Rocket 介绍:

发展历史

大约经历了三个主要版本迭代

一、Metaq 1.x ： 开源社区 killme2008 维护，开源社区非常活跃

二、Metaq 2.x ： 于 2012 年 10 月份在淘宝内部上线，并广泛使用

三、RocketMq3.x：阿里内部对其核心功能的简化。并衍生出多个消息服务项目。运用到阿里的支付、订单、充值等多个业务领域。

MQ 对比

	ActiveMQ	RabbitMQ	RocketMQ
关注度	高	高	中
成熟度	成熟	成熟	比较成熟
社区	Apache	Mozilla 开源社区	Alibaba
社区活跃度	高	高	中
文档	多	多	少
特点	功能齐全,被大量使用	由于 Erlang 语言的并发能力，使得性能很好	各个环节分布式扩展设计，主从高可用群集；支持上万个队列；多种消费模式；性能很好
开源	开源	开源	开源
语言	Java	Erlang(面向并发编程语言)	Java
Client 语言	支持 Java	支持 Java	支持 Java
支持协议	OpenWire、STOMP、REST、XMPP、AMQP	AMQP	自定义的一套，提供了支持 JMS 客户端的 API
持久化	内存,文件,数据库	内存,文件	磁盘文件
事务	支持	不支持	支持
集群和负载均衡	支持	支持	支持
管理页面	一般	好	无
部署方式	独立,嵌入	独立	独立
评价	优点：成熟的产品，已经在很多公司得到应用（非大规模场景）。有较多的文档。各种协议支持较好，	优点：由于 erlang 语言的特性，mq 性能较好；管理界面较丰富，在互联网公司也	优点：模型简单，接口易用。在阿里大规模应用。目前支付宝中的余额宝等新兴产品均使用 rocketmq。集群规模大概在 50 台左右，单日处理消息上

	有多重语言的成熟的客户端； 缺点：根据其他用户反馈，会出莫名其妙的问题，并且会丢消息。其重心放到 activemq6.0 产品—apollo 上去了，目前社区不活跃，且对 5.x 维护较少；**Activemq 不适合用于上千个队列的应用场景**	有较大规模的应用； 支持 amqp 协议，有多中语言且支持 amqp 的客户端可用。 缺点：erlang 语言难度较大。	百亿；性能非常好，可以大量堆积消息在 broker 中；支持多种消费，包括集群消费、广播消费等。开发度较活跃，版本更新很快。 缺点：**产品较新，文档比较缺乏。没有在 mq 核心中去实现 JMS 等接口，对已有系统而言不能兼容。**

RocketMQ 与 Kafka 对比

Kafka 无限消息堆积，高效的持久化速度，主要用于日志传输。
RocketMQ 广泛应用订单、交易、充值、消息推送、日志传输场景。

	RocketMQ	**Kafka**
数据可靠性	支持异步实时刷盘，同步刷盘，同步主从复制和异步主从复制	异步刷盘方式和异步主从复制
总结	总结：RocketMQ 的同步刷盘在单机可靠性上比 Kafka 更高，不会因为操作系统崩溃，导致数据丢失。同时同步 Replication 也比 Kafka 异步 Replication 更可靠，数据完全无单点。另外 Kafka 的 Replication 以 topic 为单位，支持主机宕机，备机自动切换，但是这里有个问题，由于是异步 Replication，那么切换后会有数据丢失，同时宕机的机器如果重启后，会与已经存在的主机器产生数据冲突（？）。开源版本的 RocketMQ 不支持 Master 宕机，Slave 自动切换为 Master，阿里云版本的 RocketMQ 支持自动切换特性	
性能	RocketMQ 单机写入 TPS 单实例约 7 万条/秒，单机部署 3 个 Broker，可以跑到最高 12 万条/秒，消息大小 10 个字节	Kafka 单机写入 TPS 约在百万条/秒，消息大小 10 个字节
总结	Kafka 的 TPS 跑到单机百万，主要是由于生产者端将多个小消息合并，批量发向 Broker。 RocketMQ 为什么没有这么做？ 生产者通常使用 Java 语言，缓存过多消息，GC 是个很严重的问题 生产者调用发送消息接口，消息未发送到 Broker，向业务返回成功，此时生产者宕机，会导致消息丢失，业务出错 生产者通常为分布式系统，且每台机器都是多线程发送，我们认为线上的系统单个生产者每秒产生的数据量有限，不可能上万。 缓存的功能完全可以由上层业务完成。	
单机支持队列数	RocketMQ 单机支持最高 5 万个队	Kafka 单机超过 64 个队列/分区，

	列，Load 不会发生明显变化	Load 会发生明显的飙高现象，队列越多，load 越高，发送消息响应时间变长
消息投递实时性	RocketMQ 使用长轮询，同 Push 方式实时性一致，消息的投递延时通常在几个毫秒。	Kafka 使用短轮询方式，实时性取决于轮询间隔时间
消息失败重试	RocketMQ 消费失败支持定时重试，每次重试间隔时间顺延	Kafka 消费失败不支持重试
总结	例如充值类应用，当前时刻调用运营商网关，充值失败，可能是对方压力过多，稍后在调用就会成功，如支付宝到银行扣款也是类似需求。这里的重试需要可靠的重试，即失败重试的消息不因为 Consumer 宕机导致丢失	
严格的消息顺序	RocketMQ 支持严格的消息顺序，在顺序消息场景下，一台 Broker 宕机后，发送消息会失败，但是不会乱序	Kafka 支持消息顺序，但是一台 Broker 宕机后，就会产生消息乱序
定时消息	RocketMQ 支持两类定时消息 开源版本 RocketMQ 仅支持定时 Level 阿里云 ONS 支持定时 Level，以及指定的毫秒级别的延时时间	Kafka 不支持定时消息
分布式事务消息	支持分布式事务消息	Kafka 不支持分布式事务消息
消息查询	RocketMQ 支持根据 Message Id 查询消息，也支持根据消息内容查询消息（发送消息时指定一个 Message Key，任意字符串，例如指定为订单 Id）	Kafka 不支持消息查询
总结	消息查询对于定位消息丢失问题非常有帮助，例如某个订单处理失败，是消息没收到还是收到处理出错了。	
消息回溯	RocketMQ 支持按照时间来回溯消息，精度毫秒，例如从一天之前的某时某分某秒开始重新消费消息	Kafka 理论上可以按照 Offset 来回溯消息
总结	典型业务场景如 consumer 做订单分析，但是由于程序逻辑或者依赖的系统发生故障等原因，导致今天消费的消息全部无效，需要重新从昨天零点开始消费，那么以时间为起点的消息重放功能对于业务非常有帮助。	
消息并行度	RocketMQ 消费并行度分两种情况 1.顺序消费方式并行度同 Kafka 完全一致 2.乱序方式并行度取决于 Consumer 的线程数，如 Topic 配置 10 个队列，10 台机器消费，每台机器 100 个线程，那么并行度为 1000。	Kafka 的消费并行度依赖 Topic 配置的分区数，如分区数为 10，那么最多 10 台机器来并行消费（每台机器只能开启一个线程），或者一台机器消费（10 个线程并行消费）。即消费并行度和分区数一致。

Broker 端消息过滤	RocketMQ 支持两种 Broker 端消息过滤方式 1.根据 Message Tag 来过滤，相当于子 topic 概念 2.向服务器上传一段 Java 代码，可以对消息做任意形式的过滤，甚至可以做 Message Body 的过滤拆分。	Kafka 不支持 Broker 端的消息过滤
消息堆积能力	理论上 Kafka 要比 RocketMQ 的堆积能力更强，不过 RocketMQ 单机也可以支持亿级的消息堆积能力，我们认为这个堆积能力已经完全可以满足业务需求	
成熟度	RocketMQ 在阿里集团内部有大量的应用在使用，每天都产生海量的消息，并且顺利支持了多次天猫双十一海量消息考验，是数据削峰填谷的利器。	Kafka 在日志领域比较成熟。

特性

1、支持严格的消息顺序；
2、支持 Topic 与 Queue 两种模式；
3、亿级消息堆积能力；
4、比较友好的分布式特性；
5、同时支持 Push 与 Pull 方式消费消息；

Topic 发布订阅模式
topic 数据默认不落地，是无状态的。并不保证 publisher 发布的每条数据，Subscriber 都能接受到。一般来说 publisher 发布消息到某一个 topic 时，只有正在监听该 topic 地址的 sub 能够接收到消息；如果没有 sub 在监听，该 topic 就丢失了。
一对多的消息发布接收策略，监听同一个 topic 地址的多个 sub 都能收到 publisher 发送的消息。Sub 接收完通知 mq 服务器。
Queue 点对点模式
Queue 数据默认会在 mq 服务器上以文件形式保存，也可以配置成 DB 存储。Queue 保证每条数据都能被 receiver 接收。Sender 发送消息到目标 Queue，receiver 可以异步接收这个 Queue 上的消息。Queue 上的消息如果暂时没有 receiver 来取，也不会丢失。一对一的消息发布接收策略，一个 sender 发送的消息，只能有一个 receiver 接收。receiver 接收完后，通知 mq 服务器已接收，mq 服务器对 queue 里的消息采取删除或其他操作。

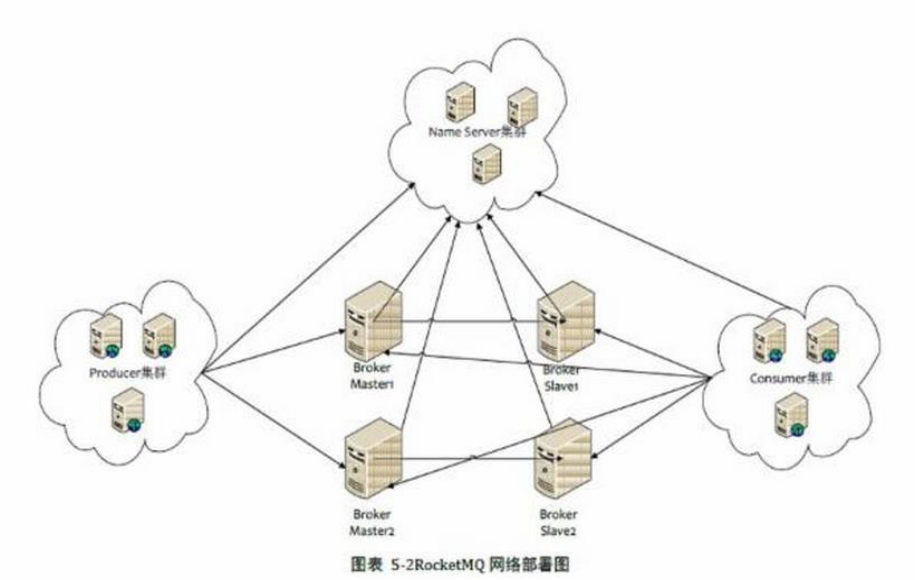
Push 推送
类似于 Broker Push 消息到 Consumer 方式，但实际仍然是 Consumer 内部后台从 Broker Pull 消息
采用长轮询方式拉消息，实时性同 push 方式一致，且不会无谓的拉消息导致 Broker、

Consumer 压力增大

Pull 拉取
短轮询方式，可以设定轮询时间间隔

版本 3.2.6

部署图



图表 5-2RocketMQ 网络部署图

RocketMq 主要包含三个服务：nameserver、broker、filterserver

Nameserver 负责维护 broker 列表和路由功能。
Broker 负责消息的读取和存储
Filterserver 负责过滤查询

1. nameserver:稳定性非常高。因为 nameserver 相互独立，彼此没有通信关系，单台 nameserver 挂掉，不影响其他 nameserver，即使全部挂掉，也不影响业务系统使用，这点类似于 dubbo 的 zookeeper。nameserver 不会有频繁的读写，所以性能开销非常小，稳定性很高。

2. broker

1) 与 nameserver 关系
连接：
单个 broker 和所有 nameserver 保持长连接
心跳：
心跳间隔：每隔 30 秒（此时间无法更改）向所有 nameserver 发送心跳，心跳包含了自

身的 topic 配置信息。

心跳超时：nameserver 每隔 10 秒钟（此时间无法更改），扫描所有还存活的 broker 连接，若某个连接 2 分钟内（当前时间与最后更新时间差值超过 2 分钟，此时间无法更改）没有发送心跳数据，则断开连接。

断开：

时机：broker 挂掉；心跳超时导致 nameserver 主动关闭连接

动作：一旦连接断开，nameserver 会立即感知，更新 topc 与队列的对应关系，但不会通知生产者和消费者

2) 负载均衡

一个 topic 分布在多个 broker 上，一个 broker 可以配置多个 topic，它们是多对多的关系。

如果某个 topic 消息量很大，应该给它多配置几个队列，并且尽量多分布在不同 broker 上，减轻某个 broker 的压力。

topic 消息量都比较均匀的情况下，如果某个 broker 上的队列越多，则该 broker 压力越大。

3) 可用性

由于消息分布在各个 broker 上，一旦某个 broker 宕机，则该 broker 上的消息读写都会受到影响。所以 rocketmq 提供了 master/slave 的结构，salve 定时从 master 同步数据，如果 master 宕机，则 slave 提供消费服务，但是不能写入消息，此过程对应用透明，由 rocketmq 内部解决。

这里有两个关键点：

一旦某个 broker master 宕机，生产者和消费者多久才能发现？受限于 rocketmq 的网络连接机制，默认情况下，最多需要 30 秒，但这个时间可由应用设定参数来缩短时间。这个时间段内，发往该 broker 的消息都是失败的，而且该 broker 的消息无法消费，因为此时消费者不知道该 broker 已经挂掉。

消费者得到 master 宕机通知后，转向 slave 消费，但是 slave 不能保证 master 的消息 100% 都同步过来了，因此会有少量的消息丢失。但是消息最终不会丢的，一旦 master 恢复，未同步过去的消息会被消费掉。

4) 可靠性

所有发往 broker 的消息，有同步刷盘和异步刷盘机制，总的来说，可靠性非常高

同步刷盘时，消息写入物理文件才会返回成功，因此非常可靠

异步刷盘时，只有机器宕机，才会产生消息丢失，broker 挂掉可能会发生，但是机器宕机崩溃是很少发生的，除非突然断电

5) 消息清理

扫描间隔：

默认 10 秒，由 broker 配置参数 cleanResourceInterval 决定

空间阈值：

物理文件不能无限制的一直存储在磁盘，当磁盘空间达到阈值时，不再接受消息，broker 打印出日志，消息发送失败，阈值为固定值 85%

清理时机：

默认每天凌晨 4 点，由 broker 配置参数 deleteWhen 决定；或者磁盘空间达到阈值

文件保留时长：

默认 72 小时，由 broker 配置参数 fileReservedTime 决定

6) 读写性能

文件内存映射方式操作文件，避免 read/write 系统调用和实时文件读写，性能非常高

永远一个文件在写，其他文件在读

顺序写，随机读

利用 linux 的 sendfile 机制，将消息内容直接输出到 sokect 管道，避免系统调用

7) 系统特性

大内存，内存越大性能越高，否则系统 swap 会成为性能瓶颈

IO 密集

cpu load 高，使用率低，因为 cpu 占用后，大部分时间在 IO WAIT

磁盘可靠性要求高，为了兼顾安全和性能，采用 RAID10 阵列

磁盘读取速度要求快，要求高转速大容量磁盘

3. 消费者

1) 与 nameserver 关系

连接：

单个消费者和一台 nameserver 保持长连接，定时查询 topic 配置信息，如果该 nameserver 挂掉，消费者会自动连接下一个 nameserver，直到有可用连接为止，并能自动重连。

心跳：

与 nameserver 没有心跳

轮询时间：

默认情况下，消费者每隔 30 秒从 nameserver 获取所有 topic 的最新队列情况，这意味着某个 broker 如果宕机，客户端最多要 30 秒才能感知。该时间由 DefaultMQPushConsumer 的 pollNameServerInteval 参数决定，可手动配置。

2) 与 broker 关系

连接

单个消费者和该消费者关联的所有 broker 保持长连接。

心跳

默认情况下，消费者每隔 30 秒向所有 broker 发送心跳，该时间由 DefaultMQPushConsumer 的 heartbeatBrokerInterval 参数决定，可手动配置。broker 每隔 10 秒钟（此时间无法更改），扫描所有还存活的连接，若某个连接 2 分钟内（当前时间与最后更新时间差值超过 2 分钟，此时间无法更改）没有发送心跳数据，则关闭连接，并向该消费者分组的所有消费者发出通知，分组内消费者重新分配队列继续消费

断开：

时机：消费者挂掉；心跳超时导致 broker 主动关闭连接

动作：一旦连接断开，broker 会立即感知到，并向该消费者分组的所有消费者发出通知，分组内消费者重新分配队列继续消费

3) 负载均衡

集群消费模式下，一个消费者集群多台机器共同消费一个 topic 的多个队列，一个队列只会被一个消费者消费。如果某个消费者挂掉，分组内其它消费者会接替挂掉的消费者继续消费。

4) 消费机制

本地队列

消费者不间断的从 broker 拉取消息，消息拉取到本地队列，然后本地消费线程消费本地消息队列，只是一个异步过程，拉取线程不会等待本地消费线程，这种模式实时性非常高。对消费者对本地队列有一个保护，因此本地消息队列不能无限大，否则可能会占用大量内存，本地队列大小由 DefaultMQPushConsumer 的 pullThresholdForQueue 属性控制，默认 1000，可手动设置。

轮询间隔

消息拉取线程每隔多久拉取一次？间隔时间由 DefaultMQPushConsumer 的 pullInterval 属性控制，默认为 0，可手动设置。

消息消费数量

监听器每次接受本地队列的消息是多少条？这个参数由 DefaultMQPushConsumer 的 consumeMessageBatchMaxSize 属性控制，默认为 1，可手动设置。

5) 消费进度存储

每隔一段时间将各个队列的消费进度存储到对应的 broker 上，该时间由 DefaultMQPushConsumer 的 persistConsumerOffsetInterval 属性控制，默认为 5 秒，可手动设置。

6) 如果一个 topic 在某 broker 上有 3 个队列，一个消费者消费这 3 个队列，那么该消费者和这个 broker 有几个连接？

一个连接，消费单位与队列相关，消费连接只跟 broker 相关，事实上，消费者将所有队列的消息拉取任务放到本地的队列，挨个拉取，拉取完毕后，又将拉取任务放到队尾，然后执行下一个拉取任务

4. 生产者

1) 与 nameserver 关系

连接：

单个生产者者和一台 nameserver 保持长连接，定时查询 topic 配置信息，如果该 nameserver 挂掉，生产者会自动连接下一个 nameserver，直到有可用连接为止，并能自动重连。

轮询时间：

默认情况下，生产者每隔 30 秒从 nameserver 获取所有 topic 的最新队列情况，这意味着某个 broker 如果宕机，生产者最多要 30 秒才能感知，在此期间，发往该 broker 的消息发送失败。该时间由 DefaultMQProducer 的 pollNameServerInteval 参数决定，可手动配置。

心跳：

与 nameserver 没有心跳

2) 与 broker 关系

连接：

单个生产者和该生产者关联的所有 broker 保持长连接。

心跳：

默认情况下，生产者每隔 30 秒向所有 broker 发送心跳，该时间由 DefaultMQProducer 的

heartbeatBrokerInterval 参数决定，可手动配置。broker 每隔 10 秒钟（此时间无法更改），扫描所有还存活的连接，若某个连接 2 分钟内（当前时间与最后更新时间差值超过 2 分钟，此时间无法更改）没有发送心跳数据，则关闭连接。
连接断开
移除 broker 上的生产者信息

3) 负载均衡
生产者时间没有关系，每个生产者向队列轮流发送消息

部署 Broker

Broker 集群有多种配置方式：
1，单 Master
优点：除了配置简单没什么优点
缺点：不可靠，该机器重启或宕机，将导致整个服务不可用
2，多 Master
一个集群无 Slave，全是 Master，例如 2 个 Master 或者 3 个 Master
优点：配置简单，消息也不会丢（异步刷盘丢失少量消息，同步刷盘一条不丢）性能最高
缺点：可能会有少量消息丢失（配置相关），单台机器重启或宕机期间，该机器下未被消费的消息在机器恢复前不可订阅，影响消息实时性
3，多 Master 多 Slave，每个 Master 配一个 Slave，有多对 Master-Slave，HA 采用异步复制方式，主备有短暂消息延迟，毫秒级
优点：性能同多 Master 几乎一样，实时性高，主备间切换对应用透明，不需人工干预
缺点：Master 宕机或磁盘损坏时会有少量消息丢失
4，多 Master 多 Slave，每个 Master 配一个 Slave，有多对 Master-Slave，HA 采用同步双写方式，主备都写成功，向应用返回成功
优点：服务可用性与数据可用性非常高
缺点：性能比异步 HA 略低，3.1.4 版本主宕备不能自动切换为主

Master 和 Slave 的配置文件参考 conf 目录下的配置文件
Master 与 Slave 通过指定相同的 brokerName 参数来配对，Master 的 BrokerId 必须是 0，Slave 的 BrokerId 必须是大于 0 的数
一个 Master 下面可以挂载多个 Slave，同一 Master 下的多个 Slave 通过指定不同的 BrokerId 来区分

1) 安装搭建环境：

软件安装包:
jdk-7u67-linux-x64.tar.gz

alibaba-rocketmq-3.1.8.tar.gz

1、 解压

2、 可执行权限：chmod +x mqadmin mqbroker mqfiltersrv mqshutdown mqnamesrv

3、 启动 mqnamesrv：mqnamesrv >/home/sre/alibaba-rocketmq/log/ns.log &

4、 启动 mqbroke：mqbroker >/home/sre/alibaba-rocketmq/log/mq.log &

关闭:

```
mqshutdown  namesrv
mqshutdown  broker
```

创建 top

sh mqadmin updateTopic -b 10.77.144.160:10911 -n 10.77.144.160:9876 -t top01

sh mqadmin updateTopic -cDefaultCluster -n 10.1.169.16:9876 -t top02

sh mqadmin updateTopic -b 10.1.169.238:10911 -n 10.1.169.16:9876 -t top03

注：http://blog.csdn.net/zhu_tianwei/article/details/40951301 命令整理

查询 topic 列表

sh mqadmin topicList –n 127.0.0.1:9876

删除 topic

sh mqadmin deleteTopic –n 10.1.169.16:9876 –c group_name1–t top003

sh mqadmin updateSubGroup–b 10.1.169.238:10911–g gn1 –n 10.1.169.16:9876

broker 地址:

#Cluster Name	#Broker Name	#BID	#Addr	#Version	#InTPS	#OutTPS
DefaultCluster	WDDS-DEV-016	0	10.1.169.238:10911	V3_1_3	0.00	0.00

删除 group

sh mqadmin deleteSubGroup -b 10.1.169.238:10911 -g please_rename_unique_group_name_4 -n 192.168.1.101:9876

2) Demo 测试

Push 推送模式：PushConsumer.java

Pull 拉取模式：PullConsumer.java

基本概念：

Topic：消息的逻辑管理单位；
Queue：消息的物理管理单位，一个 topic 下可以有多个 queue，Queue 的引入使得消息存储可以分布式集群化，

事务消息：这样的消息有多个状态，并且其发送是两个阶段，第一个阶段发送 PREPARED 状态的消息，此时 consumer 是看不见这种状态的消息的，发送完毕后回调用户的 TransactionExecutor 接口，执行相应的事务操作，当事务操作成功时，则对此条消息返回 commint，让 broker 对该消息执行 commit 操作，成为 commit 状态的消息对 consumer 是可见的。

ProducerGroup:
通常具有同样属性（处理的消息种类-topic、以及消息处理逻辑流程—分布式多个客户端）的一些 producer 可以归为同一个 group。在事务消息机制中，如果某条发送某条消息的 producer-A 宕机，使得事务消息一直处于 PREPARED 状态并超时，则 broker 会回查同一个 group 的其他 producer，确认这条消息应该 commit 还是 rollback。

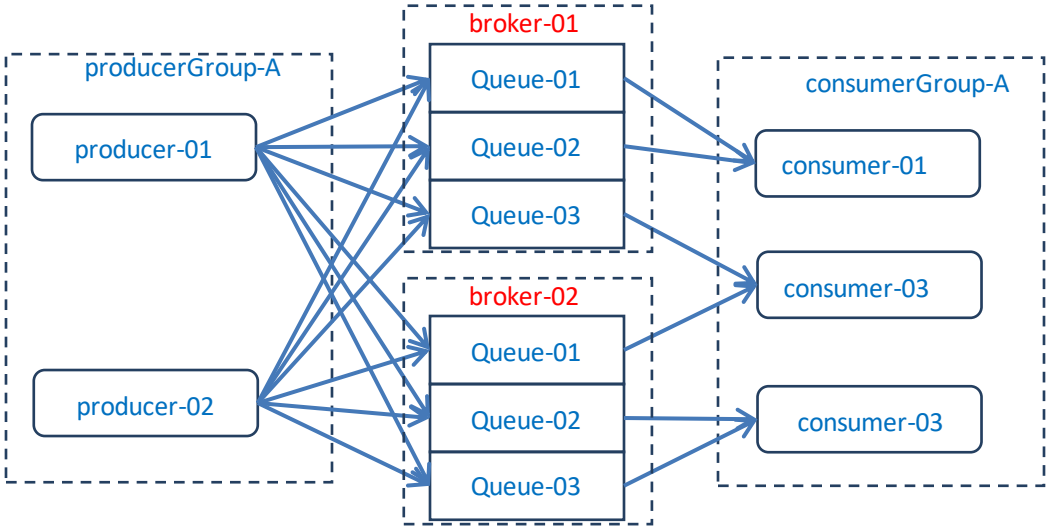
ConsumerGroup: 具有同样逻辑消费同样消息的 consumer，可以归并为一个 group。同一个 group 内的消费者，可以共同消费（CLUSTERING）对应 topic 的消息，达到分布式并行处理的功能。

消费属性管理：用户实现 MessageQueueSelector 为某一批消息（通常是有同样的唯一的标示 ID），选择同一个 Queue，则这一批消息的消费将是顺序消费（并由同一个 consumer 完成消费）。

基本原理
RocketMQ 以 Topic 来管理不同应用的消息。对于生产者而言，发送消息是，需要指定消息的 Topic，对于消费者而言，在启动后，需要订阅相应的 Topic，然后可以消费相应的消息。Topic 是逻辑上的概念，在物理实现上，一个 Topic 由多个 Queue 组成，采用多个 Queue 的好处是可以将 Broker 存储分布式化，提高系统性能。

RocketMQ 中，producer 将消息发送给 Broker 时，需要制定发送到哪一个队列中，默认情况下，producer 会轮询的将消息发送到每个队列中（所有 broker 下的 Queue 合并成一个 List 去轮询）。

对于 consumer 而言，会为每个 consumer 分配固定的队列（如果队列总数没有发生变化），consumer 从固定的队列中去拉取没有消费的消息进行处理。



广播消费（发布\订阅）：
一个消息被多个 Consumer 消费，即使这些 Consumer 属于同一个 Consumer Group，消息也会被 Consumer 都消费一次，广播消费中的 Consumer Group 概念可以认为在消息划分方面无意义。

顺序消费：
消费消息的顺序要同发送消息的顺序一致，在 RocketMQ 中，主要指的是局部顺序，即一类消息为满足顺序性，必须 Producer 单线程顺序发送，且发送到同一个队列，这样 Consumer 就可以按照 Producer 发送的顺序去消费消息。

普通顺序消费
顺序消息的一种，正常情况下可以保证完全的顺序消息，但是一旦发生通信异常，Broker 重启，由于队列总数发生变化，哈希取模后定位的队列会变化，产生短暂的消息顺序不一致。如果业务能容忍在集群异常情况（如某个 Broker 宕机或者重启）下，消息短暂的乱序，使用普通顺序方式比较合适。

严格顺序消费
顺序消息的一种，无论正常异常情况都能保证顺序，但是牺牲了分布式Failover特性，即Broker集群中只要有一台机器不可用，则整个集群都不可用，服务可用性大大降低。
如果服务器部署为同步双写模式，此缺陷可通过备机自动切换为主避免，不过仍然会存在几分钟的服务不可用。（依赖同步双写，主备自动切换，自动切换功能目前还未实现）
目前已知的应用只有数据库 binlog 同步强依赖严格顺序消息，其他应用绝大部分都可以容忍短暂乱序，推荐使用普通的顺序消息。

消息过滤：

Broker 端消息过滤
在 Broker 中，按照 Consumer 的要求做过滤，优点是减少了对于 Consumer 无用消息的网络传输。
缺点是增加了 Broker 的负担，实现相对复杂。
(1). 淘宝 Notify 支持多种过滤方式，包含直接按照消息类型过滤，灵活的语法表达式过滤，几乎可以满足最苛刻的过滤需求。
(2). 淘宝 RocketMQ 支持按照简单的 Message Tag 过滤，也支持按照 Message Header、body 进行过滤。
(3). CORBA Notification 规范中也支持灵活的语法表达式过滤。
Consumer 端消息过滤
这种过滤方式可由应用完全自定义实现，但是缺点是很多无用的消息要传输到 Consumer 端。

消息回溯：
回溯消费是指Consumer已经消费成功的消息，由于业务上需求需要重新消费，要支持此功能，Broker在向Consumer投递成功消息后，消息仍然需要保留。并且重新消费一般是按照时间维度，例如由于Consumer系统故障，恢复后需要重新消费1小时前的数据，那么Broker要提供一种机制，可以按照时间维度来回退消费进度。
RocketMQ 支持按照时间回溯消费，时间维度精确到毫秒，可以向前回溯，也可以向后回溯。

消息堆积：
消息堆积到持久化存储系统中，例如DB，KV存储，文件记录形式。
当消息不能在内存 Cache 命中时，要不可避免的访问磁盘，会产生大量读 IO，读 IO 的吞吐量直接决定了消息堆积后的访问能力。

分布式事务
已知的几个分布式事务规范，如XA，JTA等。其中XA规范被各大数据库厂商广泛支持，如Oracle，Mysql等。其中XA的TM实现佼佼者如Oracle Tuxedo，在金融、电信等领域被广泛应用。
分布式事务涉及到两阶段提交问题，在数据存储方面的方面必然需要KV存储的支持，因为第二阶段的提交回滚需要修改消息状态，一定涉及到根据Key去查找Message的动作。
RocketMQ在第二阶段绕过了根据Key去查找Message的问题，采用第一阶段发送Prepared消息时，拿到了消息的Offset，第二阶段通过Offset去访问消息，并修改状态，Offset就是数据的地址。
RocketMQ 这种实现事务方式，没有通过 KV 存储做，而是通过 Offset 方式，存在一个显著缺陷，即通过 Offset 更改数据，会令系统的脏页过多，需要特别关注。

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。
如果要支持任意的时间精度，在Broker层面，必须要做消息排序，如果再涉及到持久化，那么消息排序要不可避免的产生巨大性能开销。
RocketMQ 支持定时消息，但是不支持任意时间精度，支持特定的 level，例如定时 5s，10s，

1m 等。

消息重试:

客户端 API 形式：
DefaultMQProducer 、 TransactionMQProducer 、 DefaultMQPushConsumer 、 DefaultMQPullConsumer

客户端的公用配置：

pollNameServerInteval ： 轮询 NameServer 间隔时间，单位毫秒。
heartbeatBrokerInterval ： 向 Broker 发送心跳间隔时间，单位毫秒。

生产者配置：
sendMsgTimeout ： 发送消息超时时间，单位毫秒 。
maxMessageSize：客户端限制的消息大小，超过报错，同时服务端也会限制。

Push 消费者配置：

messageModel ： 消息模型，支持以下两种 1、集群消费 2、广播消费。
consumeMessageBatchMaxSize ： 批量消费，一次消费多少条消息。
pullBatchSize ： 批量拉消息，一次最多拉多少条

Pull 消费者配置：
messageModel ： 消息模型，支持以下两种 1、集群消费 2、广播消费 ，

Message 数据结构：

针对 Producer：
Topic：必填，线下环境不需要申请，线上环境需要申请后才能使用
Body：必填，二进制形式，序列化由应用决定，Producer 与 Consumer 要协商好序列化形式。
Tags ： 选填，类似于 Gmail 为每封邮件设置的标签，方便服务器过滤使用。
Keys ： 选填，代表这条消息的业务关键词，服务器会根据 keys 创建哈希索引，设置后，可以在 Console 系统根据 Topic、Keys 来查询消息，由于是哈希索引，请尽可能保证 key 唯一，例如订单号，商品 Id 等。
DelayTimeLevel：选填，消息延时级别，0 表示不延时，大于 0 会延时特定的时间才会被消费
WaitStoreMsgOK ： 选填，表示消息是否在服务器落盘后才返回应答

针对 Consumer：

MessageExt 在 Message 基础上增加了多个字段。

3) Broker 配置

autoCreateTopicEnable：是否允许 Broker 自动创建 Topic，建议线下开启，线上关闭
rejectTransactionMessage：是否拒绝事务消息接入
storePathConsumeQueue：消费队列存储路径
maxTransferCountOnMessageInMemory：单次 Pull 消息（内存）传输的最大条数

4) 发送顺序消息：
5) 顺序消费和乱顺消费：
6) 集群消费和广播消费
7) 消息发送失败重试
8) Broker 重启对客户端的影响
Broker 重启可能会导致正在发往这台机器的的消息发送失败，RocketMQ 提供了一种优雅关闭 Broker 的方法，通过执行以下命令会清除 Broker 的写权限，过 40s 后，所有客户端都会更新 Broker 路由信息，此时再关闭 Broker 就不会发生发送消息失败的情况，因为所有消息都发往了其他 Broker。
sh mqadmin wipeWritePerm -b brokerName -n namesrvAddr

零拷贝原理

Consumer消费消息过程，使用了零拷贝，零拷贝包含以下两种方式
1. 使用mmap + write方式 优点：即使频繁调用，使用小块文件传输，效率也很高 缺点：不能很好的利用DMA方式，会比sendfile多消耗CPU，内存安全性控制复杂，需要避免JVM Crash问题。

2. 使用sendfile方式 优点：可以利用DMA方式，消耗CPU较少，大块文件传输效率高，无内存安全新问题。 缺点：小块文件效率低于mmap方式，只能是BIO方式传输，不能使用NIO。

RocketMQ 选择了第一种方式，mmap+write 方式，因为有小块数据传输的需求，效果会比 sendfile 更好。

刷盘策略

RocketMQ 的所有消息都是持久化的，先写入系统 PAGECACHE，然后刷盘，可以保证内存与

磁盘都有一份数据，访问时，直接从内存读取。
异步刷盘：
在有RAID 卡，SAS 15000 转磁盘测试顺序写文件，速度可以达到300M 每秒左右，而线上的网卡一般都为千兆网卡，写磁盘速度明显快于数据网络入口速度，那么是否可以做到写完内存就向用户返回，由后台线程刷盘呢？
(1). 由于磁盘速度大于网卡速度，那么刷盘的进度肯定可以跟上消息的写入速度。

(2). 万一由于此时系统压力过大，可能堆积消息，除了写入IO，还有读取IO，万一出现磁盘读取落后情况， 会不会导致系统内存溢出，答案是否定的，原因如下：
a) 写入消息到PAGECACHE 时，如果内存不足，则尝试丢弃干净的PAGE，腾出内存供新消息使用，策略是LRU 方式。

b) 如果干净页不足，此时写入PAGECACHE 会被阻塞，系统尝试刷盘部分数据，大约每次尝试32 个PAGE， 来找出更多干净PAGE。

综上，内存溢出的情况不会出现。
同步刷盘：
同步刷盘与异步刷盘的唯一区别是异步刷盘写完PAGECACHE 直接返回，而同步刷盘需要等待刷盘完成才返回， 同步刷盘流程如下：
(1). 写入PAGECACHE 后，线程等待，通知刷盘线程刷盘。

(2). 刷盘线程刷盘后，唤醒前端等待线程，可能是一批线程。

(3). 前端等待线程向用户返回成功。

消息查询

按照 Message Id 查询消息
按照 Message Key 查询消息

服务器消息过滤

RocketMQ 的消息过滤方式有别于其他消息中间件，是在订阅时，再做过滤，先来看下Consume Queue 的存储结构。
在Broker 端进行Message Tag 比对，先遍历Consume Queue，如果存储的Message Tag 与订阅的Message Tag 不符合，则跳过，继续比对下一个，符合则传输给Consumer。注意：Message Tag 是字符串形式，Consume Queue 中存储的是其对应的hashcode，比对时也是比对hashcode。

Consumer 收到过滤后的消息后，同样也要执行在Broker 端的操作，但是比对的是真实的

Message Tag 字符串，而不是Hashcode。

消费堆积问题

在有 Slave 情况下，Master 一旦发现 Consumer 访问堆积在磁盘的数据时，会向 Consumer 下达一个重定向指令，令 Consumer 从 Slave 拉取数据，这样正常的发消息与正常消费的 Consumer 都不会因为消息堆积受影响，因为系统将堆积场景与非堆积场景分割在了两个不同的节点处理。这里会产生另一个问题，Slave 会不会写性能下降，答案是否定的。因为 Slave 的消息写入只追求吞吐量，不追求实时性，只要整体的吞吐量高就可以，而 Slave 每次都是从 Master 拉取一批数据，如 1M，这种批量顺序写入方式即使堆积情况，整体吞吐量影响相对较小，只是写入 RT 会变长

问题总结：

1、 发送消息注意事项；

2、 消息发送失败如何处理

3、 发送顺序消息注意

4、 消费失败处理方式

5、 消费速度慢处理方式

6、 批量方式消费

7、 跳过非重要消息