# 消息队列

## RabbitMq

1. 什么是消息?

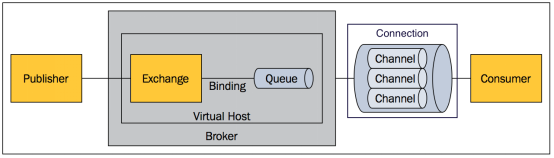
在应用间传送的数据。可以是文本字符串, 也可以是对象.

消息队列（Message Queue）是一种应用间的通信方式，消息发送后可以立即返回，由消息系统来确保消息的可靠传递。发布者只管把消息发布到 MQ 中而不用管谁来取，消息使用者只管从 MQ 中取消息而不管是谁发布的。这是一种异步协作机制. 用于解耦.

1. rabbitmq 特点:
   1. 可靠性. 如持久化、传输确认、发布确认。
   2. 灵活的路由.
   3. 集群
   4. 多种协议
   5. 管理界面
   6. 跟踪机制
2. 消息模型

消费者（consumer）订阅某个队列。生产者（producer）创建消息，然后发布到队列（queue）中，最后将消息发送到监听的消费者。

1. rabbitmq结构



Publisher : 消息的生产者

Exchange : 交换器，接收生产者发送的消息并将这些消息路由给服务器中的队列。

Binding : 消息和绑定, 用于消息队列和交换器之间的关联。一个绑定就是基于路由键将交换器和消息队列连接起来的路由规则，所以可以将交换器理解成一个由绑定构成的路由表。

Routing Key：路由关键字，exchange根据这个关键字进行消息投递。

Queue : 消息队列，用来保存消息直到发送给消费者。它是消息的容器，也是消息的终点。一个消息可投入一个或多个队列。消息一直在队列里面，等待消费者连接到这个队列将其取走。

Connection : 网络连接，比如一个TCP连接。

Channel : 信道，多路复用连接中的一条独立的双向数据流通道。信道是建立在真实的TCP连接内地虚拟连接，AMQP 命令都是通过信道发出去的，不管是发布消息、订阅队列还是接收消息，这些动作都是通过信道完成。因为对于操作系统来说建立和销毁 TCP 都是非常昂贵的开销，所以引入了信道的概念，以复用一条 TCP 连接。

consumer : 消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。

Virtual Host : 虚拟主机，表示一批交换器、消息队列和相关对象。虚拟主机是共享相同的身份认证和加密环境的独立服务器域。每个 vhost 本质上就是一个 mini 版的 RabbitMQ 服务器，拥有自己的队列、交换器、绑定和权限机制。vhost 是 AMQP 概念的基础，必须在连接时指定，RabbitMQ 默认的 vhost 是 / .

Broker : 表示消息队列服务器实体。

1. RabbitMQ常用的Exchange Type有三种：fanout、direct、topic。
   1. fanout:把所有消息都发送到与它绑定的队列中。
   2. direct:把消息投递到那些binding key与routing key完全匹配的队列中。
   3. topic:将消息路由到binding key与routing key模式匹配的队列中。匹配格式: a.b.c;匹配符号\*表示一个单词, #表示0个或多个. a.#表示所有以a.开头的都会匹配. 不能被匹配的消息会被丢弃掉.当路由键为#号时相当于fanout模式, 当没有\*和#时则相当于direct模式.
2. RabbitMQ的消息确认机制
   1. 默认情况下，如果消息已经被某个消费者正确的接收到了，那么该消息就会被从队列中移除。当然也可以让同一个消息发送到很多的消费者。
   2. 如果一个队列没有消费者，那么，如果这个队列有数据到达，那么这个数据会被缓存，不会被丢弃。当有消费者时，这个数据会被立即发送到这个消费者，这个数据被消费者正确收到时，这个数据就被从队列中删除。
   3. 什么是正确收到呢？通过ack。每个消息都要被acknowledged（确认，ack）。我们可以显示的在程序中去ack，也可以自动的ack。如果有数据没有被ack，那么：

RabbitMQ Server会把这个信息发送到下一个消费者。

如果这个app有bug，忘记了ack，那么RabbitMQServer不会再发送数据给它，因为Server认为这个消费者处理能力有限。

1. rabbitmq编程
   1. 发送与接收方共同点

ConnectionFactory factory = **new** ConnectionFactory(); //连接工厂

factory.setHost("localhost"); //设置host

Connection conn = factory.newConnection(); //创建连接

Channel channel = conn.createChannel(); //创建频道.

* 1. 消息的负载均衡 channel.basicQos(int conut), 设置每次发送消息给消费者的数量,如为1时,则每次只会给消费者发送一个消息, 当消费者处理完后才会收到下一个.
  2. 消息持久化 channel.queueDeclare("hello", durable, false, false, null); 在声明队列时定义, 当durable为true时,则持久化.当消费者挂掉后,消息不会丢失,当rabbitmq服务器挂掉仍会丢失部分消息,因为持久化过程不会时刻进行.仍要保证消息不丢失, 则可考虑生产者确认反馈机制.
  3. 消息确认(消费者) 当服务器收到消费者的确认反馈时,才会将消息从队列中删除, 如果没有收到, 服务器会重新推送.

确认有消息处理完成确认channel.basicAck(envelope.getDeliveryTag(), false);

消息消费完成确认 channel.basicConsume(TASK\_QUEUE\_NAME, false, consumer);

1. 死信队列DLX:

当消息过期, 被拒绝, 队列达到最大长度时会被放入死信队列, 死信队列也是一个普通队列, 只是它是专门用来存放死信信息的队列.它与死信交换机绑定. 死信交换机可以与任何队列进行绑定, 在业务出现死信时会将数据发送到死信队列.

x-dead-letter-exchange: 用来设置死信后发送的交换机

x-dead-letter-routing-key：用来设置死信的routingKey

1. 队列一旦定义后就不能再重新定义, 幂等原理. 其任意多次执行所产生的影响均与一次执行的影响相同

## ActiveMq