RocketMQ 基本理论及架构

课程主题：

1、为什么要学习 mq?

2、mq 产品的选型?

3、使用 mq 消息中间件的优缺点？

4、RocketMQ 消息的发送流程？

5、RocketMQ 消息存储结构？

6、消息是如何发送的？

7、消息是如何接受的？

8、RocketMQ 网络架构及其分布式组件的作用？

面试问题：

1：什么场景使用了 mq？直接调接口不行吗？

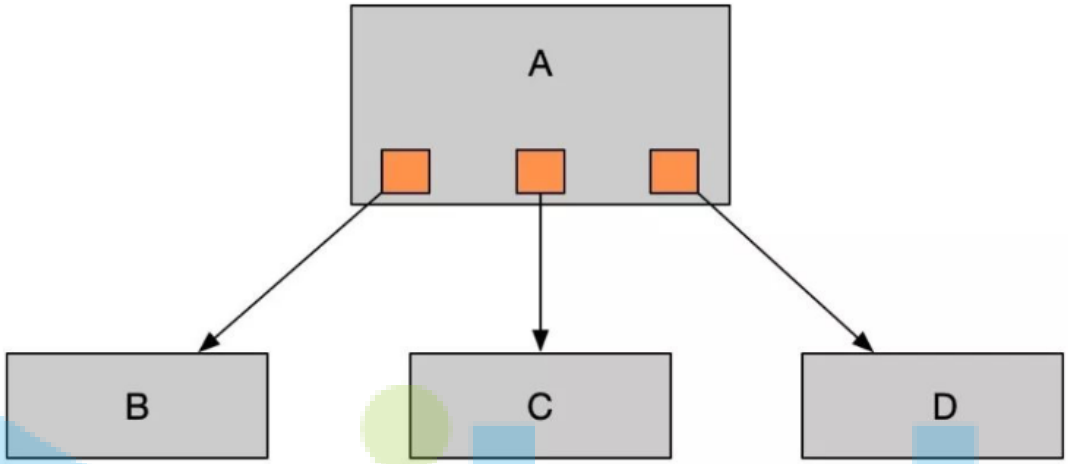
2：Kafka、ActiveMQ、RabbitMQ、RocketMQ 都有什么区别？

3：用消息队列都有什么优点和缺点？

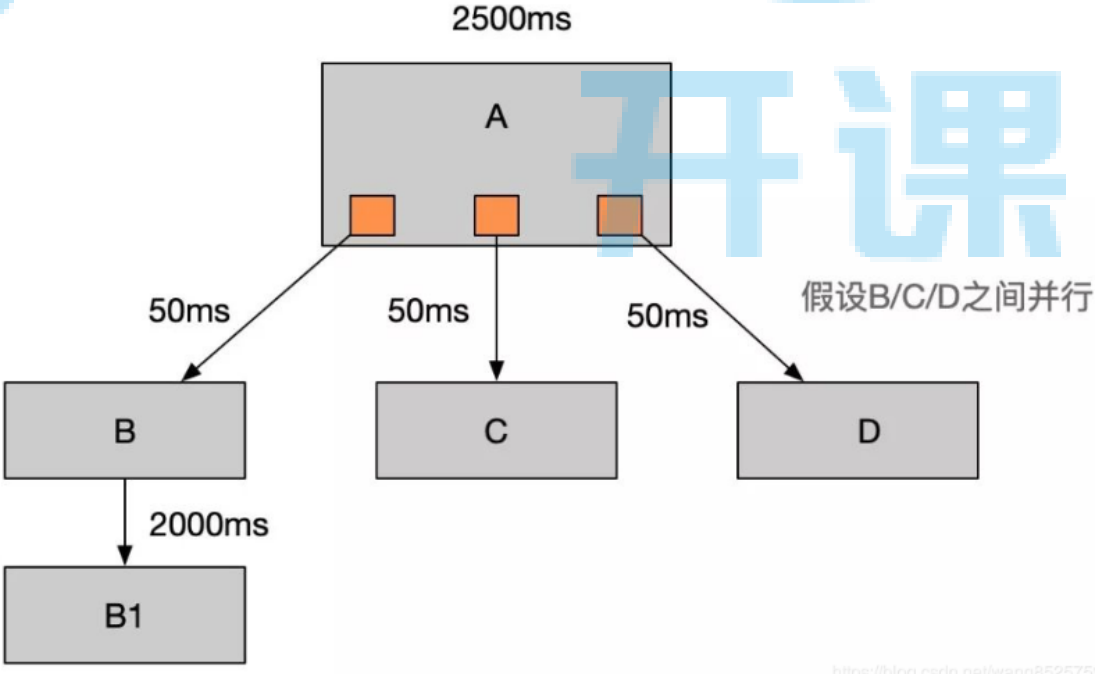
4：RocketMQ 消息是如何存储的？有什么优势？

一、前置知识

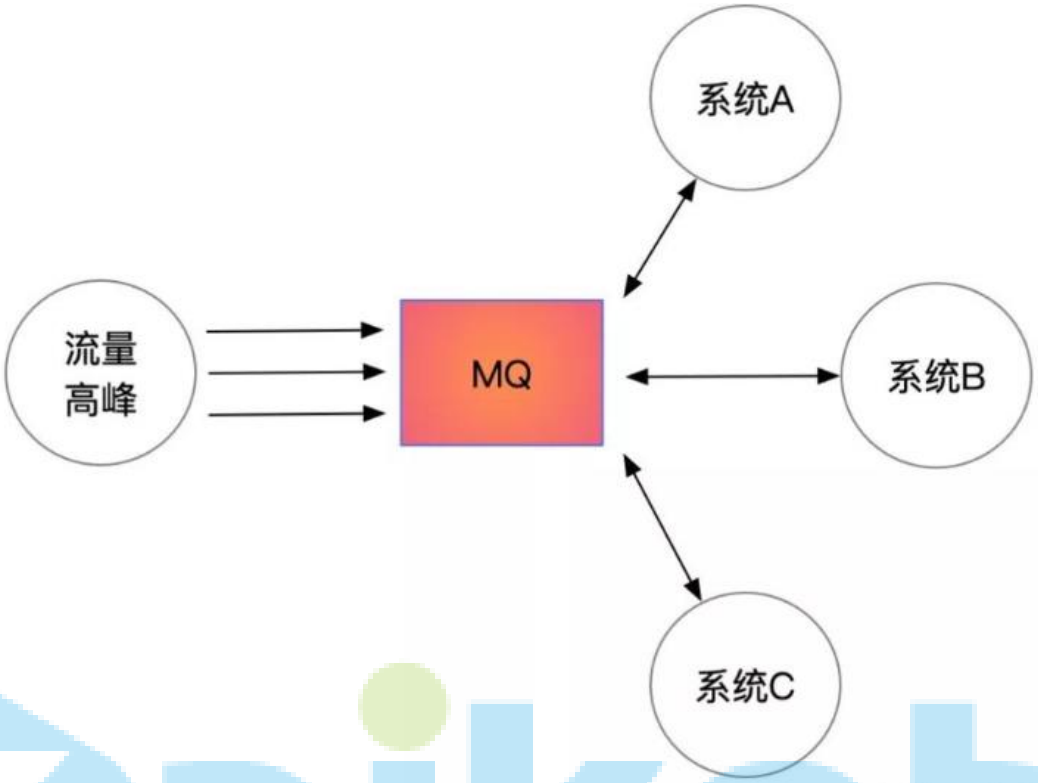
1、链式调用



2、木桶理论



3、MQ 引入



二、MQ 前置知识

1、MQ 作用

消息队列作为高并发系统的核心组件之一，能够帮助业务系统解构提升开发效率和系统稳定性。主要具有以下优势：

- 削峰填谷（大促等流量洪流突然来袭时，MQ 可以缓冲突发流量，避免下游订阅系统因突发流量崩溃）
- 系统解耦（解决不同重要程度、不同能力级别系统之间依赖导致一死全死）
- 提升性能（当存在一对多调用时，可以发一条消息给消息系统，让消息系统通知相关系统）
- 蓄流压测（线上有些链路不好压测，可以通过堆积一定量消息再放开来压测）

1.1、应用场景

MQ 可应用在多个领域，包括异步通信解耦、企业解决方案、金融支付、电信、电子商务、快递物流、广告营销、社交、即时通信、手游、视频、物联网、车联网等。从应用功能上来讲。

例如：

- 日志监控，作为重要日志的监控通信管道，将应用日志监控对系统性能影响降到最低。
- 消息推送，为社交应用和物联网应用提供点对点推送，一对多广播式推送的能力。
- 金融报文，发送金融报文，实现金融准实时的报文传输，可靠安全。
- 电信信令，将电信信令封装成消息，传递到各个控制终端，实现准实时控制和信息传递。

从功能角度考虑：

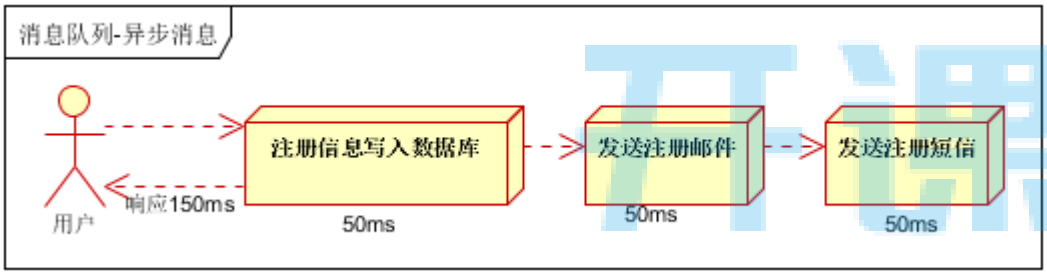
RocketMQ 在实际应用中常用的使用场景、主要有异步处理，应用解耦，流量削锋和消息通讯四个场景

1.1.1、异步处理

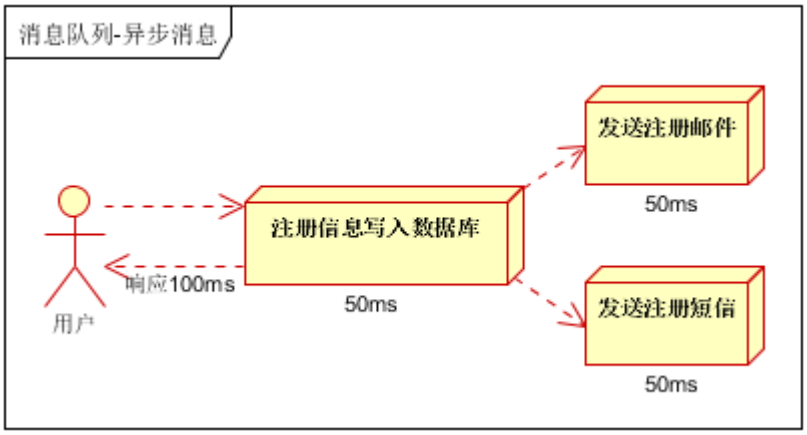
场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种：

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。

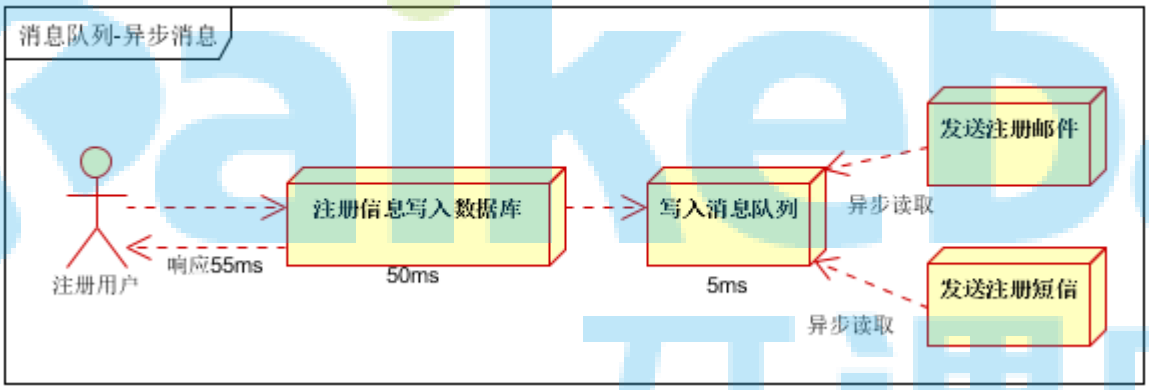


b、并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间



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？

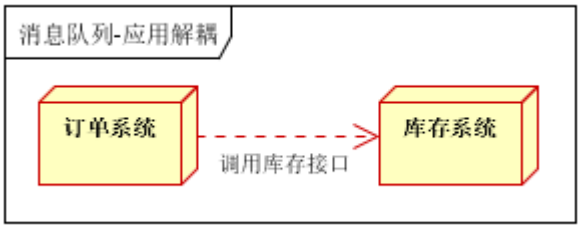
引入消息队列，将是必须的业务逻辑，异步处理。改造后的架构如下：



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。

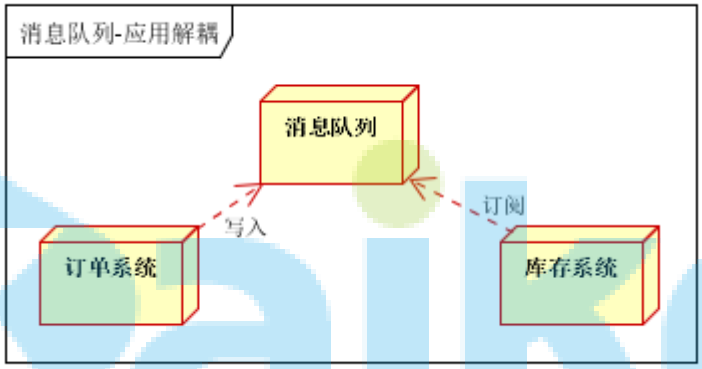
1.1.2、应用解耦

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：



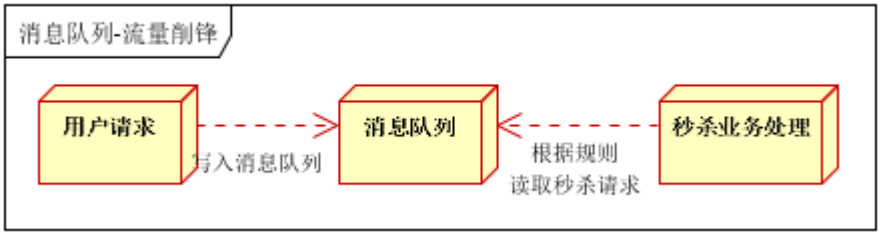
订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功

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

1.1.3、流量削锋

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。

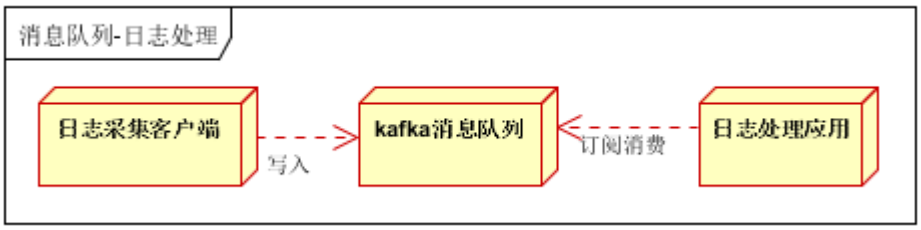
a、可以控制活动的人数 b、可以缓解短时间内高流量压垮应用



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

1.1.4、日志处理

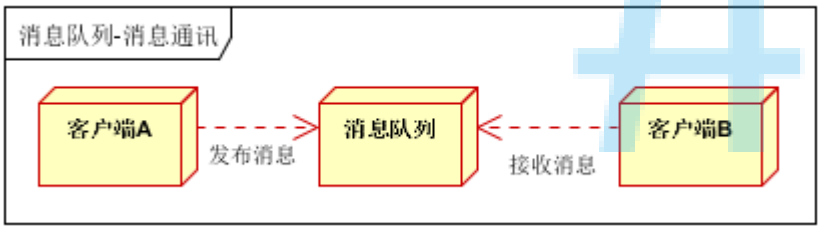
日志处理是指将消息队列用在日志处理中，比如 Kafka 的应用，解决大量日志传输的问题。架构简化如下



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

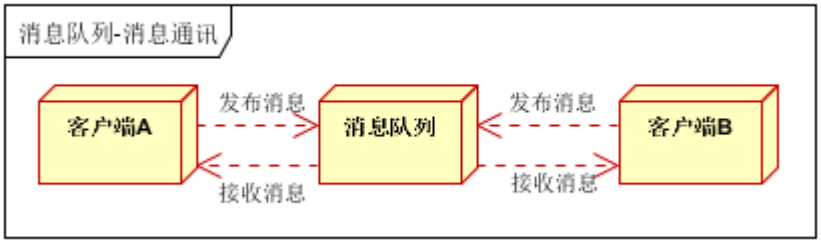
1.1.5、消息通讯

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：

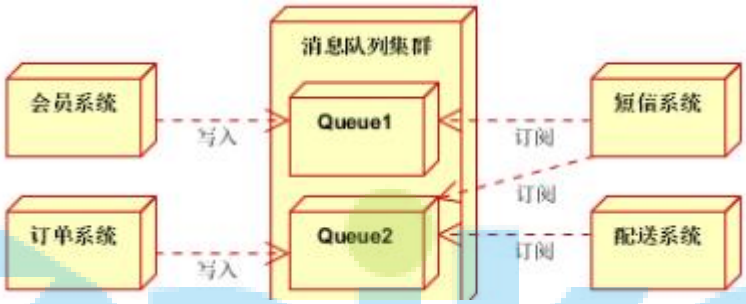


客户端 A，客户端 B，客户端 N 订阅同一主题，进行消息发布和接收。实现类似聊天室效果。

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。

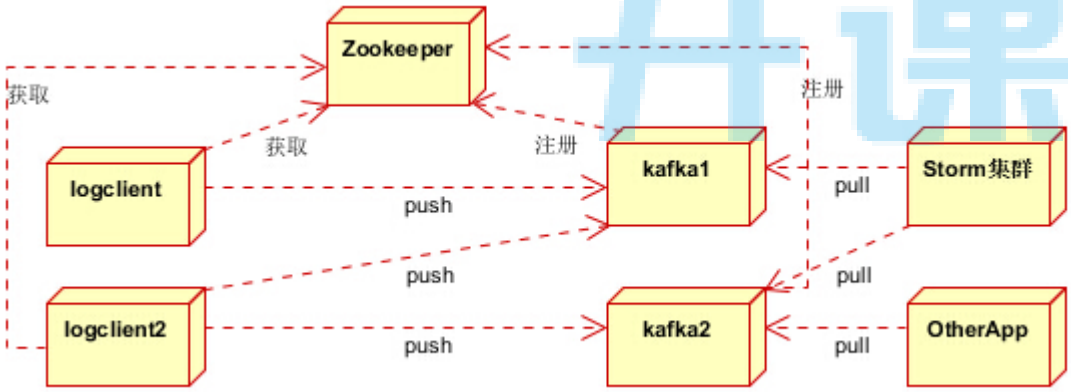
1.2、示例

1.2.1、电商系统



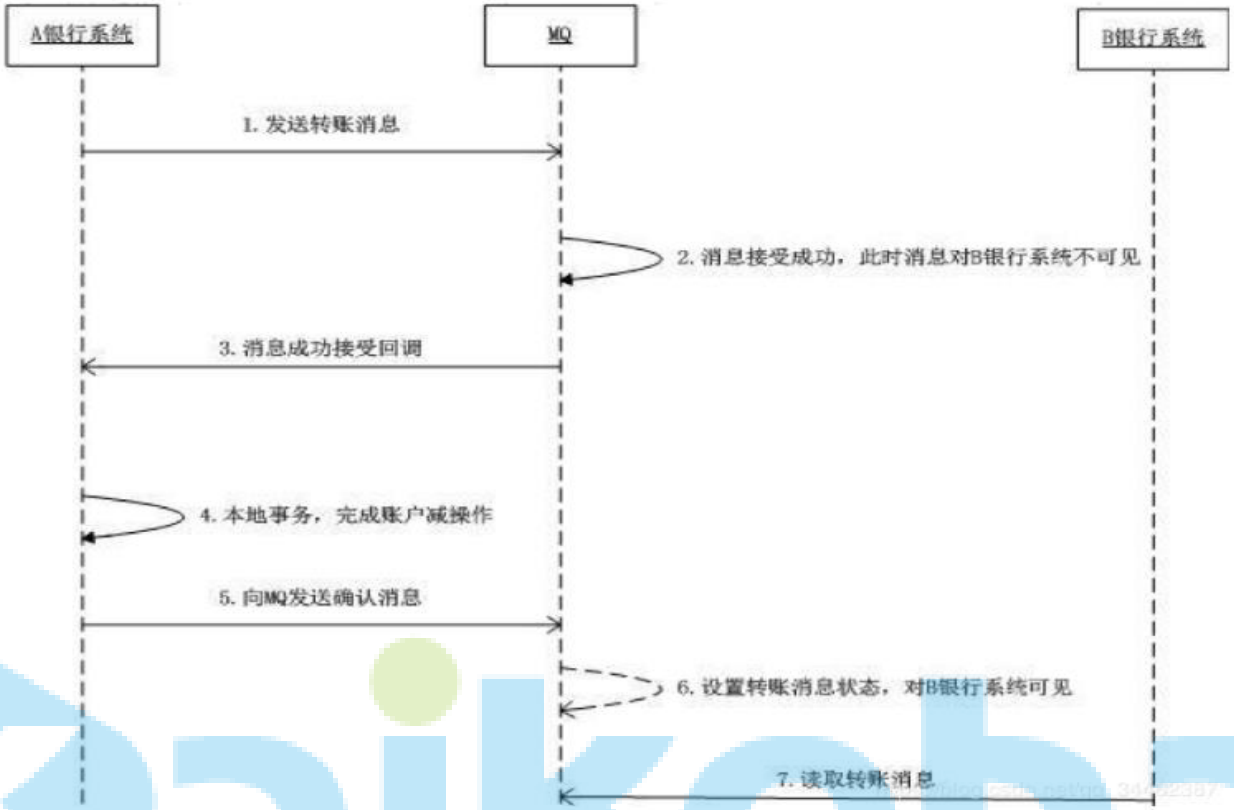
消息队列采用高可用，可持久化的消息中间件。比如 Active MQ，Rabbit MQ，Rocket Mq。

1.2.2、日志收集系统



1.2.3、事务处理

比如银行转账。



2、MQ 对比

目前主流的 MQ 主要是

- ZeroMQ
- 推特的 Distributedlog
- ActiveMQ：Apache 旗下的老牌消息引擎
- RabbitMQ、Kafka：AMQP 的默认实现。
- RocketMQ
- Artemis：Apache 的 ActiveMQ 下的子项目
- Apollo：同样为 Apache 的 ActiveMQ 的子项目的号称下一代消息引擎
- 商业化的消息引擎 IronMQ
- 实现了 JMS(Java Message Service)标准的 OpenMQ

那么其他消息中间件都有什么特性呢？

2.1、ActiveMQ

单机吞吐量：万级

时效性：ms 级

可用性：高，基于主从架构实现高可用性

消息可靠性：有较低的概率丢失数据

功能支持：MQ 领域的功能极其完备

总结：

1）非常成熟，功能强大，在早些年业内大量的公司以及项目中都有应用

2）偶尔会有较低概率丢失消息

3）现在社区以及国内应用都越来越少，官方社区现在对 ActiveMQ 5.x 维护越来越少，几个月才发布一个版本

4）主要是基于解耦和异步来用的，较少在大规模吞吐的场景中使用

2.2、RabbitMQ

单机吞吐量：万级

topic 数量都吞吐量的影响：

时效性：微秒级，延时低是一大特点。

可用性：高，基于主从架构实现高可用性

消息可靠性：

功能支持：基于 erlang 开发，所以并发能力很强，性能极其好，延时很低

总结：

1）erlang 语言开发，性能极其好，延时很低；

2）吐量到万级，MQ 功能比较完备

3）开源提供的管理界面非常棒，用起来很好用

4）社区相对比较活跃，几乎每个月都发布几个版本分

5）在国内一些互联网公司近几年用 rabbitmq 也比较多一些 但是问题也是显而易见的，RabbitMQ 确实吞吐量会低一些，这是因为他做的实现机制比较重。

6）erlang 开发，很难去看懂源码，基本职能依赖于开源社区的快速维护和修复 bug。

7）rabbitmq 集群动态扩展会很麻烦，不过这个我觉得还好。其实主要是 erlang 语言本身带来的问题。很难读源码，很难定制和掌控。

2.3、RocketMQ

单机吞吐量：十万级

topic 数量都吞吐量的影响：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。可支持大量 topic 是一大优势。

时效性：ms 级

可用性：非常高，分布式架构

消息可靠性：经过参数优化配置，消息可以做到 0 丢失

功能支持：MQ 功能较为完善，还是分布式的，扩展性好

总结：

1）接口简单易用，可以做到大规模吞吐，性能也非常好，分布式扩展也很方便，社区维护还可以，可靠性和可用性都是 ok 的，还可以支撑大规模的 topic 数量，支持复杂 MQ 业务场景

2）而且一个很大的优势在于，源码是 java，我们可以自己阅读源码，定制自己公司的 MQ，可以掌控

3）社区活跃度相对较为一般，不过也还可以，文档相对来说简单一些，然后接口这块不是按照标准 JMS 规范走的有些系统要迁移需要修改大量代码

相比于其他的 mq 消息中间件具有主要优势特性有：

- 支持事务型消息（消息发送和 DB 操作保持两方的最终一致性，rabbitmq 和 kafka 不支持）
- 支持结合 rocketmq 的多个系统之间数据最终一致性（多方事务，二方事务是前提）
- 支持 18 个级别的延迟消息（rabbitmq 和 kafka 不支持）
- 支持指定次数和时间间隔的失败消息重发（kafka 不支持，rabbitmq 需要手动确认）
- 支持 consumer 端 tag 过滤，减少不必要的网络传输（rabbitmq 和 kafka 不支持）
- 支持重复消费（rabbitmq 不支持，kafka 支持）

2.4、Kafka

单机吞吐量：十万级，最大的优点，就是吞吐量高。

topic 数量都吞吐量的影响：topic 从几十个到几百个的时候，吞吐量会大幅度下降。所以在同等机器下，kafka 尽量保证 topic 数量不要过多。如果要支撑大规模 topic，需要增加更多的机器资源

时效性：ms 级

可用性：非常高，kafka 是分布式的，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用

消息可靠性：经过参数优化配置，消息可以做到 0 丢失

功能支持：功能较为简单，主要支持简单的 MQ 功能，在大数据领域的实时计算以及日志采集被大规模使用

总结：

1）kafka 的特点其实很明显，就是仅仅提供较少的核心功能，但是提供超高的吞吐量，ms 级的延迟，极高的可用性以及可靠性，而且分布式可以任意扩展

2）同时 kafka 最好是支撑较少的 topic 数量即可，保证其超高吞吐量

3）kafka 唯一的一点劣势是有可能消息重复消费，那么对数据准确性会造成极其轻微的影响，在大数据领域中以及日志采集中，这点轻微影响可以忽略

2.5、对比图

特性	ActiveMQ	RabbitMQ	RocketMQ	ka
开发语言	java	erlang	java	scala
单机吞吐量	万级	万级	10 万级	10 万级
时效性	ms 级	us 级	ms 级	ms 级以内
可用性	高(主从架构)	高(主从架构)	非常高(分布式架构)	非常高(分布式
功能特性	成熟的产品，在很多公司得到应用；有较多的文档；各种协议支持较好	基于 erlang 开发，所以并发能力很强，性能极其好，延时很低;管理界面较丰富	MQ 功能比较完备，扩展性佳	只支持主要的些消息查询，没有提供，毕备的，在大数

3、缺点?

一个使用了 MQ 的项目，如果连这个问题都没有考虑过，就把 MQ 引进去了，那就给自己的项目带来了风险。我们引入一个技术，要对这个技术的弊端有充分的认识，才能做好预防。要记住，不要给公司挖坑！ 回答:回答也很容易，从以下两个个角度来答

- 系统可用性降低:你想啊，本来其他系统只要运行好好的，那你的系统就是正常的。现在你非要加个消息队列进去，那消息队列挂了，你的系统不是呵呵了。因此，系统可用性降低

- 系统复杂性增加:要多考虑很多方面的问题，比如一致性问题、如何保证消息不被重复消费，如何保证保证消息可靠传输。因此，需要考虑的东西更多，系统复杂性增大。

既然有这些缺点，那么是不是不敢使用 MQ 了呢？答案很明显，不是，为了提高项目的性能，构建松耦合、异步的结构，必须要使用 MQ.

4、产品选型

我们在进行中间件选型时，一般都是通过下面几点来进行产品选型的：

1）.性能

2）.功能支持程度

3）.开发语言(团队中是否有成员熟悉此中间件的开发语言，市场上此种语言的开发人员是否好招)

4）.有多少公司已经在生产环境上实际使用过，使用的效果如何

5）.社区的支持力度如何

6）.中间件的学习程度是否简单、文档是否详尽

7）.稳定性

8）.集群功能是否完备

如果从以上 8 点来选型一个消息队列，作为一名熟悉 java 的程序员，当遇到重新选择消息队列的场景时，我会毫不犹豫的选型 rocketmq，rocketmq 除了在第 5 点上表现略差(文档少，学习成本高)以及监控管理功能不友好外，从其它方面来说，它真的是一款非常优秀的消息队列中间件。

三、RocketMQ 基本理论

1、发展历史

阿里巴巴消息中间件起源 于 2001 年的五彩石项目， Notify 在这期间应运而生，用于交易核心消息的流转 。 2010 年， B2B 开始大规模使用 ActiveMQ 作为消息内核，随着阿里业务 的快速发展，急需一款支持顺序消 息，拥有海量消息堆积能力的消息中间件， MetaQ 1.0 在 2011 年诞生 。

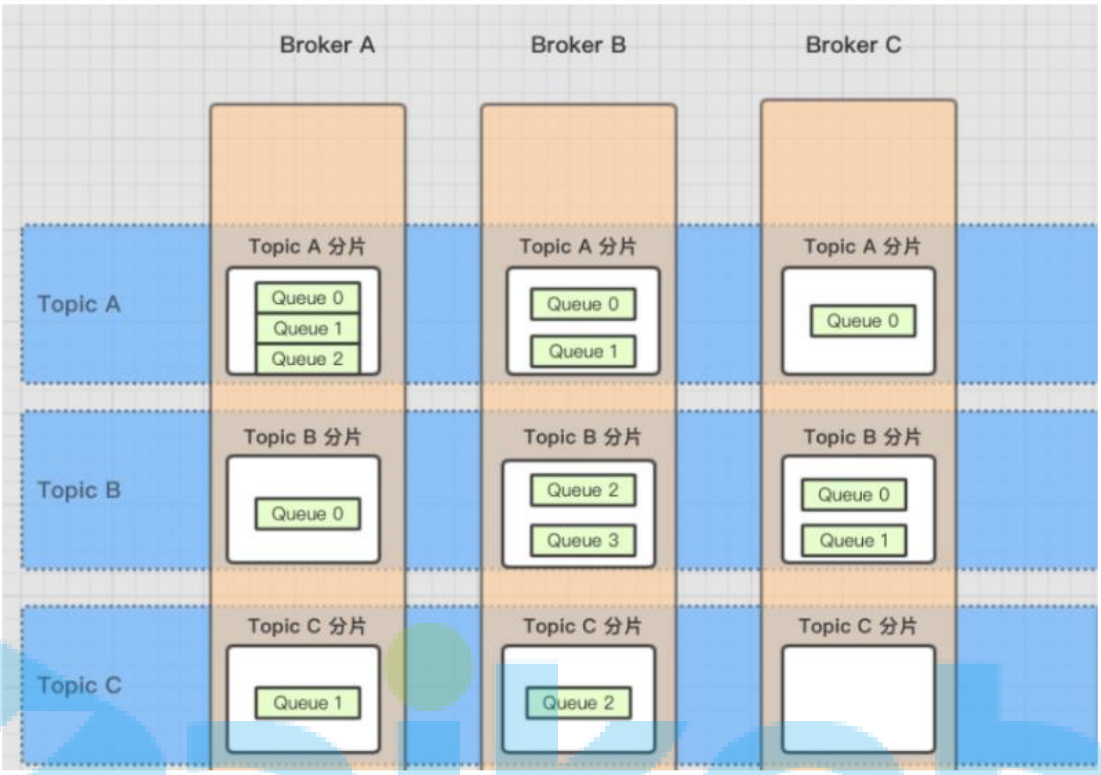
2012 年， MetaQ 已经发展到了 3.0 版本，并抽象出了通用的消息引擎 RocketMQ。 随后，对 RocketMQ 进行 了开源 ， 阿里的消息中间件正式走人了 公众视野 。

2015 年， RocketMQ 已经经历了多年双十一的洗礼，在可用性、 可靠性以 及稳定性等方面都有出色的表现。 与此同时 ， 云计算大行其道， 阿里消息中间 件基于 RocketMQ 推出了 Aliware MQ 1.0，开始为阿里云上成 千上万家企业提 供消息服务 。

2016 年， MetaQ 在双十一期间承载了万亿级消息的流转，跨越了一个新的里程碑 ， 同时 RocketMQ 进入 Apache 孵化 。

2、消息存储

Topic 是一个逻辑上的概念，实际上 Message 是在每个 Broker 上以 Queue 的形式记录。



从上面的图片可以总结下几条结论。

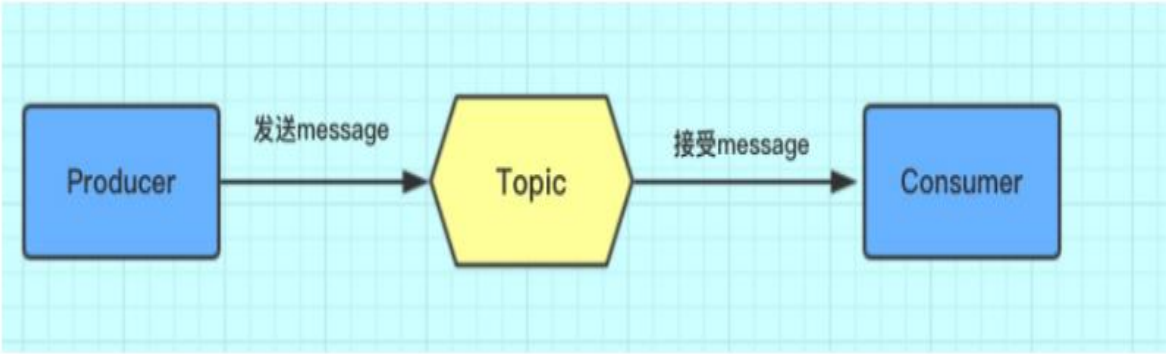
1、消费者发送的 Message 会在 Broker 中的 Queue 队列中记录
2、一个 Topic 的数据可能会存在多个 Broker 中
3、一个 Broker 存在多个 Queue

也就是说每个 Topic 在 Broker 上会划分成几个逻辑队列，每个逻辑队列保存一部分消息数据，但是保存的消息数据实际上不是真正的消息数据，而是指向 commit log 的消息索引

3、消息发送

3.1、简化流程

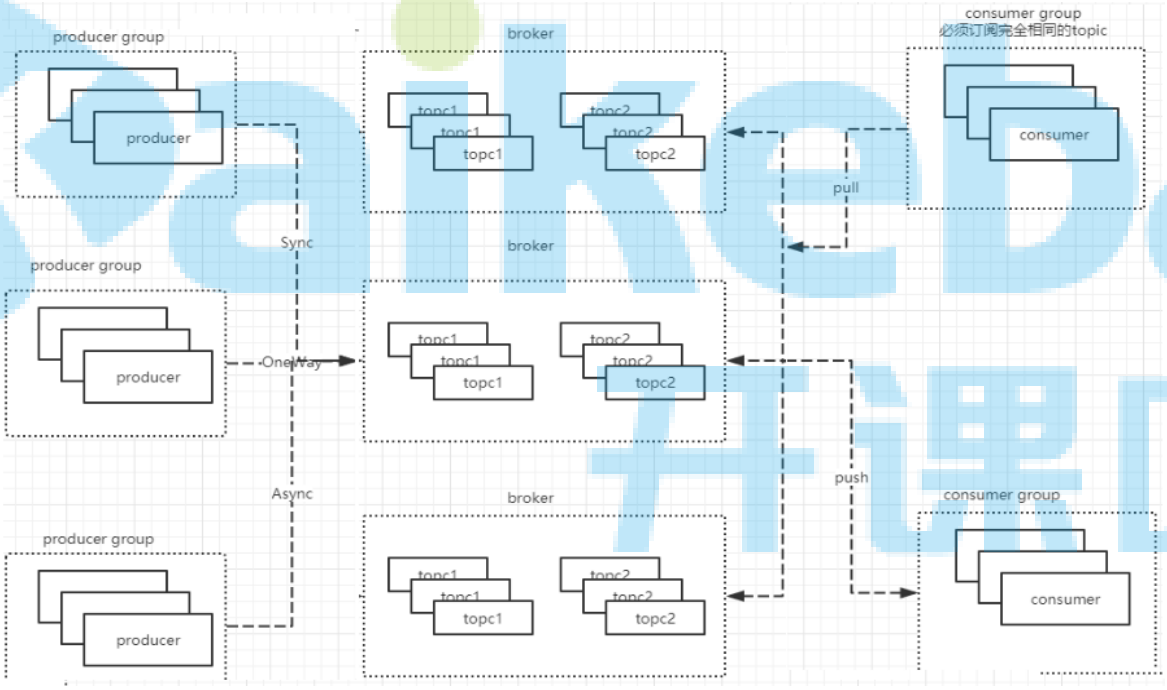
一个消息从发送，到接收，最简单的步骤：producer,topic,consumer,先由简单到复杂的来理解它的一些核心概念



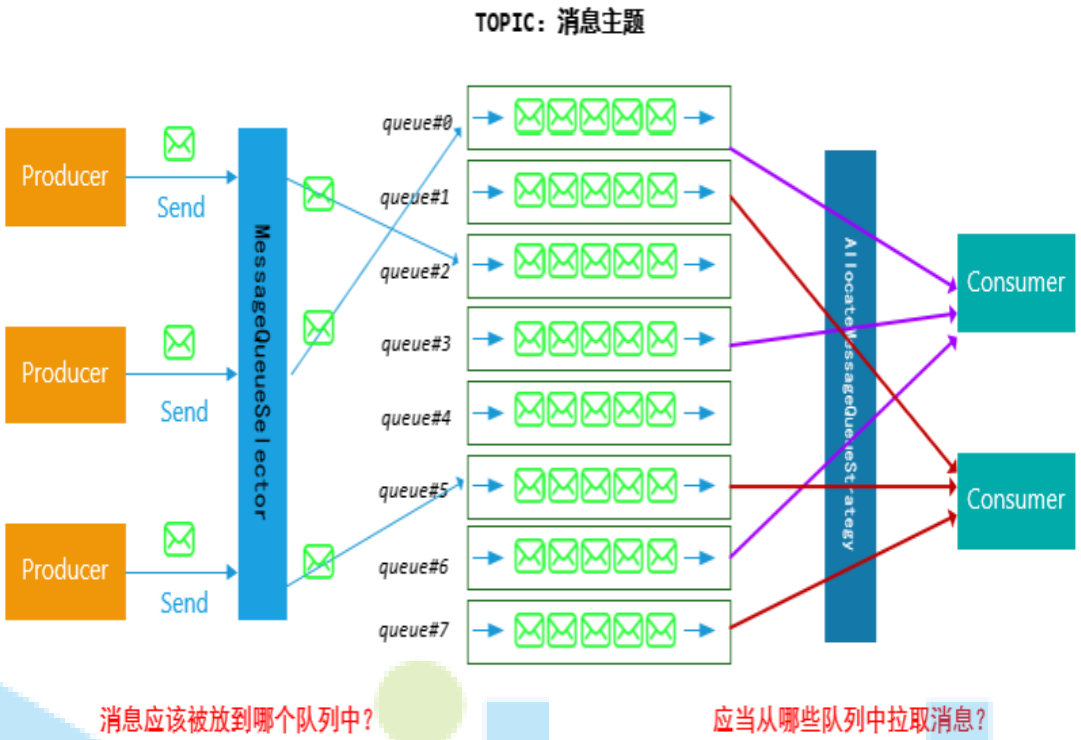
消息先发到 Topic，然后消费者去 Topic 拿消息。只是 Topic 在这里只是个概念，那它到底是怎么存储消息数据的呢，这里就要引入 Broker 概念。

3.2、细化流程

详细的消息发送及接受流程：



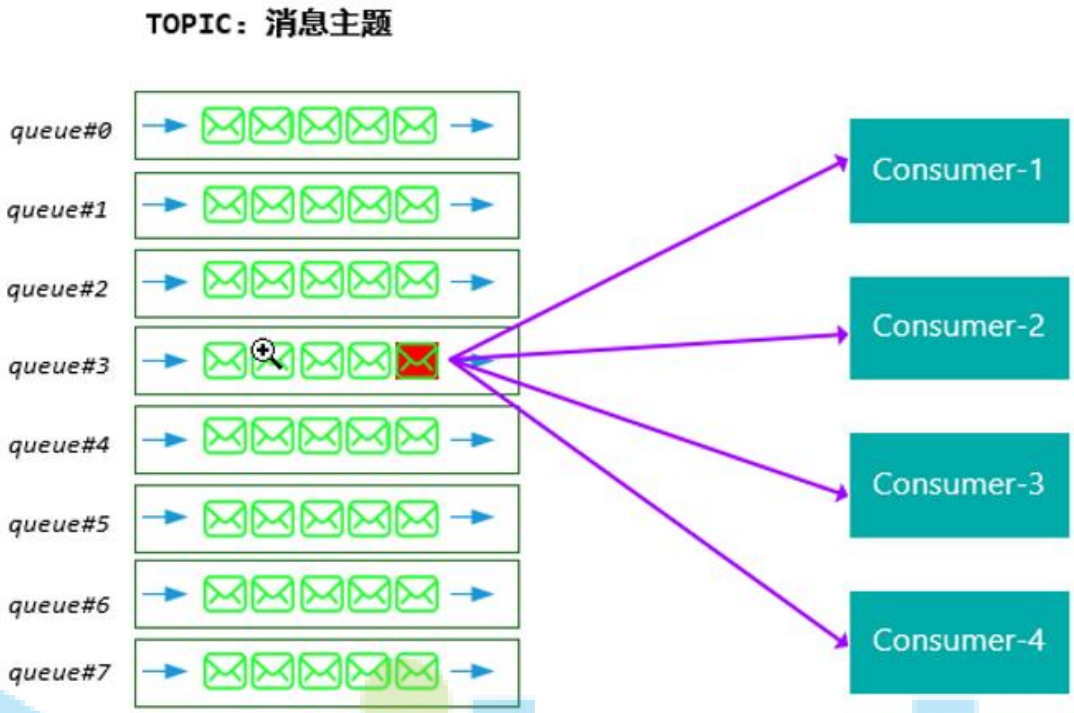
消息被发送到 queue 中进行标记：



4、消息消费

4.1、广播消费

一条消息被多个 Consumer 消费，即使这些 Consumer 属于同一个 Consumer Group，消息也会被 Consumer
Group 中的每个 Consumer 都消费一次，广播消费中的 Consumer Group 概念可以认为在消息划分方面无意义。

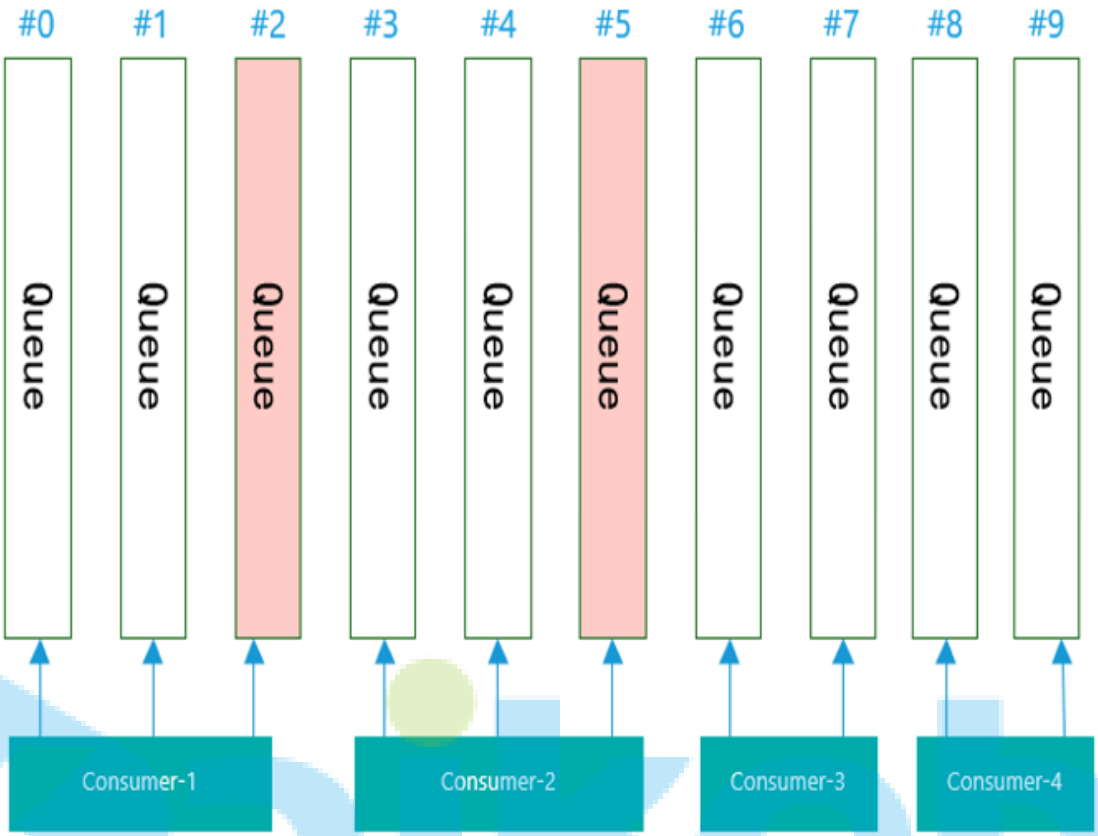


4.2、集群消费

（1）平均分配算法

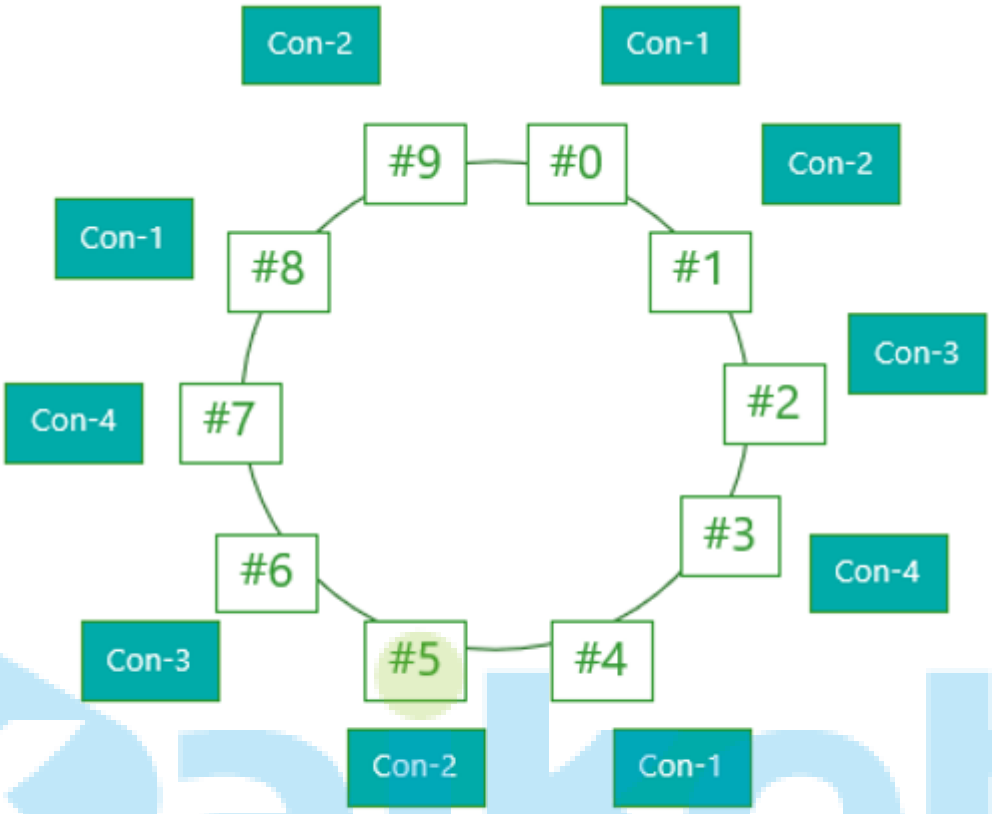
这里所谓的平均分配算法，并不是指的严格意义上的完全平均，如上面的例子中，10个queue，而消费者只有 4 个，无法是整除关系，除了整除之外的多出来的 queue,将依次根据消费者的顺序均摊。

consumer-1:3 个;consumer-2:3 个;consumer-3:2 个;consumer-4:2 个,如下图所示：



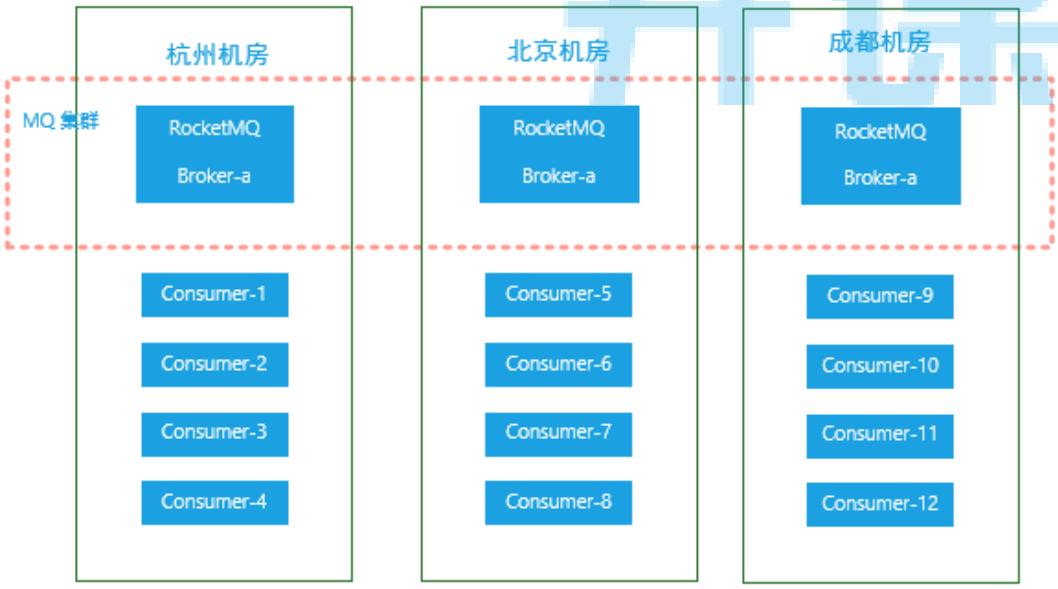
（2）环形平均算法

是指根据消费者的顺序，依次在由 queue 队列组成的环形图中逐个分配。具体流程如下所示:



环形平均算法

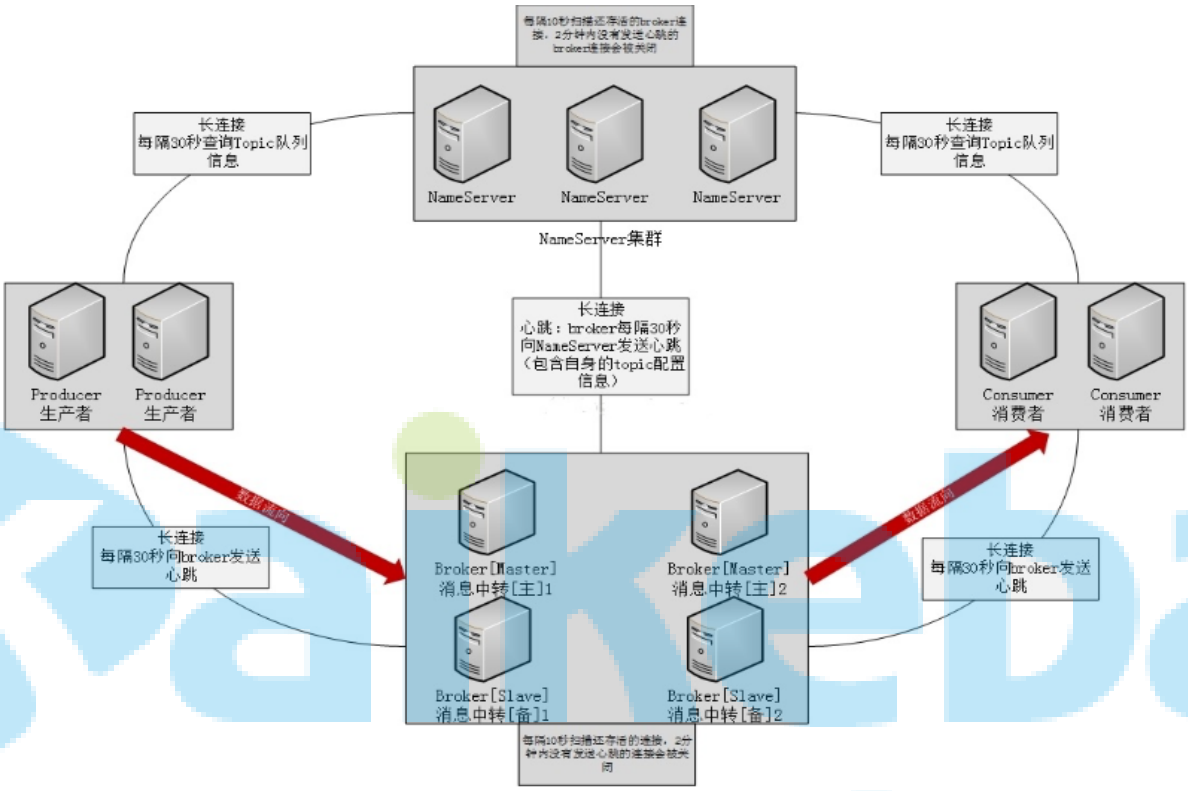
（3）机房临近法



基于机器临近算法

四、RocketMQ 架构

1、网络架构



2、何为 nameserver？

2.1、基本概念

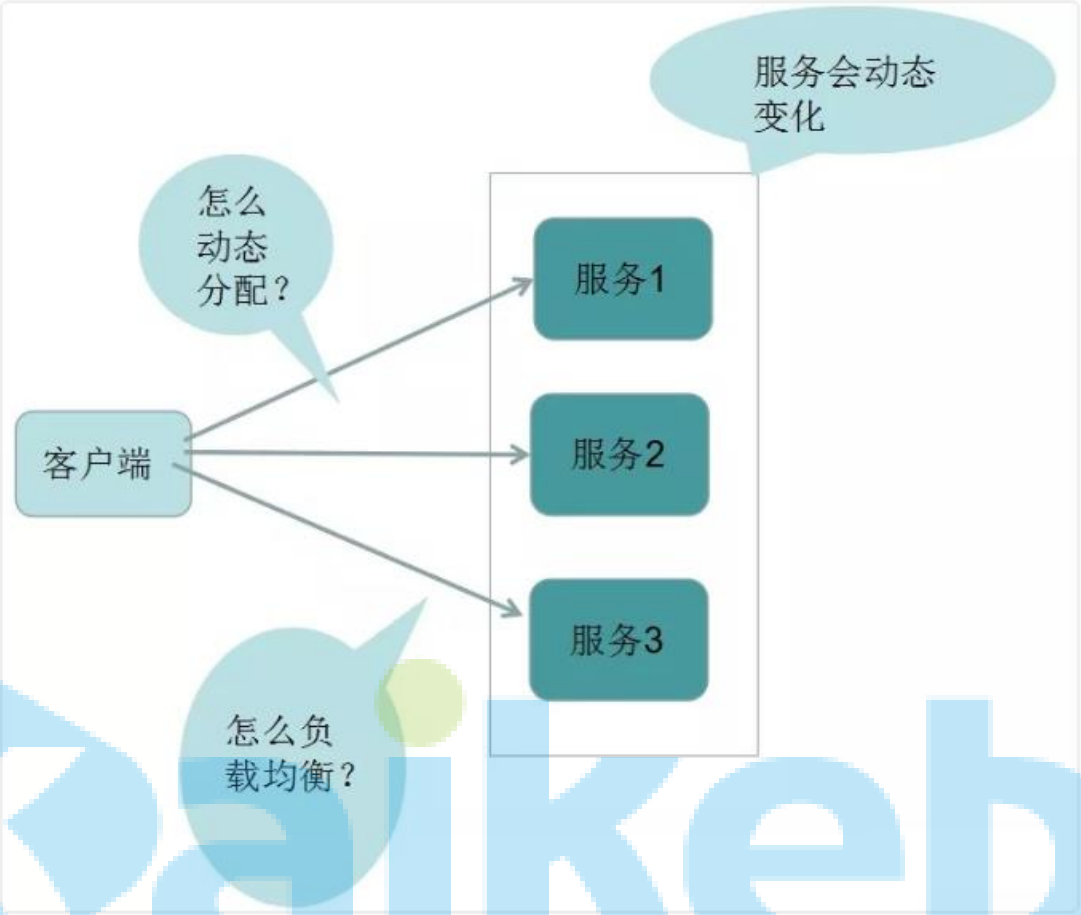
相对来说，nameserver 的稳定性非常高。原因有二：

1 ）nameserver 互相独立，彼此没有通信关系，单台 nameserver 挂掉，不影响其他 nameserver，即使全部挂掉，也不影响业务系统使用。无状态

2 ）nameserver 不会有频繁的读写，所以性能开销非常小，稳定性很高。

总结：NameServer 是一个几乎无状态的节点，可集群部署，节点之间无任何信息同步

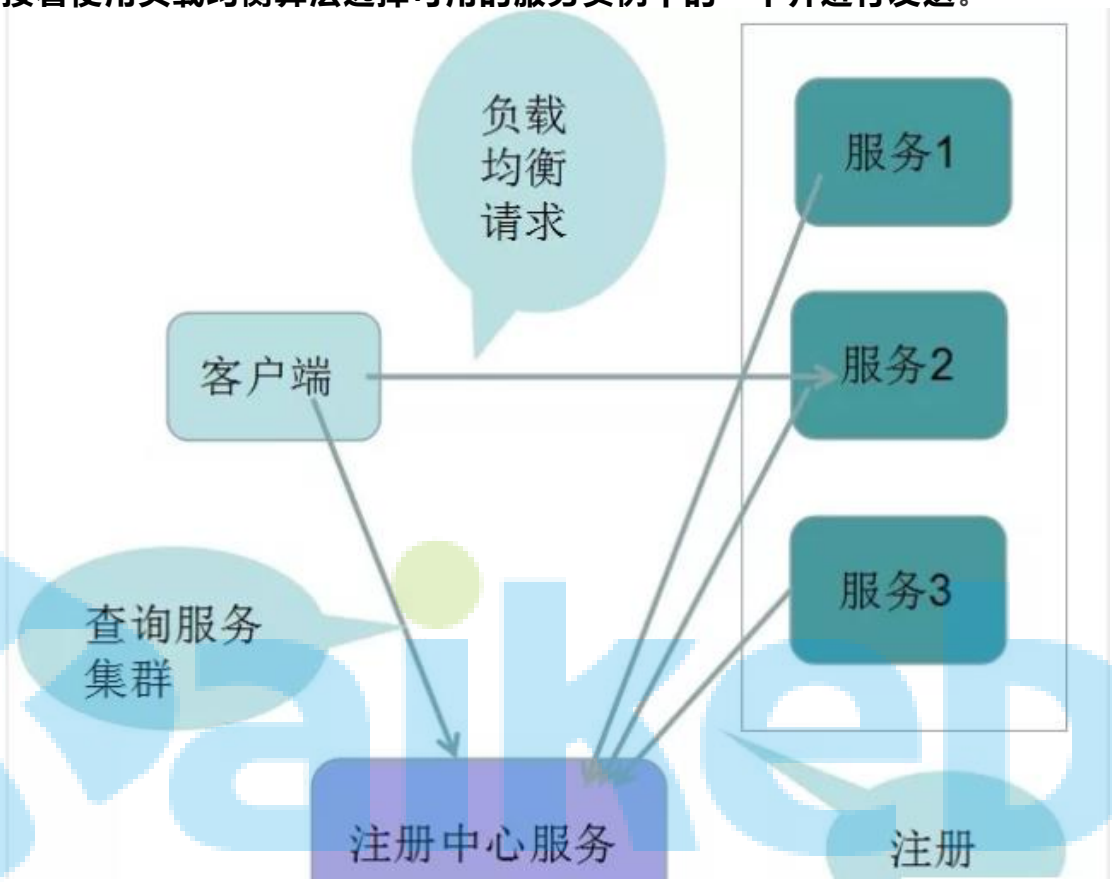
2.2、Namesrv 存在意义



服务发现机制：

当发出请求服务时,客户端通过注册中心服务知道所有的服务实例。客户端

接着使用负载均衡算法选择可用的服务实例中的一个并进行发送。



3、何为 broker?

1）基本概念
Broker 就是用来存储消息的服务。Broker 通常都是以集群的方式存在，消息发送者把消息发送给 broker 进行存储。
2） 与 nameserver 关系
3） 负载均衡
4） 可用性
5） 可靠性
6） 消息清理
总结：

Broker 部署相对复杂，Broker 分为 Master 与 Slave，一个 Master 可以对应多个 Slave，但是一个 Slave 只能对应一个 Master，Master 与 Slave 的对应关系通过指定相同的 BrokerName，不同的 BrokerId 来定义，BrokerId 为 0 表示 Master，非 0 表示 Slave。Master 也可以部署多个。每个 Broker 与 Name Server 集群中的所有节点建立长连接，定时注册 Topic 信息到所有 Name Server。

4、消费者

1）与 nameserver 关系
2）与 broker 关系
3）负载均衡
4）消费机制
总结：

Consumer 与 Name Server 集群中的其中一个节点（随机选择，但不同于上一次）建立长连接，定期从 Name Server 取 Topic 路由信息，并向提供 Topic 服务的 Master、Slave 建立长连接，且定时向 Master、Slave 发送心跳

5、生产者

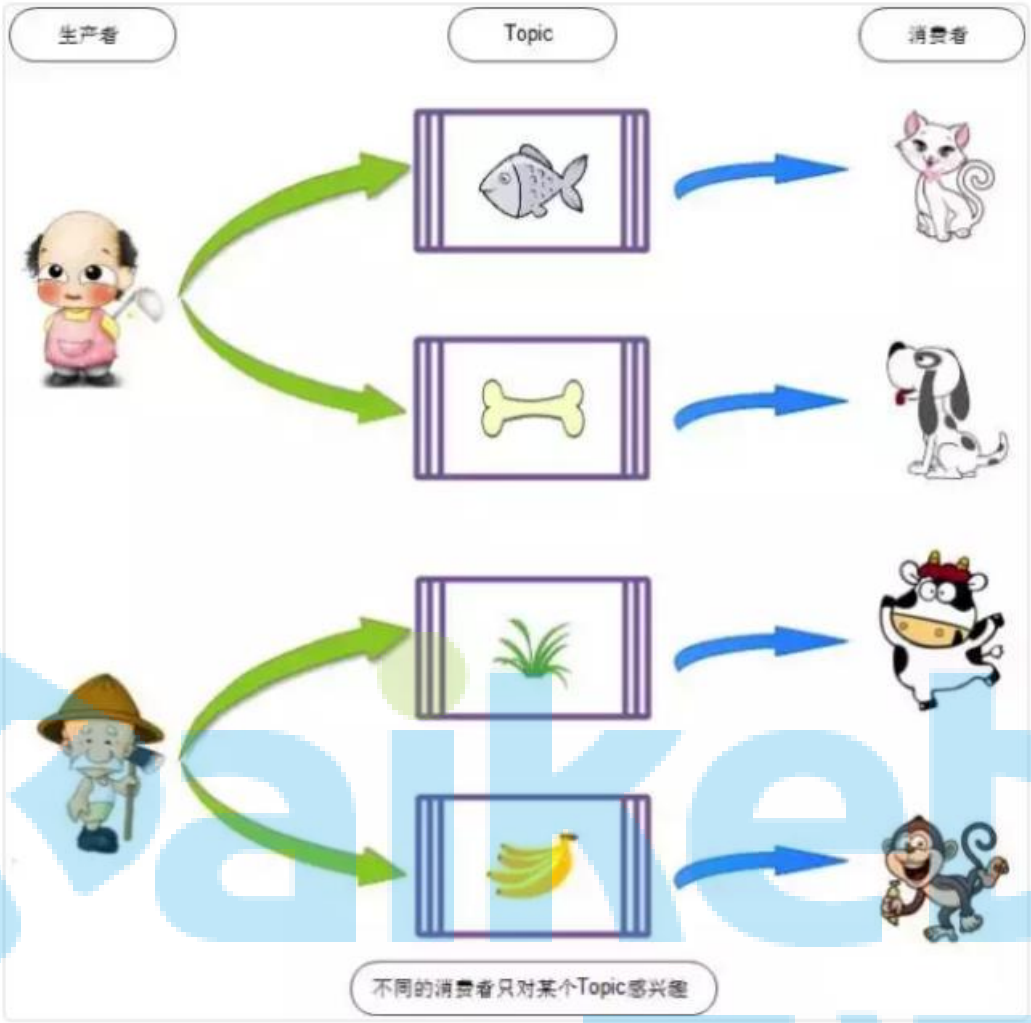
1）与 nameserver 关系
2）与 broker 关系
3）负载均衡
总结：

Producer 与 Name Server 集群中的其中一个节点（随机选择，但不同于上一次）建立长连接，定期从 Name Server 取 Topic 路由信息，并向提供 Topic 服务的 Master 建立长连接，且定时向 Master 发送心跳

6、何为 Topic？

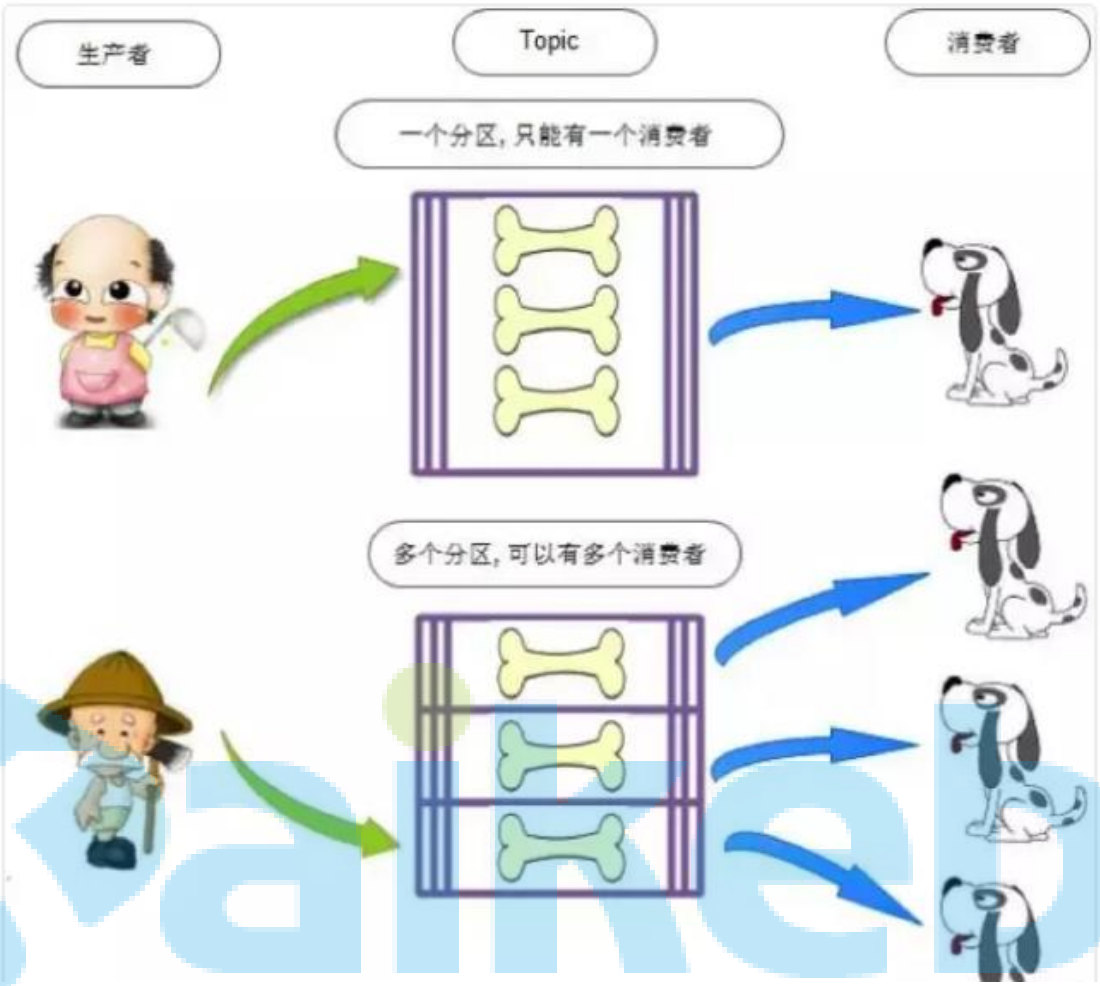
Topic 是消息中间件里一个重要的概念，每一个 Topic 代表了一类消息，有了多个 Topic，就可以对消息进行归类与隔离。

可以参照下图的动物园喂食模型，每一种动物都只能消费相对应的食品。

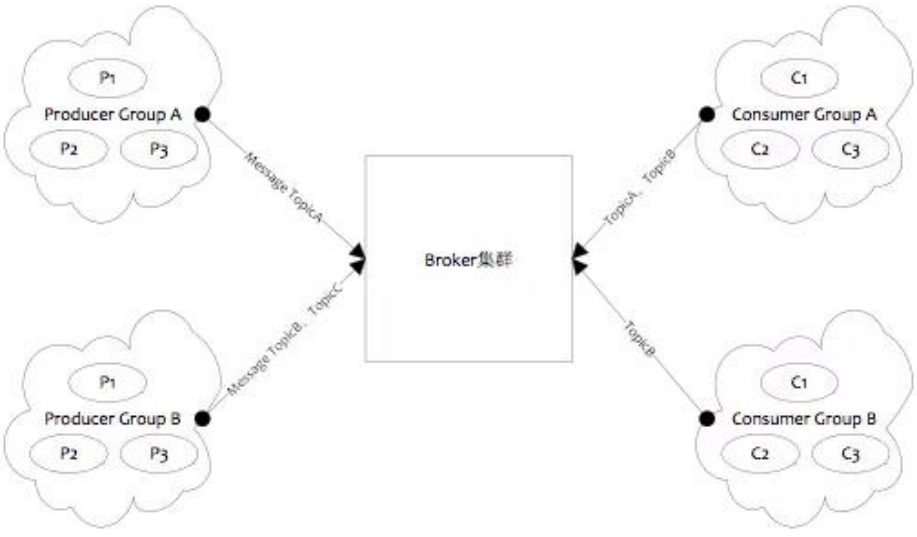


7、何为分区（queue）？

RocketMQ 是磁盘消息队列的模式，对于同一个消费组，一个分区只支持一个消费线程来消费消息。过少的分区，会导致消费速度大大落后于消息的生产速度。所以在实际生产环境中，一个 Topic 会设置成多分区的模式，来支持多个消费者，参照下图：



8、RocketMQ 名词解释



1）Producer

消息生产者，位于用户的进程内，`Producer 通过 NameServer 获取所有 Broker 的路由信息`，根据负载均衡策略选择将消息发到哪个 Broker，然后调用 Broker 接口提交消息。

2）Producer Group

生产者组，简单来说就是多个发送同一类消息的生产者称之为一个生产者组。

3）Consumer

消息消费者，位于用户进程内。Consumer 通过 NameServer 获取所有 broker 的路由信息后，向 Broker 发送 Pull 请求来获取消息数据。Consumer 可以以两种模式启动，**广播（Broadcast）和集群（Cluster）**，**广播模式下，一条消息会发送给所有 Consumer，集群模式下消息只会发送给一个 Consumer**。

4）Consumer Group

消费者组，和生产者类似，消费同一类消息的多个 Consumer 实例组成一个消费者组。

5）Topic

Topic 用于将消息按主题做划分，**Producer 将消息发往指定的 Topic，Consumer 订阅该 Topic 就可以收到这条消息**。Topic 跟发送方和消费方都没有强关联关系，发送方可以同时往多个 Topic 投放消息，消费方也可以订阅多个 Topic 的消息。在 RocketMQ 中，**Topic 是一个上逻辑概念。消息存储不会按 Topic 分开**。

Topic 表示消息的第一级类型，比如一个电商系统的消息可以分为 :交易消息、物流消息等。一条消息必须有一个 Topic。最细粒度的订阅单位，一个 Group 可以订阅多个 Topic 的消息。

6）Message

代表一条消息，使用`MessageId`唯一识别，用户在发送时可以设置 messageKey，便于之后查询和跟踪。一个 Message 必须指定 Topic，相当于寄信的地址。Message 还有一个可选的 Tag 设置，以便消费端可以基于 Tag 进行过滤消息。也可以添加额外的键值对，例如你需要一个业务 key 来查找 Broker 上的消息，方便在开发过程中诊断问题。

7）Tag

标签可以被认为是对 Topic 进一步细化。一般在相同业务模块中通过引入标签来标记不同用途的消息。

Tag 表示消息的第二级类型，比如交易消息又可以分为：交易创建消息，交易完成消息等。RocketMQ 提供 2 级消息分类，方便灵活控制。

8）Broker

Broker 是 RocketMQ 的核心模块，`负责接收并存储消息`，同时提供 Push/Pull 接口来将消息发送给 Consumer。Consumer 可选择从 Master 或者 Slave 读取数据。多个主/从组成 Broker 集群，集群内的 Master 节点之间不做数据交互。Broker 同时提供消息查询的功能，可以通过 MessageID 和 MessageKey 来查询消息。Borker 会将自己的 Topic 配置信息实时同步到 NameServer。

9）Queue

Topic 和 Queue 是 1 对多的关系，**一个 Topic 下可以包含多个 Queue**，主要用于负载均衡。发送消息时，用户只指定 Topic，Producer 会根据 Topic 的路由信息选择具体发到哪个 Queue 上。Consumer 订阅消息时，会根据负载均衡策略决定订阅哪些 Queue 的消息。

消息的物理管理单位。一个 Topic 下可以有多个 Queue，Queue 的引入使得消息的存储可以分布式集群化，具有了水平扩展能力。

在 RocketMQ 中，所有消息队列都是持久化，长度无限的数据结构，所谓长度无限是指队列中的每个存储单元都是定长，访问其中的存储单元使用 Offset 来访问，offset 为 java long 类型，64 位，理论上在 100 年内不会溢出，所以认为是长度无限。

也可以认为 Message Queue 是一个长度无限的数组，Offset 就是下标。

10）Offset

RocketMQ 在存储消息时会为每个 Topic 下的每个 Queue 生成一个消息的索引文件，每个 Queue 都对应一个 Offset**记录当前 Queue 中消息条数**