RabbitMQ基础入门

1 消息中间件(MQ)

1.1 消息队列回顾

消息队列中间件是分布式系统中重要的组件，主要解决应用解耦，异步消息，流量削锋等问题，实现高性能，高可用，可伸缩和最终一致性架构。目前使用较多的消息队列有ActiveMQ，RabbitMQ，ZeroMQ，Kafka，MetaMQ，RocketMQ。消息中间件到底该如何使用，何时使用这是一个问题，胡乱地使用消息中间件增加了系统的复杂度，如果用不好消息中间件还不如不用。

消息中间件利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。对于消息中间件，常见的角色大致也就有`Producer（生产者）`、`Consumer（消费者）`。

1.2 消息队列特点有哪些?

1.2.1 先进先出

> 先进先出是队列的一个特性

不能先进先出，都不能说是队列了。消息队列的顺序在入队的时候就基本已经确定了，一般是不需人工干预的。而且，最重要的是，**数据是只有一条数据在使用中。** 这也是MQ在诸多场景被使用的原因。

1.2.2 发布订阅

> 一般是MQ的广播模式，类似于java的观察者模式

发布订阅是一种很高效的处理方式，如果不发生阻塞，基本可以当做是同步操作。这种处理方式能非常有效的提升服务器利用率，这样的应用场景非常广泛。

1.2.3 持久化

持久化确保MQ的使用不只是一个部分场景的辅助工具，而是让MQ能像数据库一样存储核心的数据。

1.2.4 分布式

在现在大流量、大数据的使用场景下，只支持单体应用的服务器软件基本是无法使用的，支持分布式的部署，才能被广泛使用。而且，MQ的定位就是一个高性能的中间件。

1.3 消息队列通讯模式是什么?

1.3.1 点对点通讯点

对点方式是最为传统和常见的通讯方式，它支持一对一、一对多、多对多、多对一等多种配置方式，支持树状、网状等多种拓扑结构。

1.3.2 发布/订阅(Publish/Subscribe)模式

发布/订阅功能使消息的分发可以突破目的队列地理指向的限制，使消息按照特定的主题甚至内容进行分发，用户或应用程序可以根据主题或内容接收到所需要的消息。发布/订阅功能使得发送者和接收者之间的耦合关系变得更为松散，发送者不必关心接收者的目的地址，而接收者也不必关心消息的发送地址，而只是根据消息的主题进行消息的收发。在MQ家族产品中，MQEventBroker是专门用于使用发布/订阅技术进行数据通讯的产品，它支持基于队列和直接基于TCP/IP两种方式的发布和订阅。

1.3.3 群集(Cluster)

为了简化点对点通讯模式中的系统配置，MQ提供Cluster(群集)的解决方案。群集类似于一个域(Domain)，群集内部的队列管理器之间通讯时，不需要两两之间建立消息通道，而是采用群集(Cluster)通道与其它成员通讯，从而大大简化了系统配置。此外，群集中的队列管理器之间能够自动进行负载均衡，当某一队列管理器出现故障时，其它队列管理器可以接管它的工作，从而大大提高系统的高可靠性。

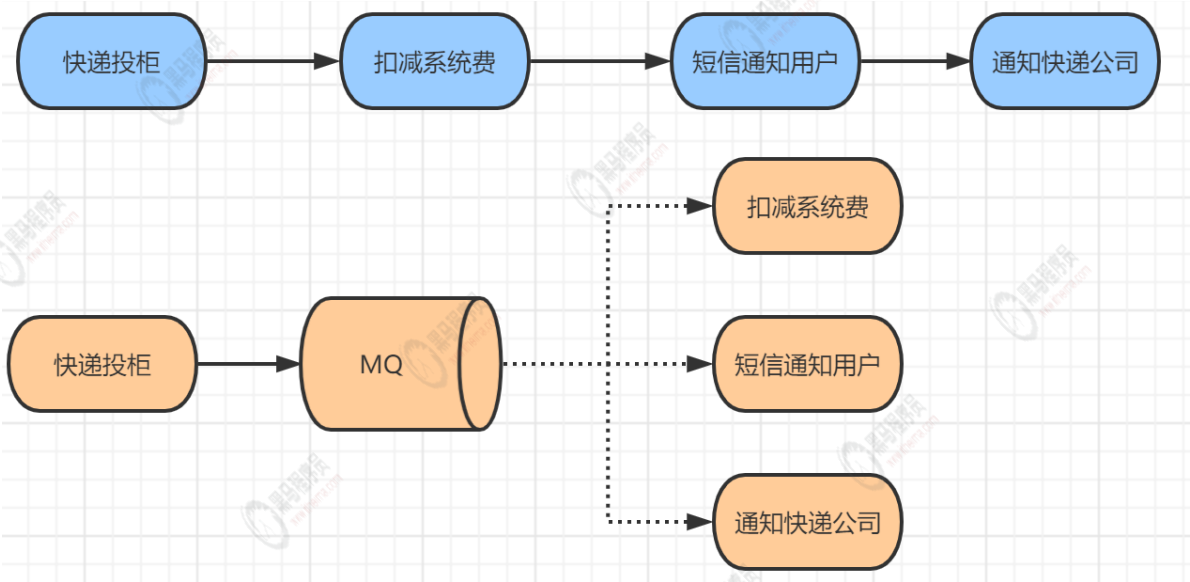
1.4 为什么使用消息队列?

> 其实就是问问你消息队列都有哪些使用场景，然后你项目里具体是什么场景，说说你在这个场景里用消息队列是什么?

先说一下消息队列常见的使用场景吧，其实场景有很多，但是比较核心的有 3 个：**解耦**、**异步**、**削峰**。

1.4.1 应用解耦

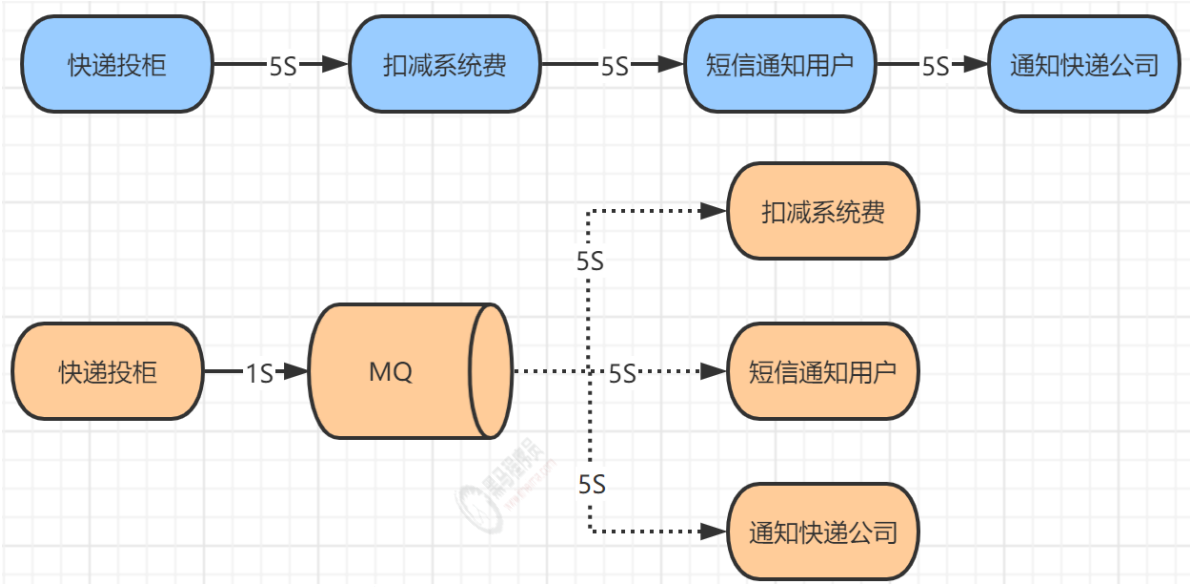
在快递柜业务流程中，快递员投柜后需要经历扣减系统费、短信通知用户和推送通知快递公司三个业务动作。传统做法需要依次执行这些业务东西，如果其中某一步异常（例如用户手机未开机或者快递公司接口故障），将会延迟甚至中断整个投柜流程，严重影响用户体验。



如果接口层收到投柜数据后，写入消息到MQ，后续三个子系统各自消费处理，将可以完美解决该问题，并且子系统故障不影响上游系统！此为 **解耦**！

1.4.2 异步

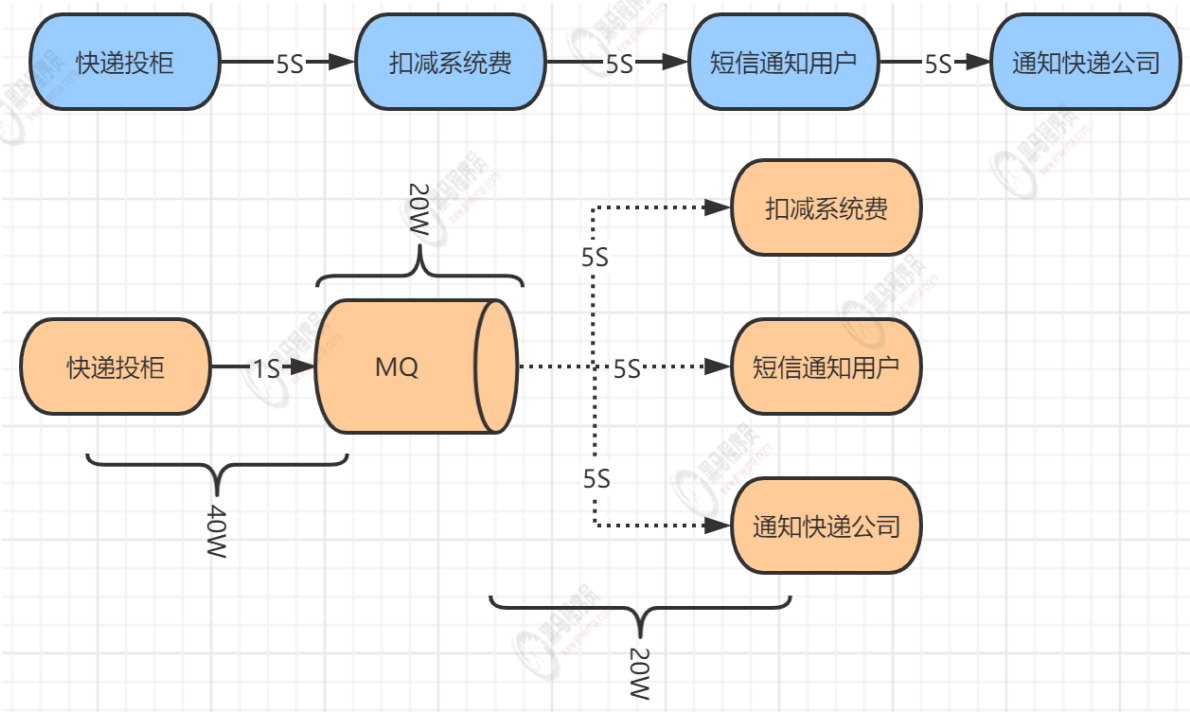
比如快递投柜后，用户马上就结束了，不会等待到发送短信或者通知快递公司结束的，直接将消息投递到MQ,然后就直接结束，具体到扣减系统费以及后续的通知，都是异步操作的，不需要用户关心的，着就是将用户的同步操作转换为异步操作。



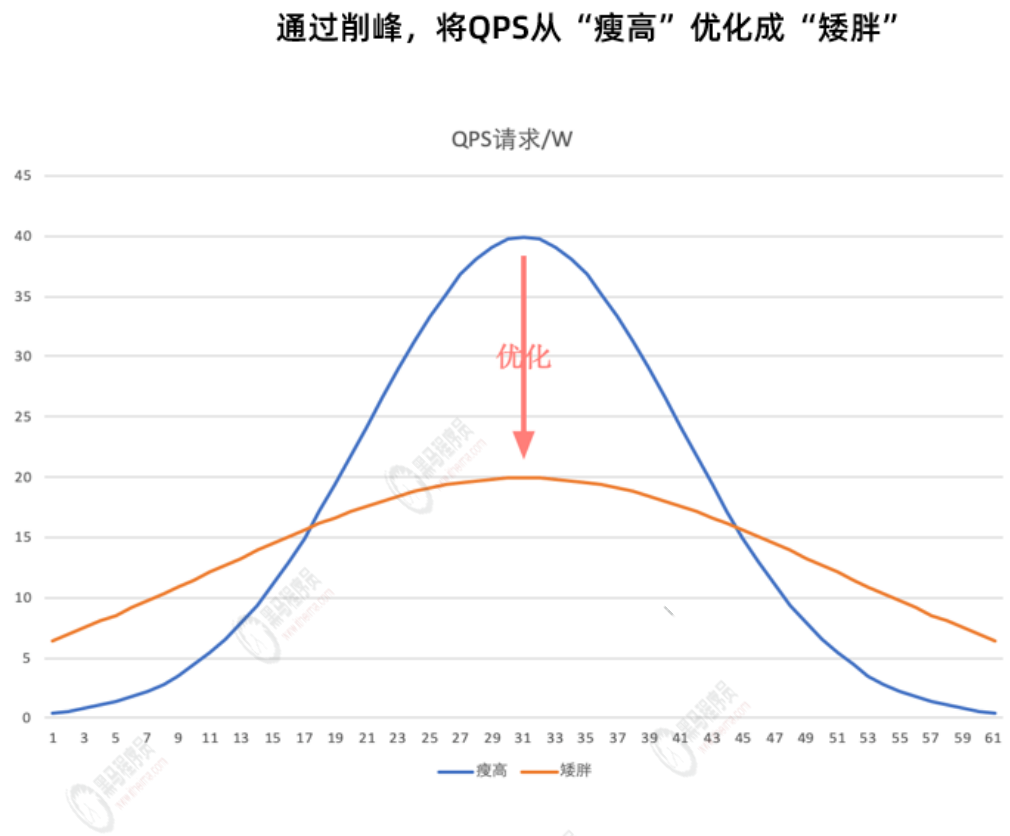
如果全部同步操作需要15S，而发送到MQ后交给系统异步处理用户只需要1S就可以完成操作。

1.4.3 流量削峰

> 就像用户投递快递，高峰到40W每秒，但是我们的后续处理业务每秒之恶能20W，还剩下20W在MQ进行堆积，这就是MQ很重要的流量削峰的能力，将用户的洪峰流量，让后台慢慢来处理，MQ承担一个缓冲的左右



就像这个波形图一样，如果用户请求的并发量的最高峰时40W，系统的承载能力只能达到30W，就可以使用MQ进行削峰，将系统最高40W的并发削峰为最高只有20万，就是时间换空间的做法。



> 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。

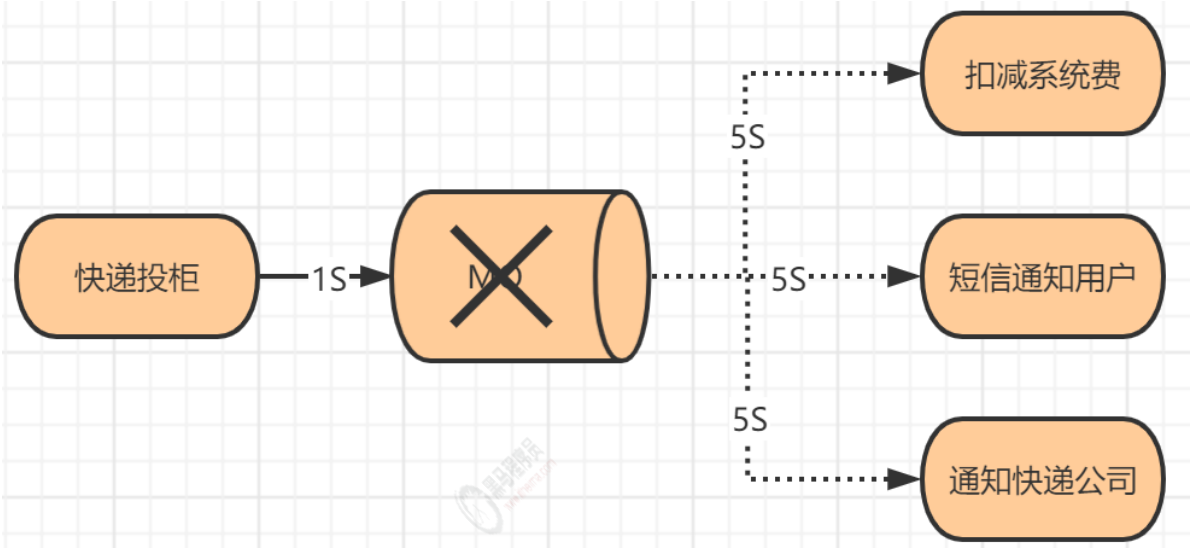
1. 可以控制活动人数，超过此一定阀值的订单直接丢弃
2. 可以缓解短时间的高流量压垮应用(应用程序按自己的最大处理能力获取订单)
3. 用户的请求,服务器收到之后,首先写入消息队列,加入消息队列长度超过最大值,则直接抛弃用户请求或跳转到错误页面.
4. 秒杀业务根据消息队列中的请求信息，再做后续处理.

1.5 消息队列有什么缺点?

> 优点上面已经说了，就是**在特殊场景下有其对应的好处，解耦、异步、削峰。**

1.5.1 系统可用性降低

消息队列在系统中充当一个中间人的身份，如果该中间人突然失联了，那其他两方就不知所措了，最后也就导致系统之间无法互动。



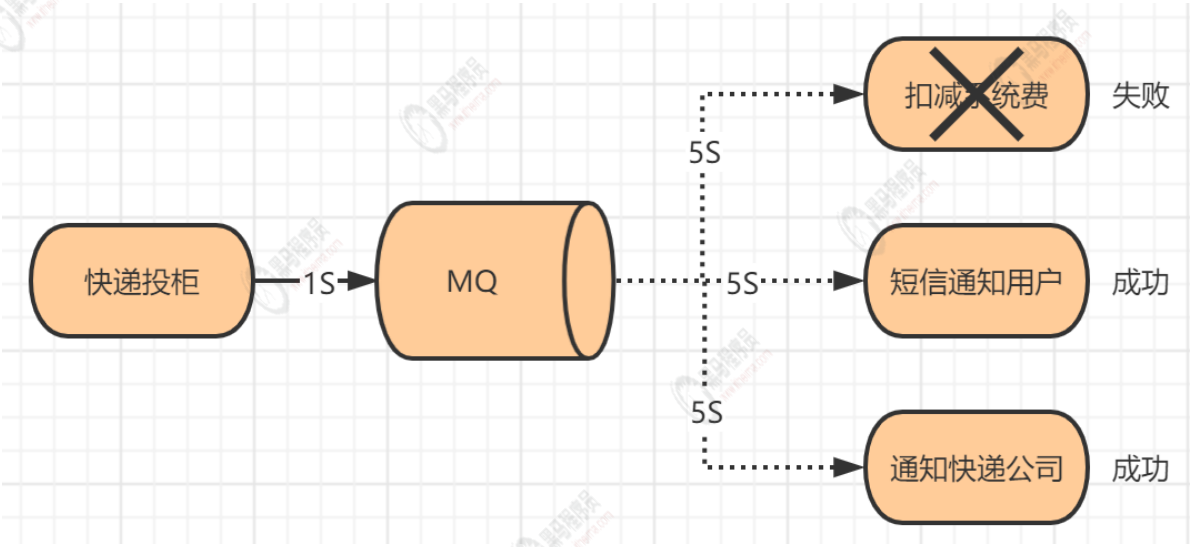
就像我们投递快递，如果MQ出现问题，那麽我们整个系统调用链路就会断开，前后端将无法通讯

1.5.2 系统复杂性提高

在使用消息队列的过程中，难免会出现生产者、MQ、消费者宕机不可用的情况，那么随之带来的问题就是消息重复、消息乱序、消息堆积等等问题都需要我们来控制。

1.5.3 一致性问题

如下图所示，系统需要保证快递投递，扣减系统费，通知等之间的数据一致性，如果系统短信通知，快递通知执行成功，扣减系统费执行失败时，就会出现数据不一致问题



如果出现这种问题，我们需要有应对方案，比如重试等方案。

所以消息队列实际是一种非常复杂的架构，你引入它有很多好处，但是也得针对它带来的坏处做各种额外的技术方案和架构来规避掉，做好之后，你会发现，妈呀，系统复杂度提升了一个数量级，也许是复杂了 10 倍。但是关键时刻，用，还是得用的。

2 常用消息队列有哪些?

消息队列是分布式应用间交换信息的重要组件，消息队列可驻留在内存或磁盘上, 队列可以存储消息直到它们被应用程序读走。

通过消息队列，应用程序可以在不知道彼此位置的情况下独立处理消息，或者在处理消息前不需要等待接收此消息。

所以消息队列可以解决应用解耦、异步消息、流量削锋等问题，是实现高性能、高可用、可伸缩和最终一致性架构中不可以或缺的一环。

现在比较常见的消息队列产品主要有ActiveMQ、RabbitMQ、ZeroMQ、Kafka、RocketMQ等

2.1 ActiveMQ

> ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现，尽管JMS规范出台已经是很久的事情了，但是JMS在当今的J2EE应用中间仍然扮演着特殊的地位。

2.1.1 特性

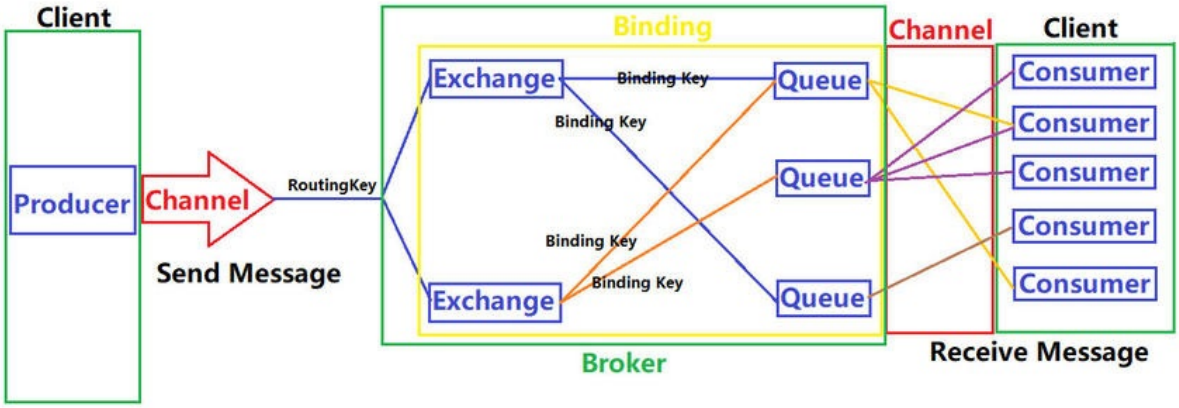
1. 多种语言和协议编写客户端。语言: Java,C,C++,C#,Ruby,Perl,Python,PHP。应用协议: OpenWire,Stomp REST,WS Notification,XMPP,AMQP
2. 完全支持JMS1.1和J2EE 1.4规范 （持久化，XA消息，事务)
3. 对Spring的支持，ActiveMQ可以很容易内嵌到使用Spring的系统里面去，而且也支持Spring2.0的特性
4. 通过了常见J2EE服务器（如 Geronimo,JBoss 4,GlassFish,WebLogic)的测试，其中通过JCA 1.5 resource adaptors的配置，可以让ActiveMQ可以自动的部署到任何兼容J2EE 1.4 商业服务器上
5. 支持多种传送协议：in-VM,TCP,SSL,NIO,UDP,JGroups,JXTA
6. 支持通过JDBC和journal提供高速的消息持久化
7. 从设计上保证了高性能的集群，客户端-服务器，点对点
8. 支持Ajax
9. 支持与Axis的整合
10. 可以很容易得调用内嵌JMS provider，进行测试

2.1.2 使用建议

一般的业务系统要引入 MQ，最早大家都用 ActiveMQ，但是现在确实大家用的不多了，没经过大规模吞吐量场景的验证，社区也不是很活跃，所以大家还是算了吧，我个人不推荐用这个了；

2.2 RabbitMQ

> RabbitMQ是流行的开源消息队列系统，用erlang语言开发。RabbitMQ是AMQP（高级消息队列协议）的标准实现。支持多种客户端，如：Python、Ruby、.NET、Java、JMS、C、PHP、ActionScript、XMPP、STOMP等，支持AJAX，持久化。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



2.2.1 特定

- erlang语言开发，性能极其好，延时很低
- 吞吐量到万级，MQ功能比较完备
- 开源提供的管理界面非常棒，用起来很好用
- 社区相对比较活跃，几乎每个月都发布几个版本分
- 在国内一些互联网公司近几年用rabbitmq也比较多一些

2.2.2 使用建议

大家开始用 RabbitMQ，但是确实 erlang 语言阻止了大量的 Java 工程师去深入研究和掌控它，对公司而言，几乎处于不可控的状态，但是确实人家是开源的，比较稳定的支持，活跃度也高，所以**中小型公司**，技术实力较为一般，技术挑战不是特别高，用 RabbitMQ 是不错的选择，对自家技术没有过高自信的话，可以使用RabbitMQ，人家有活跃的开源社区，绝对不会黄。

2.3 Kafka

APACHE kafka®
A distributed streaming platform

> Kafka是一种高吞吐量的分布式发布订阅消息系统，它可以处理消费者规模的网站中的所有动作流数据。 这种动作（网页浏览，搜索和其他用户的行动）是在现代网络上的许多社会功能的一个关键因素。 这些数据通常是由于吞吐量的要求而通过处理日志和日志聚合来解决。 对于像Hadoop的一样的日志数据和离线分析系统，但又要求实时处理的限制，这是一个可行的解决方案。Kafka的目的是通过Hadoop的并行加载机制来统一线上和离线的消息处理，也是为了通过集群机来提供实时的消费。

2.3.1 特性

- 通过O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以TB的消息存储也能够保持长时间的稳定性能。（文件追加的方式写入数据，过期的数据定期删除）
- 高吞吐量：即使是非常普通的硬件Kafka也可以支持每秒数百万的消息
- 支持通过Kafka服务器和消费机集群来分区消息
- 支持Hadoop并行数据加载

2.3.2 使用建议

如果是**大数据领域**的实时计算、日志采集等场景，用 Kafka 是业内标准的，绝对没问题，社区活跃度很高，绝对不会黄，何况几乎是全世界这个领域的事实性规范。

2.3.3 应用场景

> 一般应用在大数据日志处理或对实时性（少量延迟），可靠性（少量丢数据）要求稍低的场景使用。

2.3.3.1 日志收集

一个公司可以用Kafka可以收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer；

2.3.3.2 消息系统

解耦生产者和消费者、缓存消息等；

2.3.3.3 用户活动跟踪

kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后消费者通过订阅这些topic来做实时的监控分析，亦可保存到数据库；

2.3.3.4 运营指标

kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告；

2.3.3.5 流式处理

比如spark streaming和storm。

2.4 RocketMQ

> RocketMQ是阿里开源的消息中间件，纯Java开发，具有高吞吐量、高可用性、适合大规模分布式系统应用的特点。RocketMQ思路起源于Kafka，但并不是简单的复制，它对消息的可靠传输及事务性做了优化，目前在阿里集团被广泛应用于交易、充值、流计算、消息推送、日志流式处理、binglog分发等场景，支撑了阿里多次双十一活动。
>
> 因为是阿里内部从实践到产品的产物，因此里面很多接口、api并不是很普遍适用。可靠性毋庸置疑，而且与Kafka一脉相承（甚至更优），性能强劲，支持海量堆积。

2.4.1 特点

- 接口简单易用，源码是阿里出品，可自定义MQ
- topic可以达到几百，几千个的级别，吞吐量会有较小幅度的下降
- 消息可用性极高，经过参数优化配置，可以做到0丢失
- MQ功能较为完善，还是分布式的，扩展性好

2.4.2 使用建有

现在确实越来越多的公司会去用 RocketMQ，确实很不错，毕竟是阿里出品，但社区可能有突然黄掉的风险（目前 RocketMQ 已捐给 Apache，但 GitHub 上的活跃度其实不算高）对自己公司技术实力有绝对自信的，推荐用 RocketMQ，否则回去老老实实用 RabbitMQ 吧，**大型公司**，基础架构研发实力较强，用 RocketMQ 是很好的选择。

3 怎样选型MQ?

> 到底应该哪个方案，还是要看具体的需求。在我们的设计中，MQ的功能与业务无关，因此优先考虑使用已有的中间件搭建。那么具本选择哪个中间件呢?

3.1 需求分析

3.1.1 功能需求

除了最基本生产消费模型，还需要MQ能支持REQUEST-REPLY模型，以提供对同步调用的支持。 此外，如果MQ能提供PUBLISH-SUBSCRIBE模型，则事件代理的实现可以更加简单。

3.1.2 性能需求

考虑未来一到两年内产品的发展，消息队列的吞吐量预计不会超过 1W qps，但由单条消息延迟要求较高，希望尽量的短。

3.1.3 可用性需求

因为是在线服务，因此需要较高的可用性，但充许有少量消息丢失。

3.1.4 易用性需求

包括学习成本、初期的开发部署成本、日常的运维成本等。

3.2 横向对比

特性	ActiveMQ	RabbitMQ	Kafka	RocketMQ
PRODUCER-COMSUMER	支持	支持	支持	支持
PUBLISH-SUBSCRIBE	支持	支持	支持	支持
REQUEST-REPLY	支持	支持	-	支持
API完备性	高	高	高	低（静态配置）
多语言支持	支持，JAVA优先	语言无关	支持，JAVA优先	支持
单机吞吐量	万级	万级	十万级	单机万级
消息延迟	毫秒级	微秒级	毫秒级	毫秒级
可用性	高（主从）	高（主从）	很高（分布式）	非常高（分布式）
消息丢失	-	低	理论上不会丢失	理论上不会丢失
消息重复	-	可控制	理论上会有重复	允许重复
文档的完备性	高	高	高	中
提供快速入门	有	有	有	无
首次部署难度	-	低	中	高

注： - 表示尚未查找到准确数据

3.2.1 个人建议

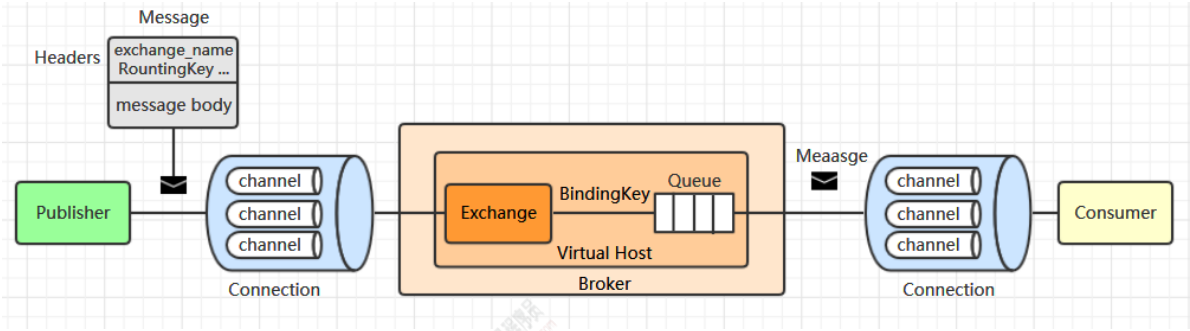
- 中小型公司，技术一般，可以考虑用 RabbitMQ；
- 大型公司，基础架构研发实力较强，用 RocketMQ 是很好的选择
- 实时计算、日志采集：使用 kafka；

3.3 如何选型?

MQ	描述
RabbitMQ	erlang开发，对消息堆积的支持并不好，当大量消息积压的时候，会导致 RabbitMQ 的性能急剧下降。每秒钟可以处理几万到十几万条消息。
RocketMQ	java开发，面向互联网集群化功能丰富，对在线业务的响应时延做了很多的优化，大多数情况下可以做到毫秒级的响应，每秒钟大概能处理几十万条消息。
Kafka	Scala开发，面向日志功能丰富，性能最高。当你的业务场景中，每秒钟消息数量没有那么多的时候，Kafka 的时延反而会比较高。所以，Kafka 不太适合在线业务场景。
ActiveMQ	java开发，简单，稳定，性能不如前面三个。小型系统用也ok，但是不推荐。推荐用互联网主流的。

4 RabbitMQ基本使用

> RabbitMQ 与 AMQP 遵循相同的模型架构，其架构示例图如下



4.1 重要概念

4.1.1 Publisher

> 消息生产者，就是投递消息的程序

发布者 (或称为生产者) 负责生产消息并将其投递到指定的交换器上。

4.1.2 Message

消息由消息头和消息体组成。消息头用于存储与消息相关的元数据：如目标交换器的名字 (exchange_name) 、路由键 (RountingKey)和其他可选配置 (properties) 信息。消息体为实际需要传递的数据。

4.1.3 Exchange

> 交换器负责接收来自生产者的消息，并将将消息路由到一个或者多个队列中，如果路由不到，则返回给生产者或者直接丢弃，这取决于交换器的 mandatory 属性：

- 当 mandatory 为 true 时：如果交换器无法根据自身类型和路由键找到一个符合条件的队列，则会将该消息返回给生产者；
- 当 mandatory 为 false 时：如果交换器无法根据自身类型和路由键找到一个符合条件的队列，则会直接丢弃该消息。

4.1.4 BindingKey

交换器与队列通过 BindingKey 建立绑定关系。

4.1.5 Routingkey

> 基于交换器类型的规则相匹配时，消息被路由到对应的队列中

生产者将消息发给交换器的时候，一般会指定一个 RountingKey，用来指定这个消息的路由规则。当 RountingKey 与 BindingKey

4.1.6 Queue

> 消息队列载体，每个消息都会被投入到一个或多个队列。

用于存储路由过来的消息。多个消费者可以订阅同一个消息队列，此时队列会将收到的消息将以轮询 (round-robin)

的方式分发给所有消费者。即每条消息只会发送给一个消费者，不会出现一条消息被多个消费者重复消费的情况。

4.1.7 Consumer

> 消息消费者，就是接受消息的程序

消费者订阅感兴趣的队列，并负责消费存储在队列中的消息。为了保证消息能够从队列可靠地到达消费者，RabbitMQ 提供了消息确认机制 (messageacknowledgement)，并通过 autoAck 参数来进行控制

- 当 autoAck 为 true 时：此时消息发送出去 (写入TCP套接字) 后就认为消费成功，而不管消费者是否真正消费到这些消息。当 TCP 连接或 channel 因意外而关闭，或者消费者在消费过程之中意外宕机时，对应的消息就丢失。因此这种模式可以提高吞吐量，但会存在数据丢失的风险。
- 当 autoAck 为 false 时：需要用户在数据处理完成后进行手动确认，只有用户手动确认完成后，RabbitMQ 才认为这条消息已经被成功处理。这可以保证数据的可靠性投递，但会降低系统的吞吐量。

4.1.8 Connection

用于传递消息的 TCP 连接。

4.1.9 Channel

> 消息通道，在客户端的每个连接里，可建立多个channel，每个channel代表一个会话任务。

RabbitMQ 采用类似 NIO (非阻塞式 IO) 的设计，通过 Channel 来复用 TCP 连接，并确保每个Channel的隔离性，就像是拥有独立的 Connection 连接。当数据流量不是很大时，采用连接复用技术可以避免创建过多的 TCP 连接而导致昂贵的性能开销。

4.1.10 Virtual Host

> 虚拟主机，一个broker里可以开设多个vhost，用作不同用户的权限分离

RabbitMQ 通过虚拟主机来实现逻辑分组和资源隔离，一个虚拟主机就是一个小型的 RabbitMQ服务器，拥有独立的队列、交换器和绑定关系。用户可以按照不同业务场景建立不同的虚拟主机，虚拟主机之间是完全独立的，你无法将 vhost1 上的交换器与vhost2 上的队列进行绑定，这可以极大的保证业务之间的隔离性和数据安全。默认的虚拟主机名为 `/` 。

4.1.11 Broker

简单来说就是消息队列服务器实体。

4.2 RabbitMQ安装

> 参考：https://baiyp.ren/RabbitMQ%E5%AE%89%E8%A3%85.html

4.3 如何使用RabbitMQ发送消息?

> exchange接收到消息后，就根据消息的key和已经设置的binding，进行消息路由，将消息投递到一个或多个队列里。

1. 客户端连接到消息队列服务器，打开一个channel。
2. 客户端声明一个exchange，并设置相关属性。
3. 客户端声明一个queue，并设置相关属性。
4. 客户端使用routing key，在exchange和queue之间建立好绑定关系。
5. 客户端投递消息到exchange。

4.4 消息怎么路由?

从概念上来说，消息路由必须有三部分：**交换器、路由、绑定。**生产者把消息发布到交换器上；绑定决定了消息如何从路由器路由到特定的队列；消息最终到达队列，并被消费者接收。

消息发布到交换器时，消息将拥有一个路由键（routing key），在消息创建时设定，通过队列路由键，可以把队列绑定到交换器上。

消息到达交换器后，RabbitMQ会将消息的路由键与队列的路由键进行匹配（针对不同的交换器有不同的路由规则）。如果能够匹配到队列，则消息会投递到相应队列中；如果不能匹配到任何队列，消息将进入“黑洞”。

4.5 常用交换器有哪些?

> 交换器是消息被发送的 AMQP 实体。交换器拿到消息然后把它路由给0或多个队列。路由算法基于交换器的类型和 bindings。

常用的交换器主要分为一下三种：

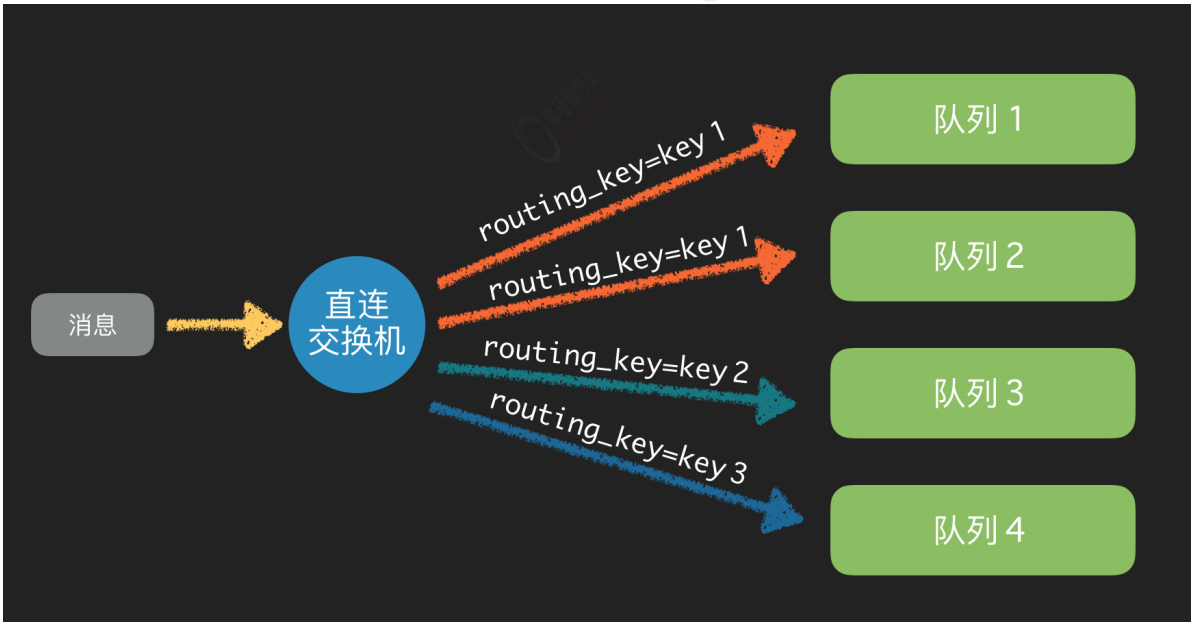
Exchange type(交换器类型)	Default pre-declared names(预声明默认名称)
Direct exchange(直连交换器)	(Empty string) and amq.direct
Fanout exchange(扇形交换器)	amq.fanout
Topic exchange(主题交换器)	amq.topic
Headers exchange(头信息交换器)	amq.match (and amq.headers in RabbitMQ)

4.5.1 直连交换机

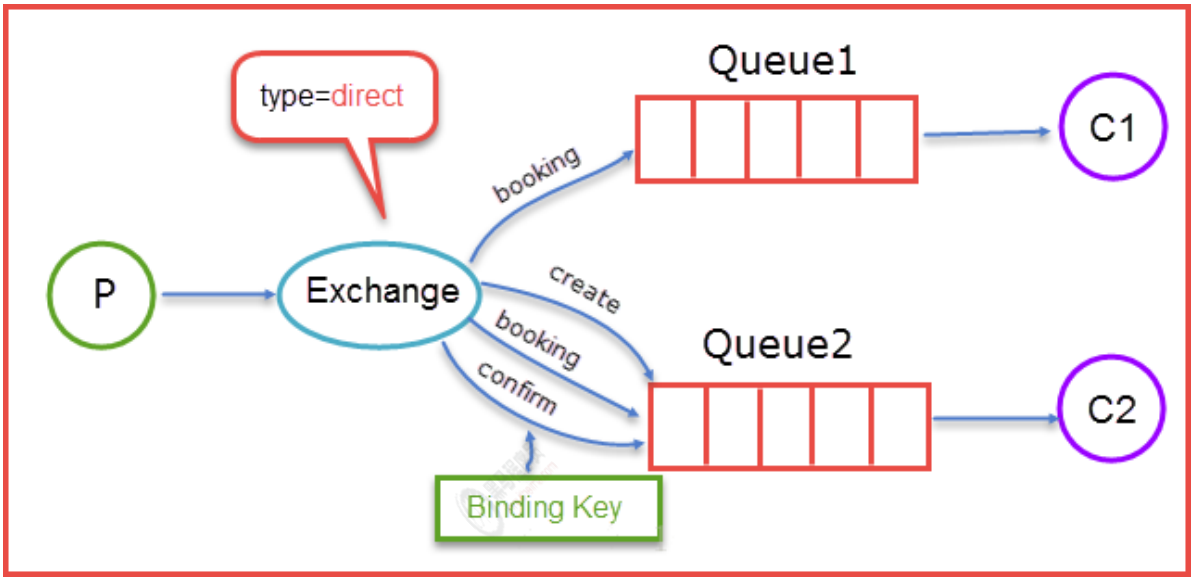
> 如果路由键完全匹配，消息就被投递到相应的队列

直连型交换机（direct exchange）是根据消息携带的路由键（routing key）将消息投递给对应绑定键的队列。

直连交换机是一种带路由功能的交换机，一个队列会和一个交换机绑定，除此之外再绑定一个routing_key，当消息被发送的时候，需要指定一个binding_key，这个消息被送达交换机的时候，就会被这个交换机送到指定的队列里面去。同样的一个binding_key也是支持应用到多个队列中的。



> 直连型交换机图例：



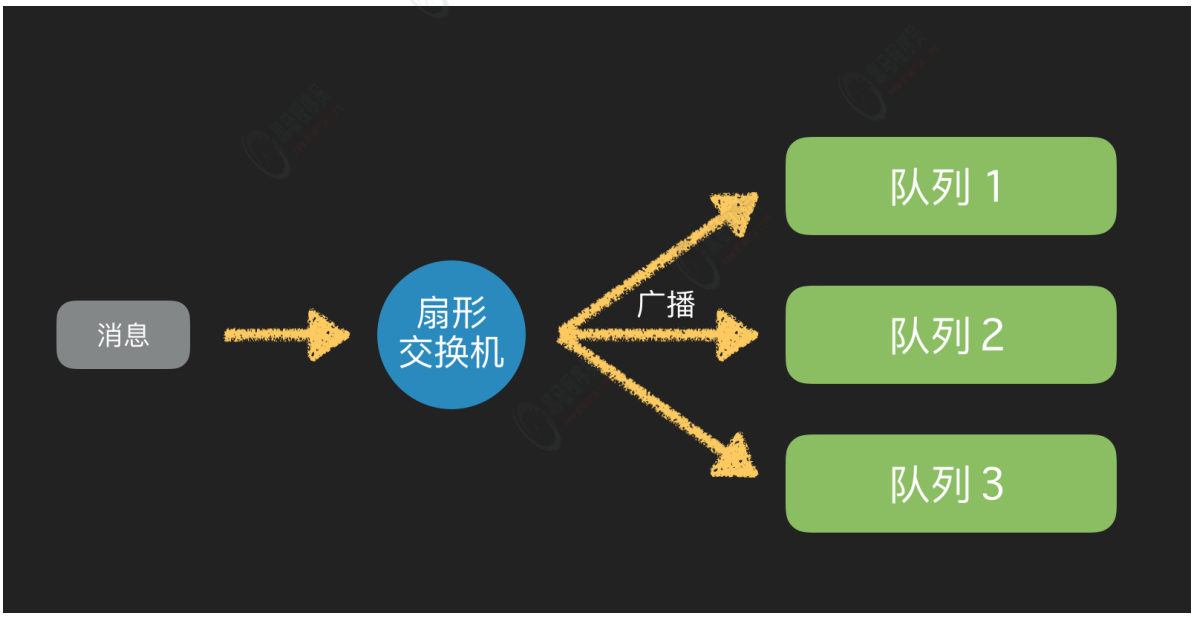
当生产者（P）发送消息时 Rotuing key=booking 时，这时候将消息传送给 Exchange，Exchange 获取到生产者发送过来消息后，会根据自身的规则进行与匹配相应的 Queue，这时发现 Queue1 和 Queue2 都符合，就会将消息传送给这两个队列。

如果我们以 Rotuing key=create 和 Rotuing key=confirm 发送消息时，这时消息只会被推送到 Queue2 队列中，其他 Routing Key 的消息将会被丢弃。

4.5.2 扇型交换机

> 如果交换器收到消息，将会广播到所有绑定的队列上

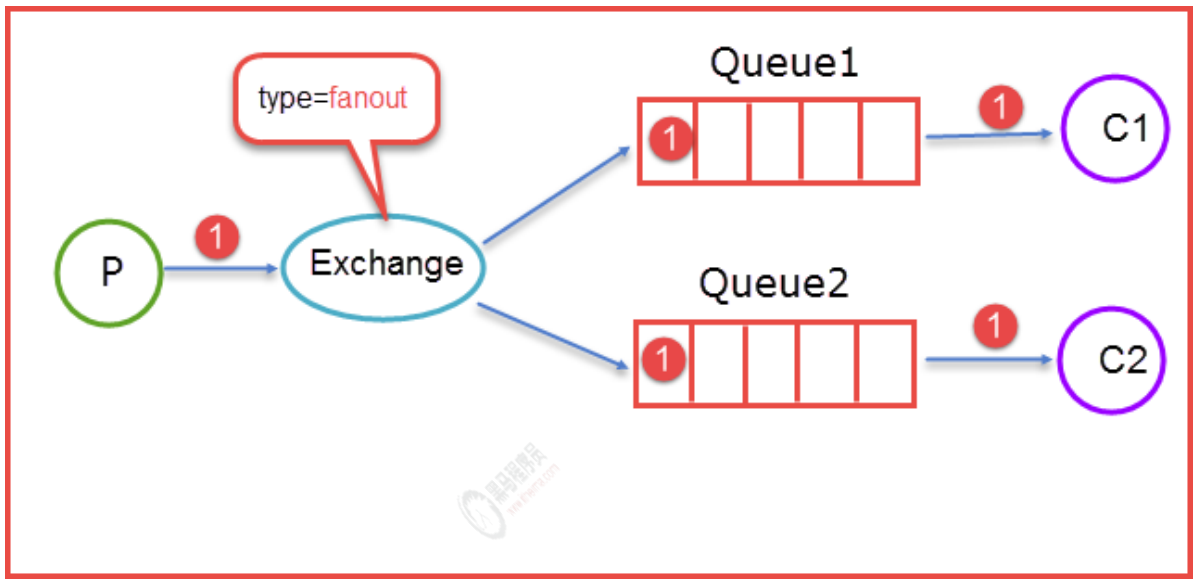
扇型交换机（fanout exchange）将消息路由给绑定到它身上的所有队列，而不理会绑定的路由键。如果 N 个队列绑定到某个扇型交换机上，当有消息发送给此扇型交换机时，交换机会将消息的拷贝分别发送给这所有的 N 个队列。扇型用来交换机处理消息的广播路由（broadcast routing）。



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：



上图所示，生产者（P）生产消息 1 将消息 1 推送到 Exchange，由于 Exchange Type=fanout 这时候会遵循 fanout 的规则将消息推送到所有与它绑定 Queue，也就是图上的两个 Queue 最后两个消费者消费。

4.5.3 主题交换机

> 可以使来自不同源头的消息能够到达同一个队列。使用topic交换器时，可以使用通配符。

基于消息的 routing key 与绑定到该交换器的队列的 pattern 进行匹配，路由消息到一个或多个队列。常用于复杂的发布/订阅场景。
当出现多消费者/应用的场景，消费者选择性地接收消息时，应该考虑使用 topic exchange

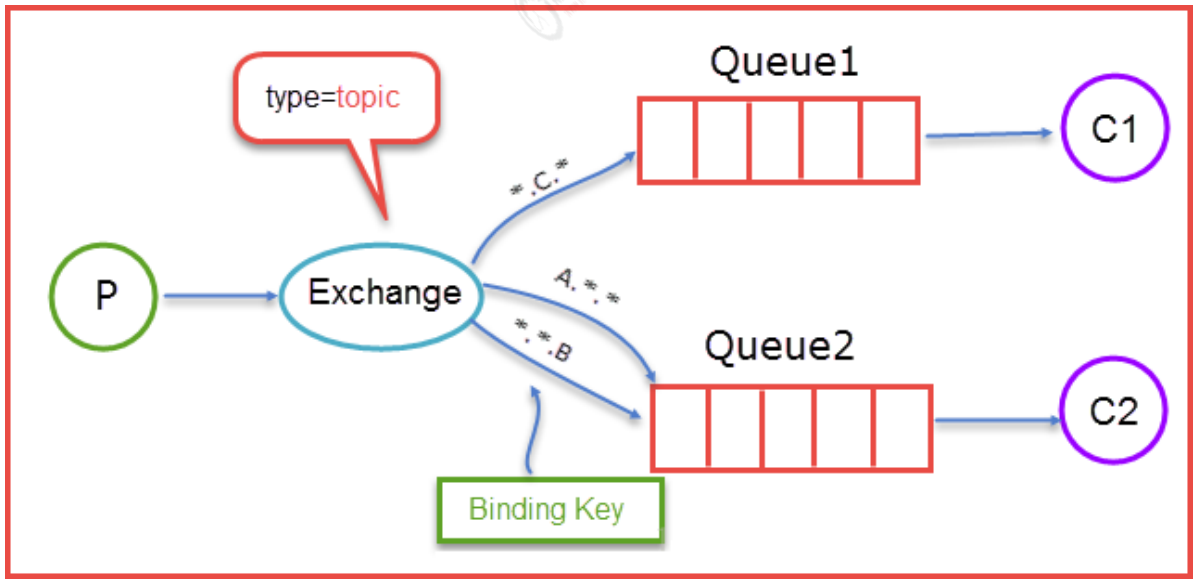
前面提到的 direct 规则是严格意义上的匹配，换言之 Routing Key 必须与 Binding Key 相匹配的时候才将消息传送给 Queue.

而Topic 的路由规则是一种模糊匹配，可以通过通配符满足一部分规则就可以传送。

4.5.3.1 约束条件

1. binding key 中可以存在两种特殊字符 “*” 与“#”，用于做模糊匹配，其中 “*” 用于匹配一个单词，“#”用于匹配多个单词（可以是零个）
2. routing key 为一个句点号 “.” 分隔的字符串（我们将被句点号 “. ” 分隔开的每一段独立的字符串称为一个单词），如“stock.usd.nyse”、“nyse.vmw”、“quick.orange.rabbit”
   binding key 与 routing key 一样也是句点号 “.” 分隔的字符串

> 主题交换机图例：



当生产者发送消息 Routing Key=F.C.E 的时候，这时候只满足 Queue1，所以会被路由到 Queue 中，如果 Routing Key=A.C.E 这时候会被同是路由到 Queue1 和 Queue2 中，如果 Routing Key=A.F.B 时，这里只会发送一条消息到 Queue2 中。

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。

4.5.3.2 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。

4.5.4 头交换机(不常用)

> headers 类型的 Exchange 不依赖于 routing key 与 binding key 的匹配规则来路由消息，而是根据发送的消息内容中的 headers 属性进行匹配。

头交换机可以视为直连交换机的另一种表现形式。但直连交换机的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。灵活性更强（但实际上我们很少用到头交换机）。工作流程：

1. 绑定一个队列到头交换机上时，会同时绑定多个用于匹配的头（header）。
2. 传来的消息会携带header，以及会有一个 “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。

4.5.5 交换机小结

类型名称	路由规则
Default	自动命名的直交换机
Direct	把消息路由到BindingKey和RoutingKey完全匹配的队列中，Routing Key==Binding Key，严格匹配
Fanout	发送到该交换机的消息都会路由到与该交换机绑定的所有队列上，可以用来做广播
Topic	topic和direct类似，也是将消息发送到RoutingKey和BindingKey相匹配的队列中，只不过可以模糊匹配
Headers	根据发送的消息内容中的 headers 属性进行匹配，性能差，基本不会使用