

MQTT与RMQ和KAFKA对比

MQTT（Message Queuing Telemetry Transport）是一种基于发布/订阅（Pub/Sub）模式的轻量级通讯协议，专为资源受限的物联网（IoT）设备和低带宽、高延迟、不稳定的网络环境设计。

相对于 RabbitMQ 和 Kafka，我们直接从它们的核心差异切入：**Kafka 是为了处理海量数据的“吞吐”和“存储”设计的，RabbitMQ 是为了企业级系统间的“解耦”和“复杂路由”设计的，而 MQTT 是为了“百万级海量边缘连接”和“极弱网络环境”设计的。**

一、核心概念（与 Kafka/RabbitMQ 的映射）

- Broker（代理服务端）**：类似 Kafka 的 Broker 或 RabbitMQ 的 Server。它负责接收所有消息，过滤并分发给订阅者。常见的 MQTT Broker 有 EMQX、Mosquitto、HiveMQ 等。
- Topic（主题）**：与 Kafka 预先创建的强约束 Topic 或 RabbitMQ 复杂的 Exchange/Binding 不同，**MQTT 的 Topic 是极其轻量且动态的层级字符串**。例如 `grid/substationA/sensor/temp`。你不需要提前在 Broker 上创建它，客户端发布或订阅时自动生效，并支持通配符订阅（`+` 匹配单层，`#` 匹配多层）。
- 发布/订阅机制**：MQTT 默认**没有传统意义上的“队列（Queue）”缓冲**。如果一个 Topic 没有订阅者，消息会被直接丢弃（除非使用了保留消息机制）。它强调的是即时路由，而不是持久化堆积。

二、MQTT 的四大“杀手级”特性（为物联网而生）

这些特性是 Kafka 和 RabbitMQ 难以在设备端直接替代 MQTT 的原因：

1. QoS（服务质量）等级 针对不稳定的移动网络（如 4G/NB-IoT），MQTT 在协议层内置了三种消息投递保证：

- QoS 0（最多一次）**：发完即忘，消息可能丢失。适用于高频且允许丢失的传感器数据（如每秒上报的温度）。
- QoS 1（至少一次）**：确保消息到达，但可能重复。接收端需返回 PUBACK。适用于重要告警。
- QoS 2（只有一次）**：确保消息到达且仅到达一次，开销最高（四步握手）。适用于计费、开关控制等严苛场景。

2. Retained Message（保留消息） 如果 Broker 收到一条标志为 Retained 的消息，它会**记住该 Topic 下的最后一条消息**。当有新的订阅者上线并订阅该 Topic 时，会立刻收到这条消息。这对于随时上线的设备获取“最新系统配置”或“最后已知状态”非常有用。

3. LWT（最后遗嘱 - Last Will and Testament） 设备在建立连接时，可以向 Broker 登记一份“遗嘱”（指定一个 Topic 和一段 Payload）。如果设备因为断网、没电等原因**非正常断开连接**，Broker 会自动将这份“遗嘱”发布出去。这是监控海量设备在线/离线状态的核心机制。

4. 极低的协议开销 MQTT 运行在 TCP/IP 之上，它的协议头部最小只有 **2 个字节**，相比于 HTTP 或 Kafka 沉重的客户端 SDK，MQTT 客户端可以在几百 KB 内存的单片机（MCU）上流畅运行。

三、MQTT vs RabbitMQ vs Kafka 选型对比

对比维度	MQTT	RabbitMQ	Kafka
主要定位	物联网设备接入、端到云通信	企业级微服务解耦、任务队列	大数据管道、流式日志处理
客户端/网络开销	极低（几KB代码，2字节报文头）	中等（AMQP协议较重）	高（客户端重，需拉取元数据）
并发连接数能力	千万级（如 EMQX 单集群）	万级 - 十万级	万级（按 Partition 连接）
消息吞吐量	中高（十万至百万级 TPS）	中（万级至十万级 TPS）	极高（千万级 TPS，零拷贝）
持久化与堆积	弱（依靠会话状态，非核心功能）	中（内存/磁盘队列堆积）	极强（基于磁盘的 Append Log）
路由灵活性	树状层级 Topic，支持通配符过滤	极强（Direct/Topic/Fanout）	弱（按 Topic 严格分区拉取）

四、 典型架构组合

在大型物联网平台中，通常不会只用一种中间件，而是**组合使用**：

1. **接入层（边缘到云）**： 使用 **MQTT Broker**（如 **EMQX**） 承接十万/百万台智能电表的长连接并发，处理心跳保活、遗嘱解析和 QoS 确保。
2. **缓冲层（云端内部）**： MQTT Broker 通过规则引擎，将海量设备数据桥接（Bridge）或转发推送到 **Kafka**。
3. **消费层（业务处理）**： 业务微服务或 Flink 集群从 **Kafka** 订阅数据进行时序存储、流计算和业务落盘。对于一些需要强一致性且非海量的工单派发，可能会引入 **RabbitMQ**。