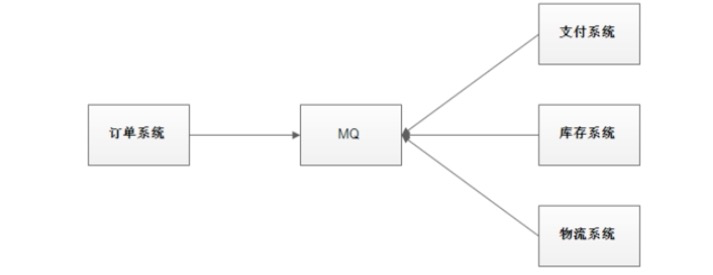
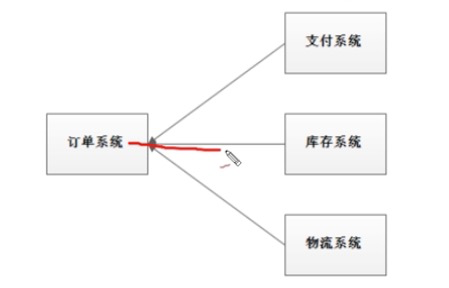
**MQ介绍**

消息队列是一种先进先出的数据结构

**应用场景**

1. 应用解耦

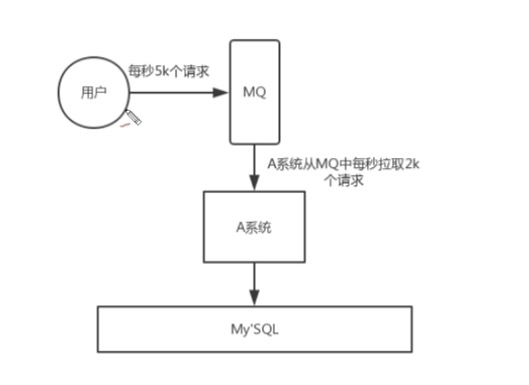
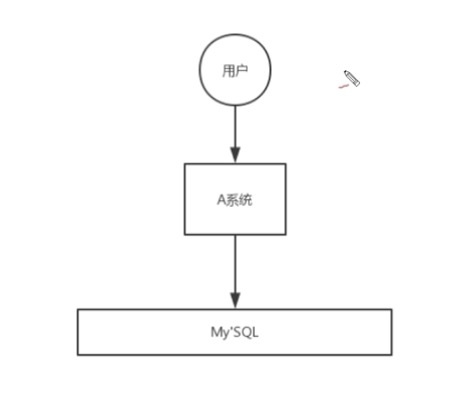
系统的耦合性越高，容错性就越低。以电商应用为例，如果某个子系统出现故障，那么整个系统都将不可用。



使用消息队列解耦合，如果物流系统发生了故障，需要几分钟才能恢复，这段时间内，物流系统要处理的数据被缓存到消息队列中，用户下单等操作可以正常完成。 当物流系统恢复后，补充处理存在消息队列中的订单消息即可，终端系统感知不到物流系统发生过几分钟故障。

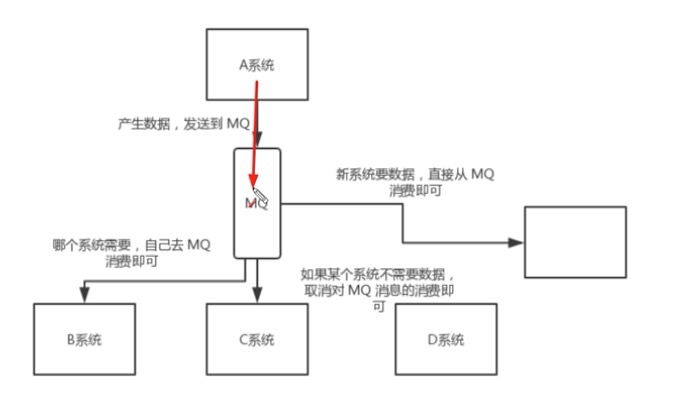
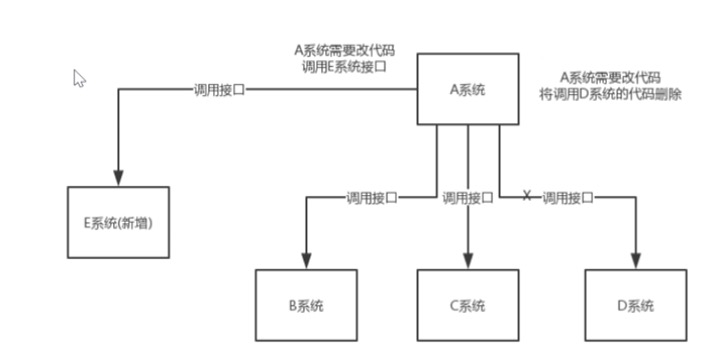
1. 流量削峰

应用系统如果遇到系统请求流量瞬间猛增，有可能将系统压垮。有了消息队列，可以将大量请求缓存起来，分散到很长一段时间处理，这样可以大大提供系统的稳定性和可靠性



一般情况，为了保证系统的温蒂定型，如果系统负载超过阈值，就会组织用户请求，这回影响用户体验，而如果使用消息队列额将请求缓存起来，等待系统处理完毕后通知用户下单完毕，这样总不能下单体验要好。

1. 数据分发



通过消息队列可以让数据在多个系统间进行流通，数据的生产方不需要谁来使用数据，只需要将数据发送到消息队列，数据使用方直接在消息队列中直接获取数据即可

**缺点**

1. 系统可用性降低

系统一如的外部依赖越多，系统稳定性越差，一旦mq宕机，就会对业务造成影响。

如何保证mq的高可用

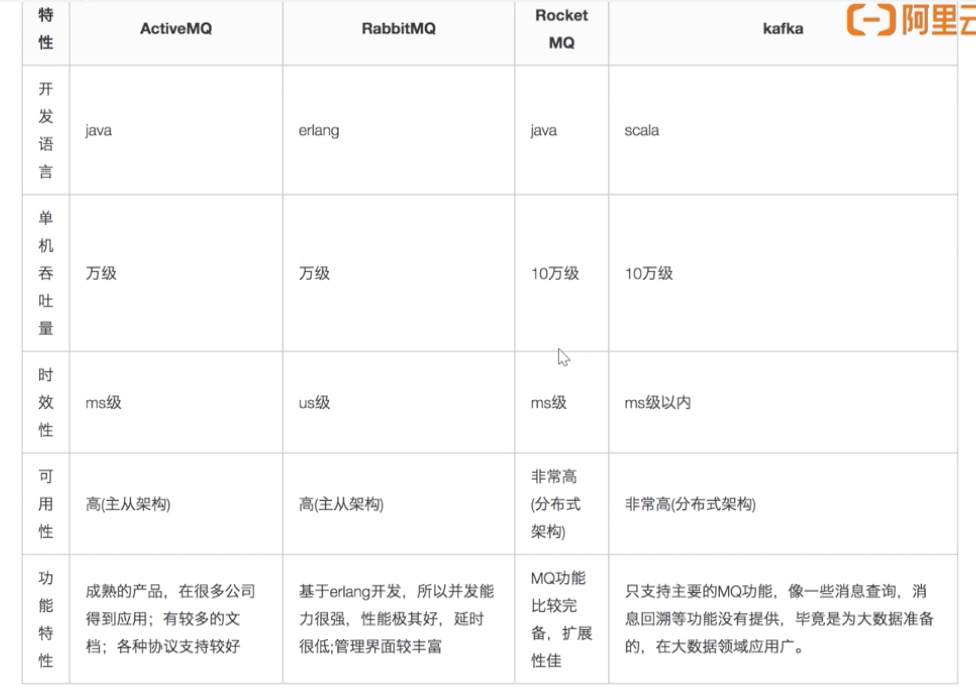
1. 系统复杂度提高

mq的加入大大增加了系统的复杂度，以前系统间是同步的远程调用，现在是通过mq进行一步调用。 如何保证消息没有被重复消费，怎么处理消息丢失的情况，怎么保证消息传递的顺序

1. 一致性问题

A系统处处理完业务，通过mq给B、c、d 三个系统发送消息数据；如果b、c系统处理成功，d处理失败，如何保证消息数据处理的一致性

**mq产品比较**



**快速入门**

**安装**

1. 安装jdk
2. 下载[官网](https://www.apache.org/dyn/closer.cgi?path=rocketmq/4.9.0/rocketmq-all-4.9.0-bin-release.zip)
3. 解压二进制
   1. bin 启动脚本，包括shell，cmd
   2. conf 实例配置文件，包括broker配置文件，logback配置文件
   3. lib 依赖的jar包，包括netty commins-lan, fastjson
4. 启动

#在bin目录下启动nameserver

nohup sh bin/mqnamesvr &

#查看启动日志

tail -f ~/logs/rocketmqlogs/namesvr.log

#启动broker

nohup sh bin/mqbroker -n localhost:9876 &

tail -f ~logs/rocketmqlogs/broker.log

rocketmq 默认的虚拟机内存较大，启动broker 如果因为内存不足失败，需要编辑如下两个文件，修改jvm 内存大小

runbroker.sh

runserver.sh

JAVA\_OPT="${JAVA\_OPT} -server -Xms2g -Xmx2g -Xmn1g"

1. 关闭

#关闭nameserver

sh bin/mqshutdown namesrv

#关闭broker

sh bin/mqshudown broker

**测试**

发送消息

export NAMESRV\_ADDR=localhost:9876

sh bin/tools.sh org.appache.rocketmq.example.quickstart.Producer

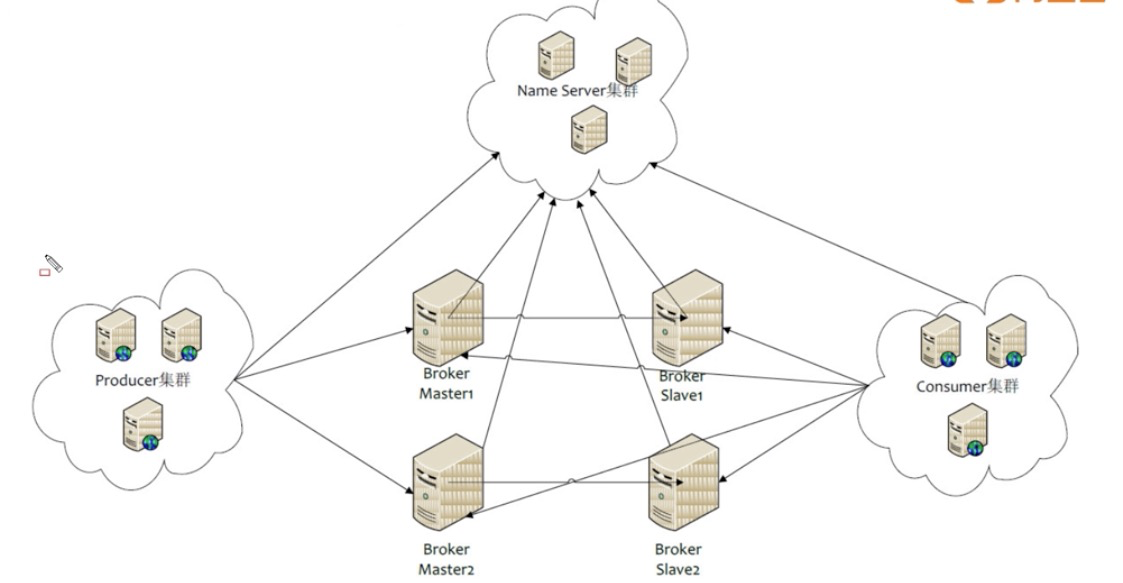
接收消息

export NAMESRV\_ADDR=localhost:9876

sh bin/tools.sh org.appache.rocketmq.example.quickstart.Consumer

**rocketmq 集群搭建**

**各个角色介绍**



producer: 消息发送者

consumer： 消息接受者

broker： 暂存和传输消息

nameserver： 管理broker

topic： 区分消息的种类，一个发送者可以给一个或者多个topic，一个消息的接受者可以订阅一个或者多个topic

message queue： 相当于是topic 的分区；用于并行发送和接收消息

1. broker 主动上报自己的信息到namesvr
2. producer 向namesvr 询问，分配一个broker 地址
3. producer 向broker 发送消息
4. 消费者询问namesvr，找哪个broker
5. 消费者从broker 获取消息

**集群特点**

1. nameserver 是一个几乎无状态节点，可集群部署，节点之间无任何信息同步。
2. broker 部署相对复杂，broker分为master 与slave，一个master可以对应多个slave，但是一个slave只能对应一个master。 master 与slave的对应关系通过制定相同的brokerName，不通的brokerid 来定义。 brokerid 为0 表示master，非0 表示slave。 master 也可以部署多个，每个broker与nameserver 集群中的所有节点建议长连接，定时注册topic 信息到所有nameserver。
3. 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，并想提供topic服务的master建立长连接，且定时向master发送心跳。 producer完全无状态，可以集群部署
4. consumer 与namesever 中的其中一个节点（随机选择）简历长连接，定期从nameserver 取topic 路由信息，并想提供topic的master、slae建立长连接，且定时向master、slave发送心跳。consumer 既可以从master订阅信息，也可以从slave订阅信息，订阅规则有broker配置决定。

消费者可以从broker 拉信息，也可以broker 推消息到消费者

**集群模式**

**单master模式**

这种方式风险较大，一旦broker重启或者宕机时，会导致整个服务不可用。不建议线上环境使用，可以用于本地测试。

**多master模式**

一个集群无slave，全是master，例如2个master或者3个master，这种模式的优缺点如下

优点： 配置简单，单个master宕机或者重启维护对应用无影响，在磁盘配置为raid10时，及时机器宕机不可恢复的情况下，由于raid10 磁盘确实非常可靠，消息也不会丢（一步刷磁盘丢少量信息，同步刷磁盘一条不丢），性能最高。

缺点： 单台机器宕机期间，这台机器上违背消费的消息在机器恢复之前不可订阅，消息实时性会受到影响，

**多master多slave 模式（异步）**

每个master 配置一个slave，有多对master-slave， ha 采用异步复制方式，主备有短暂消息延时（毫秒级），这种模式的优缺点如下

优点：及时磁盘损坏，消息丢失的非常少，且消息实时性不会受影响，同时master 宕机后，消费者仍然可以从slave消费。而且此过程对应用透明，不需要人工干预，性能通多master模式几乎一样；

缺点： master宕机，磁盘损坏情况下会丢失少量消息。

**多master 多slave模式（同步）**

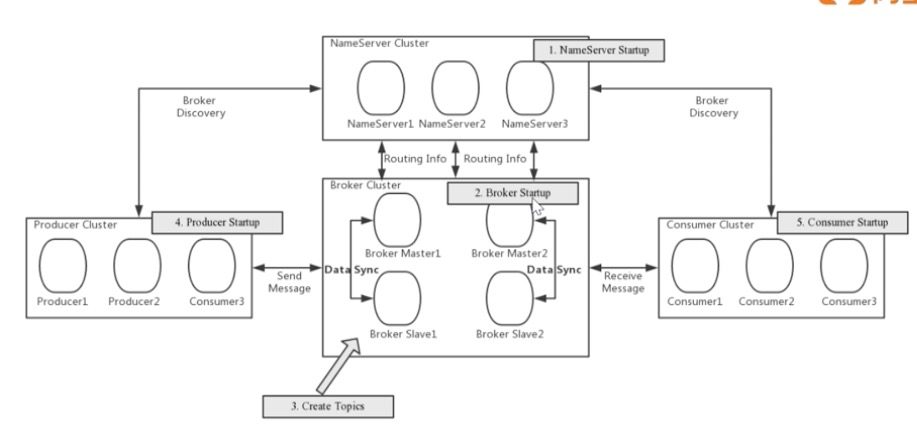
每个master配置一个slave，有多对master-slave， ha 采用异步双写方式，即只有主备都写成功了，才能向应用返回。

优点： 数据与服务都无单点故障，master宕机情况下，消息无延迟，服务可用性与数据可用性都非常高。

缺点： 性能比异步复制模式略低（大约10%），发送单个消息的RT会略高，且目前版本在主节点宕机后，备机不能自动切换为主机

**双主双从集群搭建（同步）**

架构图



集群工作流程：

1. 启动nameserver，nameserver起来后监听端口，等待proker、producer、consumer连接，相当于一个路由控制中心
2. broker启动，跟所有的nameserver 保持长连接，定时发送心跳包。心跳包中包含当前broker信息（IP+端口）以及存储所有topic信息。注册成功之后，nameserver集群中就有topic跟briker的映射关系
3. 收发消息前，先创建topic，创建topic是需要制定该topic要存储在哪些broker上，也可以在发送消息是自动创建topic
4. produser发送消息，启动是先跟nameserver集群中的一台建立长连接，并从nameserver中获取当前发送的topic存在哪些broker上，轮询从队列列表中选择一个队列，然后与队列所在的broker建立长连接从而向broker发消息
5. consumer 跟producer 类似，跟其中一个nameserver建立长连接，获取当前订阅topic存在哪些broker上，然后直接跟broker建立连接通道，开始消费信息。

服务器信息

|  |  |  |
| --- | --- | --- |
| IP | 角色 |  |
| 10.0.0.95 | nameserver，brokerserver | master1，slave2 |
| 10.0.0.96 | nameserver，brokerserver | master2，slave1 |

配置hosts文件

#nameserver

10.0.0.95 rocketmq-nameserver1

10.0.0.96 rocketmq-nameserver2

#broker

10.0.0.95 rocketmq-master1

10.0.0.96 rocketmq-slave1

10.0.0.95 rocketmq-slave2

10.0.0.95 rocketmq-master2

修改/etc/profile

#set rocketmq

export ROCKETMQ\_HOME=/usr/local/rocketmq/rocketmq-all-4.8.0-bin-release

PATH=$PATH:$ROCKETMQ\_HOME/bin

创建消息存储路径

mkdir -p /usr/local/rocketmq/store/{commitlog,consumequeue,index}

修改broker配置文件

在conf/2m-2s-sync/broker-a.properties 这是master1 的配置文件

#所属集群名字，四个配置文件都是这个

brokerClusterName=rocketmq-cluster

#broker名字，注意此处不通的配置文件填写不一样。 不同组的需要不一样

brokerName=broker-a

#0表示Master，>0表示slave

brokerId=0

#nameserver地址，分号分隔。一样的。

namesrvAddr=rocketmq-nameserver1:9876;rocketmq-nameserver2:9876

#在发送消息时，自动创建服务器不存在的topic，默认创建的队列数

defaultTopicQueueNum=4

#是否允许broker自动创建topic，建议线下开启，线上关闭

autoCreateTopicEnable=true

#是否允许broker自动穿件订阅组，建议线下开启，线上关闭

autoCreateSubscriptionGroup=true

#broker 对外服务的监听端口

listenPort=10911

#删除文件时间点，默认凌晨4点

deleteWhen=04

#文件保留时间，默认48小时

fileReservedTime=120

#commitlog 每个文件的默认大小1G

mapedFileSizeCommitLog=10737417824

#consumequeue 每个文件默认存30w条，根据业务情况调整。

mapedFileSizeConsumeQueue=300000

#destroyMapedFIleIntervalForcibly=120000

#redeleteHangedFileInterval=120000

#检查物理文件磁盘空间

diskMaxUsedSpaceRatio=88

#存储路径 注意如果一个节点上部署多个broker，这里得不一样

storePathRootDir=/usr/local/rocketmq/store

#commitlog 存储路径

storePathCommitLog=/usr/local/rocketmq/store/commitlog

#消费队列存储路径

#消费队列存储路径

storePathConsumeQueue=/usr/local/rocketmq/store/consumequeue

#消息索引存储路径

storePathIndex=/usr/local/rocketmq/store/index

#checkpoint 文件存储路径

storeCheckPoint=/usr/local/rocketmq/store/checkpoint

#abort 文件存储路径

abortFIle=/usr/local/rocketmq/store/abort

#限制消息大小

maxMessageSize=65536

#flushCommitLogLeastPages=4

#flushConsumeQueueLeastPages=2

#flushCommitLogThoroughInterval=10000

#flueshConsumeQueueThoroughInterval=60000

#broker 的角色

#- ASYNC\_MASTER 异步复制master

#- SYNC\_MASTER 同步复制master

#- SLAVE

brokerRole=SYNC\_MASTER

#刷盘方式

#- Async\_Flush 异步刷盘

#- SYNC\_FLUSH 同步刷盘

flushDiskType=SYNC\_Flush

#checkTransactionMessageEnable=false

#发送消息线程池数量

#sendMessageThreadPoolNum=128

#拉消息线程池数量

#PullMEssageThreadPoolNum=128

注意端口不要冲突。95 节点配置broker-a.properties broker-b-s.properties，96 节点配置 broker-a-s.properties broker-b.properties

**启动服务**

修改runserver.sh/runbroker.sh 里面的java\_opt 使用内存大小

启动nameserver集群

分别在两个节点上启动

cd /usr/local/rocketmq/bin/

nohuo sh mqnamesrv &

启动broker

master1

cd /usr/local/rocketmq/bin/

nohup sh mqbroker -c /usr/local/rocketmq/conf/2m-2s-sync/broker-a.properties &

其他broker类似

查看日志

~/logs/rocketmq/broker.log

~/logs/rocketmq/

**mqadmin 管理工具**

具体用法参见：

<https://blog.csdn.net/wulitaot/article/details/79551499>

./mqadmin {command} {args}

**集群监控平台搭建**

<https://github.com/apache/rocketmq-externals>

这个包含rocketmq 的一些子项目， rocketmq-console 也在这里，需要提前下载下来

git clone https://github.com/apache/rocketmq-externals

cd rocketmq-console

mvn clean package -Dmaven.test.skip=true

java -jar rocketmq-console-ng-2.0.0.jar

注意：打包前需要先修改src/main/resources/application.properties 下面的namesrcAddr 字段。

执行成功后通过8080 端口就可以访问管理页面

**消息发送样例**

<https://developer.aliyun.com/lesson_2184_21656#_21656>

**发送消息者步骤分析**

1. 创建消息生产者producer，并制定生产者组名
2. 指定nameserver地址
3. 启动producer
4. 创建消息对象，指定主题topic，tag，和消息体
5. 发送消息
6. 关闭生产者producer

**消息消费者步骤分析**

1. 创建小飞飞这consumer，指定消费者组名
2. 指定nameserver地址
3. 订阅主题topic 跟tag
4. 设置回调函数，处理消息
5. 启动消费者consumer

**基本消息**

同步消息（需要得到发送结果）、异步消息（返回结果通过回调函数）、单向消息（不关心发送结果）

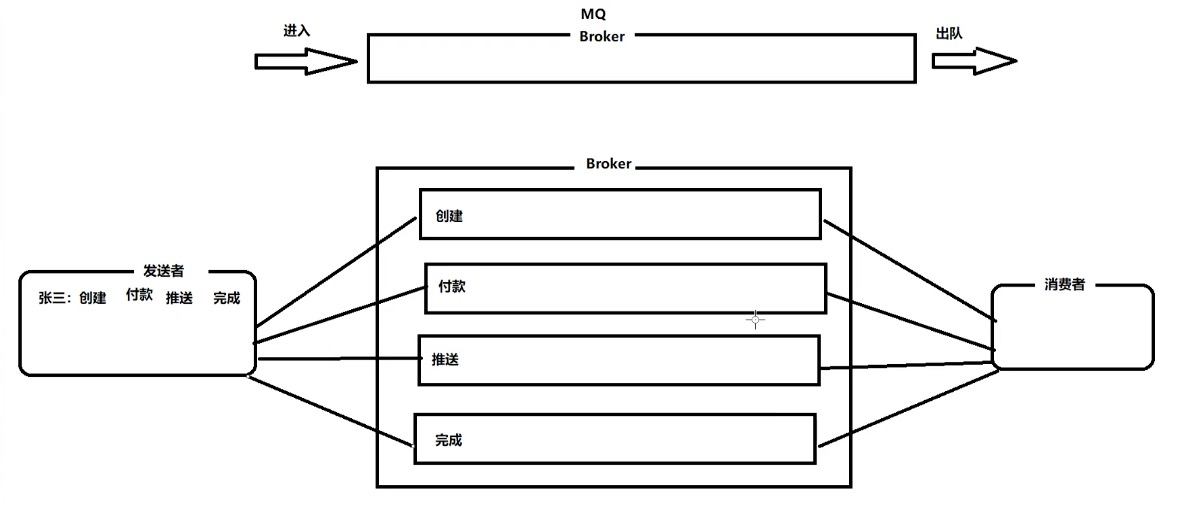
消息消费可以分为广播模式跟负载均衡。有多个消费者订阅相同的消息，广播模式就是每个消息都被每个消费者使用；负载均衡模式就是 每个消息只被一个消费者使用。 默认是负载均衡。

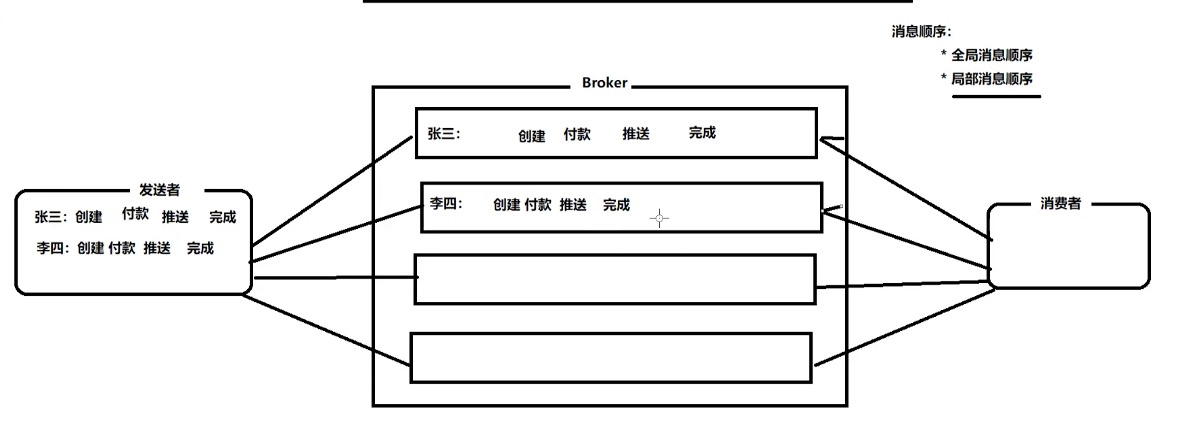
可以通过setMessageModel 来设置。

**顺序消息**

消息有序值得是可以按照消息的发送顺序来消费（FIFO）。rocketmq可以严格保证消息有序，可以分为分区有序或者全局有序。

书序消费的原理解析，在默认情况下消息发送会才去round robin轮询方式把消息发送到不通的queue（分区队列）；而消费消息的时候从多个queue上拉取消息，这种情况发送和消费是不能保证顺序。但是如果控制发送的顺序消息只依次发送到同一个queue中，消费的时候只能从这个queue上依次拉取，则就保证了顺序。 当发送和消费参与的queue只有一个，则是全局有序；如果多个queue参与，则为分区有序，即相对于每个queue消息都是有序的。





**延迟消息**

现在rocketmq并不支持任意时间的延时，需要设置几个固定的延时等级，有18 个

**批量消息**

如果每次只发送不超过4M的消息，则很容易使用批处理。

如果消息的总长度可能大于4M，这时候最好把消息进行分割

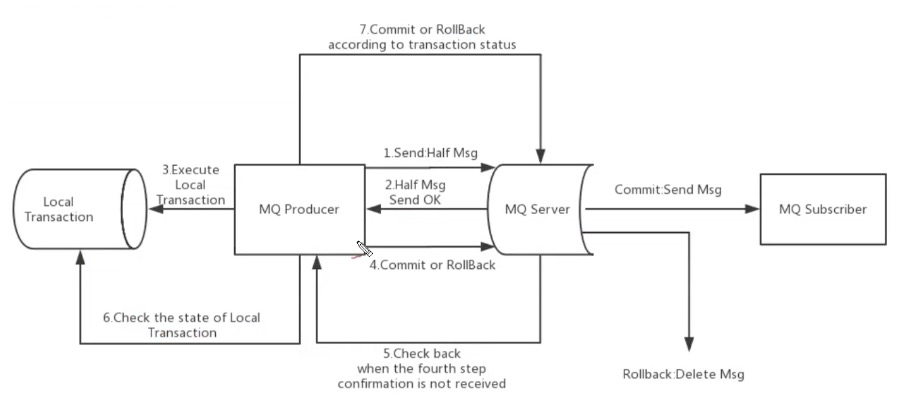
**过滤消息**

1. tag 过滤。大多数情况下，tag是一个简单有用的设计，其可以来选择您想要的消息。"tag1 || tag2"
2. sql语法过滤。 只有使用push 模式的消费者才能使用

**事务消息**

前面讲的都是没有事务控制的消息，称为half 消息

带事务的消息如果没有commit，消费者消费不了



上图说明了事务消息的大致方案，其中分为两个流程： 正常事务消息的发送及提交、事务消息的补偿流程（chackback）

**事务消息发送及提交**

1. 发送消息（half消息）
2. 服务端相应消息写入结果
3. 根据发送结果执行本地事务（如果写入失败，此时half消息对业务不可见，本地逻辑不执行）。
4. 根据本地事务状态执行commit 或者rollback（commit操作生成消息索引，消息对消费者课件）

**事务补偿**

1. 对没有commit/rollback 的事务消息（pending状态的消息），从服务端发起一次”回查“
2. producer收到回查消息，检查回查消息对应的本地事务的状态。
3. 根据本地事务状态，重新commit或者rollback

其中，补偿阶段用于解决消息commit或者rollback发生超时或者失败的情况

**事务消息的状态**

transactionstatus.committransaction:提交状态,他允许消费者消费此消息

transactionstatus。rollbacktrancation：回滚状态、他代表改消息江北删除，不允许被消费。

transactionstatus.unknown 中间状态， 他岱庙需要检查消息队列来确定状态