消息中间件的应用场景

提高系统性能首先考虑的是数据库的优化，但是数据库因为历史原因，横向扩展是一件非常复杂的工程，所有我们一般会尽量把流量都挡在数据库之前。

不管是无限的横向扩展服务器，还是纵向阻隔到达数据库的流量，都是这个思路。阻隔直达数据库的流量，缓存组件和消息组件是两大杀器。这里就重点说说MQ的应用场景。

1. MQ简介

MQ：Message queue，消息队列，就是指保存消息的一个容器。具体的定义这里就不类似于数据库、缓存等，用来保存数据的。当然，与数据库、缓存等产品比较，也有自己一些特点，具体的特点后文会做详细的介绍。

现在常用的MQ组件有activeMQ、rabbitMQ、rocketMQ、zeroMQ，当然近年来火热的kafka，从某些场景来说，也是MQ，当然kafka的功能更加强大，虽然不同的MQ都有自己的特点和优势，但是，不管是哪种MQ，都有MQ本身自带的一些特点，下面，咱们就先聊聊MQ的特点。

2. MQ特点

l 先进先出
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。

l 发布订阅
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。

l 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。

l 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。

3. 应用场景

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

3.1. 消息中间件监控

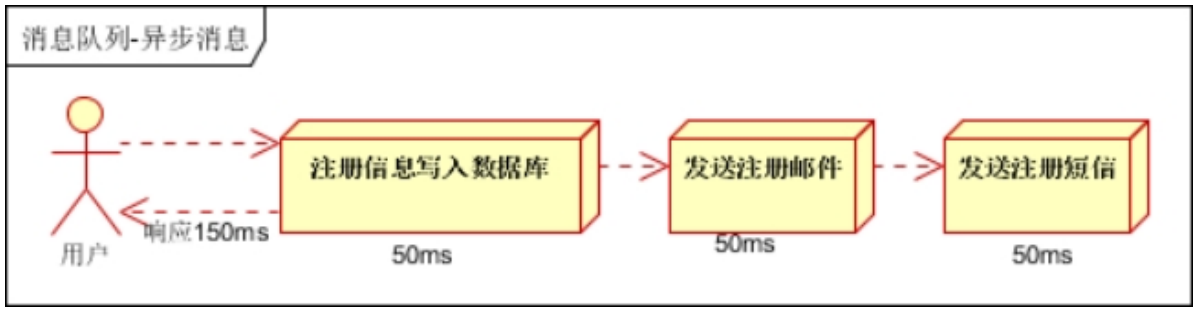
Activemq 监控

Rabbitmq 监控

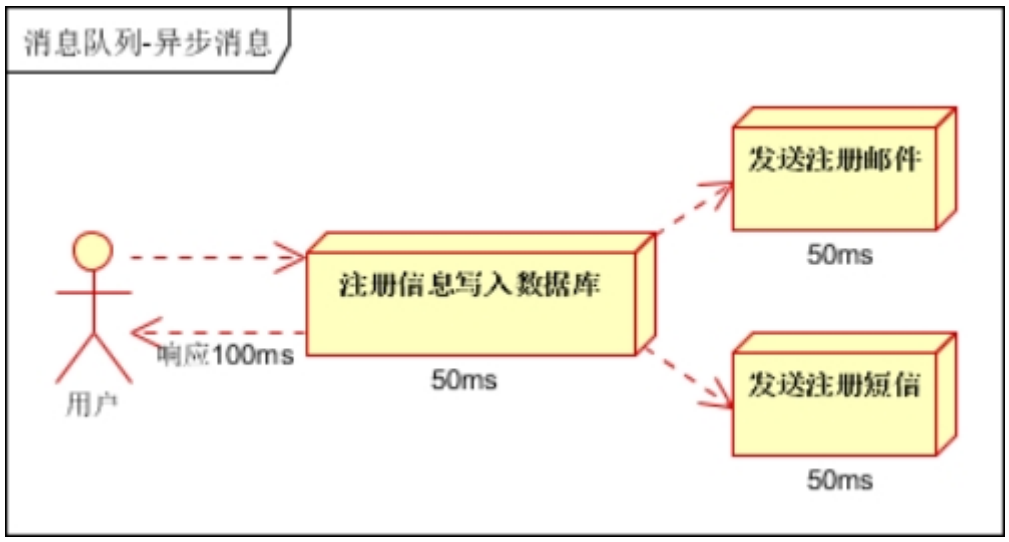
Kafka 监控

3.2. 异步处理

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.并行方式
a、串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端。

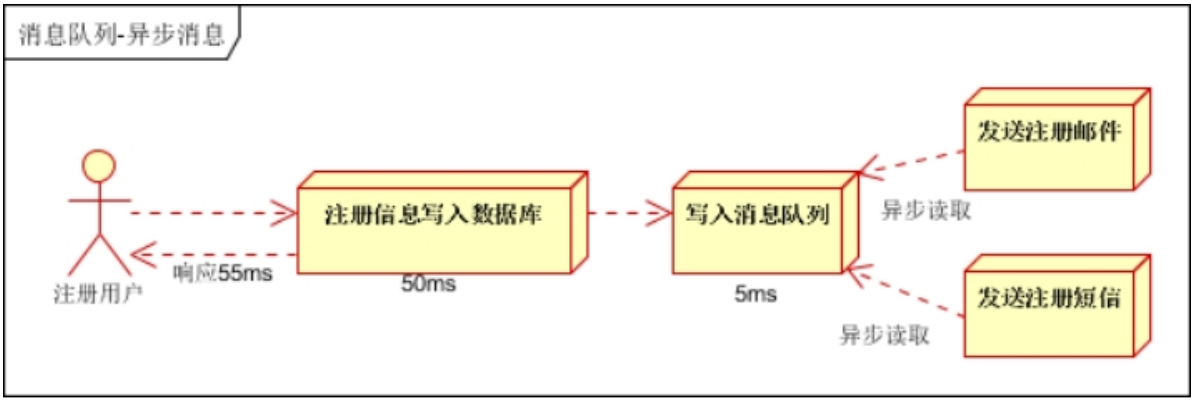


b、并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间



假设三个业务节点每个使用50毫秒钟，不考虑网络等其他开销，则串行方式的时间是150毫秒，并行的时间可能是100毫秒。
因为CPU在单位时间内处理的请求数是一定的，假设CPU1秒内吞吐量是100次。则串行方式1秒内CPU可处理的请求量是7次（1000/150）。并行方式处理的请求量是10次（1000/100）
小结：如以上案例描述，传统的方式系统的性能（并发量，吞吐量，响应时间）会有瓶颈。如何解决这个问题呢?

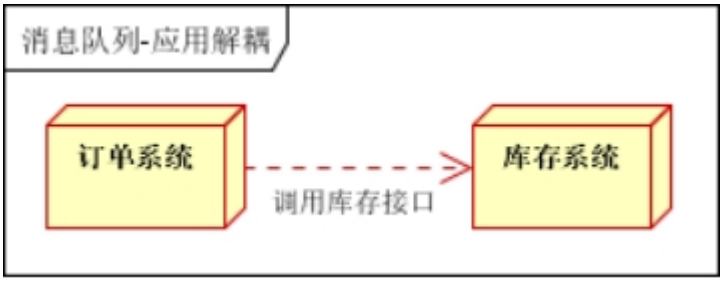
引入消息队列，将不是必须的业务逻辑，异步处理。改造后的架构如下：



按照以上约定，用户的响应时间相当于是注册信息写入数据库的时间，也就是50毫秒。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是50毫秒。因此架构改变后，系统的吞吐量提高到每秒20 QPS。比串行提高了3倍，比并行提高了两倍。

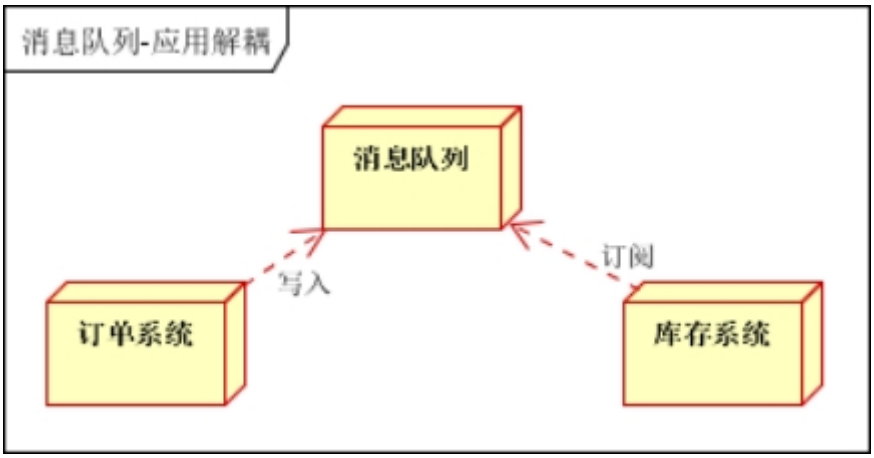
3.3. 应用解耦

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。如下图：



传统模式的缺点：假如库存系统无法访问，则订单减库存将失败，从而导致订单失败，订单系统与库存系统耦合

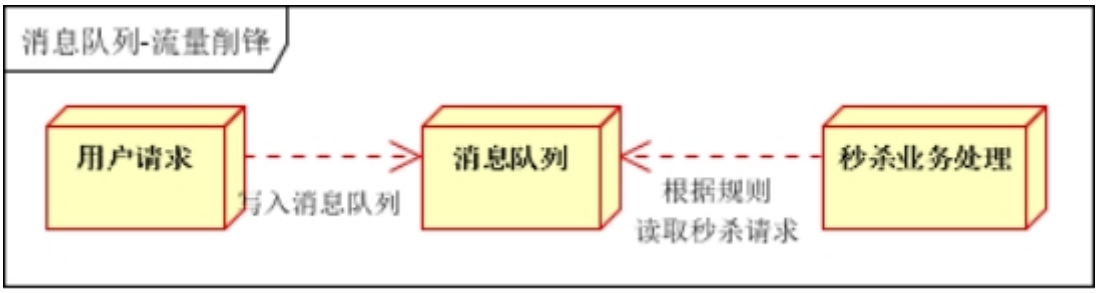
如何解决以上问题呢？引入应用消息队列后的方案，如下图：



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，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作
假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦

3.4. 流量削峰

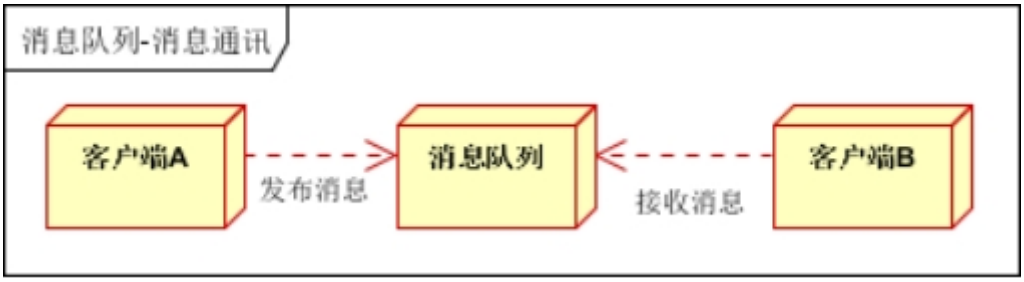
流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛。
应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。
a、可以控制活动的人数
b、可以缓解短时间内高流量压垮应用



用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面。
秒杀业务根据消息队列中的请求信息，再做后续处理。

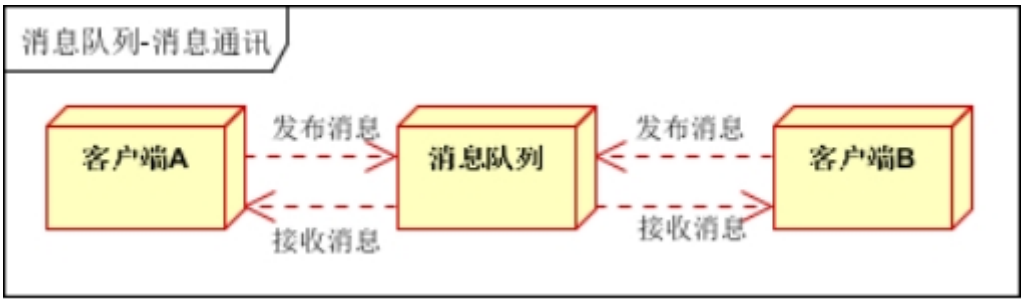
3.5. 消息通讯

消息通讯是指，消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等。
点对点通讯：



客户端A和客户端B使用同一队列，进行消息通讯。

聊天室通讯：



客户端A，客户端B，客户端N订阅同一主题，进行消息发布和接收。实现类似聊天室效果。

以上实际是消息队列的两种消息模式，点对点或发布订阅模式。模型为示意图，供参考。

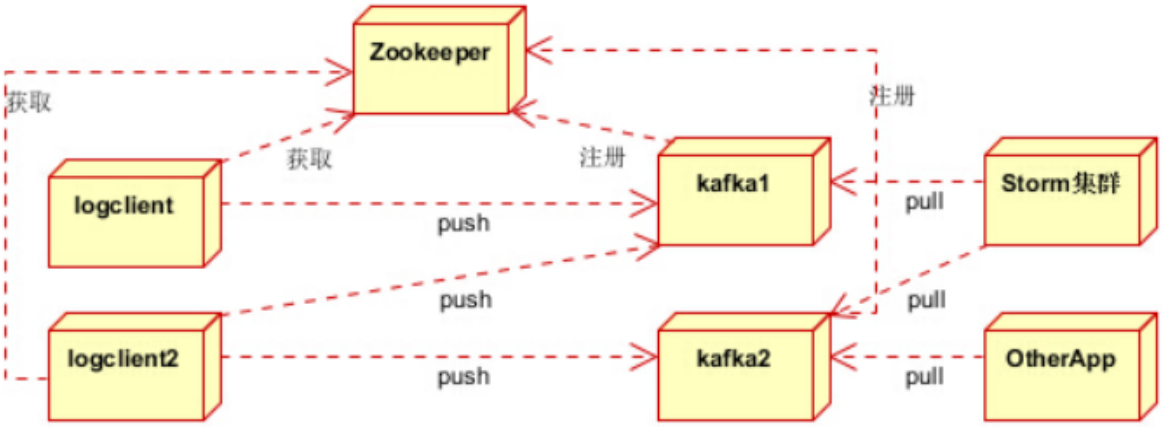
具体例子可以参考官网：https://www.rabbitmq.com/web-stomp.html

3.6. 海量数据同步（日志）

日志处理是指将消息队列用在日志处理中，比如Kafka的应用，解决大量日志传输的问题。架构简化如下

日志采集客户端，负责日志数据采集，定时写受写入Kafka队列
Kafka消息队列，负责日志数据的接收，存储和转发
日志处理应用：订阅并消费kafka队列中的日志数据

eg：日志收集系统



分为Zookeeper注册中心，日志收集客户端，Kafka集群和Storm集群（OtherApp）四部分组成。
Zookeeper注册中心，提出负载均衡和地址查找服务
日志收集客户端，用于采集应用系统的日志，并将数据推送到kafka队列
Kafka集群：接收，路由，存储，转发等消息处理
Storm集群：与OtherApp处于同一级别，采用拉的方式消费队列中的数据

n 网易使用案例：

网易NDC-DTS系统在使用，应该是最典型的应用场景，主要就是binlog的同步，数据表的主从复制。简单一点就是：MySQL进程写binlog文件 -> 同步应用去实时监控binlog文件读取发送到Kafka -> 目标端处理binlog 。原理上与阿里开源的canal, 点评的puma大同小异。

3.7. 任务调度

参考延时队列

> 结合代码进行讲解

3.8. 分布式事物

> 结合代码进行讲解

分布式事务有强一致，弱一致，和最终一致性这三种：

n 强一致：

当更新操作完成之后，任何多个后续进程或者线程的访问都会返回最新的更新过的值。这种是对用户最友好的，就是用户上一次写什么，下一次就保证能读到什么。根据 CAP 理论，这种实现需要牺牲可用性。

n 弱一致：

系统并不保证续进程或者线程的访问都会返回最新的更新过的值。系统在数据写入成功之后，不承诺立即可以读到最新写入的值，也不会具体的承诺多久之后可以读到。

n 最终一致：

弱一致性的特定形式。系统保证在没有后续更新的前提下，系统最终返回上一次更新操作的值。在没有故障发生的前提下，不一致窗口的时间主要受通信延迟，系统负载和复制副本的个数影响。DNS 是一个典型的最终一致性系统。

在分布式系统中，同时满足“CAP定律”中的“一致性”、“可用性”和“分区容错性”三者是几乎不可能的。在互联网领域的绝大多数的场景，都需要牺牲强一致性来换取系统的高可用性，系统往往只需要保证“最终一致性”，只要这个最终时间是在用户可以接受的范围内即可，这时候我们只需要用短暂的数据不一致就可以达到我们想要效果。

Ø 实例描述

比如有订单，库存两个数据，一个下单过程简化为，加一个订单，减一个库存。 而订单和库存是独立的服务，那怎么保证数据一致性。

这时候我们需要思考一下，怎么保证两个远程调用“同时成功”，数据一致？

请大家先注意一点远程调用最郁闷的地方就是，结果有3种，成功、失败和超时。 超时的话，成功失败都有可能。

一般的解决方案，大多数的做法是借助mq来做最终一致。

Ø 如何实现最终一致

我们是怎么利用Mq来达到最终一致的呢？

首先，拿我们上面提到的订单业务举例：

在我们进行加订单的过程中同时插入logA（这个过程是可以做本地事务的）

然后可以异步读取logA，发mqA

B端接收mqA，同时减少库存，B这里需要做幂等(避免因为重复消息造成的业务错乱)

***那么我们通过上面的分析可能联想到这样的问题**？*

本地先执行事务，执行成功了就发个消息过去，消费端拿到消息执行自己的事务
比如a，b，c，a异步调用b，c如果b失败了，或者b成功，或者b超时，那么怎么用mq让他们最终一致呢？b失败就失败了，b成功之后给c发一个消息，b和c对a来讲都是异步的，且他们都是同时进行的话，而且需要a，b，c同时成功的情况，那么这种情况用mq怎么做？

其实做法还是参照于本地事务的概念的。

l 第一种情况:假设a,b,c三者都正常执行，那整个业务正常结束

l 第二种情况:假设b超时,那么需要a给b重发消息（记得b服务要做幂等），如果出现重发失败的话，需要看情况，是终端服务，还是继续重发，甚至人为干预（所有的规则制定都需要根据业务规则来定）

l 第三种情况:假设a,b,c三者之中的一个失败了,失败的服务利用MQ给其他的服务发送消息,其他的服务接收消息，查询本地事务记录日志，如果本地也失败，删除收到的消息(表示消息消费成功)，如果本地成功的话，则需要调用补偿接口进行补偿(需要每个服务都提供业务补偿接口)。

注意事项：

mq这里有个坑，通常只适用于只允许第一个操作失败的场景，也就是第一个成功之后必须保证后面的操作在业务上没障碍，不然后面失败了前面不好回滚，只允许系统异常的失败，不允许业务上的失败，通常业务上失败一次后面基本上也不太可能成功了，要是因为网络或宕机引起的失败可以通过重试解决，如果业务异常，那就只能发消息给a和c让他们做补偿了吧？通常是通过第三方进行补偿，ABC提供补偿接口，设计范式里通常不允许消费下游业务失败。

怎么理解呢，举个例子：
比如A给B转账，A先自己扣钱，然后发了个消息，B这边如果在这之前销户了，那重试多少次也没用，只能人工干预。

Ø 网易在分布式事务采用的解决方式

网易部分业务是用MQ实现了最终一致性，目前教育产品，例如：网易云课堂。

也有一部分业务用了TCC事务，但是TCC事务用的比较少，因为会侵染业务，开发成本比较高，如果体量不大的话直接用JPA或MQ支持事务就好。

网易的产品中使用分布式事务基于技术

TCC，FMT（Framework-managed transactions），事务消息都有。

开源产品myth：https://gitee.com/shuaiqiyu/myth

4. 常用消息队列（ActiveMQ、RabbitMQ、RocketMQ、Kafka）比较

特性MQ	ActiveMQ	RabbitMQ	RocketMQ	Kafka
生产者消费者模式	支持	支持	支持	支持
发布订阅模式	支持	支持	支持	支持
请求回应模式	支持	支持	不支持	不支持
Api完备性	高	高	高	高
多语言支持	支持	支持	java	支持
单机吞吐量	万级	万级	万级	十万级
消息延迟	无	微秒级	毫秒级	毫秒级
可用性	高（主从）	高（主从）	非常高（分布式）	非常高（分布式）
消息丢失	教低	低	理论上不会丢失	理论上不会丢失
文档的完备性	高	高	教高	高
提供快速入门	有	有	有	有
社区活跃度	高	高	中	高
商业支持	无	无	商业云	商业云

总体来说：

l ActiveMQ 历史悠久的开源项目，已经在很多产品中得到应用，实现了JMS1.1规范，可以和spring-jms轻松融合，实现了多种协议，不够轻巧（源代码比RocketMQ多），支持持久化到数据库，对队列数较多的情况支持不好。

l RabbitMQ 它比Kafka成熟，支持AMQP事务处理，在可靠性上，RabbitMQ超过Kafka，在性能方面超过ActiveMQ。

l RocketMQ RocketMQ是阿里开源的消息中间件，目前在Apache孵化，使用纯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。其可靠性毋庸置疑，而且与Kafka一脉相承（甚至更优），性能强劲，支持海量堆积。

· Kafka Kafka设计的初衷就是处理日志的，不支持AMQP事务处理，可以看做是一个日志系统，针对性很强，所以它并没有具备一个成熟MQ应该具备的特性。Kafka的性能（吞吐量、tps）比RabbitMQ要强，如果用来做大数据量的快速处理是比RabbitMQ有优势的。