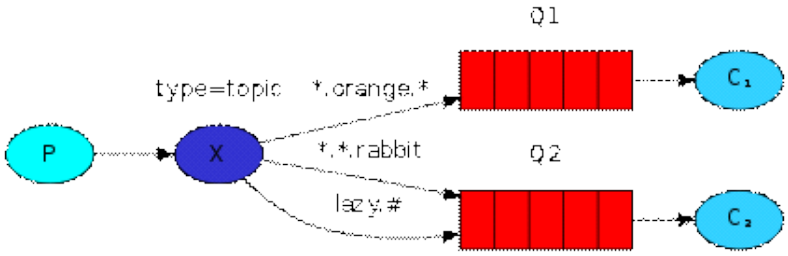
我们将会使用这种模型在我们的路由系统中，我们将会使用直接交换机替代fanout来发送消息。我们提供颜色来当作路由关键字。那种方式使接收程序可以通过颜色来选择想要接受的消息(或者订阅).让我们先关注于发送消息。

主题模式

前面我们提升了消息的灵活性。替代了使用fanout交换机只能广播。我们使用direct交换机。获得了通过路由迷失选择接受消息的可能性。 尽管使用direct交换机提升我们系统，他依然有限制，不能路由根据多个标准。 在我们消息系统中我们不仅仅想要根据路由密匙订阅队列，我们也想根据生产消息的源头订阅。你或许知道这个概念从syslog unix工具。它路由日志根据严重性(消息/警告……)和facility（设备？）(auth/cron/kern……)。我们的例子类似。 那个例子会给我们很多灵活性。我们只想监听严重错误来自于'cron' 和所有来自于'kern'. 为了实现那种灵活性在我们的日志系统中，我们需要学习更复杂的topic交换机。

发送到主题交换器的消息不能含有随意的路由关键字-它必须是一组单词，由**.** 分割。单词可以是任何东西，但是通常它们指定一些和消息关联的属性。一个由价值的路由密匙例子: "stock.usd.nyse", "nyse.vmw", "quick.orange.rabbit". 可以由很多个单词在路由关键字中如果你喜欢，上限时255字节。 绑定关键字必须是同样的格式。topic exchange背后的逻辑跟direct exchage类似-消息携带特定路由关键字传递给所有队列那些绑定了匹配绑定关键字的。（从对象的角度讲，交换机是个对象，队列时个对象，绑定也是个对象，消息携带的关键字和绑定的关键字匹配，消息发送到相应队列。嗯，topic 交换机和队列是一对多，和绑定也是一对多，队列和绑定是一对一）。然而又两个特别重要的例子关于绑定关键字。

- *(星)可以替代确定的一个单词
- #(哈希)可以替代0个或多个单词 这在例子中很好解释



此例中，我们发送的消息都是描述动物的。这些消息北宋宋的时候携带的路由密匙又三个单词组成(两个.)。路由密匙中第一个单词描述速度，第二个是颜色，第三个是物种："..". 我们创建三个绑定:Q1队列绑定关键字 ".orange."和Q2的"..rabbit"和"lazy.#". 这些分类总结为：

- Q1 队列对所有黄颜色的动物感兴趣。
- Q2 想要监听所有关于兔子，和懒的动物。

一个消息路由关键字被设置为"quick.orange.rabbit"将会被传递给两个队列。消息"lazy.orange.elephant"也会。另一方面"quick.orange.fox"只会去队列1, "lazy.brown.fox"只会去队列2. "lazy.pink.rabbit"指挥去第二队列一次,尽管匹配了两次. "quick.brown.fox"没有匹配任何一个，所以会被丢弃。 如果我们破坏了协议发送了一个消息有四个单词会发生什么呢，比如"orage"或者"quick.orange.male.rabbit"?好吧，这些消息不匹配任何绑定然后会丢失。 另一方面"lazy.orange.male.rabbit",尽管它有四个单词，会匹配最后一个绑定会被传递到队列2中去。

五、消息队列关键字解释

生产者：发送消息到队列

消费者：接受消息

交换机：指定消息按照什么规则到哪个队列

队列：存储消息

绑定：把交换机和队列按照路由规则绑定起来，也就是交换机和队列的虚拟连接

路由关键字：交换机根据它来进行消息投递

频道:“多个轻量级连接共享一个TCP连接”，每一个协议操作都是发生在频道里的。类似于电视频道。

其它:

消息持久化是默认的。

Rabbitmq工作模式例子 https://github.com/coldbloodanimal/demo.git 中的demo-rabbitmq

rabbitmq订单例子地址：https://github.com/coldbloodanimal/demo-rabbitmq.git

docker安装:

https://blog.csdn.net/dang7758/article/details/89634749

参考与引用:

消息队列：https://juejin.im/post/5cb025fb5188251b0351ef48#heading-5

异步解释：https://www.cnblogs.com/IT-CPC/p/10898871.html

异步解释：https://www.zhihu.com/question/34243607/answer/1023686807