引言

我们继续来了解服务通信的异步方式——消息队列（Message Queue,MQ）。
这种方式通常应用于异步处理、流量削峰\缓冲、数据广播、事件分发、最终一致性保障等场景，追求高性能、应用解耦。
所以一般设计时考虑高可靠性、高可用性、高性能、可扩展性、消息有序性。还有可运维、安全性与其他丰富功能。

先不了解那么多，仅了解一些消息队列的共同基础。

MQ基础

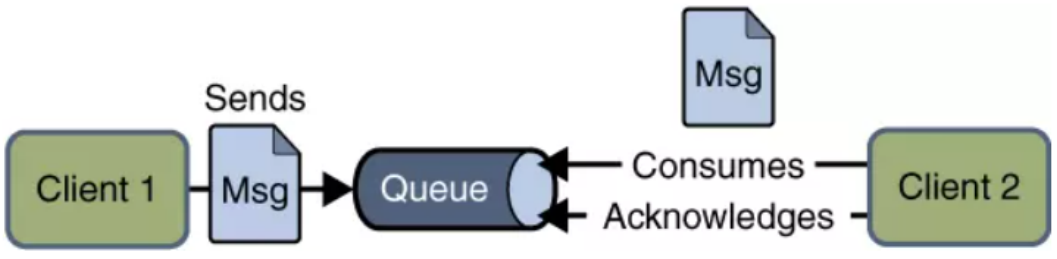
消息队列模型

消息队列（Message Queue, MQ）作为异步通信的核心组件，其最基本和最经典的使用模型就是点对点（Point-to-Point, P2P）模型和发布/订阅（Publish/Subscribe, Pub/Sub）模型。

点对点模型（Point-to-Point,P2P）模型

消息生产者（Producer）发送消息到一个特定的队列（Queue），消息消费者（Consumer）从这个队列中拉取或接收消息。
一条消息对应一个消费者，消费者成功消费则消息从队列中移除（或标记已处理）， 未被消费消息在队列中保留直到被消费或超时。
可以有多个消费者监听同一个队列。当消息到达时，队列通常会将消息分发给其中一个活跃的消费者（具体分发策略取决于MQ实现，如轮询、公平分发等）。即消息队列的负载均衡。

应用场景：任务队列（如发送邮件、生成报表、短信通知等后台耗时任务）、确保消息被处理一次的场景。

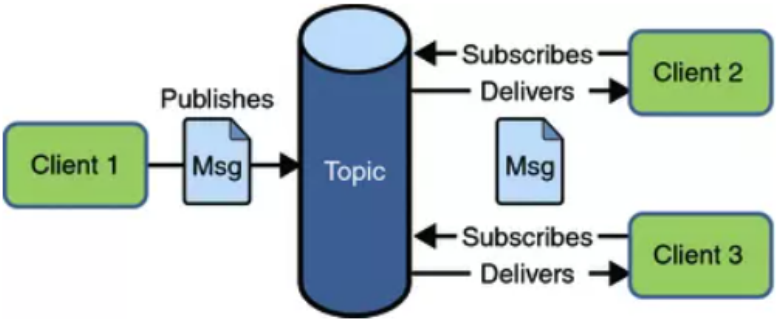


点到点（P2P）模型

发布/订阅模型(Publish/Subscribe)

消息生产者（Publisher）将消息发布到一个主题（Topic）或交换机（Exchange），所有订阅了该主题的消费者（Subscriber）都会收到这条消息的副本。
一对多关系，一条消息可以被多个订阅者接收和处理。
主题作为中介，发布者将消息发送到主题，订阅者从主题订阅消息。发布者和订阅者互相不知道对方的存在。
更适合事件驱动架构，当事件发生时，所有关心这个事件的系统都可以独立做出响应。

应用场景：事件通知（如用户注册成功后，通知邮件服务、积分服务、营销服务等）、实时数据分发、日志聚合等。



发布/订阅（Pub/Sub）模型

基础概念

消息

Q：消息是什么?
A：消息队列的数据单元。生产者想要发送给消费者的业务数据+控制消息行为的元数据。

消息 = 消息体 + 消息头

消息队列系统通常不需要关心消息的数据结构，但生产者和消费者需要约定消息体的格式。
常见消息体格式有：

- **JSON (JavaScript Object Notation)** : 轻量级的数据交换格式，易于人阅读和编写，也易于机器解析和生成。非常流行。
- **XML (Extensible Markup Language)**: 另一种标记语言，结构化，但相对JSON更冗长。
- **Protocol Buffers (Protobuf)**: Google 开发的一种语言无关、平台无关、可扩展的序列化结构数据的方法，通常用于RPC系统和持久存储。性能高，体积小。
- **Apache Avro**: 类似于Protobuf，也是一种数据序列化系统，特别适合于大数据处理场景（如Hadoop生态）。
- **MessagePack**: 一种高效的二进制序列化格式，像JSON一样，但更快更小。
- **纯文本 (Plain Text)**: 简单的字符串。
- **自定义二进制格式**: 应用自定义的二进制编码。
  消息头一般是消息的元数据，用于描述消息或控制消息的行为，而不是业务数据本身。通常以键值对形式存在。

生产者在发送前将业务对象序列化成上述某种格式的字节流；消费者在接收后需要将字节流反序列化成业务对象。

消息代理（Broker）的角色

不管模型是P2P还是Pub/Sub，通常都有一个中间件，即消息代理（Message Broker），它负责接受、存储和转发消息。

消息持久化

为了防止消息丢失，MQ通常支持消息持久化。确保MQ服务器发生故障或重启消息也不会丢失。
在MQ设计时，一般根据设计目标（吞吐量、延迟、可靠性、功能特性）选择不同的数据模型和底层存储技术。
常见的有以下方案：

- **顺序追加日志 (Append-only Log / Commit Log)**
  这是高性能MQ的主流模型（直接使用文件系统）。所有消息（通常不区分Topic/Queue）被顺序写入一个或多个大的日志文件中。每个消息都有一个在此日志中的唯一偏移量（Offset）。

Kafka：消息写入分区（Partition）的Segment Log文件中。消费者通过Offset顺序读取。
RocketMQ：消息写入CommitLog文件。然后为每个逻辑队列（Message Queue，是Topic下的一个读写单元）创建ConsumeQueue文件，ConsumeQueue中存储的是指向CommitLog中消息的物理偏移量、消息大小和Tag的Hash值，相当于消息的索引。

日志的清理通常通过日志分段（Log Segmentation）和定期删除旧的、已消费的日志段，或者通过日志压缩（Log Compaction）策略来实现。

- **关系型数据库|NoSQL数据库**
  能利用事务特性或利用高读写性能，但是并不主流，吞吐量低性能低或一致性不能满足MQ。

消息确认机制

消息队列有 At-Least-Once Delivery至少一次送达语义。这意味着消息队列会确保消息至少被消费者处理一次。“确认消费”（Acknowledgement，简称ack）机制就是实现这一点的关键。消费者在成功处理消息后发送ack，消息队列收到ack后才会将消息标记为已处理（比如从队列中删除或移到别处）。

- **自动确认模式（Auto-ACK）**
  在这种模式下，消息队列在将消息发送给消费者后立即或很快就认为消息已被消费，而不管消费者是否真的处理成功。
  这种模式简单，但**丢失消息的风险很高**（比如消费者收到消息后、处理完成前就崩溃了）。
  对于像订单、支付这样重要的场景，**绝对不推荐**使用自动确认。
- **手动确认模式（Manual-ACK）**
  标准的消息队列机制，它要求**应用层（消费者）在成功处理完业务逻辑后，显式地调用API向消息队列发送确认回执。**
  这**不是**应用层在消息队列之外“再行设计一个”机制，而是**正确使用**消息队列提供的可靠消息机制。
  正是通过这种应用层控制的确认，才能确保消息在被业务逻辑成功处理后才被标记为消费，从而防止消息丢失。
  如果处理失败，应用层可以选择不发送ack或者发送nack（negative acknowledgement），让消息队列重新投递或放入死信队列。