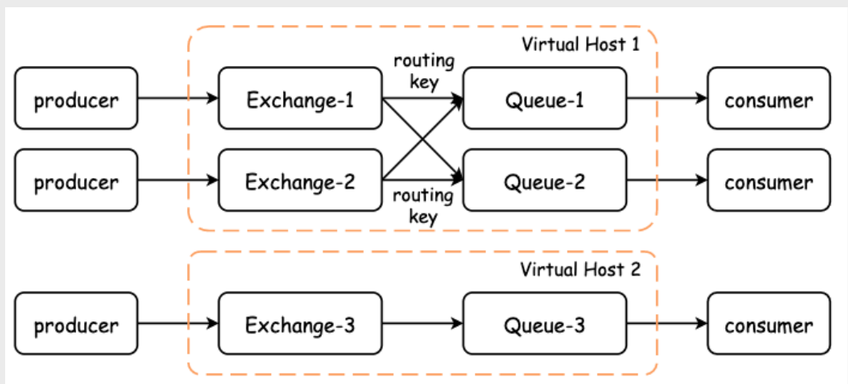
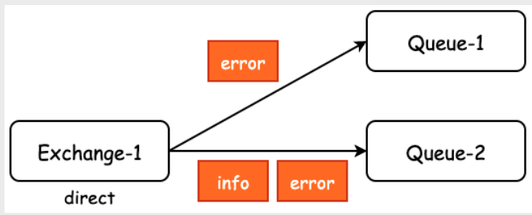
RabbitMQ

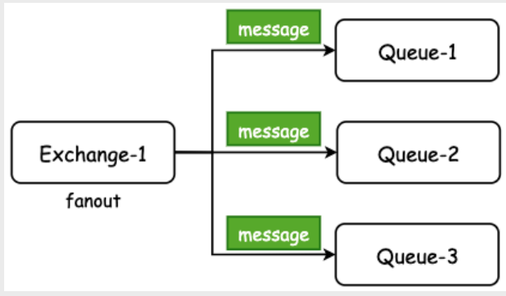
基本架构



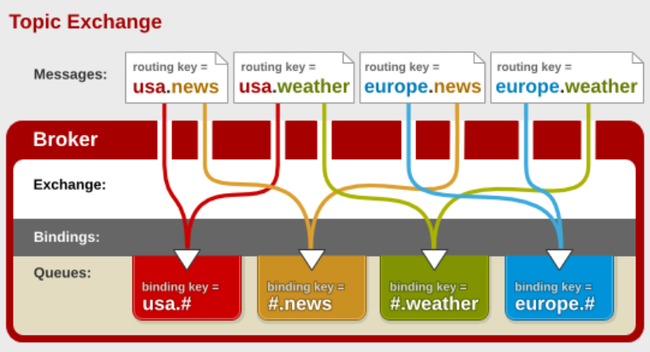
- 基本组件
  - Virtual Host — 其实就是一个字符串，默认的 virtual host 为 "/"，作用就是逻辑隔离，如 Redis 中的 db-0、db-1 一样
  - Exchange — 即交换机，根据 Routing Key 将消息投递到不同的队列中去，RabbitMQ 中 一共有 3 种 Exchange 类型
  - Queue — 消息的最终存储队列，消费者从 Queue 中取出数据并消费。每一个 Queue 都有自己的属性，例如队列的过期时间，是否持久化，以及队列的最大长度等等，其配置非常多样且灵活
- Exchange
  - Exchange 可以看作是对 Queue 的一个前置抽象，我们可以根据设置不同的 Exchange 类型来决定如何处理一条消息。比如只把该消息发送给一个队列，或者使用类似广播的机制将消息发送给与该交换机绑定的所有队列
  - 类型
    - ① direct
      - **direct 类型的 Exchange 会严格按照 Routing Key 进行精确匹配地投递**



        - **Exchange 和 Queue 之间可以使用多个 Routing Key 进行关联**
        - 当 Routing Key 为 "error" 时，消息将会投递到 Queue-1 以及 Queue-2 中，因为这两个队列的 Binding Key 与 Routing Key 完全匹配
        - 当 Routing Key 为 "info" 时，消息仅会投递到 Queue-1 队列中
    - ② fanout
      - **fanout 类型的 Exchange 提供广播功能，即将消息投递到所有与该 Exchange 所绑定的 Queue 上，此时所指定 Routing Key 将会被忽略**



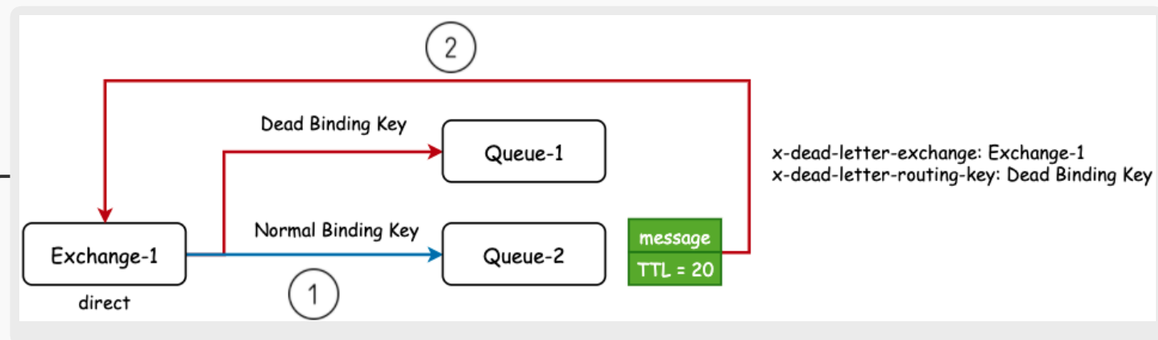
        - **Exchange 和 Queue 之间并不会使用 Routing Key 进行关联**
        - 在广播业务中我们就会使用 fanout 类型的交换机，使得每一个消费者都会有一个属于自己的 Message Queue
    - ③ topic
      - **topic 类型的 Exchange 将提供通配符匹配的功能，将消息投递至与 Routing Key 所匹配的 Queue 中**



        - **Exchange 和 Queue 之间使用带有通配符的 Binding Key 进行关联**
        - "#" 匹配一个或多个单词，"*" 则匹配不多不少一个单词
        - 当消息的 Routing Key 为 "usa.wheather" 时，既能匹配到 "usa.#" 又可以匹配到 "#.weather"，因此消息将会投递到这两个队列中
        - 使用方式更为灵活，消费者可以根据自己感兴趣的 Queue 进行"主题消费"
  - 属性
    - Durability — 是否持久化该 Exchange，绝大部分情况都需要将 Exchange 持久化，保证 RabbitMQ 在异常重启后不会导致 Exchange 的丢失
    - Auto delete — 表示是否自动删除，当某一个 Exchange 所关联的 Queue 均与其解绑时，该 Exchange 将会被自动删除
- Queue
  - 相较于"花里胡哨"的 Exchange，Queue 就朴实无华的多了，通常会通过 Queue 的属性来其进行设置
  - 属性
    - x-message-ttl — 指定队列中所有消息的存活时间
    - x-max-length — 指定当前队列所能容纳的最大消息数量
    - **x-dead-letter-exchange** — 用于指定当队列中有 "dead letter" 产生时，将消息投递至哪个 Exchange
    - **x-dead-letter-routing-key** — 与 x-dead-letter-exchange 相配合，指定死信队列的 Routing Key
    - 死信队列严格意义上来说其实是一个 Exchange、Binding Key 以及 Queue 所组成的整体，我们用这个整体来保存一些"特殊"的消息

死信队列

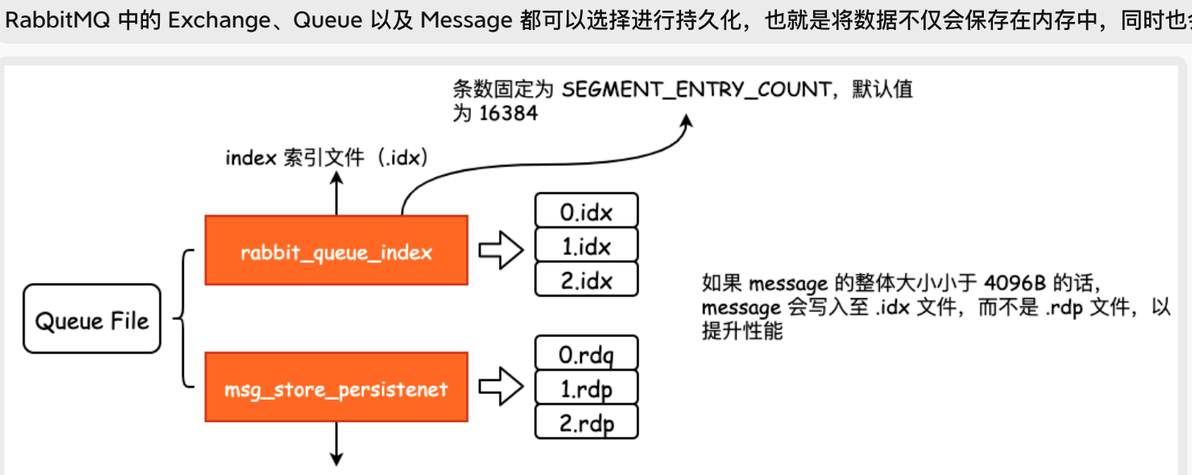
- 死信队列其实是 Exchange、Binding Key 以及 Queue 所组成的一个整体，并不单指哪一个具体的 Queue。也就是说，只要我们愿意，可以将任何的一个 Exchange、Binding Key 以及 Queue 指定为死信队列
- 首先来看一条消息在什么情况下会变成死信队列
  - ① 消息被拒绝（basic.reject/basic.nack），并且 requeue 为 false — 当我们关闭了自动 ACK 以后，若某一条消息未正确回送 ACK 或者回送了 NACK 时，消息将会被重新投递到指定的队列中（Exchange-Binding Key-Queue）
  - ② 消息TTL过期
  - ③ 队列达到最大长度或队列空间已满 — 此时 Queue 的拒绝策略需设置为 reject-publish-dlx，那么后续投递至该队列中的消息会被重新投递到死信队列中
- 延迟消息投递



  - 利用 Message 的 TTL 以及 Dead Letter Exchange，可以非常轻松的实现延迟消息投递功能，只不过此时我们需要两个队列

Question

- RabbitMQ 如何确保消息的可靠投递?
  - **RabbitMQ 借鉴了 TCP 协议的 Seq + ACK 机制，同时也会有 ACK/NACK 丢失后的消息重传机制**
  - 当我们收到 MQ 投递的消息以后，在处理成功后可向 MQ 回送一个 ACK 消息，表示当前 message 已成功消费。如果业务处理出现了异常导致该消息无法被正常消费时，可回送一个 NACK 消息
  - **同时，我们也可以配置 RabbitMQ 在收到 NACK 的行为，包括重新投递该消息，也就是重回队列，或者直接丢弃该消息，又或者是重新投递至死信队列中**
- 如何保证消息消费的幂等性?
  - **消息的幂等性消费永远是在 Consumer 中进行保证的，我们无法让 MQ 中间件来保证**
  - 假如说 ACK 在网络中丢失，RabbitMQ 因迟迟没有收到该 ACK 所以重新推送该消息，那么此时同一条消息就会向消费者投递多次
  - 解决的办法通常都是在 Message 中添加一个唯一 ID，比如唯一的 MessageID，OrderID，等等。如果同一个唯一 ID 存在多种状态，例如商品订单，那么我们也可以通过 OrderID + status 或者是更新时间、数据版本号等综合进行去重
- RabbitMQ 如何保证重启后消息不丢失?
  - RabbitMQ 中的 Exchange、Queue 以及 Message 都可以选择进行持久化，也就是将数据不仅会保存在内存中，同时也会保存在硬盘中



  - RabbitMQ 并不需要进行范围查询，因此消息在硬盘中全部使用顺序追加的方式进行写入，因此写入效率较高
- RabbitMQ 有哪些应用场景? — 解耦，广播，削峰