An overview of Apache RocketMQ

研发处

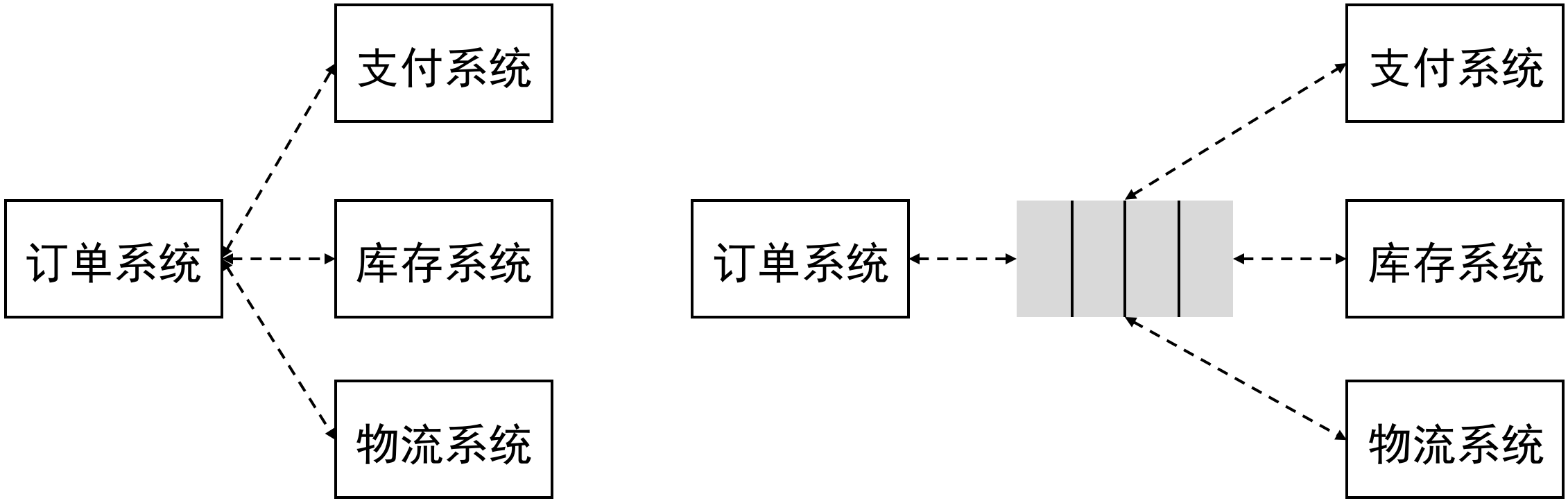
2019. 09. 05

Content

- RocketMQ 产品概述
- RocketMQ 名称服务
- RocketMQ 消息发送
- RocketMQ 消息存储
- RocketMQ 消息消费
- RocketMQ 部署运维
- RocketMQ 应用实战
- RocketMQ 源码阅读

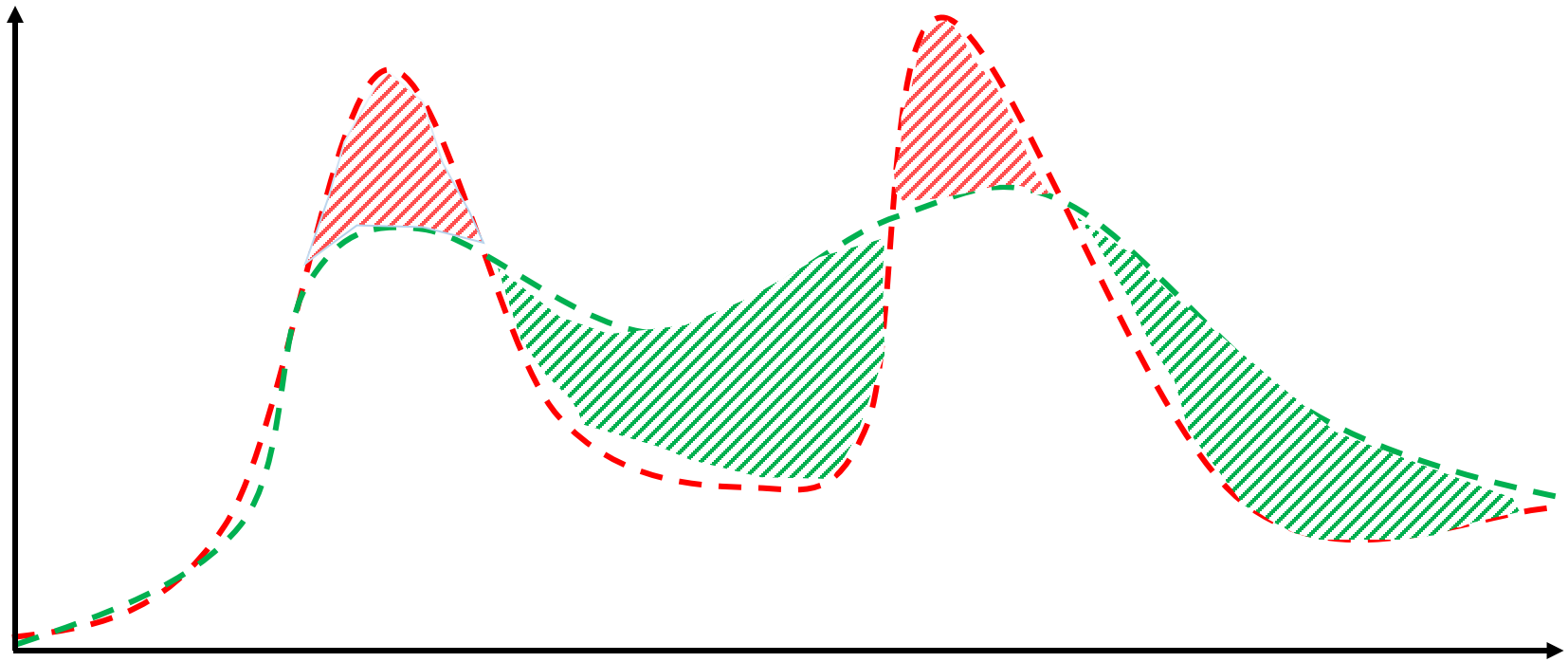
为什么需要消息队列 消息队列核心功能

- 应用解耦，提高系统可用性



为什么需要消息队列 消息队列核心功能

- 流量削峰，将高峰流量缓冲存储到消息队列，保证系统稳定



为什么需要消息队列 消息队列核心功能

- 弹性扩容和缩容
- 快速存储和持久化
- 分布式事务
- 回溯消息
- 过滤消息
- 定时消息
- ···

什么是RocketMQ

- 阿里巴巴开源可支撑万亿级数据洪峰的分布式消息和流计算平台，于2016年捐赠给Apache Software Foundation，2017年9月25日成为Apache 顶级项目
- 大规模应用于金融、互联网、物流公司核心交易支付、实时位置追踪、大数据分析等场景，同时也被电力、交通、汽车、零售等十几个行业的数万家企业广泛使用，企业数字化转型的核心基础性软件

RocketMQ特点

Low Latency
More than 99.6% response latency within 1 millisecond under high pressure.

Vendor Neutral
A new open distributed messaging and streaming standard since latest 4.1 version.

Finance Oriented
High availability with tracking and auditing features.

BigData Friendly
Batch transferring with versatile integration for flooding throughput.

Industry Sustainable
Trillion-level message capacity guaranteed.

Massive Accumulation
Given sufficient disk space, accumulate messages without performance loss.

RocketMQ特点

低延迟
高负载场景下99.6%响应延迟在1ms以内

无关供应商
自最新4.1版本以来的新开放式分布式消息传递和流媒体标准 OpenMessaging

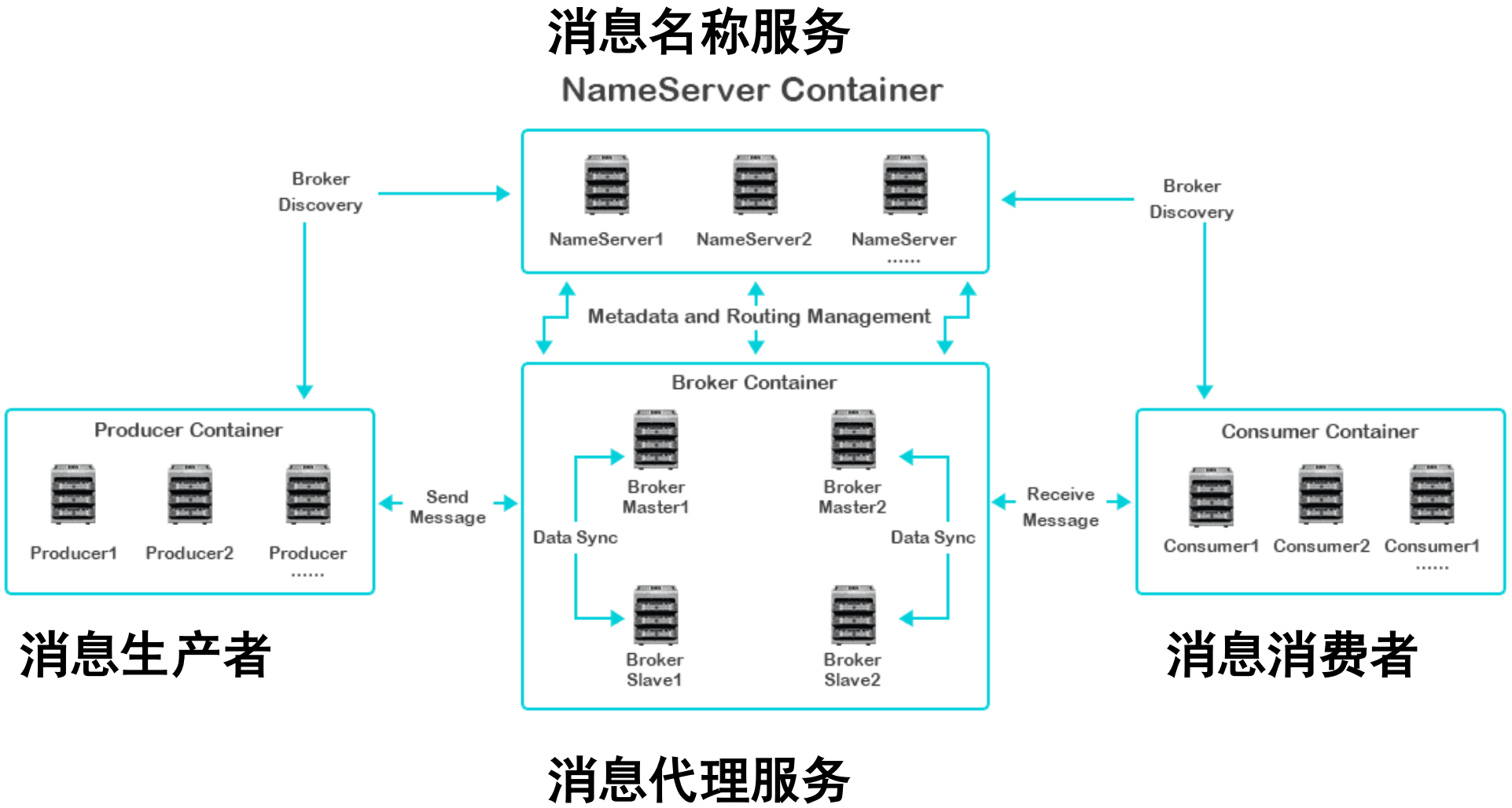
面向财务
具有跟踪和审计功能的高可用性

大数据友好
支持批量传输以实现洪峰吞吐

工业级别稳定
保证万亿级消息容量

大规模累积
给定足够的磁盘空间，累积大量消息而不会丢失性能

RocketMQ系统组成

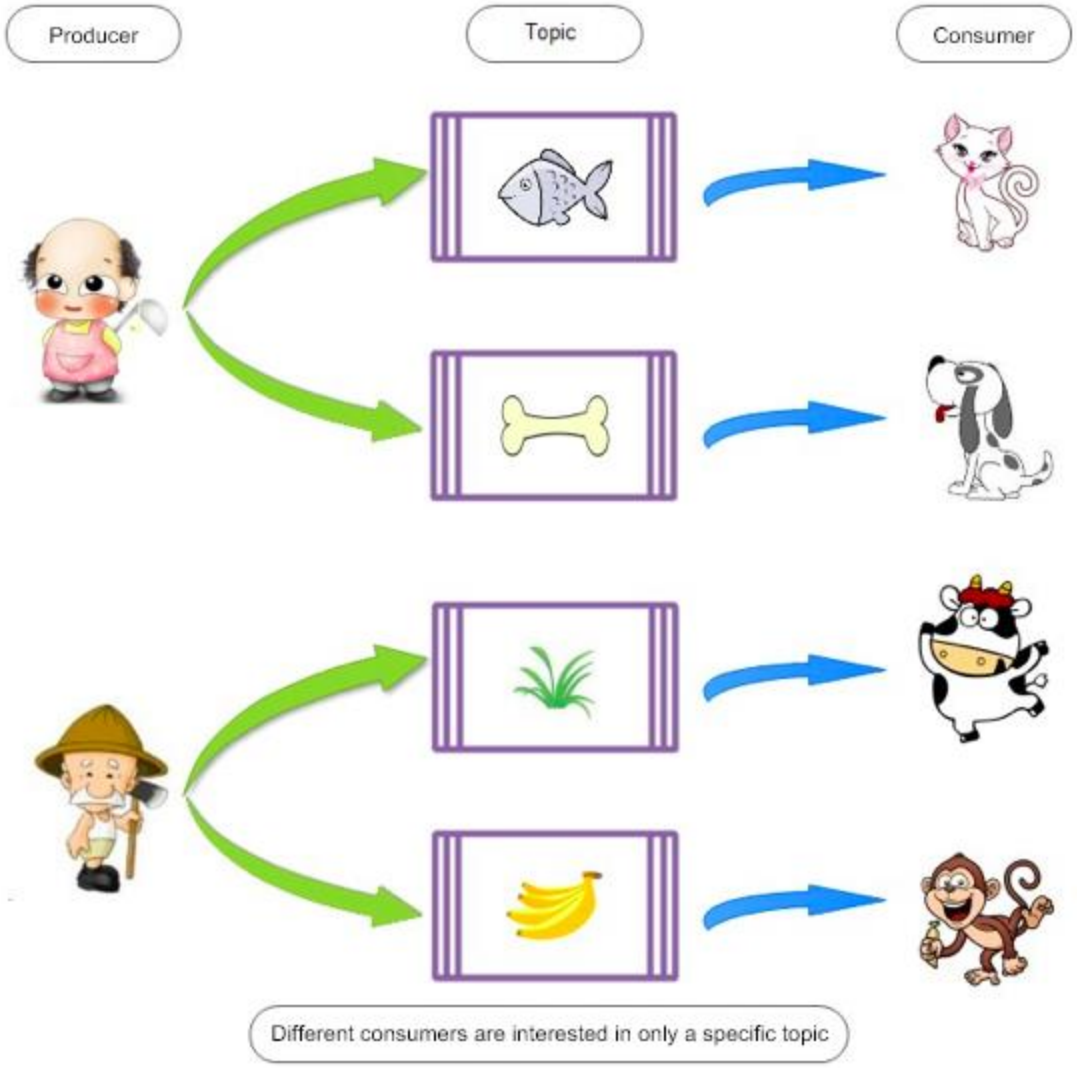


生产者和消费者组 RocketMQ概念模型

- 生产者组
  - 同一类Producer的集合，这类Producer发送同一类消息且发送逻辑一致。如果发送的是事物消息且原始生产者在发送之后崩溃，则Broker服务器会联系同一生产者组的其他生产者实例以提交或回溯消费
- 消费者组
  - 同一类Consumer的集合，这类Consumer通常消费同一类消息且消费逻辑一致。消费者组使得在消息消费时，实现负载均衡和容错的目标变得非常容易

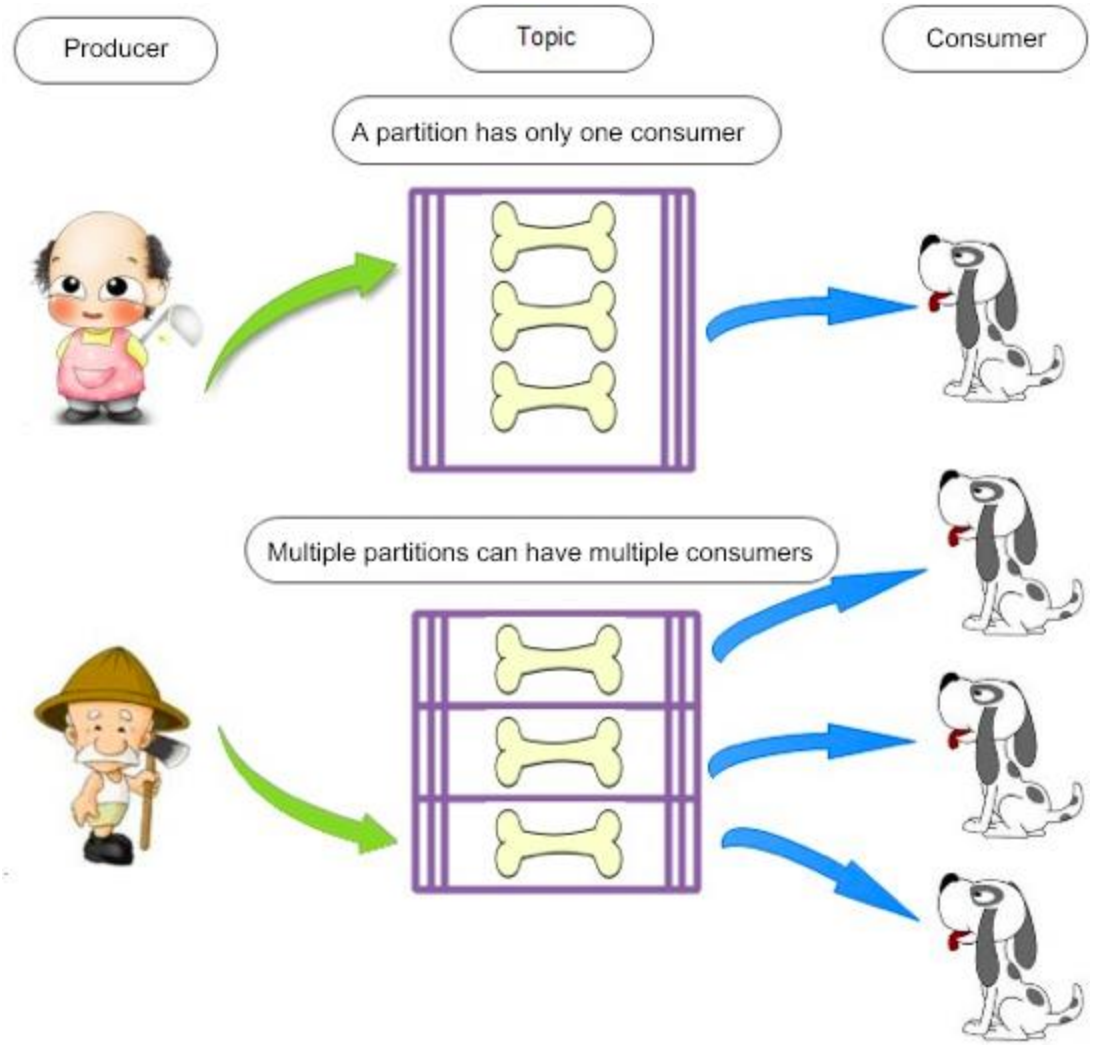
Topic 主题 RocketMQ 概念模型

- 每个主题代表一个消息类别
- 通过多个主题可以实现对消息进行分类和隔离
- RocketMQ还提供了Tag标签将主题进一步分类和隔离



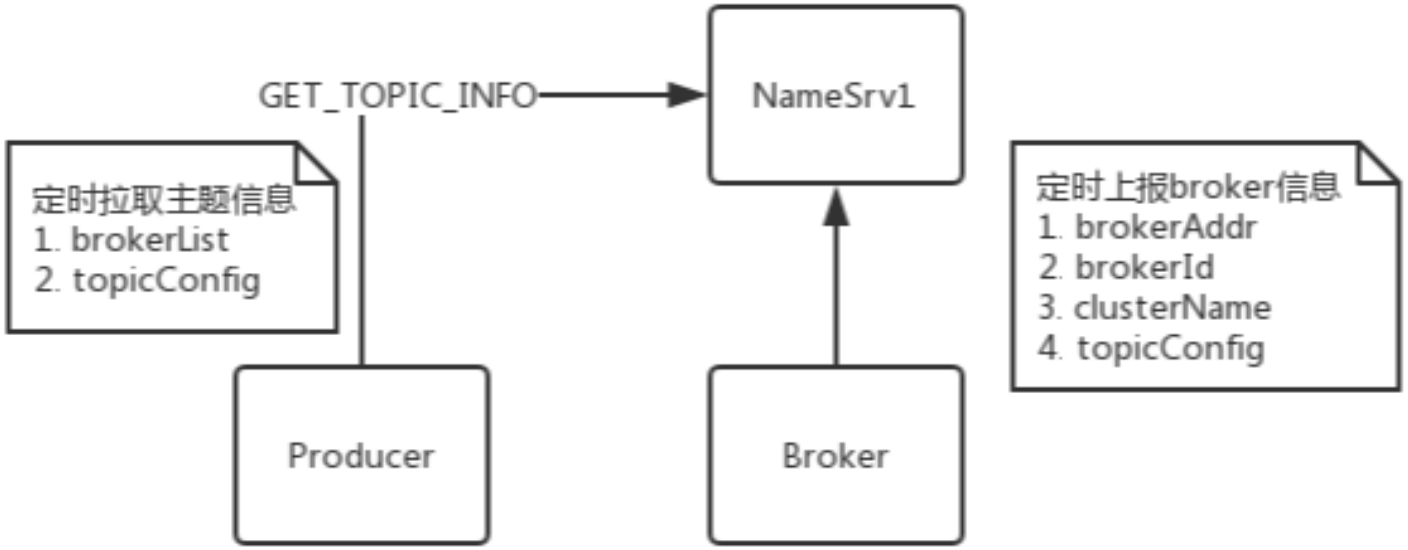
Message Queue队列 RocketMQ 概念模型

- 一个主题包含若干个消息队列
- 一个消息队列仅支持一个消费者消费



RocketMQ 名称服务

- Broker管理
  - 心跳接收Broker注册信息
  - 检查Broker是否存活
- TopicRouteinfo管理
  - 保存Broker路由信息
  - 客户端通过名称服务寻址消息队列路由信息

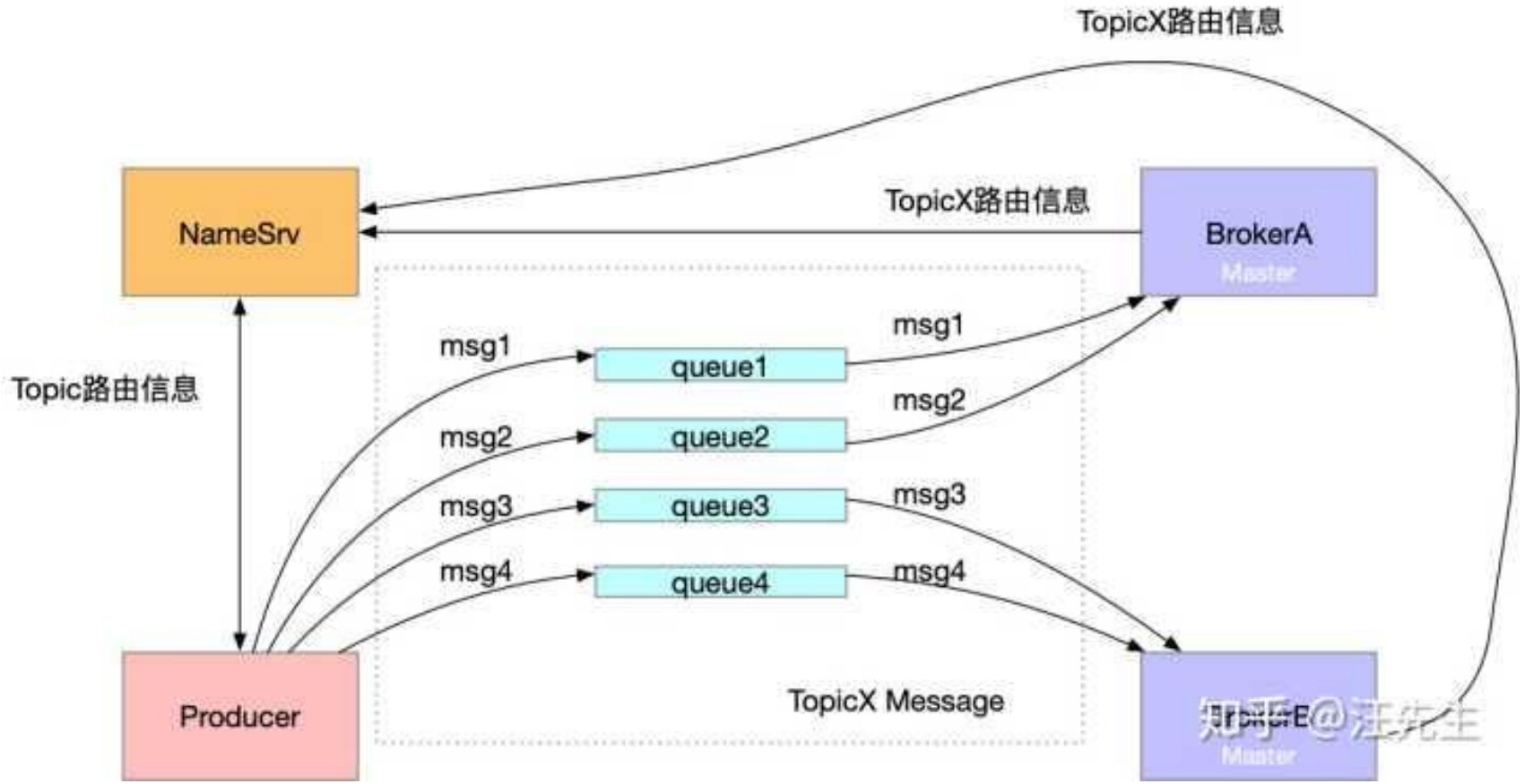


特点 RocketMQ名称服务

- 稳定性
  - 互相独立，无互相通信，单台挂掉，不影响其他节点
  - 不会有频繁读写，性能开销非常小，稳定性很高
- 可用性
  - broker异常退出后，名称服务不能及时将broker移除，Producer根据主题获取到的路由信息包含已经宕机的broker，会导致消息发送失败，那么消息发送是否无法高可用
- KISS(Keep it simple,stupid)原则

RocketMQ 消息发送

- 发送过程
  - 查找主题路由信息
  - 选择队列
  - 消息发送
  - 失败重试



主题路由信息 RocketMQ 消息发送

- 队列数据

{"brokerName":"broker-a","queueId":0,"topic":"dev%rocketmq-plain-example-topic"},

{"brokerName":"broker-a","queueId":1,"topic":"dev%rocketmq-plain-example-topic"},

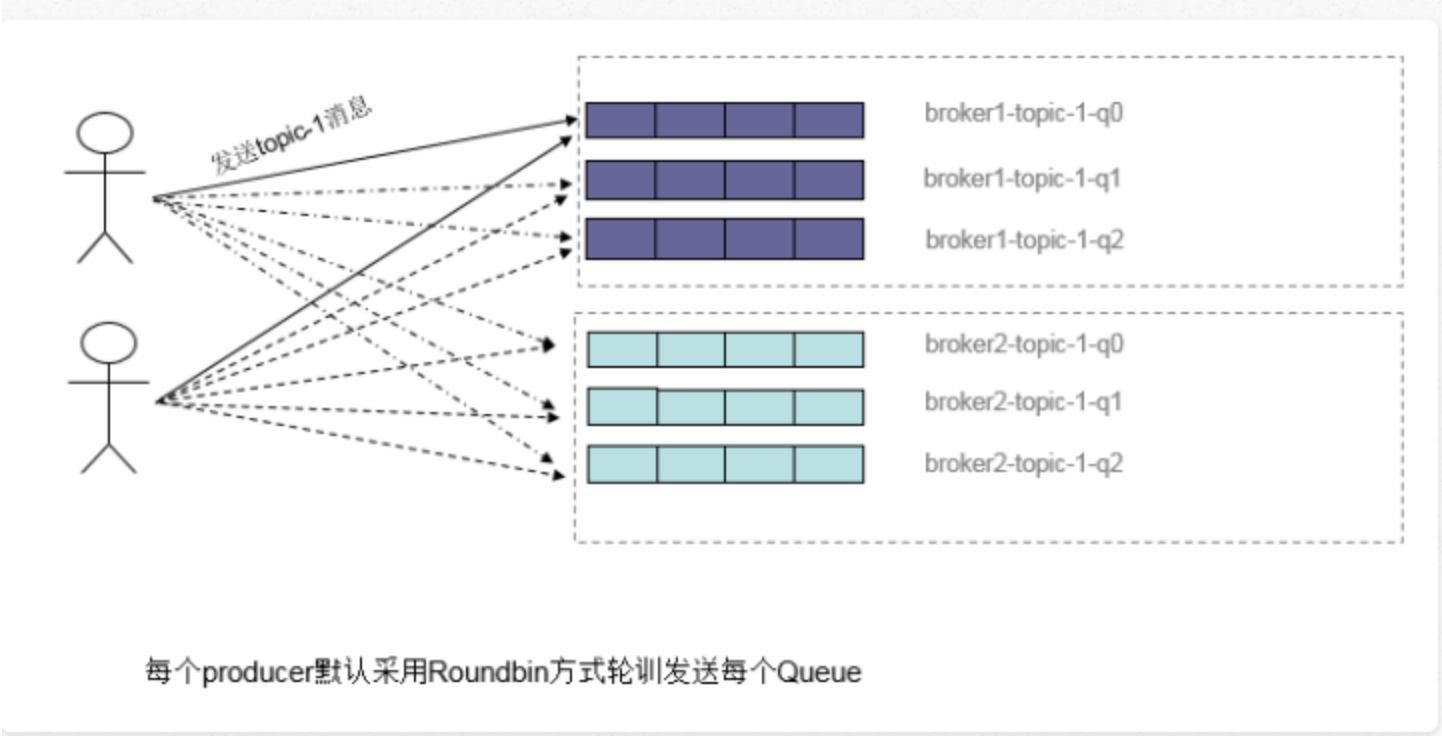
{"brokerName":"broker-b","queueId":0,"topic":"dev%rocketmq-plain-example-topic"},

{"brokerName":"broker-b","queueId":1,"topic":"dev%rocketmq-plain-example-topic"}

- 如何选择

选择队列和负载均衡 RocketMQ 消息发送

- 默认通过轮询方式
  - 保证所有消息队列均衡均衡
- 自定义选择队列方式
  - 实现消息全局有序或者局部有序



发送方式 RocketMQ 消息发送

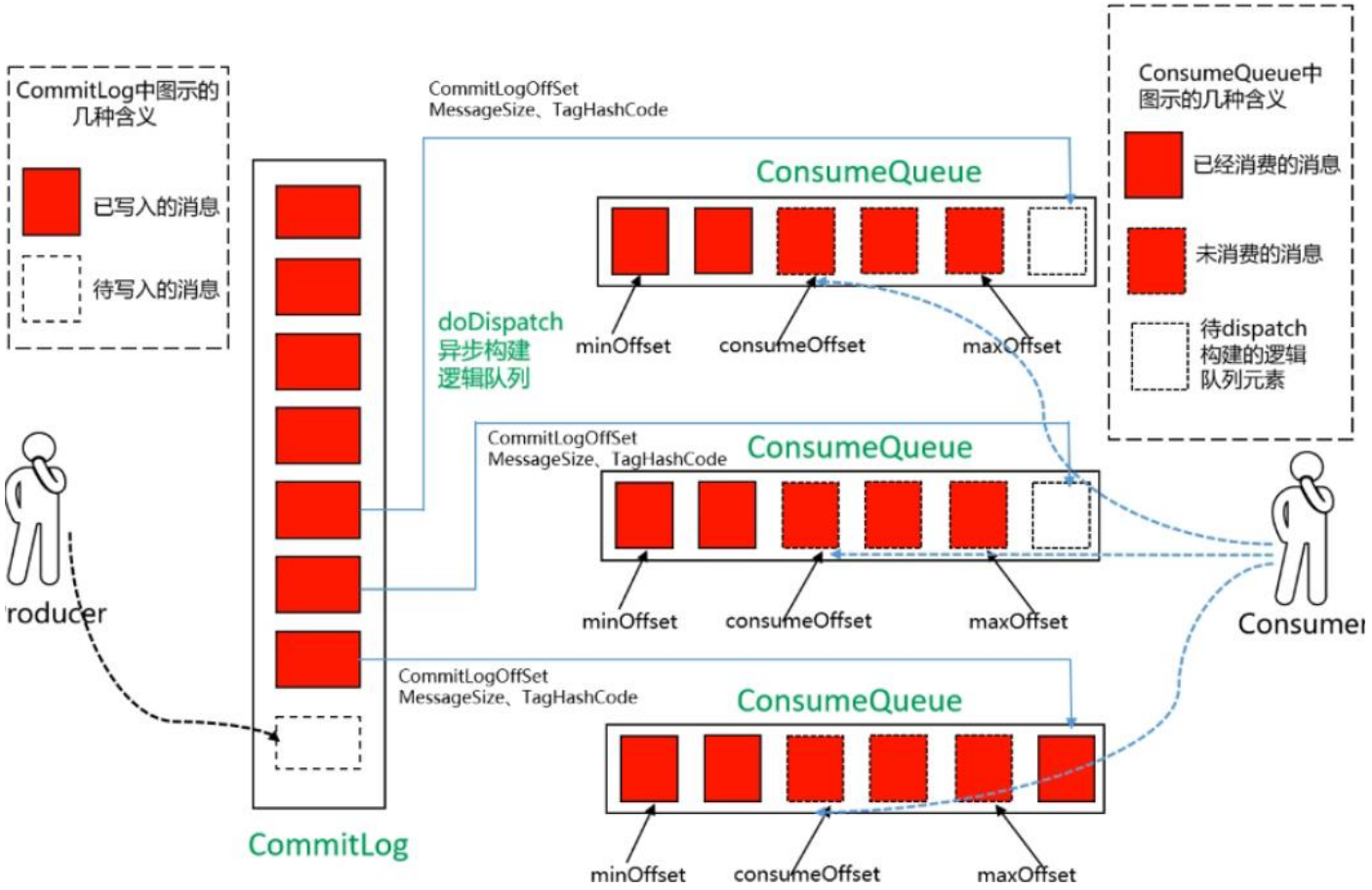
- 同步发送
  - 同步等待broker返回发送结果
- 异步发送
  - 消息发送结果返回后执行异步回调
- 单向发送
  - 只发送请求不等待应答
- 应用场景

失败重试机制 RocketMQ 消息发送

- 禁用broker故障延迟
  - 默认选项
  - 如果发生重试，会尝试在队列列表中寻找不属于lastBrokerName的MessageQueue，作为发送消息的目标队列
- 启用broker故障延迟
  - 将上次不可用Broker规避notAvailbleDuration时长后，再次参与路由计算，规避期间不选择该Broker，如果没有可选broker，返回MQ=null

RocketMQ 消息存储

- 消息存储文件
  - CommitLog
  - ConsumeQueue
  - IndexFile

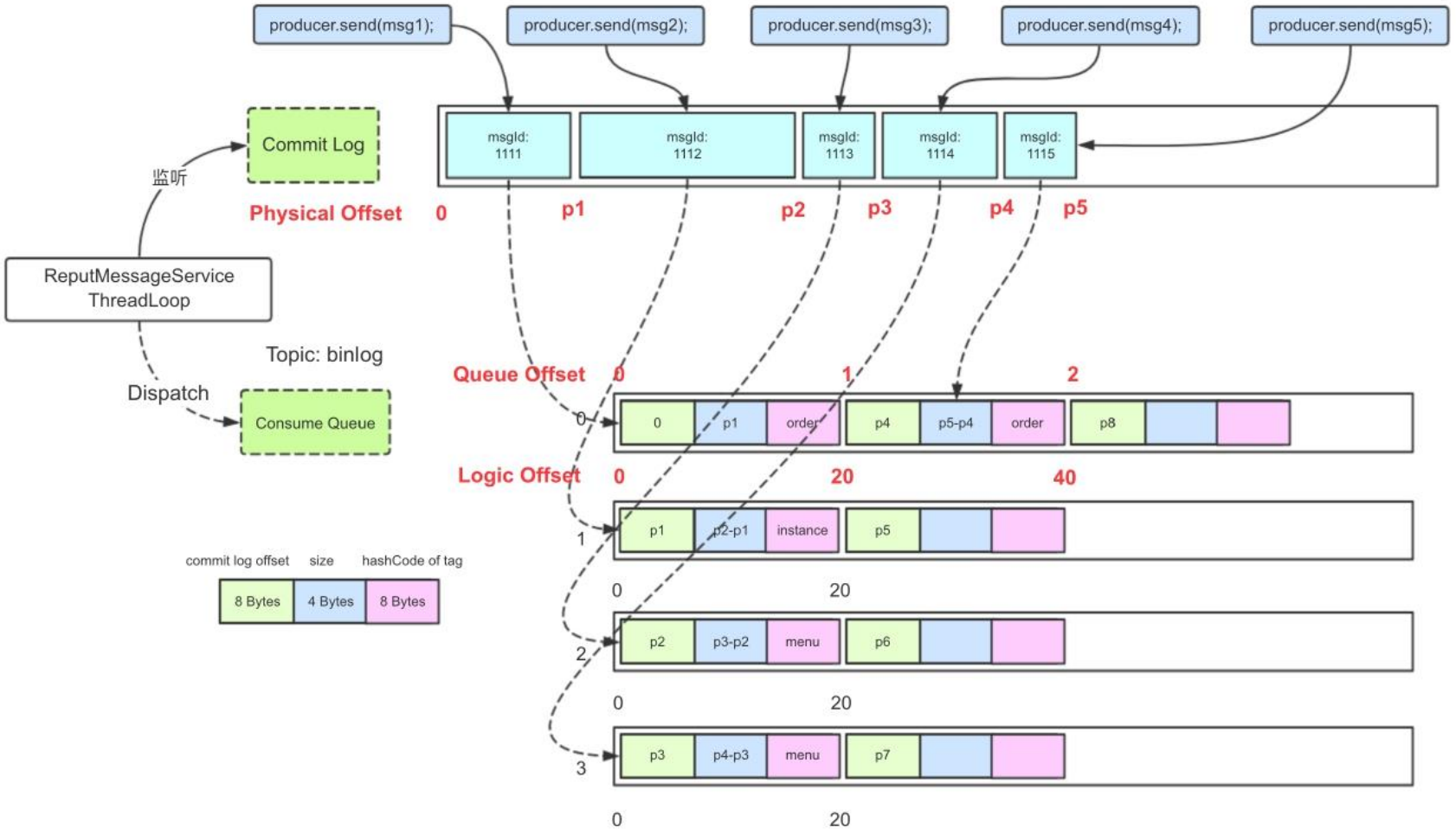


CommitLog RocketMQ 消息存储

- 消息主体以及元数据的存储主体
- 单文件大小定长1G，文件名长度为20位，左边补零，剩余为起始偏移量
- 单文件顺序追加，当文件满了，写入下一个文件

ConsumeQueue RocketMQ 消息存储

- 消费队列
  - 提高消息消费的性能
- 存储
  - 消息物理偏移
  - 消息大小
  - Tag hash值

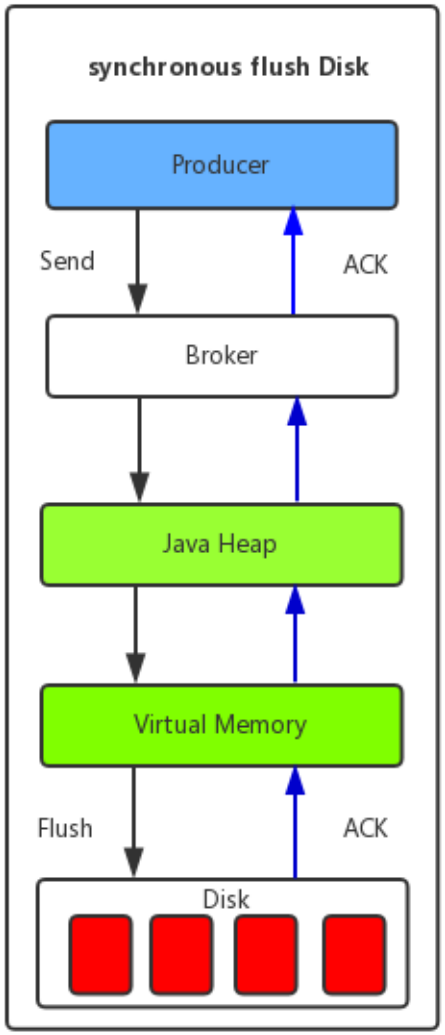


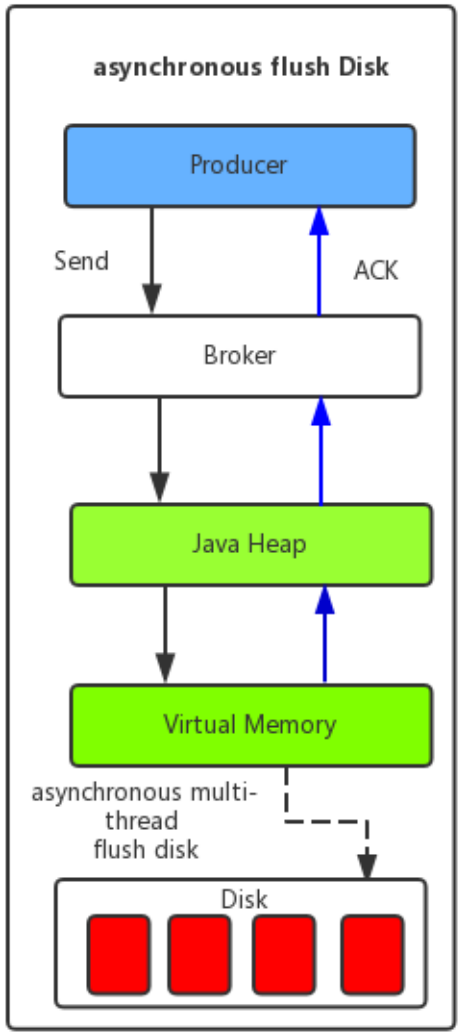
辅助文件 RocketMQ 消息存储

- 索引文件
  - 给RocketMQ提供了根据自定义业务KEY查询消息功能
- 配置文件
  - 消息主题
  - 消息过滤
  - 主题订阅

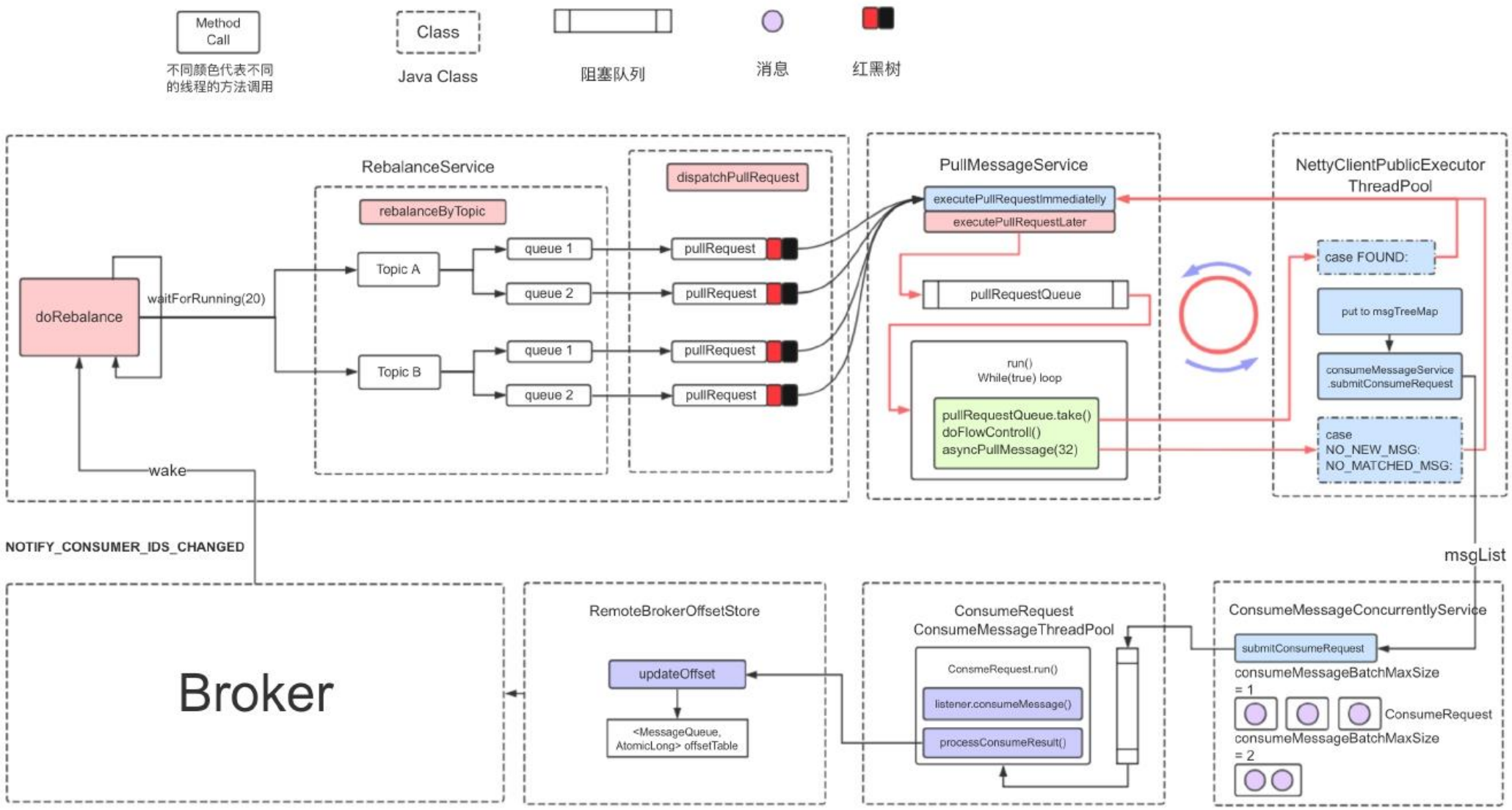
消息刷盘方式 RocketMQ 消息存储

- 同步刷盘
  - 消息持久化至磁盘后Broker端才返回给Producer端ACK响应
  - 提高MQ消息可靠性
- 异步刷盘
  - 只要消息写入PageCache即可将成功的ACK返回给Producer端
  - 降低读写延迟，提高MQ的性能和吞吐量





RocketMQ 消息消费

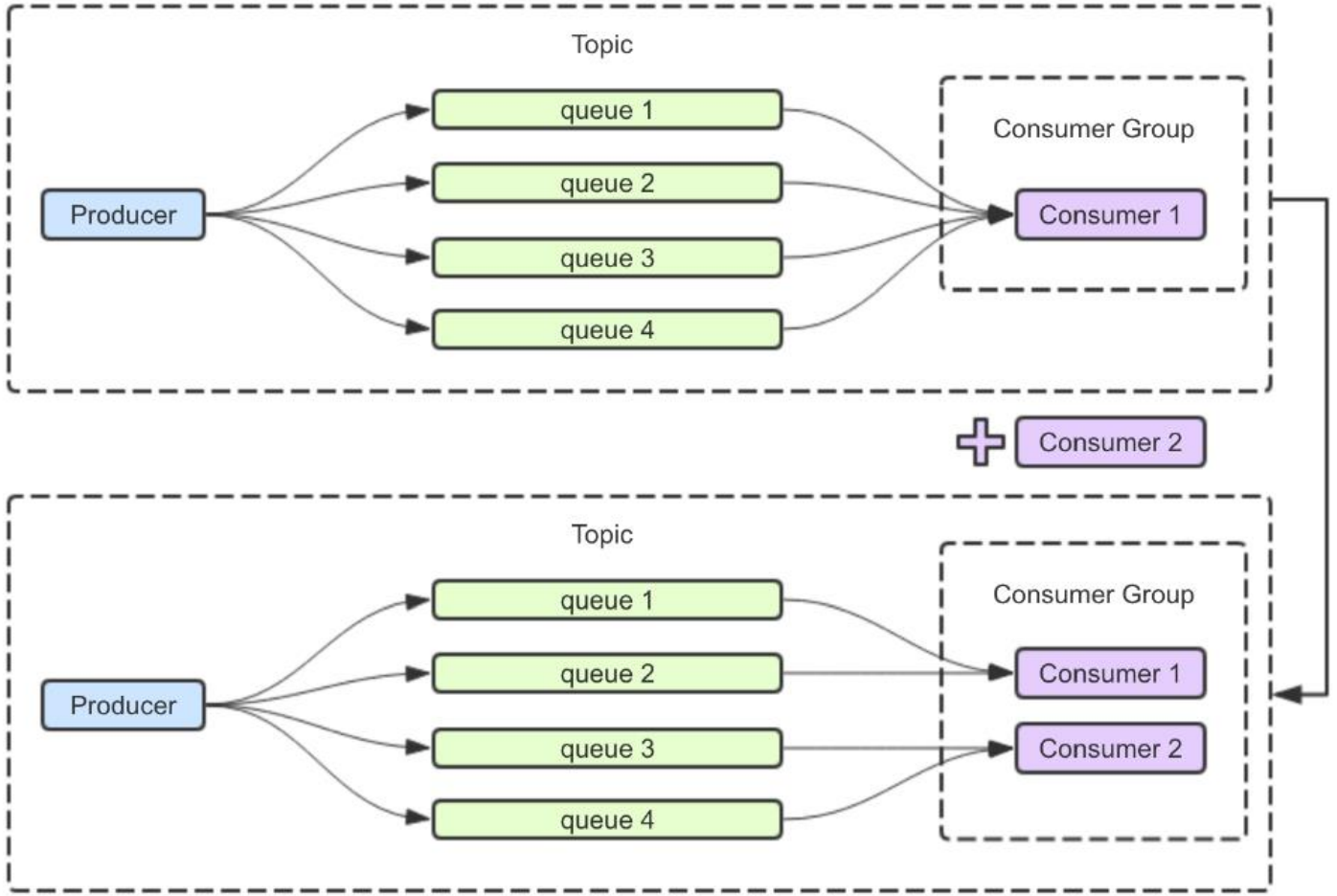


消息订阅 RocketMQ 消息消费

- 广播订阅模型
  - 同一个消费者组的消费者需要去拉取订阅主题下所有消息队列的消息
- 集群订阅模型
  - 同一个消费者组根据负载均衡机制均衡消费订阅主题下所有消息队列的消息

队列选择和负载均衡 RocketMQ 消息消费

- 负载均衡触发
  - 消费者数量变更
  - 消息队列数量变更
- 客户端负载
  - 同一个消费者组内看到的视图(cids, mqids)保持一致
  - Broker端保证一个队列对应一个消费者



消息确认和重试（Cluster） RocketMQ 消息消费

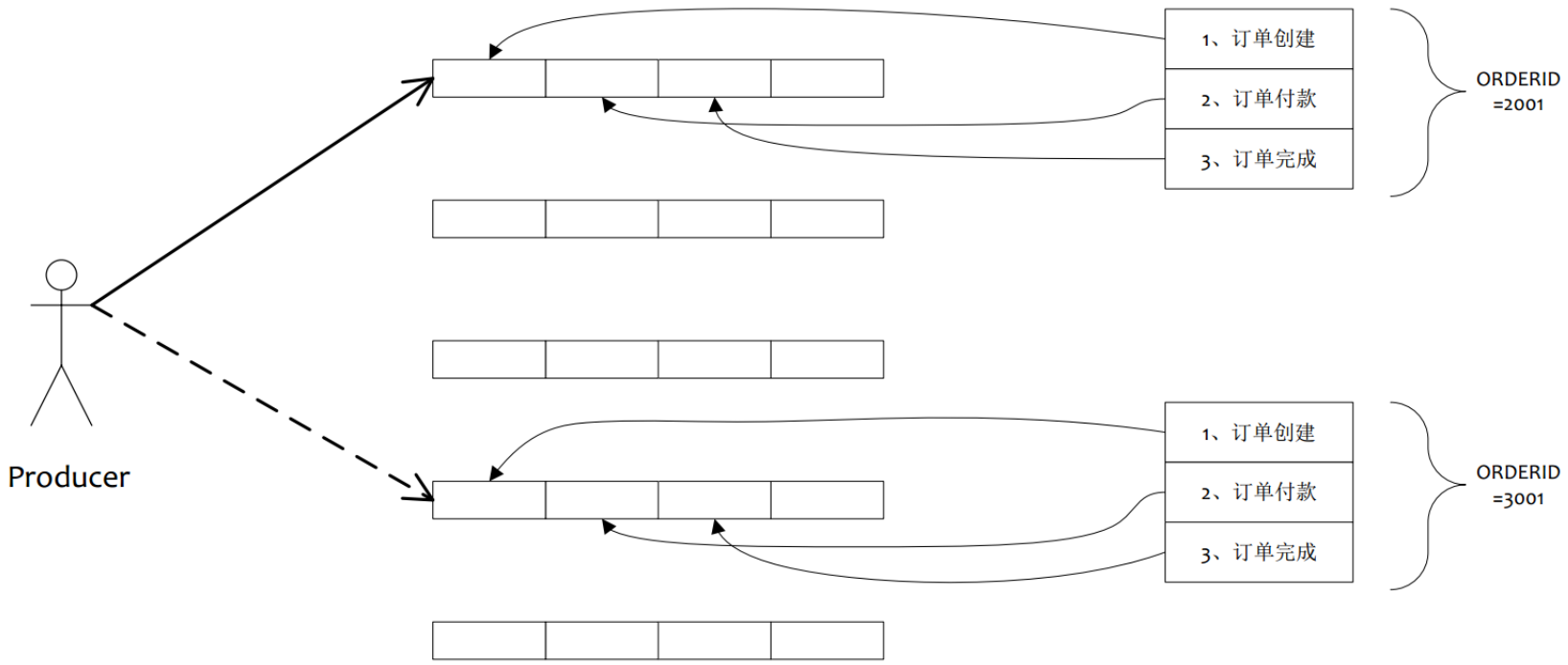
- 消费成功
  - 更新消费进度
  - 上报broker当前消费进度
- 消费失败
  - 上报broker延时消费，如10s/30s
  - 消息在broker端加入到消费组的重试队列中
  - 重试次数超越阈值，将消息加入死信队列dead-letter-queue

消费进度管理 RocketMQ 消息消费

- 消息进度根据消息订阅模式管理
  - 广播模式：同一个消费组的所有消息消费者需要消费主题下的所有消息，消费行为独立互不影响，存储到本地
  - 集群模式：同一个消费组的所有消息消费者共享消息主题下的所有消息，同一消息在同一时间只会被消费者组的一个消费者消息，且随消费队列、消费者的动态变化重新负载，所以消费进度保存在broker端
- 问题
  - 若先到来消息阻塞可能会导致队列进度卡住，可能会导致无法继续消费消息，所以消息消费一定要提供有超时机制

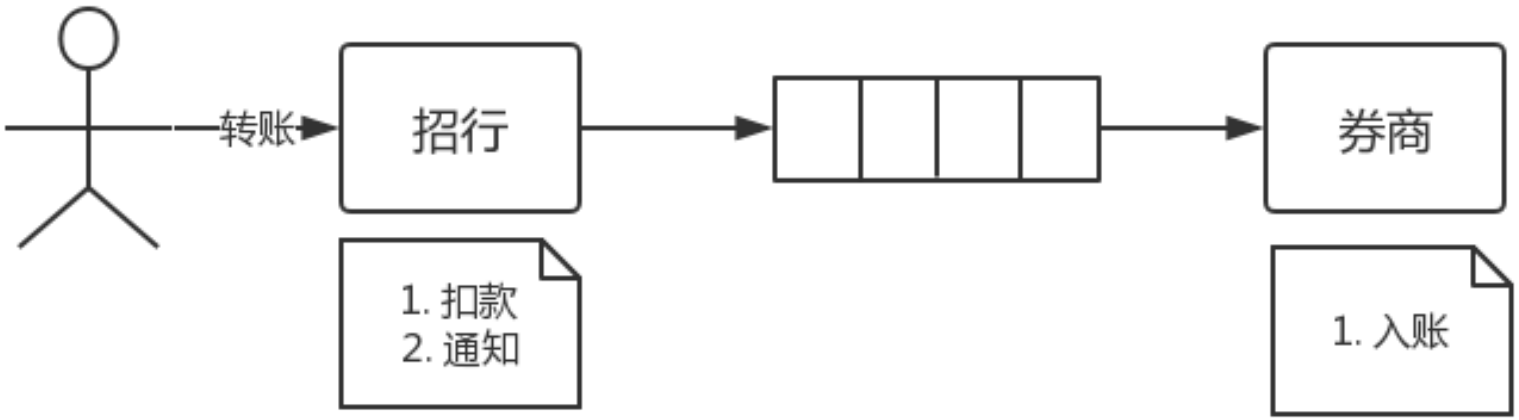
顺序消息 RocketMQ 消息消费

- 顺序消息意义
- 普通顺序消息
  - Hash取模选择队列
  - 宕机时短暂无序
- 严格顺序消息
  - 宕机时短暂不可用



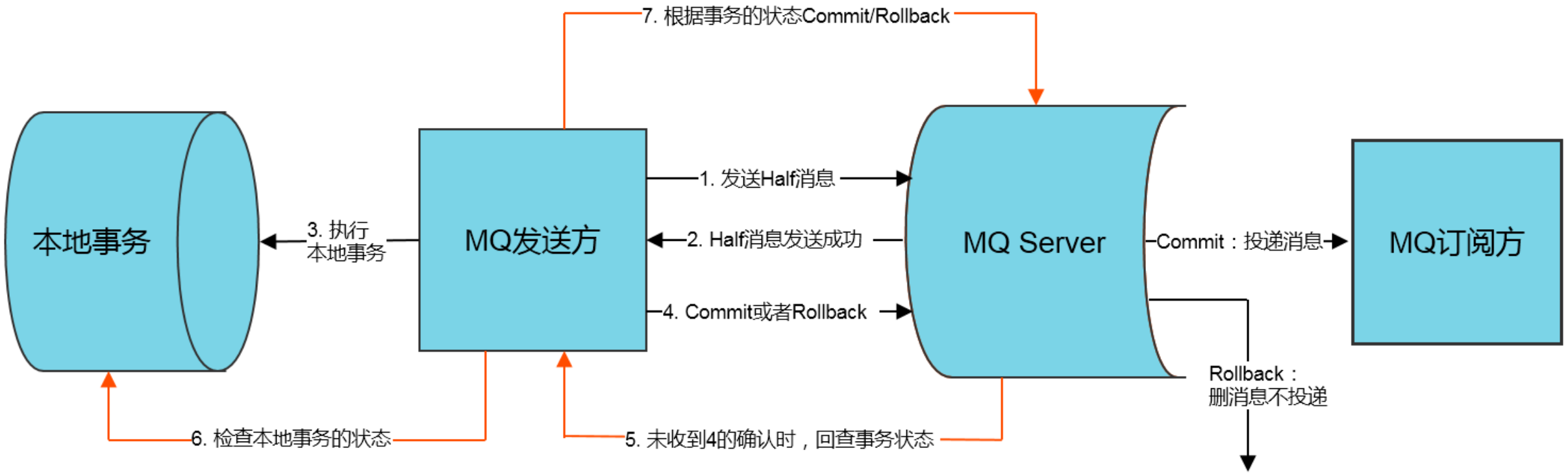
事务消息 RocketMQ 消息消费

- 分布式事务场景
  - 订单付款->扣减库存
  - 招行扣款->券商入账



事务消息 RocketMQ 消息消费

- 事务三种状态
- 半消息和回查机制



消费推拉模型 RocketMQ 消息消费

- 推模式
  - 消息延迟低，适用于低延迟需求的场景
  - 若消费者消费速度低，推送失败
- 拉模式
  - 按需消费，根据处理能力从broker拉取消息
  - 消息延迟，难点在于判断何时去拉取最新消息
- RocketMQ消费模型
  - Consumer 都是从 Broker 拉消息来消费，但是为了能做到实时收消息，RocketMQ 使用长轮询方式，可以保证消息实时性同Push方式一致

Master模式 RocketMQ 部署运维

- 单Master模式
  - 风险较大，一旦Broker重启或者宕机，会导致整个服务不可用，不建议线上环境使用，可以用于本地测试
- 多Master模式
  - 单个Master宕机或重启维护对应用无影响，在磁盘配置为RAID10时，即使机器宕机不可恢复情况下，由于RAID10磁盘非常可靠，消息也不会丢（异步刷盘丢失少量消息，同步刷盘一条不丢），性能最高

多Master多Slave异步复制 RocketMQ 部署运维

- 每个Master配置一个Slave，有多对Master-Slave，HA采用异步复制方式，主备有短暂消息延迟（毫秒级）
- Master宕机后，消费者仍然可以从Slave消费，而且此过程对应用透明，不需要人工干预
- Master磁盘损坏情况下会丢失少量消息

多Master多Slave同步双写 RocketMQ 部署运维

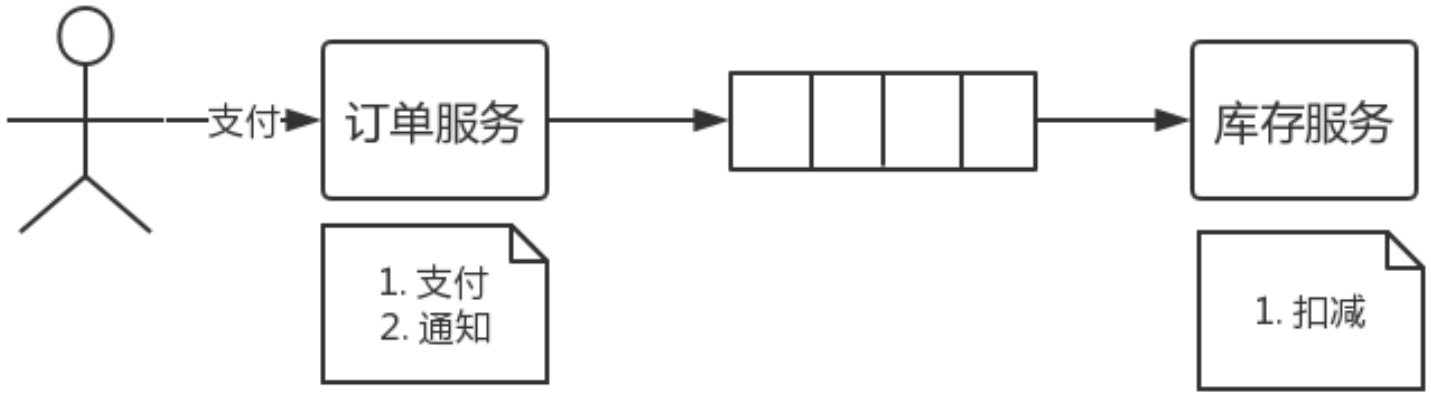
- 每个Master配置一个Slave，有多对Master-Slave，HA采用同步双写方式，即只有主备都写成功，才向应用返回成功
- 数据与服务都无单点故障，Master宕机情况下，消息无延迟，服务可用性与数据可用性都非常高
- 性能比异步复制模式略低（大约低10%左右），发送单个消息的RT会略高，且目前版本在主节点宕机后，备机不能自动切换为主机

RocketMQ 部署运维

- Admin控制台管理工具
- Web页面可视化管理工具

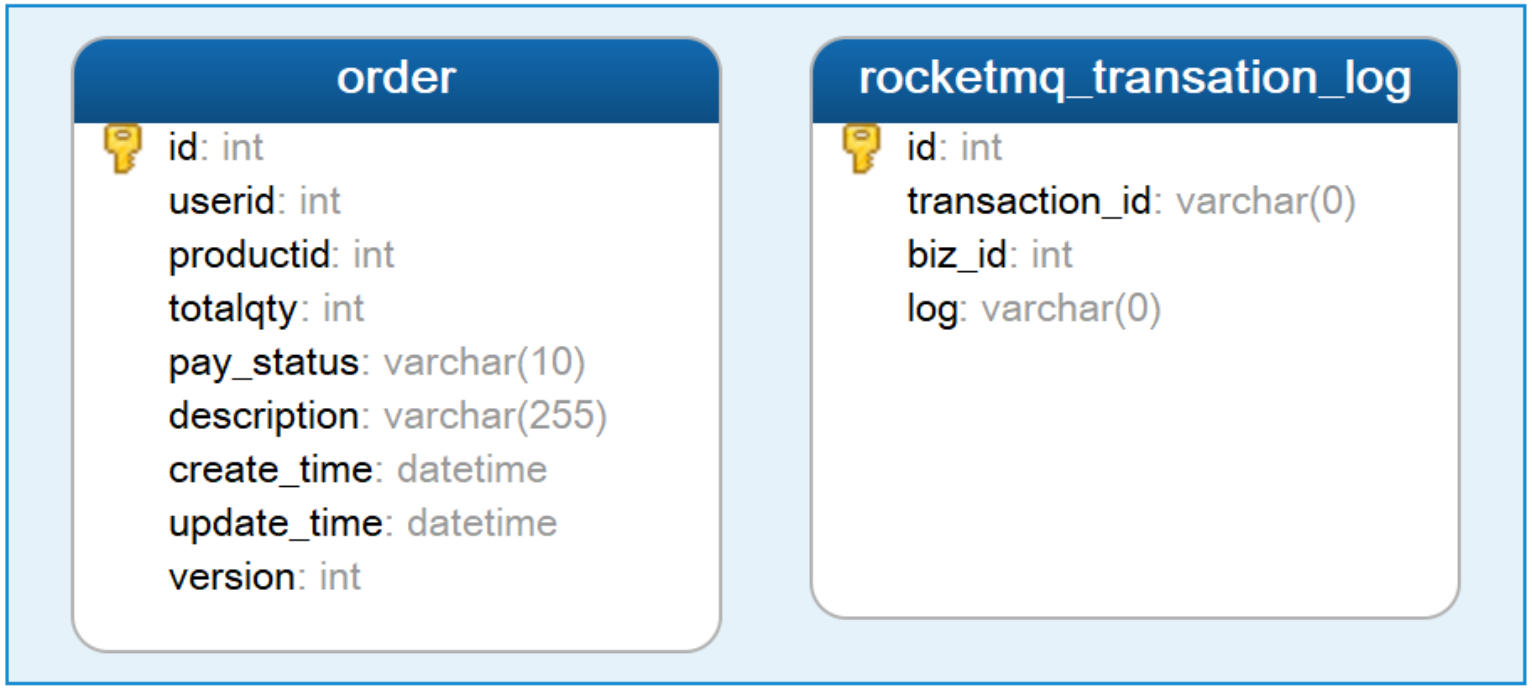
应用场景 RocketMQ 应用实战

- 简化的订单服务和库存服务场景



数据库表设计 RocketMQ 应用实战

- 订单服务
- 库存服务



RocketMQ 源码导读

- 项目代码结构
- 阅读源代码
  - 名称服务
  - 生产服务
  - 消费服务

RocketMQ 总结

- 消息中间件核心功能
  - 异步和解耦
- Apache RocketMQ功能
- Apache RocketMQ架构
  - 名称服务
  - 生产者
  - 消费者
- Apache RocketMQ部署

Q & A of Apache RocketMQ