# MQ

为什么使用消息队列？ --- 解耦（系统间依赖）、异步（）、削峰（高并发时可按数据库能处理的并发量从消息队列拉取消息处理，允许短暂高峰期积压）

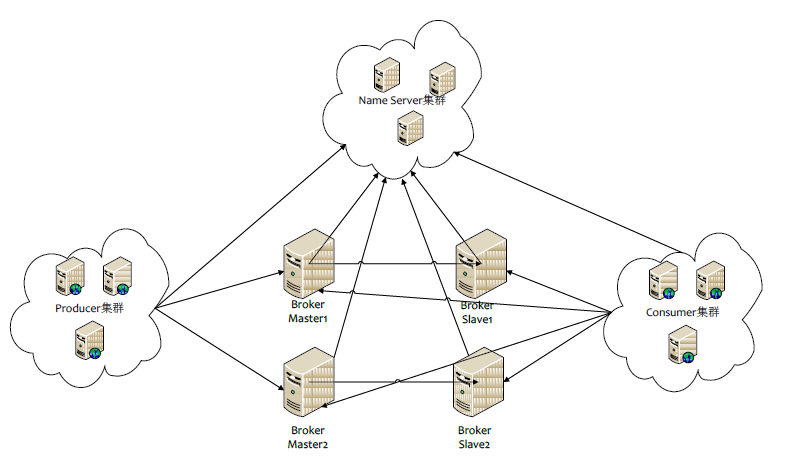
1. 缺点：

系统可用性降低:你想啊，本来其他系统只要运行好好的，那你的系统就是正常的。现在你非要加个消息队列进去，那消息队列挂了，你的系统不是呵呵了。因此，系统可用性降低

系统复杂性增加:要多考虑很多方面的问题，比如一致性问题、如何保证消息不被重复消费，如何保证保证消息可靠传输。因此，需要考虑的东西更多，系统复杂性增大。

| **特性** | **ActiveMQ** | **RabbitMQ** | **RocketMQ** | **kafka** |
| --- | --- | --- | --- | --- |
| 开发语言 | java | erlang | java | scala |
| 单机吞吐量 | 万级 | 万级 | 10万级 | 10万级 |
| 时效性 | ms级 | us级 | ms级 | ms级以内 |
| 可用性 | 高(主从架构) | 高(主从架构) | 非常高(分布式架构) | 非常高(分布式架构) |
| 功能特性 | 成熟的产品，在很多公司得到应用；有较多的文档；各种协议支持较好 | 基于erlang开发，所以并发能力很强，性能极其好，延时很低;管理界面较丰富 | MQ功能比较完备，扩展性佳 | 只支持主要的MQ功能，像一些消息查询，消息回溯等功能没有提供，毕竟是为大数据准备的，在大数据领域应用广。 |

## 2. 保证消息队列高可用



rcoketMQ：他的集群就有多master 模式、多master多slave异步复制模式、多 master多slave同步双写模式。

Producer：生产发送消息

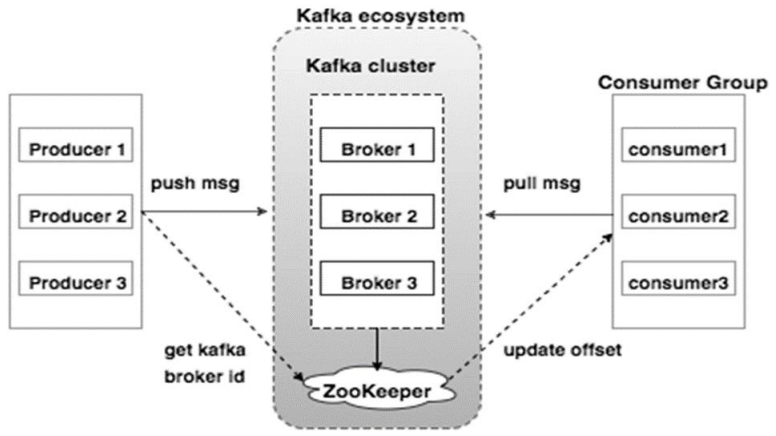
Broker：存储Producer发送过来的消息

Consumer：从Broker拉取消息并进行消费

NameServer：为Producer或Consumer路由到Broker

RocketMQ的消息发送方式主要含syncSend()同步发送、asyncSend()异步发送、sendOneWay()三种方式，sendOneWay()也是异步发送，区别在于不需等待Broker返回确认，所以可能会存在信息丢失的状况，但吞吐量更高，具体需根据业务情况选用。性能：sendOneWay > asyncSend > syncSend RocketMQTemplate的send()方法默认是同步(syncSend)

————————————————



Kafka：只是NameServer集群，在kafka中是用zookeeper代替，都是用来保存和发现master和slave用的。通信过程如下:  
Producer 与 NameServer集群中的其中一个节点（随机选择）建立长连接，定期从 NameServer 获取 Topic 路由信息，并向提供 Topic 服务的 Broker Master 建立长连接，且定时向 Broker 发送心跳。Producer 只能将消息发送到 Broker master，但是 Consumer 则不一样，它同时和提供 Topic 服务的 Master 和 Slave建立长连接，既可以从 Broker Master 订阅消息，也可以从 Broker Slave 订阅消息。

## 3. 如何保证消息不被重复消费？

RabbitMQ是发送一个ACK确认消息，RocketMQ是返回一个CONSUME\_SUCCESS成功标志，kafka实际上有个offset的概念，简单说一下(如果还不懂，出门找一个kafka入门到精通教程),就是每一个消息都有一个offset，kafka消费过消息后，需要提交offset，让消息队列知道自己已经消费过了。那造成重复消费的原因?，就是因为网络传输等等故障，确认信息没有传送到消息队列，导致消息队列不知道自己已经消费过该消息了，再次将该消息分发给其他的消费者。  
  如何解决?这个问题针对业务场景来答分以下几点  
  (1)比如，你拿到这个消息做数据库的insert操作。那就容易了，给这个消息做一个唯一主键，那么就算出现重复消费的情况，就会导致主键冲突，避免数据库出现脏数据。  
  (2)再比如，你拿到这个消息做redis的set的操作，那就容易了，不用解决，因为你无论set几次结果都是一样的，set操作本来就算幂等操作。  
  (3)如果上面两种情况还不行，上大招。准备一个第三方介质,来做消费记录。以redis为例，给消息分配一个全局id，只要消费过该消息，将<id,message>以K-V形式写入redis。那消费者开始消费前，先去redis中查询有没消费记录即可。

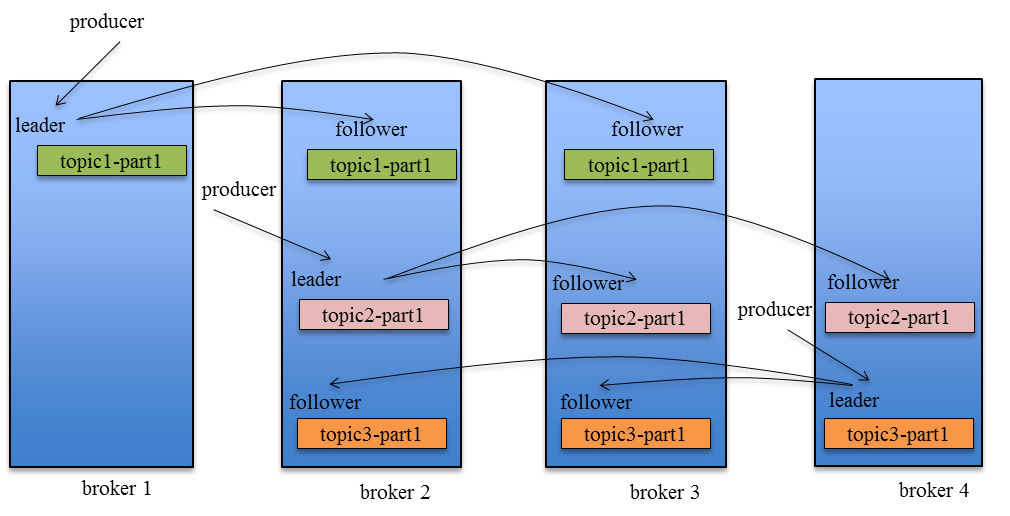
## 4． 如何保证消费的可靠性传输?

#### RabbitMQ

(1)生产者丢数据  
从生产者弄丢数据这个角度来看，RabbitMQ提供transaction和confirm模式来确保生产者不丢消息。  
transaction机制就是说，发送消息前，开启事物(channel.txSelect())，然后发送消息，如果发送过程中出现什么异常，事物就会回滚(channel.txRollback())，如果发送成功则提交事物(channel.txCommit())。  
然而缺点就是吞吐量下降了。因此，按照博主的经验，生产上用confirm模式的居多。一旦channel进入confirm模式，所有在该信道上面发布的消息都将会被指派一个唯一的ID(从1开始)，一旦消息被投递到所有匹配的队列之后，rabbitMQ就会发送一个Ack给生产者(包含消息的唯一ID)，这就使得生产者知道消息已经正确到达目的队列了.如果rabiitMQ没能处理该消息，则会发送一个Nack消息给你，你可以进行重试操作

(2)消息队列丢数据  
处理消息队列丢数据的情况，一般是开启持久化磁盘的配置。这个持久化配置可以和confirm机制配合使用，你可以在消息持久化磁盘后，再给生产者发送一个Ack信号。这样，如果消息持久化磁盘之前，rabbitMQ阵亡了，那么生产者收不到Ack信号，生产者会自动重发。  
那么如何持久化呢，这里顺便说一下吧，其实也很容易，就下面两步  
1、将queue的持久化标识durable设置为true,则代表是一个持久的队列  
2、发送消息的时候将deliveryMode=2  
这样设置以后，rabbitMQ就算挂了，重启后也能恢复数据  
(3)消费者丢数据  
消费者丢数据一般是因为采用了自动确认消息模式。这种模式下，消费者会自动确认收到信息。这时rahbitMQ会立即将消息删除，这种情况下如果消费者出现异常而没能处理该消息，就会丢失该消息。  
至于解决方案，采用手动确认消息即可。

#### kafka

这里先引一张kafka Replication的[数据流向图](http://www.infoq.com/cn/articles/kafka-analysis-part-2/" \t "_blank)  
  
Producer在发布消息到某个Partition时，先通过ZooKeeper找到该Partition的Leader，然后无论该Topic的Replication Factor为多少（也即该Partition有多少个Replica），Producer只将该消息发送到该Partition的Leader。Leader会将该消息写入其本地Log。每个Follower都从Leader中pull数据。  
针对上述情况，得出如下分析  
(1)生产者丢数据  
在kafka生产中，基本都有一个leader和多个follwer。follwer会去同步leader的信息。因此，为了避免生产者丢数据，做如下两点配置

1. 第一个配置要在producer端设置acks=all。这个配置保证了，follwer同步完成后，才认为消息发送成功。
2. 在producer端设置retries=MAX，一旦写入失败，这无限重试

(2)消息队列丢数据  
针对消息队列丢数据的情况，无外乎就是，数据还没同步，leader就挂了，这时zookpeer会将其他的follwer切换为leader,那数据就丢失了。针对这种情况，应该做两个配置。

1. replication.factor参数，这个值必须大于1，即要求每个partition必须有至少2个副本
2. 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

这两个配置加上上面生产者的配置联合起来用，基本可确保kafka不丢数据

(3)消费者丢数据  
这种情况一般是自动提交了offset，然后你处理程序过程中挂了。kafka以为你处理好了。再强调一次offset是干嘛的  
offset：指的是kafka的topic中的每个消费组消费的下标。简单的来说就是一条消息对应一个offset下标，每次消费数据的时候如果提交offset，那么下次消费就会从提交的offset加一那里开始消费。  
比如一个topic中有100条数据，我消费了50条并且提交了，那么此时的kafka服务端记录提交的offset就是49(offset从0开始)，那么下次消费的时候offset就从50开始消费。