消息中间件：



一、Rabbit MQ

一对一模式

1. 根据接入系统，创建队列
2. 发送方调用API将消息发送至队列
3. 接收方调用API接收完成后，调用回执方法。
4. 服务端接收回执，删除队列消息，并记录回执日志。

一对多模式

1. 创建fanout交换器，将多个系统的队列进行绑定。
2. 发送方将消息发送到交换器再到绑定的队列
3. 多个接收队列完成后，调用回执方法，生成回执消息。服务端记录日志。

Exchange可以理解为交换器，RoutingKey可以理解为路由，Queue作为真实存储消息的队列和某个Exchange绑定，具体如何路由到感兴趣的Queue则由Exchange的三种模式决定：

（1）Exchange为fanout时，生产者往此Exchange发送的消息会发给每个和其绑定的Queue，此时RoutingKey并不起作用；  
（2）Exchange为topic时，生产者可以指定一个支持通配符的RoutingKey（如demo.\*）发向此Exchange，凡是Exchange上RoutingKey满足此通配符的Queue就会收到消息；  
（3）direct类型的Exchange是最直接最简单的，生产者指定Exchange和RoutingKey，然后往其发送消息，消息只能被绑定的满足RoutingKey的Queue接受消息。(通常如果不指定RoutingKey的具体名字，那么默认的名字其实是Queue的名字）

普通集群模式

镜像集群模式（queue存在于多个实例上）

rabbitmqctl stop\_app

# --ram 是指以内存的方式运行 不加这个参数的话是默认以 disk 硬盘的方式执行

rabbitmqctl join\_cluster --ram rabbit@master

rabbitmqctl start\_app

# 在其他服务器节点上移除节点rabbit@slave1

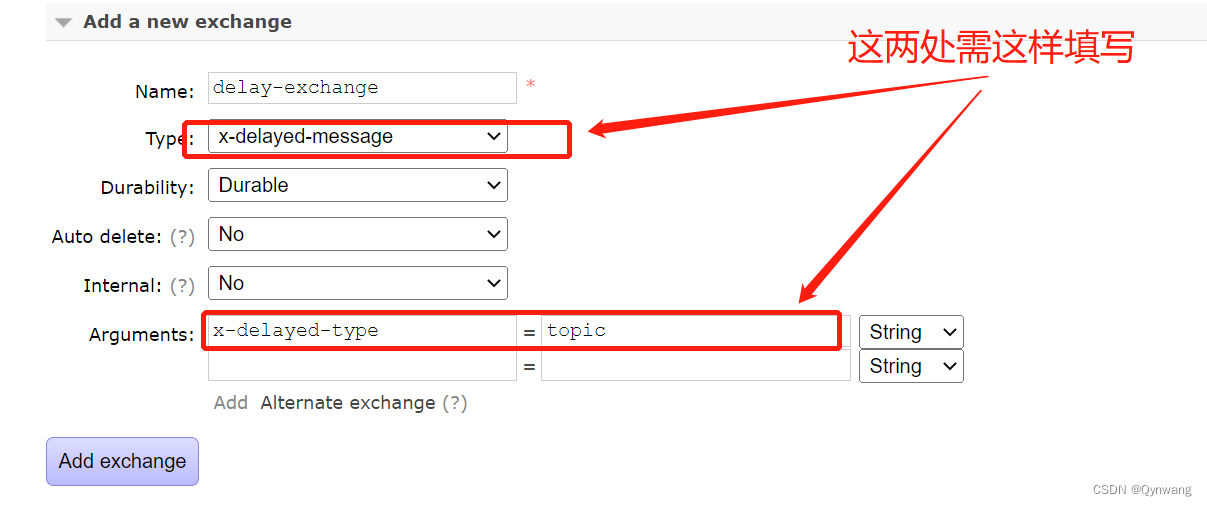
rabbitmqctl forget\_cluster\_node rabbit@slave1

设置镜像队列策略（在其中任意一个节点上执行即可）

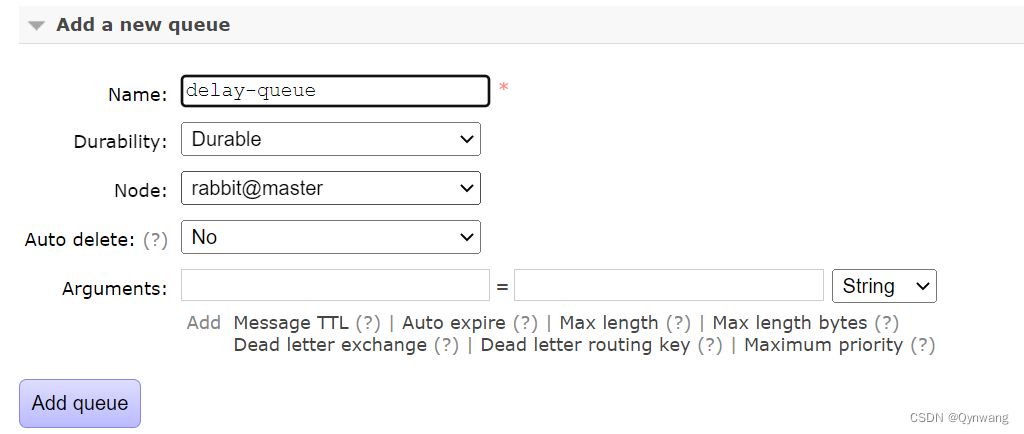
rabbitmqctl set\_policy ha-all "^" '{"ha-mode":"all"}'

添加延迟队列（取消订单的场景）

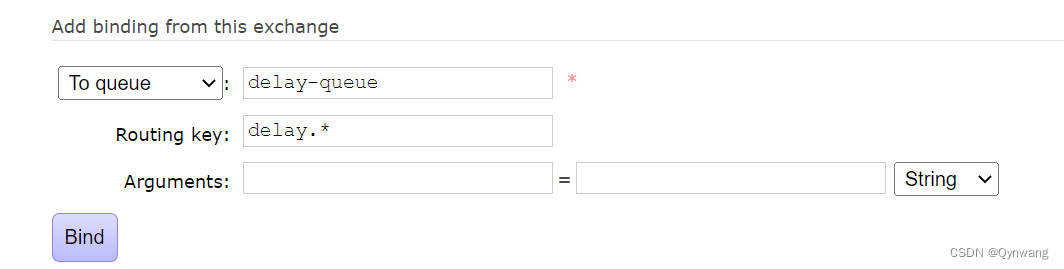
1. 配置exchange，type和args



2.配置queue，并和该交换机按照 routingKey 绑定



3.绑定



配置优化

系统级调整：

vi /etc/sysctl.conf

fs.file-max=10240

vi /etc/security/limits.conf

openstack  soft  nofile  10240

openstack  hard  nofile   10240

配置调整

vi /usr/lib/systemd/system/rabbitmq-server.service

LimitNOFILE=10240

☆. 消息持久化:

  exchange：durable=True

  queue：durable=True

  message：BasicProperties(delivery\_mode=2) 2是持久化，1是不持久化

☆.发送确认

☆. 手动消费确认

  ack：消费成功

  nack：消费失败

## **如何保证消息的可靠性传输？或者说，如何处理消息丢失的问题？**

开启 confirm 模式，在生产者那里设置开启 confirm 模式之后，你每次写的消息都会分配一个唯一的 id，然后如果写入了 RabbitMQ 中，RabbitMQ 会给你回传一个 ack 消息，告诉你说这个消息 ok 了。如果 RabbitMQ 没能处理这个消息，会回调你的一个 nack 接口，告诉你这个消息接收失败，你可以重试。而且你可以结合这个机制自己在内存里维护每个消息 id 的状态，如果超过一定时间还没接收到这个消息的回调，那么你可以重发。

**事务机制和 confirm 机制最大的不同在于**，**事务机制是同步的**，你提交一个事务之后会阻塞在那儿，但是 **confirm 机制是异步的**，你发送个消息之后就可以发送下一个消息，然后那个消息 RabbitMQ 接收了之后会异步回调你的一个接口通知你这个消息接收到了。

所以一般在生产者这块避**免数据丢失，都是用 confirm 机制**的。

**设置持久化有两个步骤**

1. **1. 创建 queue 的时候将其设置为持久化 这样就可以保证 RabbitMQ 持久化 queue 的元数据，但是它是不会持久化 queue 里的数据的。**
2. **2. 第二个是发送消息的时候将消息的 deliveryMode 设置为 2 就是将消息设置为持久化的，此时 RabbitMQ 就会将消息持久化到磁盘上去。**

**必须要同时设置这两个持久化才行，RabbitMQ 哪怕是挂了，再次重启，也会从磁盘上重启恢复 queue，恢复这个 queue 里的数据。**

注意，哪怕是你给 RabbitMQ 开启了持久化机制，也有一种可能，就是这个消息写到了 RabbitMQ 中，但是还没来得及持久化到磁盘上，结果不巧，此时 RabbitMQ 挂了，就会导致内存里的一点点数据丢失。

所以，持久化可以跟生产者那边的 confirm 机制配合起来，只有消息被持久化到磁盘之后，才会通知生产者 ack 了，所以哪怕是在持久化到磁盘之前，RabbitMQ 挂了，数据丢了，生产者收不到 ack ，你也是可以自己重发的。

### **Kafka 弄丢了数据**

这块比较常见的一个场景，就是 Kafka 某个 broker 宕机，然后重新选举 partition 的 leader。大家想想，要是此时其他的 follower 刚好还有些数据没有同步，结果此时 leader 挂了，然后选举某个 follower 成 leader 之后，不就少了一些数据？这就丢了一些数据啊。

生产环境也遇到过，我们也是，之前 Kafka 的 leader 机器宕机了，将 follower 切换为 leader 之后，就会发现说这个数据就丢了。

所以此时一般是要求起码设置如下 4 个参数：

给 topic 设置 replication.factor 参数：这个值必须大于 1，要求每个 partition 必须有至少 2 个副本。在 Kafka 服务端设置 min.insync.replicas 参数：这个值必须大于 1，这个是要求一个 leader 至少感知到有至少一个 follower 还跟自己保持联系，没掉队，这样才能确保 leader 挂了还有一个 follower 吧。在 producer 端设置 acks=all ：这个是要求每条数据，必须是写入所有 replica 之后，才能认为是写成功了。在 producer 端设置 retries=MAX （很大很大很大的一个值，无限次重试的意思）：这个是要求一旦写入失败，就无限重试，卡在这里了。我们生产环境就是按照上述要求配置的，这样配置之后，至少在 Kafka broker 端就可以保证在 leader 所在 broker 发生故障，进行 leader 切换时，数据不会丢失。

## **如何避免消息重复投递或重复消费？**

在消息生产时，MQ 内部针对每条生产者发送的消息生成一个 inner-msg-id，作为去重的依据（消息投递失败并重传），避免重复的消息进入队列；在消息消费时，要求消息体中必须要有一个 bizId（对于同一业务全局唯一，如支付 ID、订单 ID、帖子 ID 等）作为去重的依据，避免同一条消息被重复消费。

## **无法被路由的消息去了哪里**

无设置的情况下，无法路由（Routing key错误）的消息会被直接丢弃 解决方案：将mandatory设置为true，并配合ReturnListener，实现消息的回发

## **RabbitMQ如何实现延时队列**

利用TTL（队列的消息存活时间或者消息存活时间），加上死信交换机

监控 Prometheus

1. Kafka

### **Kafka与Zookeeper是什么关系？**

Kafka的数据会存储在zookeeper上。包括broker和消费者consumer的信息 其中,

**broker信息**：包含各个broker的服务器信息、Topic信息

**消费者信息**：主要存储每个消费者消费的topic的offset的值

### **Kafka的性能好在什么地方？**

Kafka的性能好在两个方面：**顺序写**和**零拷贝**

1）**顺序写**

我们知道，操作系统每次从磁盘读写数据的时候，都需要找到数据在磁盘上的地址，再进行读写。而如果是机械硬盘，寻址需要的时间往往会比较长

而一般来说，如果把数据存储在内存上面，少了寻址的过程，性能会好很多；但Kafka 的数据存储在磁盘上面，依然性能很好，这是为什么呢？

这是因为，Kafka采用的是顺序写，直接追加数据到末尾。实际上，磁盘顺序写的性能极高，在磁盘个数一定，转数一定的情况下，基本和内存速度一致

因此，磁盘的顺序写这一机制，极大地保证了Kafka本身的性能

Redis

能不能和Oracle配套使用