下载地址：<http://rocketmq.apache.org/release_notes/release-notes-4.4.0/>



上传到虚拟机

**rz**

**yum install unzip –y**

**unzip rocketmq-all-4.4.0-bin-release.zip -d /usr/local/**

**cd /usr/local/**

# 扩展小知识

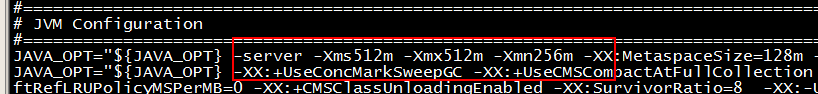
‘

|  |
| --- |
| 使用&后台运行程序：   * 结果会输出到终端 * 使用Ctrl + C发送SIGINT信号，程序免疫 * 关闭session发送SIGHUP信号，程序关闭   使用nohup运行程序：   * 结果默认会输出到nohup.out * 使用Ctrl + C发送SIGINT信号，程序关闭 * 关闭session发送SIGHUP信号，程序免疫   平日线上经常使用nohup和&配合来启动程序：   * 同时免疫SIGINT和SIGHUP信号   同时，还有一个最佳实践：   * 不要将信息输出到终端标准输出，标准错误输出，而要用日志组件将信息记录到日志里 |

# Start Name Server

注意：这里要先修改内存，默认4g，请修改（学习时）

修改**bin/runserver.sh**



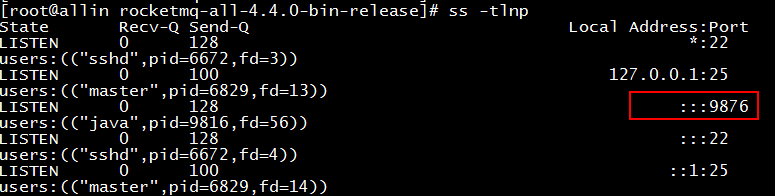
> nohup sh bin/mqnamesrv &

> tail -f ~/logs/rocketmqlogs/namesrv.log

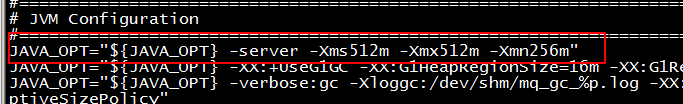
The Name Server boot success...



**开了个端口 9876**



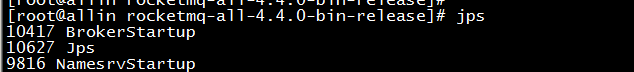
# Start Broker



> **nohup sh bin/mqbroker -n localhost:9876 &**

> **tail -f ~/logs/rocketmqlogs/broker.log**

The broker[%s, 172.30.30.233:10911] boot success...



# Send & Receive Messages

Before sending/receiving messages, we need to tell clients the location of name servers. RocketMQ provides multiple ways to achieve this. For simplicity, we use environment variable NAMESRV\_ADDR

> **export NAMESRV\_ADDR=localhost:9876**

> **sh bin/tools.sh org.apache.rocketmq.example.quickstart.Producer**

SendResult [sendStatus=SEND\_OK, msgId= ...

> **sh bin/tools.sh org.apache.rocketmq.example.quickstart.Consumer**

ConsumeMessageThread\_%d Receive New Messages: [MessageExt...

# Shutdown Servers

> **sh bin/mqshutdown broker**

The mqbroker(36695) is running...

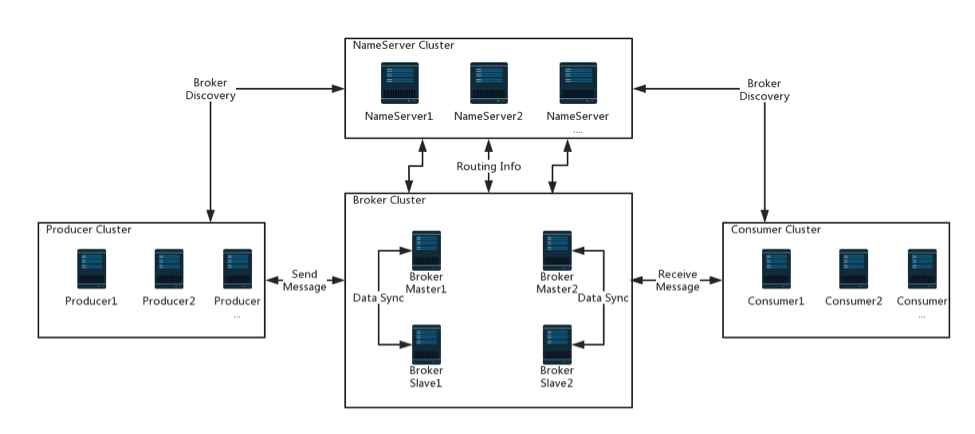
Send shutdown request to mqbroker(36695) OK

> **sh bin/mqshutdown namesrv**

The mqnamesrv(36664) is running...

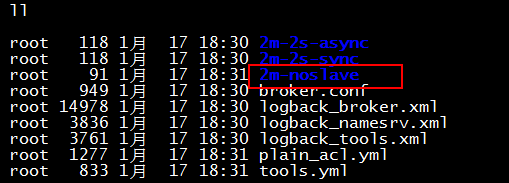
Send shutdown request to mqnamesrv(36664) OK

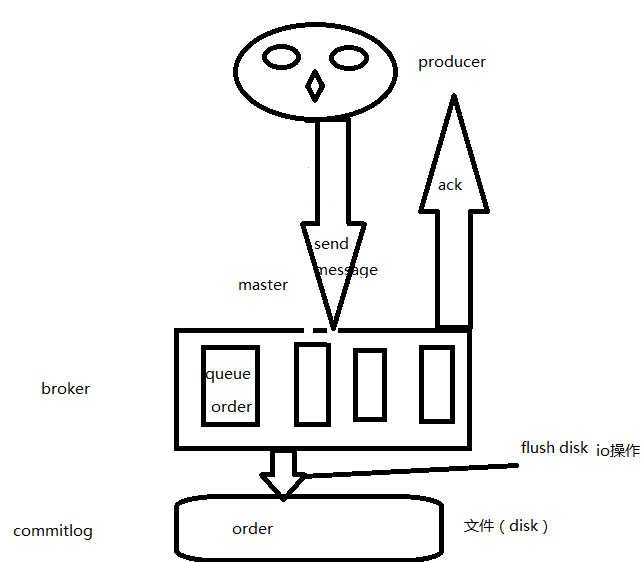
# 理论



# 集群部署

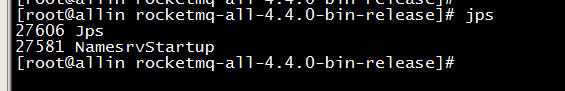
我们部署的模式是：





## 启动一个name server

**nohup sh bin/mqnamesrv &**

****

## 配置broker

**mkdir /rocketmq/2m-a /store –p**

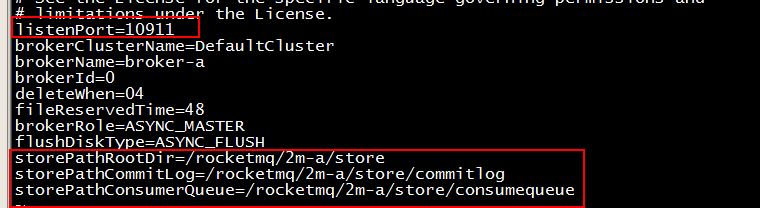
**mkdir /rocketmq/2m-a/ store/commitlog**

**mkdir /rocketmq/2m-a/ store/consumequeue**

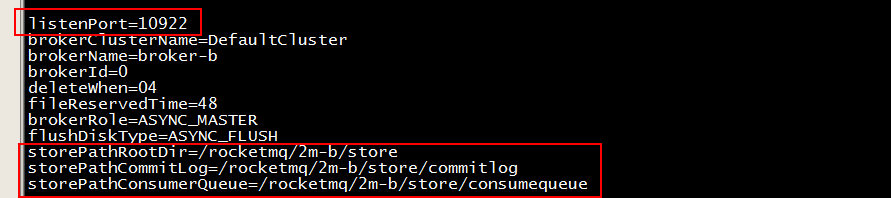
**mkdir /rocketmq/2m-b/store/consumequeue -p**

**mkdir /rocketmq/2m-b/ store/commitlog**

**vi 2m-noslave/broker-a.properties**

****

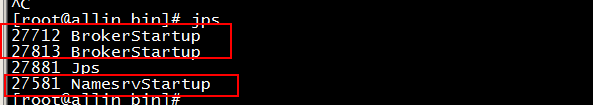
**vi ../conf/2m-noslave/broker-b.properties**



## 启动broker

**nohup bash mqbroker -c ../conf/2m-noslave/broker-a.properties -n 192.168.57.100:9876 &**

**nohup bash mqbroker -c ../conf/2m-noslave/broker-b.properties -n 192.168.57.100:9876 &**

****

## 发送消息

**export NAMESRV\_ADDR=localhost:9876**

**sh bin/tools.sh org.apache.rocketmq.example.quickstart.Producer**

## 接受消息

**sh bin/tools.sh org.apache.rocketmq.example.quickstart.Consumer**

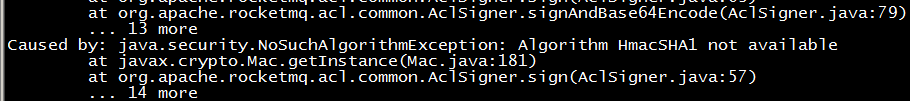
## 关闭集群

**sh bin/mqshutdown broker**

**sh bin/mqshutdown namesrv**

## 查看集群

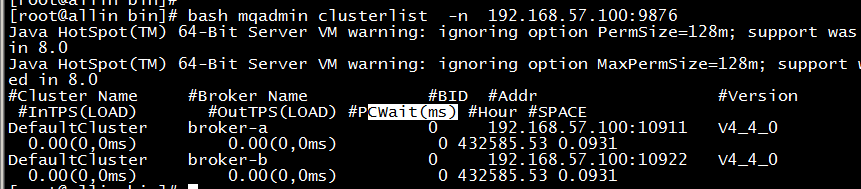
**bash mqadmin clusterlist -n 192.168.57.100:9876**

****

**如果出现以上问题，请配置java\_home环境变量：**

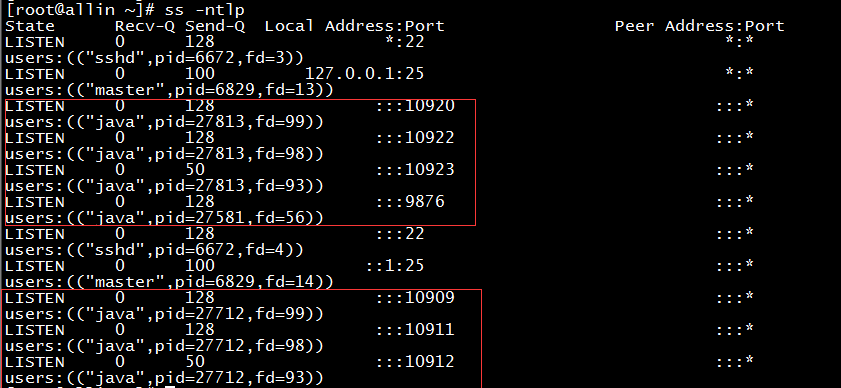
**export JAVA\_HOME=/usr/java/jdk1.8.0\_202-amd64**

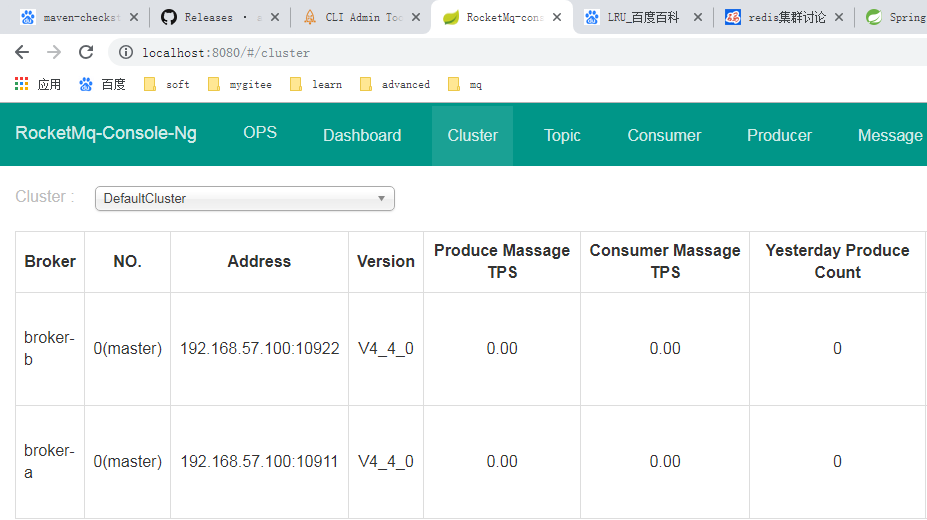
**bash mqadmin clusterlist -n 192.168.57.100:9876**

****

## 图形界面

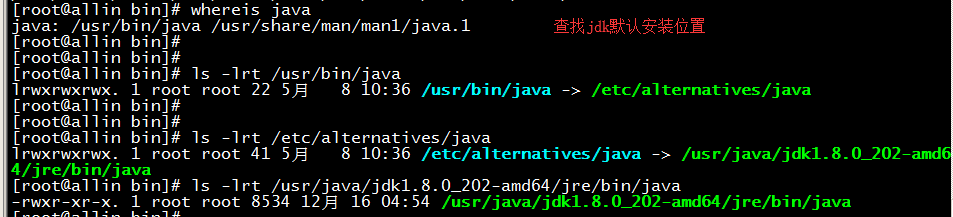
<https://github.com/apache/rocketmq-externals/releases>



****

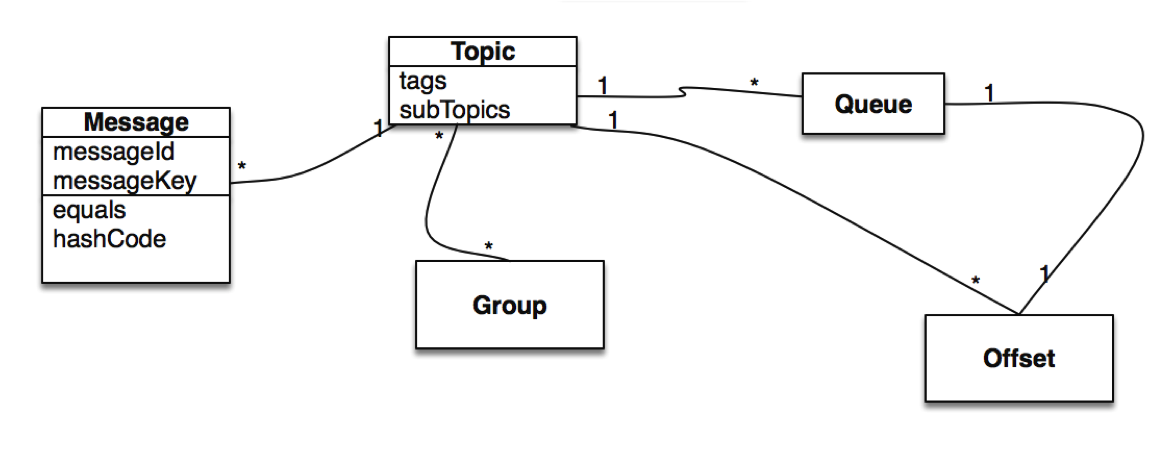
# 扩展知识

Jdk默认安装路径：

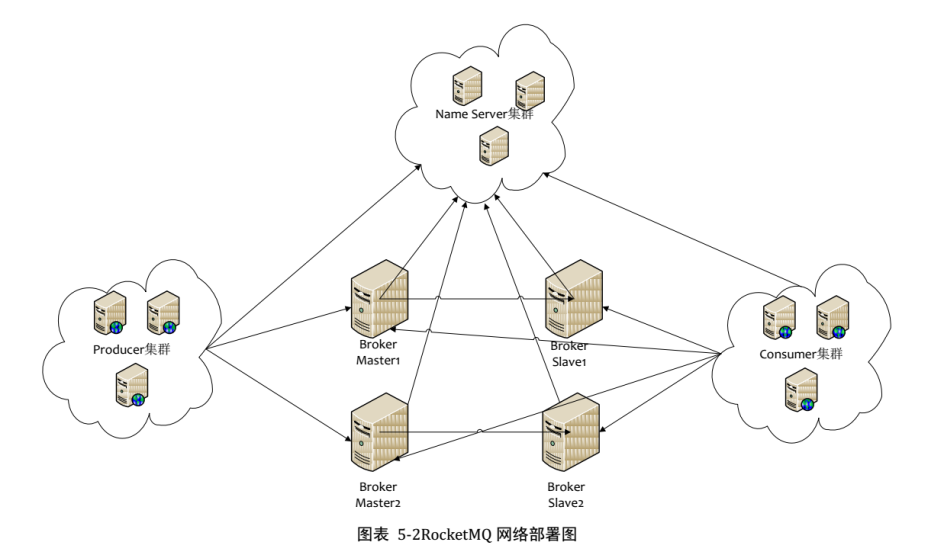


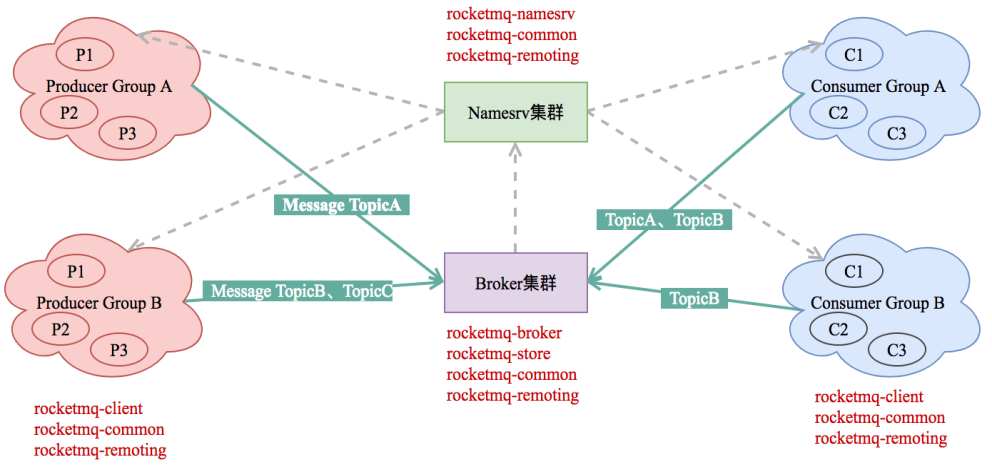


# 核心概念

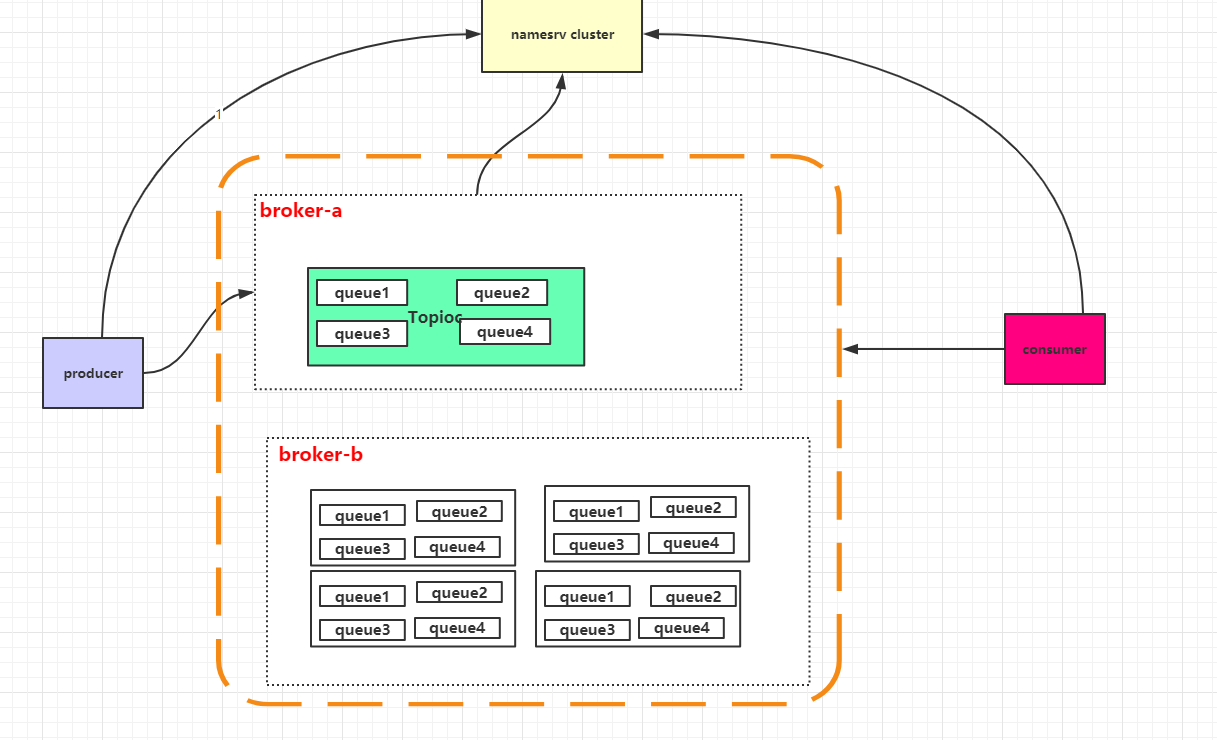


## 架构：





**整体认识：**远程通讯，发送消息，存储消息。



## 概念：

**Producer**：消息生产者，负责产生消息，一般由业务系统负责产生消息

**Consumer**：消息消费者，负责消费消息，一般是后台系统负责异步消费

**Topic：**消息主题，负责标记一类消息，生产者将消息发送到Topic，消费者从该Topic消费消息

**Broker：**消息中转角色，负责存储消息，转发消息，一般也称为 Server，在 JMS 规范中称为 Provider

**NameServer：**服务发现Server，用于生产者和消费者获取Broker的服务；

## Rocketmq模块划分：

|  |  |
| --- | --- |
| **名称** | **作用** |
| broker | broker模块：c和p端消息存储逻辑 |
| client | 客户端api：produce、consumer端 接受与发送api |
| common | 公共组件：常量、基类、数据结构 |
| tools | 运维tools：命令行工具模块 |
| store | 存储模块：消息、索引、commitlog存储 |
| namesrv | 服务管理模块：服务注册topic等信息存储 |
| remoting | 远程通讯模块：netty+fastjson |
| logappender | 日志适配模块 |
| example | Demo列子 |
| filtersrv | 消息过滤器模块 |
| srvutil | 辅助模块 |
| filter | 过滤模块：消息过滤模块 |
| distribution | 部署、运维相关zip包中的代码 |
| openmessaging | 兼容openmessaging分布式消息模块 |

# Rocketmq高可用：



# 面试：

# Producer端：

## 发送方式：

**Sync：**同步的发送方式，会等待发送结果后才返回

**Async：**异步的发送方式，发送完后，立刻返回。Client 在拿到 Broker 的响应结果后，会回调指定的 callback. 这个 API 也可以指定 Timeout，不指定也是默认的 3000ms.

**Oneway：**比较简单，发出去后，什么都不管直接返回。Ps:日志

## 发送结果：

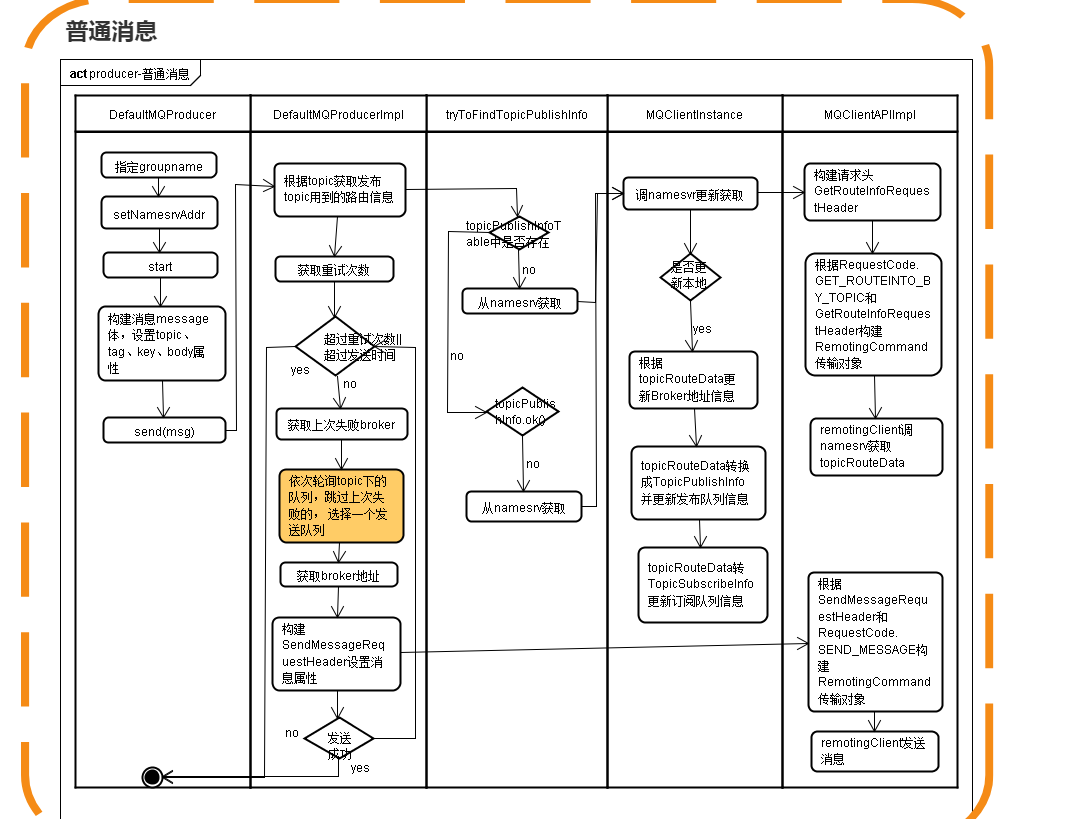
org.apache.rocketmq.client.producer.SendStatus

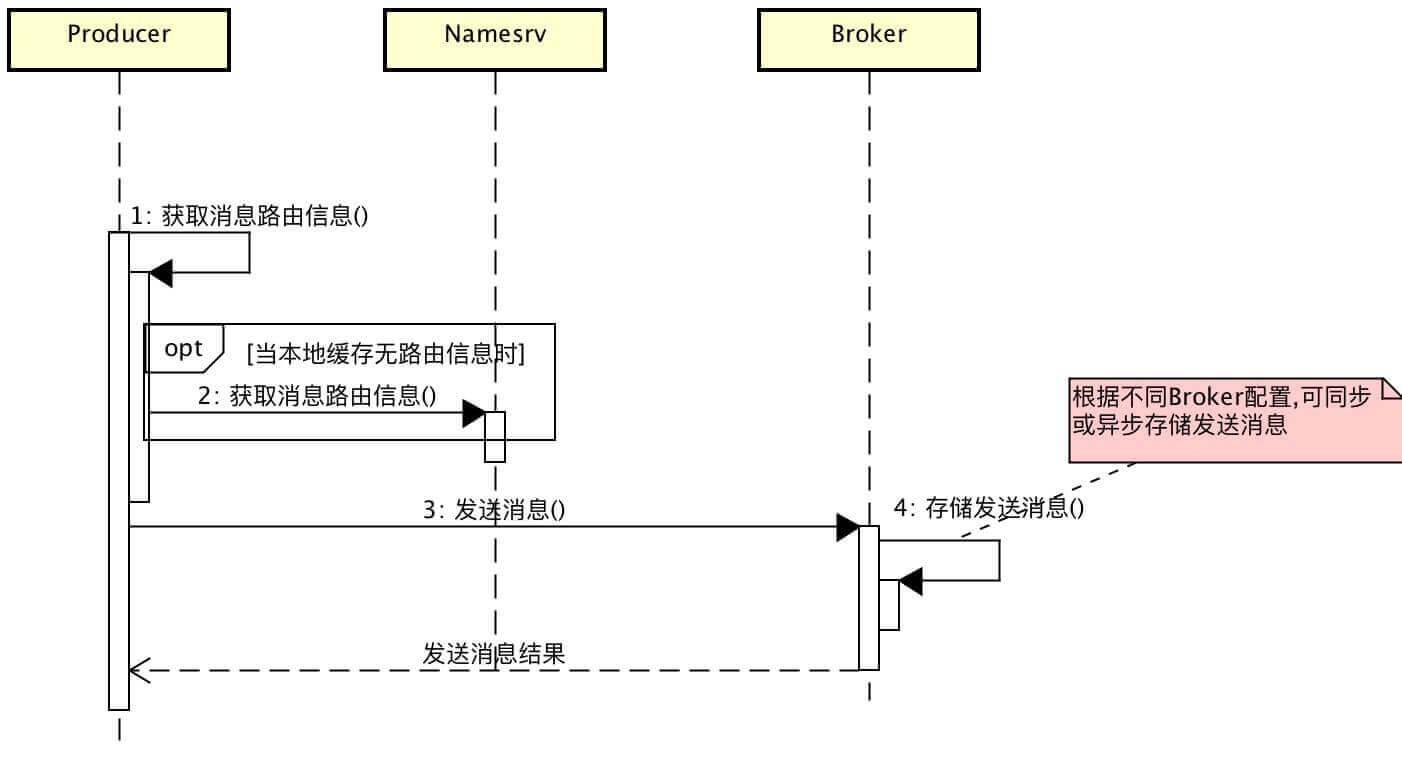
SEND\_OK,：消息发送成功   
FLUSH\_DISK\_TIMEOUT,消息发送成功，但是服务器刷盘超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失   
FLUSH\_SLAVE\_TIMEOUT,消息发送成功，但是服务器同步到 Slave 时超时，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢失   
SLAVE\_NOT\_AVAILABLE,

消息发送成功，但是此时 slave 不可用，消息已经进入服务器队列，只有此时服务器宕机，消息才会丢

## 普通消息：

org.apache.rocketmq.client.impl.producer.DefaultMQProducerImpl#sendDefaultImpl





1. 准备工作 mesasge、网络相关、线程相关
2. 从namesrv获取topic路由（缓存机制）
3. 组装数据，broker需要的序列化数据（json）
4. Netty发送（源码）

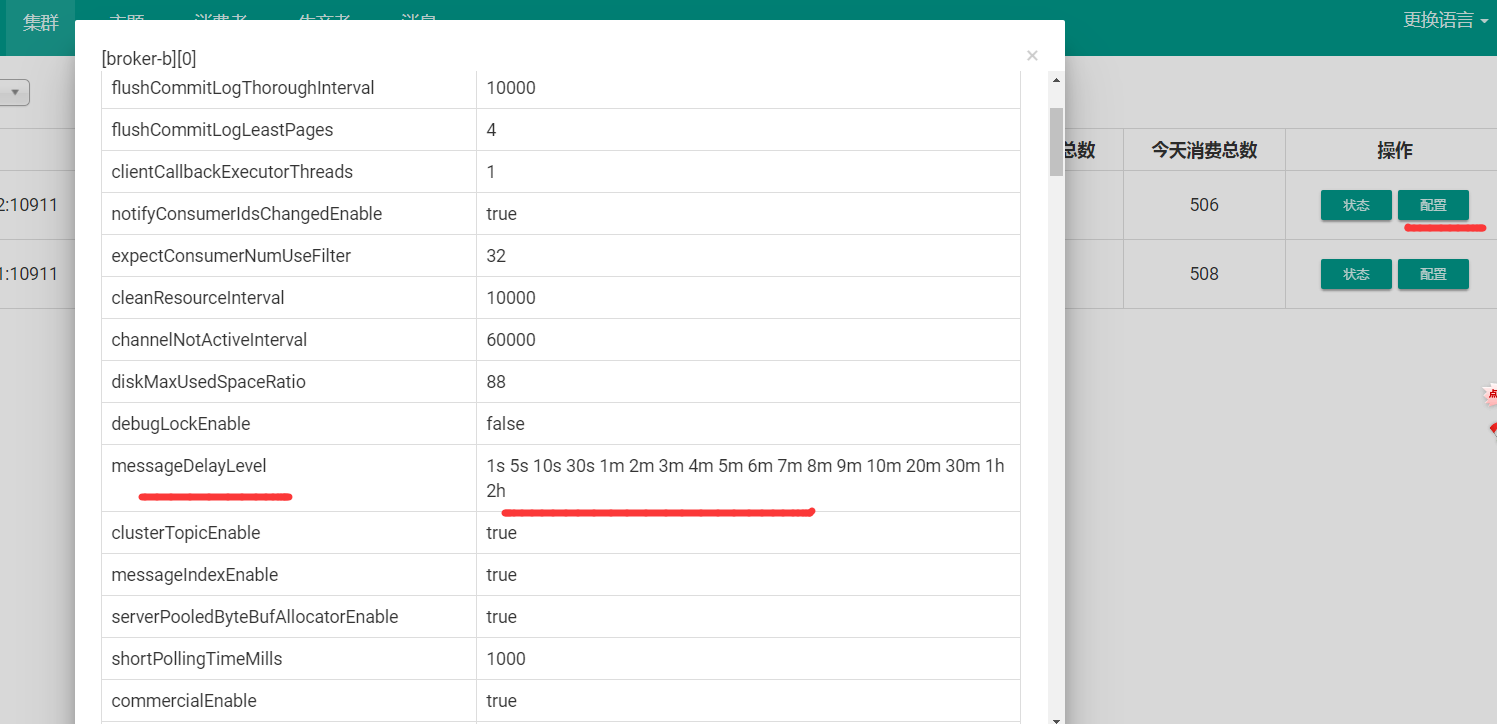
## 定时消息

定时消息是指消息发到 Broker 后，不能立刻被 Consumer 消费，要到特定的时间点或者等待特定的时间后才能被消费。（第三方 job 步长）

固定精度：

**1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h**

org.apache.rocketmq.store.config.MessageStoreConfig#messageDelayLevel



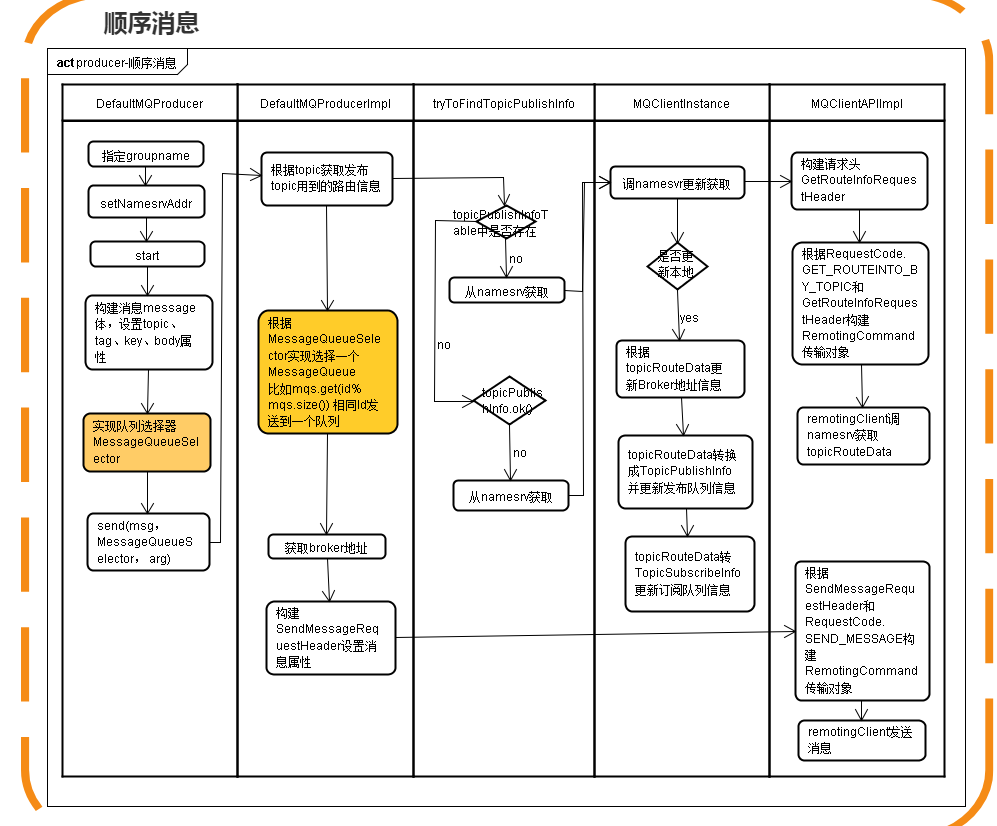
## 顺序消息：

org.apache.rocketmq.client.impl.producer.DefaultMQProducerImpl#sendSelectImpl

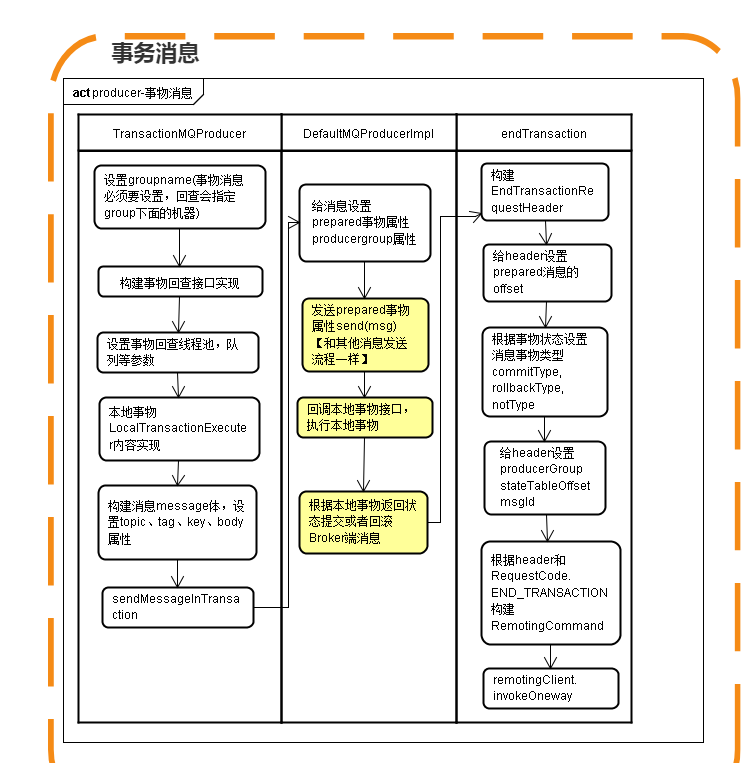
场景：订单》下单》支付》配送》签收

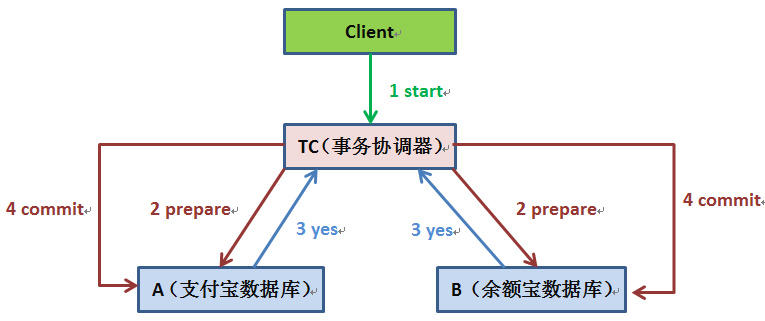
底层原理：4个队列，一个订单下面不同状态的消息是顺序的只需要发到一个队列中

org.apache.rocketmq.client.producer.MessageQueueSelector 如何选择一个队列

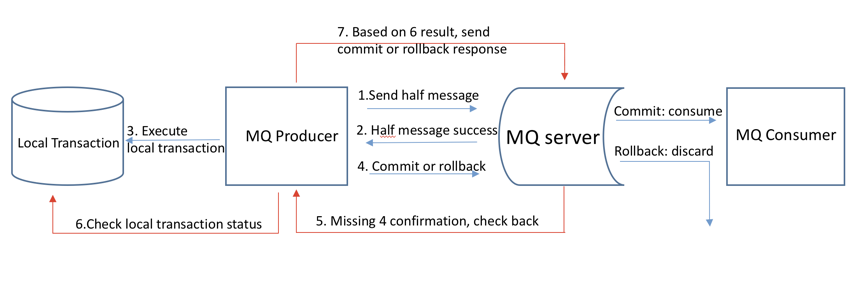


## 事务消息





事务：ACID xa 二段阶段 三阶段 tcc 阿里



http://rocketmq.apache.org/rocketmq/the-design-of-transactional-message/

# Consumer端：

## 消费模型：

org.apache.rocketmq.common.protocol.heartbeat.MessageModel#BROADCASTING

org.apache.rocketmq.common.protocol.heartbeat.MessageModel#CLUSTERING

## 消费选择：

org.apache.rocketmq.common.consumer.ConsumeFromWhere#CONSUME\_FROM\_LAST\_OFFSET

第一次启动从队列**最后**位置消费，后续再启动接着上次消费的进度开始消费

org.apache.rocketmq.common.consumer.ConsumeFromWhere#CONSUME\_FROM\_FIRST\_OFFSET

第一次启动从队列**初始位置**消费，后续再启动接着上次消费的进度开始消费

org.apache.rocketmq.common.consumer.ConsumeFromWhere#CONSUME\_FROM\_TIMESTAMP

第一次启动从**指定时间点位置**消费，后续再启动接着上次消费的进度开始消费

以上所说的第一次启动是指从来没有消费过的消费者，如果该消费者消费过，那么会在broker端记录该消费者的消费位置，如果该消费者挂了再启动，那么自动从上次消费的进度开始

## 消息重复幂等：

RocketMQ无法避免消息重复，所以如果业务对消费重复非常敏感，务必要在业务层面去重

Ps：见开发文档

**为什么要有组group：?**

sh mqshutdown broker

sh mqshutdown namesrv