JMQ 原理及实现

MQ(Message Queue):

是一种高效、可靠、安全、可扩展的分布式消息服务。它能够帮助应用开发者在他们应用的分布式组件上自由的传递数据，构建松耦合系统

MQ 包含几个关键要素

- **Producer**：消息的发送方；
- **Consumer**：消息的接收方；
- **Topic**：消息中间件里的数据分类的标示，一个 Topic 也就代表一类消息，producer 和 consumer 通过 Topic 建立关联。
- **Broker**：消息中间件服务端的一个实例；

Jmq 是京东自研的分布式消息中间件，相对于 **ActiveMQ,**性能功能都有提升

- ActiveMQ

Producer：

- 发送到单一 broker

Consumer：

- 采用 push 模型
- 对于消费失败的消息，放入死信队列（DLQ）

Broker：

- 使用了 KahaDB 存储引擎进行存储，BTree 索引
- 对于有多个订阅的消息，会在 broker 内复制多份（即：多个虚拟队列）

Archive：不支持

- JMQ

Producer：

- 随机发送/加权随机发送

Consumer：

- 采用 pull 模型
- 对于消费失败的消息，放入重试服务

Broker：

- 存储分为日志文件（journal）、消息队列文件（queue）及消费位置文件（offset），这三类文件都存储在 Broker 所在机器的本地磁盘上。消息只写一份，**同一个 broker 上的 topic 都写入一个 Journal 文件，queue 文件为每一个 topic 的索引文件（即：该 topic 的消息在 Journal 文件中的位置），offset 记录每个消费者的消费位置。**

重试：

- 异常消息入重试库
- 定期重试，broker 端重试间隔为 3^n 秒（n=当前重试次数-1）

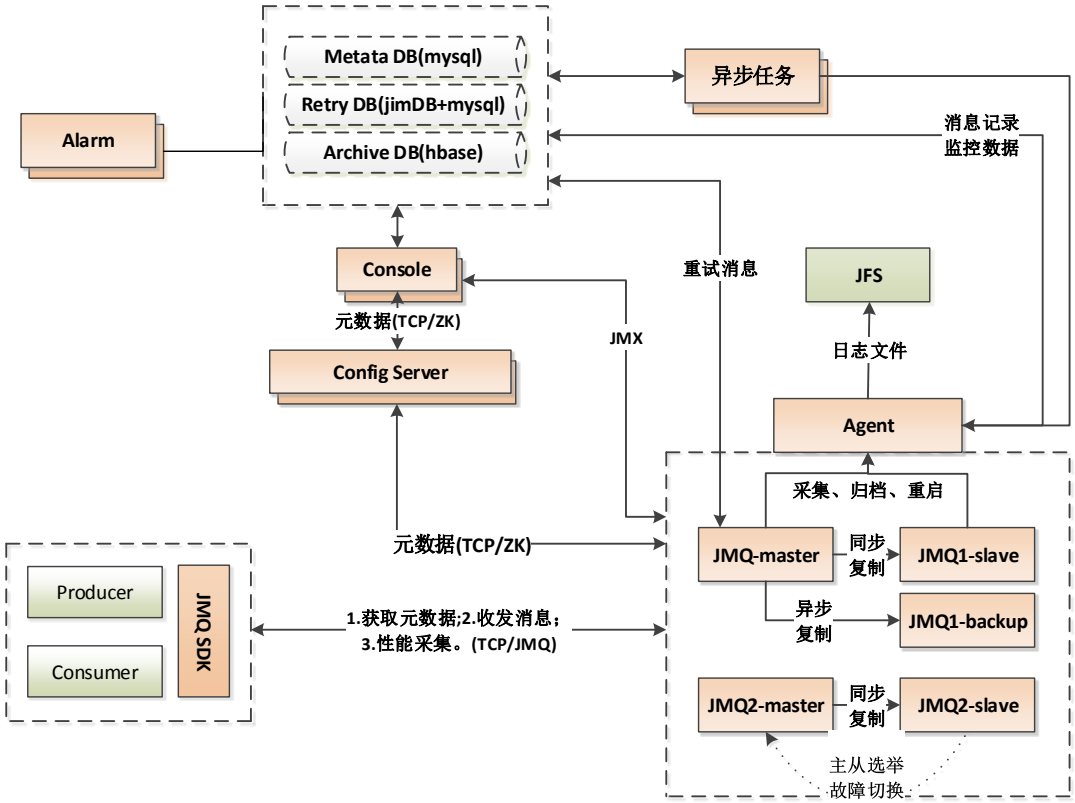
归档:

- 出入队成功消息记录日志
- Agent 采集日志进行归档，元信息(消息 id、发送时间、接收时间等)HBase，消息内容写入 JFS

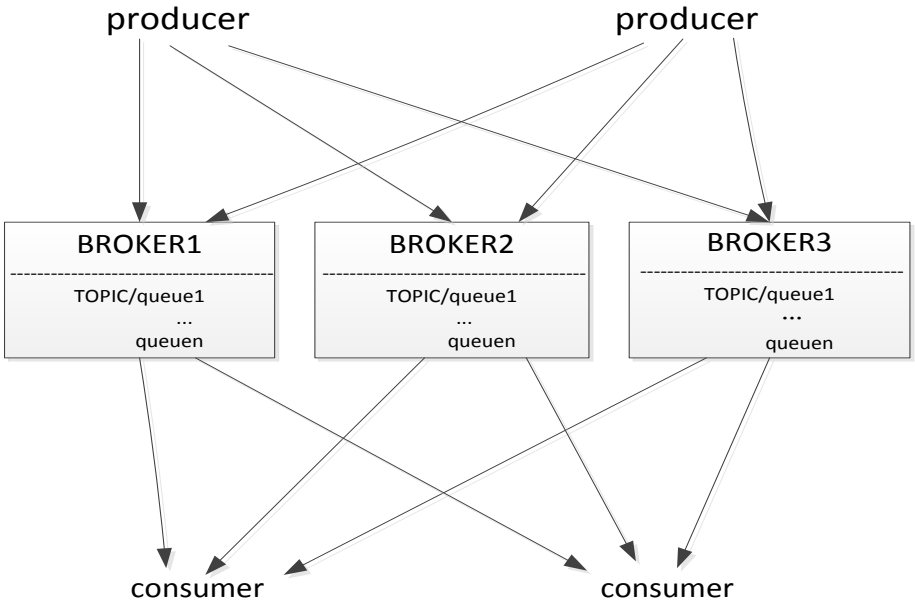
注册中心:

- 元数据（topic 与 broker 对应关系）存储至 MYSQL 和 zk，zk 定期同步到每个 Broker

JMQ 系统结构

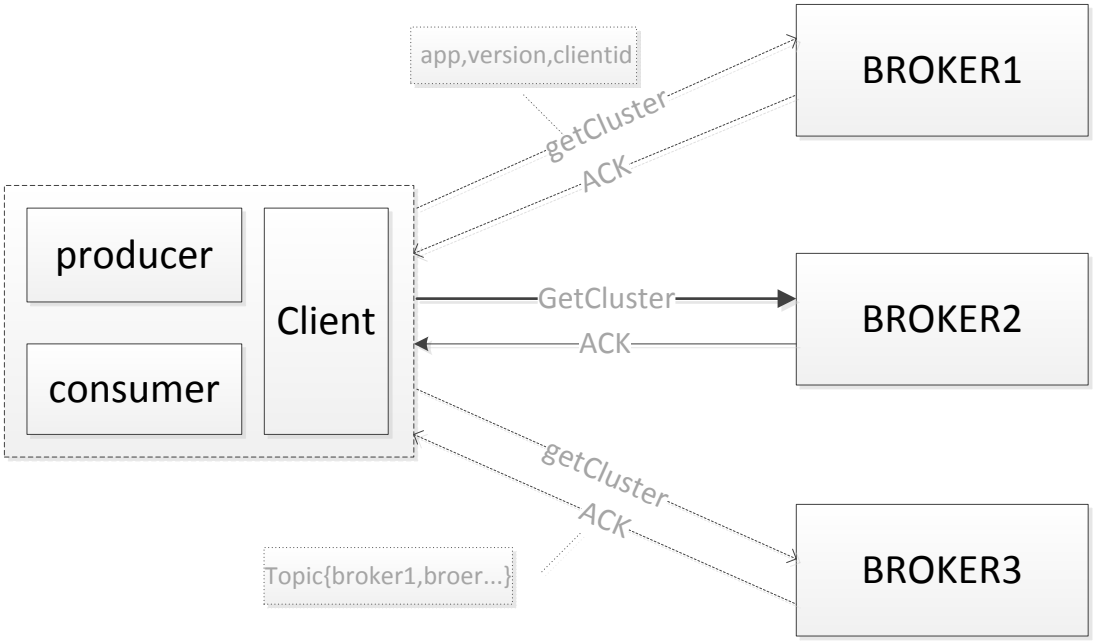


JMQ 交互模型



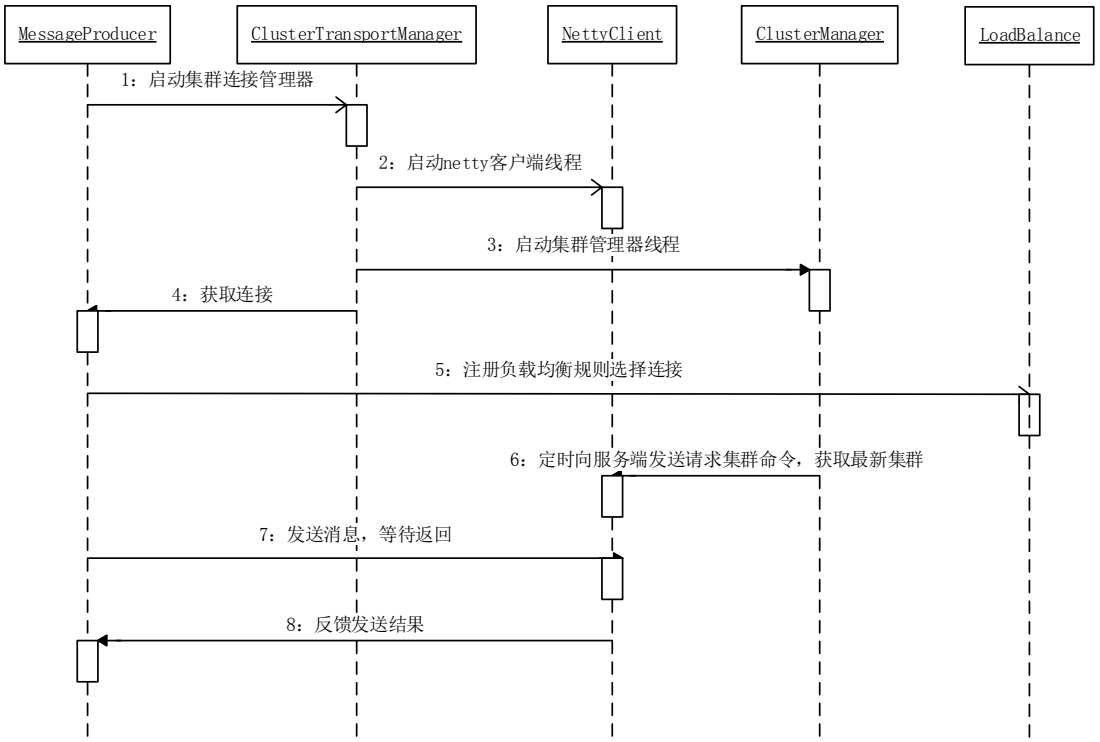
从图中可以看出，一个应用实例的每个消息队列都会创建到所有 broker 的链接，如果一个系统生产 100 个消息队列，每个消息队列的 broker 为 5，这个系统的每个应用实例都是有 100*5=500 个 jmq 的生产链接，底层使用的 netty，所以 jstack 查看应用实例的 java 线程中 ChannelEvent 的线程数很多

JMQ 客户端



Jmq 提供客户端，负责和 jmq 服务端交互，客户端支持 producer 和 cunsumer，客户端再启动是根据 app、version、clientId 等信息尝试链接服务端，服务端接受请求后给客户端 ack 响应，长链接生效

JMQ 客户端-发送



时序图如图所示，发送几个特性

随机：

对于非顺序消息，则选择任意 broker 上的任意 queue 发送；对于顺序消息（Message 对象中 ordered 设为 true），则根据 businessId 的 hashCode 取模，保证同一个 businessId 的数据发送到同一个 broker 上的同一个 queue，从而实现消息基本有序。

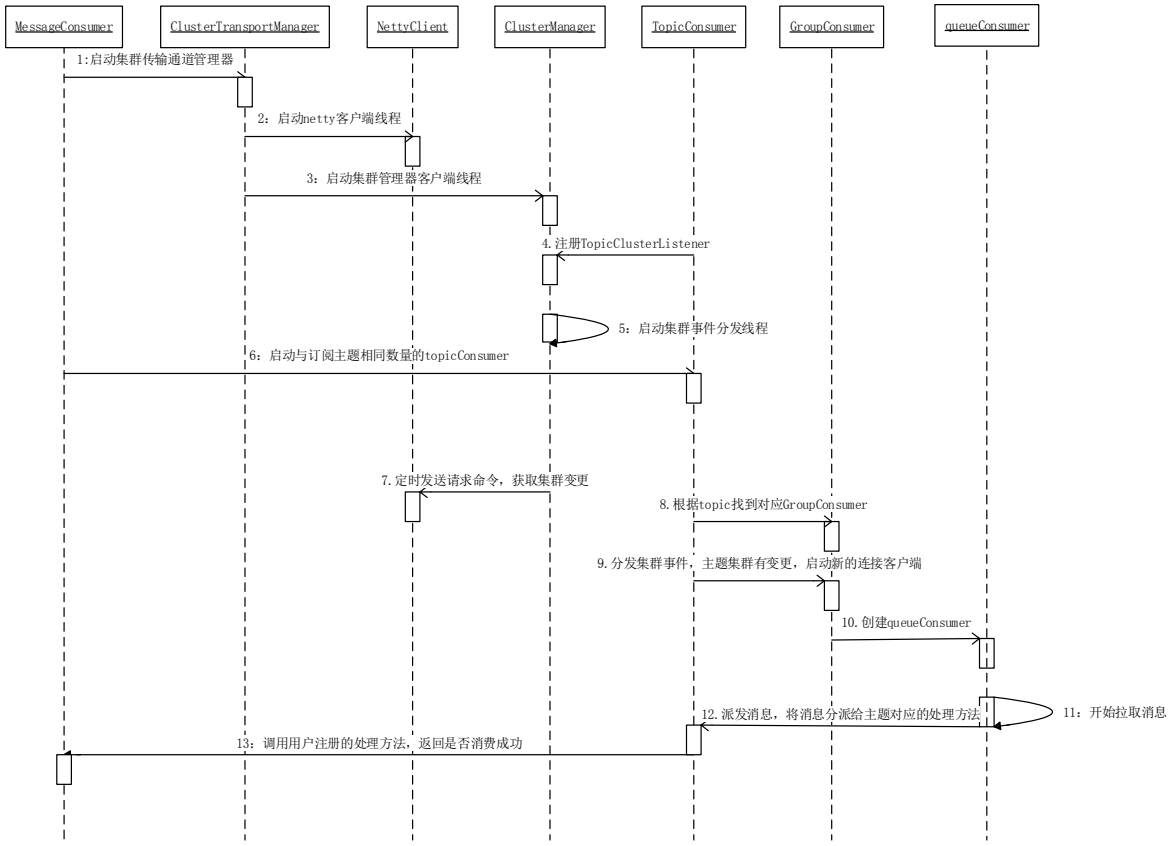
压缩：

消息体超过 100 字节时会被压缩，压缩后消息超过 4M 将会发送失败。

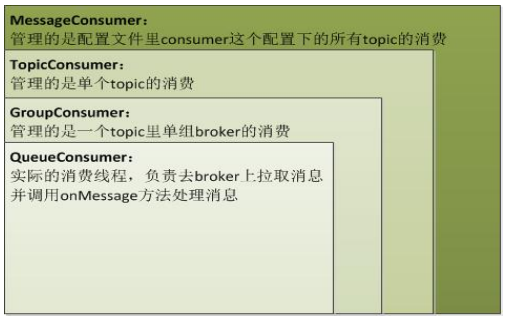
同步：

客户端重试三次（不同 broker），三次后仍失败的情况，抛出异常。消息发送超时时长默认为 5000ms。

JMQ 客户端-消费

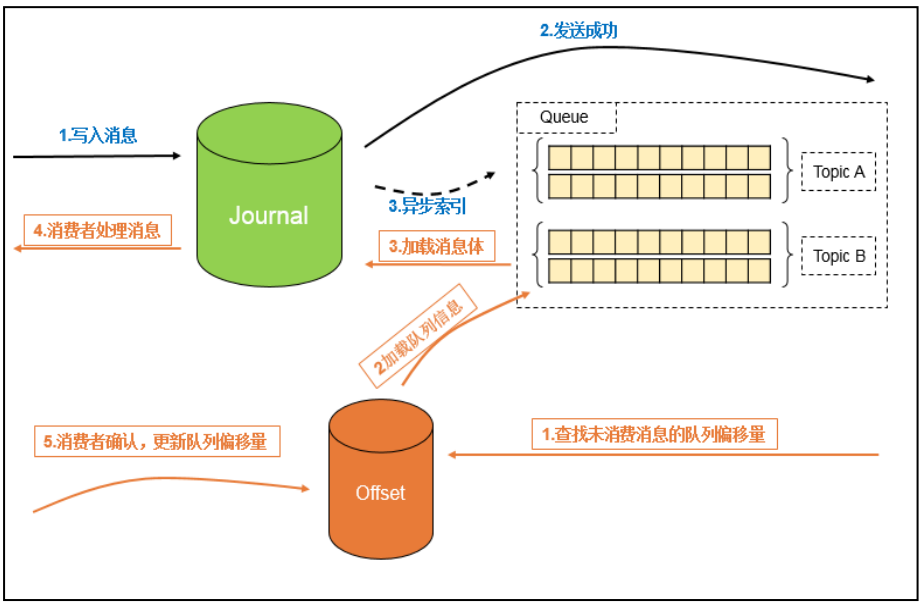


说明：消费采用的 pull 拉取模式，客户端不停的向服务端发送请求，客户端从服务器获取批量数据，处理过程中出现异常，会本地重试这批数据，重试失败才抛出异常返回给服务端，所以如果重试比例过高，会严重影响整个 topic 的消费能力

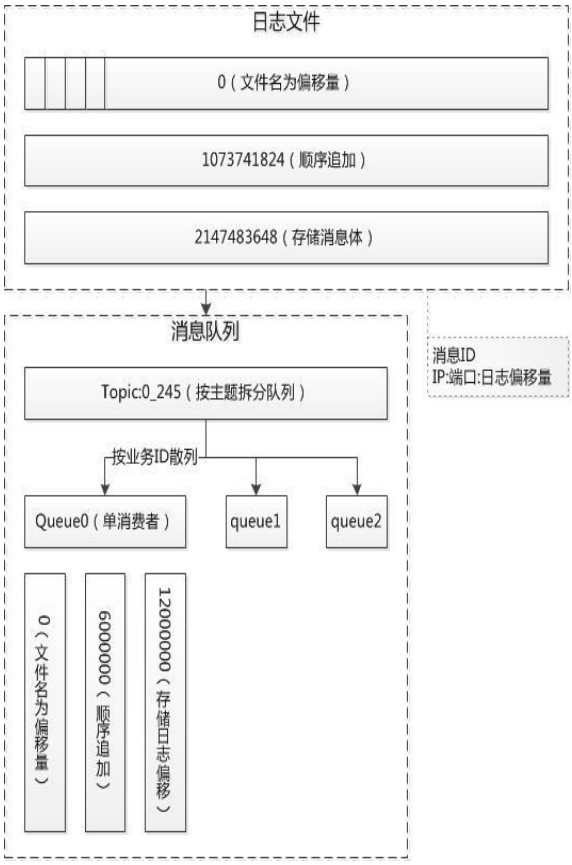


JMQ 服务端

上面有提到过 broker 存储分为日志文件（journal）、消息队列文件（queue）及消费位置文件（offset），这三类文件都存储在 Broker 所在机器的本地磁盘上。消息只写一份，**同一个 broker 上的 topic 都写入一个 Journal 文件，queue 文件为每一个 topic 的索引文件（即：该 topic 的消息在 Journal 文件中的位置），offset 记录每个消费者的消费位置**



日志文件格式及日志、queue 格式



Queue（定长）

8 字节的日志偏移量

4 字节的消息总长度

2 字节标识位（未使用）

8 字节索引 CRC

Journal（定长+变长）

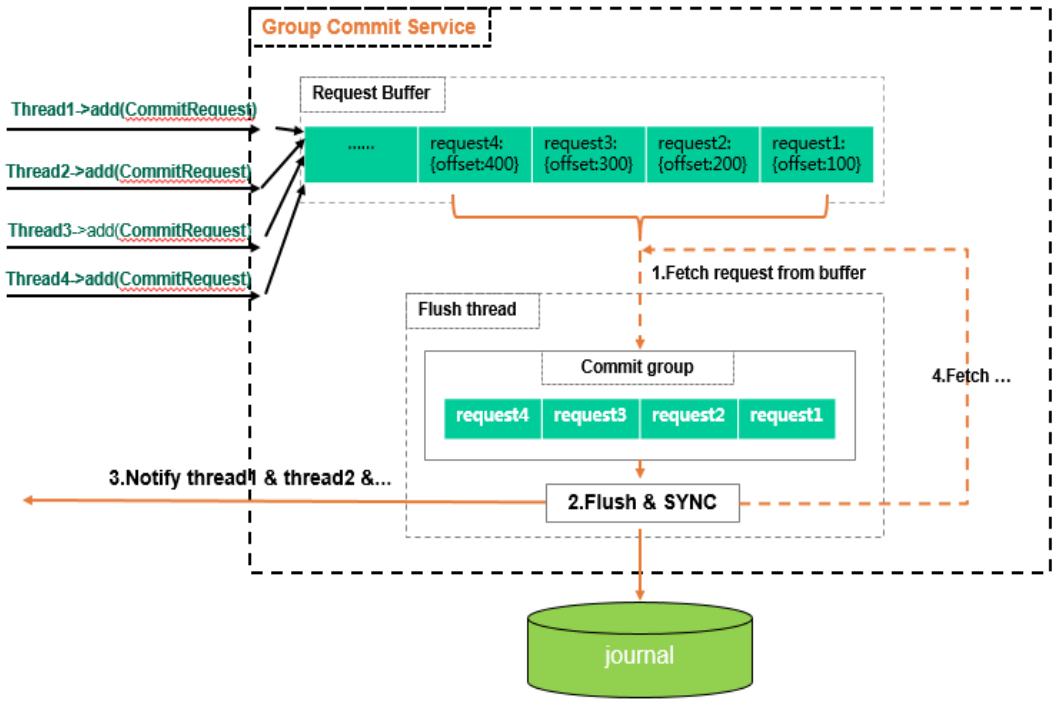
定长部分：

- 4 字节的消息总长度（需要计算出来）
- 2 字节的魔法标识
- 1 字节的系统字段 1-1：压缩标识 2-2：顺序消息 3-8：其他,预留未用
- 2 字节业务标签
- 1 字节优先级
- 8 字节日志偏移
- 1 字节队列
- 8 字节队列偏移
- 6 字节的客户端地址
- 6 字节的服务端地址
- 8 字节发送时间
- 4 字节接受时间（相对发送时间的偏移）
- 4 字节存储时间（相对发送时间偏移）
- 8 字节消息体 CRC

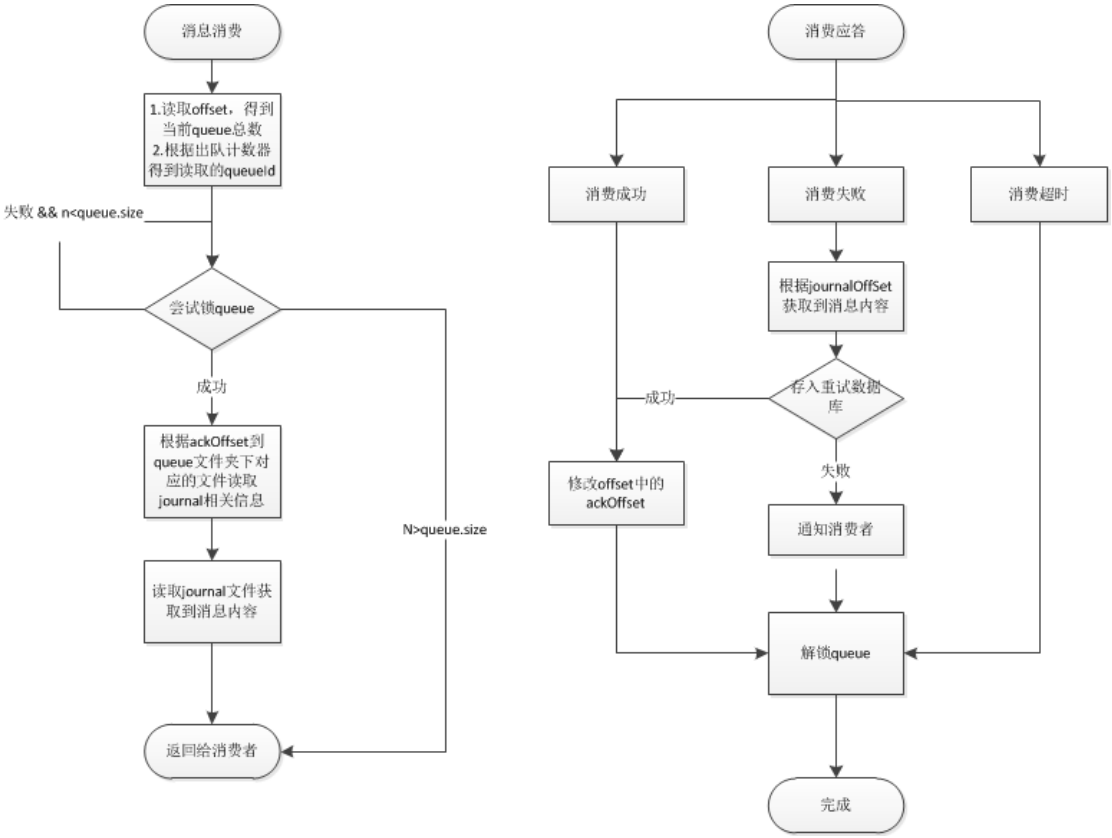
变长部分：

- 4 字节消息体大小 + 消息体（字节数组）
- 1 字节主题长度 + 主题（字节数组）
- 1 字节应用长度 + 应用（字节数组）
- 1 字节业务 ID 长度 + 业务 ID（字节数组）
- 2 字节属性长度 + 属性（字节数组）

JMQ 服务端-发送



JMQ 服务端-消费



JMQ 特点

高性能

- 批量发送和接收
- 轻量级存储模型，减少序列化，组提交，快速索引，积压不影响写入
- 内存镜像文件，减少内存拷贝操作
- 自定义消息序列化，默认开启压缩

监控&配置

- 灵活配置消费、生产策略
- 丰富的监控图表

丰富特性

- 消息回放
- 并行消费

  一个 queue 允许并行处理数据量，根据上面的介绍已经了解到消费时按 queue 锁定，最大的并行数也就是 broker*queue，并行消费是每个 queue 可以并行处理的最大数据是多少，消费的批量数设置是 10，如果是把并行消费开启，并行数量 20，则每个线程抓取数据量为 10，每个 queue 可并行的线程为 20/10=2，消费能力也会提高
- 严格顺序消息