



微信搜一搜

Java架构师进阶编程

<http://jm.taobao.org/2017/01/12/rocketmq-quick-start-in-10-minutes/>

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, 消溯等功能没有提供, 是为大数据准备的, 数据领域应用广。

1. 中小型公司首选RabbitMQ: 管理界面简单, 高并发。
2. 大型公司可以选择RocketMQ: 更高并发, 可对rocketmq进行定制化开发。
3. 日志采集功能, 首选kafka, 专为大数据准备。

1. 消息可靠性:

影响消息可靠性的情况:

- Broker正常关闭
- Broker异常Crash
- OS Crash
- 机器掉电, 但是能立即恢复供电情况。
- 机器无法开机 (可能是cpu、主板、内存等关键设备损坏)
- 磁盘设备损坏。

1、(1)、(2)、(3)、(4)四种情况都属于硬件资源可立即恢复情况, RocketMQ在这四种情况下能保证消息不丢, 或者丢失少量数据 (依赖刷盘方式是同步还是异步)。

2、(5)、(6)属于单点故障, 且无法恢复, 一旦发生, 在此单点上的消息全部丢失。RocketMQ在这两种情况下, 通过异步复制, 可保证99%的消息不丢, 但是仍然会有极少量的消息可能丢失。通过同步双写技术可以完全避免单点, 同步双写势必会影响性能, 适合对消息可靠性要求极高的场合, 例如与Money相关的应用。

2. 消息低延迟:

在消息不堆积情况下, 消息到达Broker后, 能立刻到达Consumer。RocketMQ使用长轮询Pull方式, 可保证消息非常实时, 消息实时性不低于Push。

3. 每个消息至少投递一次:

RocketMQ Consumer先pull消息到本地, 消费完成后, 才向服务器返回ack, 如果没有消费一定不会ack消息, 所以RocketMQ可以很好的支持此特性。

4. 每个消息只消费一次:

1、前提:

- 发送消息阶段, 不允许发送重复的消息。
- 消费消息阶段, 不允许消费重复的消息。

2、只有以上两个条件都满足情况下, 才能认为消息是“Exactly Once”, 而要实现以上两点, 在分布式系统环境下, 不可避免要产生巨大的开销。所以RocketMQ为了追求高性能, 并不保证此特性, 要求在业务上进行去重, 也就是说消费消息要做到幂等性。RocketMQ虽然不能严格保证不重复, 但是正常情况下很少会出现重复发送、消费情况, 只有网络异常, Consumer启停等异常情况下会出现消息重复。

5. Broker的Buffer满了怎么办?

a. 下面是CORBA Notification规范中处理方式:

- RejectNewEvents 拒绝新的消息, 向Producer返回RejectNewEvents错误码。
- 按照特定策略丢弃已有消息
 - AnyOrder - Any event may be discarded on overflow. This is the default setting for this property.
 - FifoOrder - The first event received will be the first discarded.
 - LifoOrder - The last event received will be the first discarded.

4. PriorityOrder - Events should be discarded in priority order, such that lower priority events will be discarded before higher priority events.

5. DeadlineOrder - Events should be discarded in the order of shortest expiry deadline first.

b. RocketMQ没有内存Buffer概念, RocketMQ的队列都是持久化磁盘, 数据定期清除。

对于此问题的解决思路, RocketMQ同其他MQ有非常显著的区别, RocketMQ的内存Buffer抽象成一个无限长度的队列, 不管有多少数据进来都能装得下, 这个无限是有前提的, Broker会定期删除过期的数据, 例如Broker只保存3天的消息, 那么这个Buffer虽然长度无限, 但是3天前的数据会被从队尾删除。

此问题的本质原因是网络调用存在不确定性, 即既不成功也不失败的第三种状态, 所以才产生了消息重复性问题。

6. 回溯消息:

a. 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。
b. 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。

7. 消息堆积:

a. 消息中间件的主要功能是异步解耦, 还有个重要功能是挡住前端的数据洪峰, 保证后端系统的稳定性, 这就要求消息中间件具有一定的消息堆积能力, 消息堆积分以下两种情况:

i. 消息堆积在内存Buffer, 一旦超过内存Buffer, 可以根据一定的丢弃策略来丢弃消息, 如CORBA Notification规范中描述。适合能容忍丢弃消息的业务, 这种情况消息的堆积能力主要在于内存Buffer大小, 而且消息堆积后, 性能下降不会太大, 因为内存中数据多少对于对外提供的访问能力影响有限。
ii. 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。

b. 评估消息堆积能力主要有以下四点:

i. 消息能堆积多少条, 多少字节? 即消息的堆积容量。
ii. 消息堆积后, 发消息的吞吐量大小, 是否会受堆积影响?
iii. 消息堆积后, 正常消费的Consumer是否会受影响?
iv. 消息堆积后, 访问堆积在磁盘的消息时, 吞吐量有多大?

8. 分布式事务:

1. 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。

2. 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。

3. 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。

9. 定时消息:

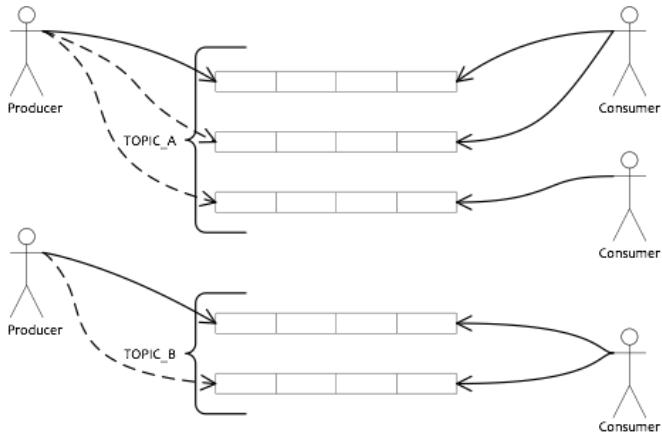
a. 定时消息是指消息发到Broker后, 不能立刻被Consumer消费, 要到特定的时间点或者等待特定的时间后才能被消费。
b. 如果要支持任意的时间精度, 在Broker层面, 必须要做消息排序, 如果再涉及到持久化, 那么消息排序要不可避免的产生巨大性能开销。
c. RocketMQ支持定时消息, 但是不支持任意时间精度, 支持特定的level, 例如定时5s, 10s, 1m等。

10. 消息重试:

Consumer消费消息失败后, 要提供一种重试机制, 令消息再消费一次。Consumer消费消息失败通常可以认为有以下几种情况:

i. 由于消息本身的原因, 例如反序列化失败, 消息数据本身无法处理 (例如话费充值, 当前消息的手机号被注销, 无法充值) 等。这种错误通常需要跳过这条消息, 再消费其他消息, 而这条失败的消息即使立刻重试消费, 99%也不成功, 所以最好提供一种定时重试机制, 即过10s秒后再重试。
ii. 由于依赖的下游应用服务不可用, 例如db连接不可用, 外系统网络不可达等。遇到这种错误, 即使跳过当前失败的消息, 消费其他消息同样也会报错。这种情况建议应用sleep 30s, 再消费下一条消息, 这样可以减轻Broker重试消息的压力。

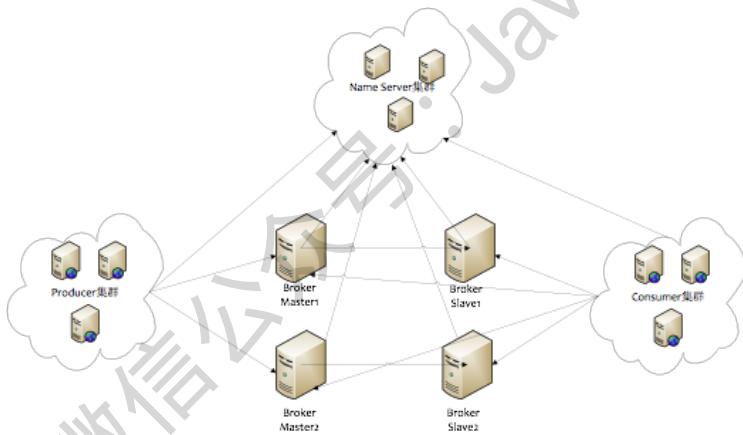
11. RocketMq是什么？



上图是一个典型的消息中间件收发消息的模型，RocketMQ也是这样的设计，简单说来，RocketMQ具有以下特点：

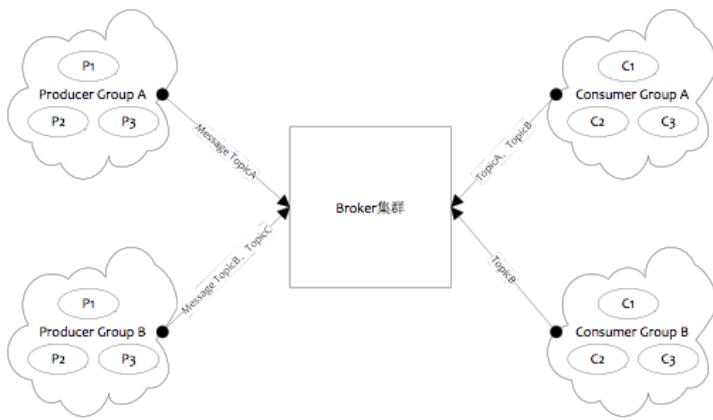
- 是一个队列模型的消息中间件，具有高性能、高可靠、高实时、分布式特点。
- Producer、Consumer、队列都可以分布式。
- Producer向一些队列轮流发送消息，队列集合称为Topic， Consumer如果做广播消费，则一个consumer实例消费这个Topic对应的所有队列，如果做集群消费，则多个Consumer实例平均消费这个topic对应的队列集合。
- 能够保证严格的消息顺序
- 提供丰富的消息拉取模式
- 高效的订阅者水平扩展能力
- 实时的消息订阅机制
- 亿级消息堆积能力
- 较少的依赖

12. RocketMq物理部署结构



- Name Server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。
- 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的对应关系通过指定相同的BrokerName，不同的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Name Server集群中的所有节点建立长连接，定时注册Topic信息到所有Name Server。
- Producer与Name Server集群中的其中一个节点（随机选择）建立长连接，定期从Name Server取Topic路由信息，并向提供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，可集群部署。
- Consumer与Name Server集群中的其中一个节点（随机选择）建立长连接，定期从Name Server取Topic路由信息，并向提供Topic服务的Master、Slave建立长连接，且定时向Master、Slave发送心跳。Consumer既可以从Master订阅消息，也可以从Slave订阅消息，订阅规则由Broker配置决定。

13. RocketMq逻辑结构



1. Producer Group

用来表示一个发送消息应用，一个Producer Group下包含多个Producer实例，可以是多台机器，也可以是一台机器的多个进程，或者一个进程的多个Producer对象。一个Producer Group可以发送多个Topic消息，Producer Group作用如下：

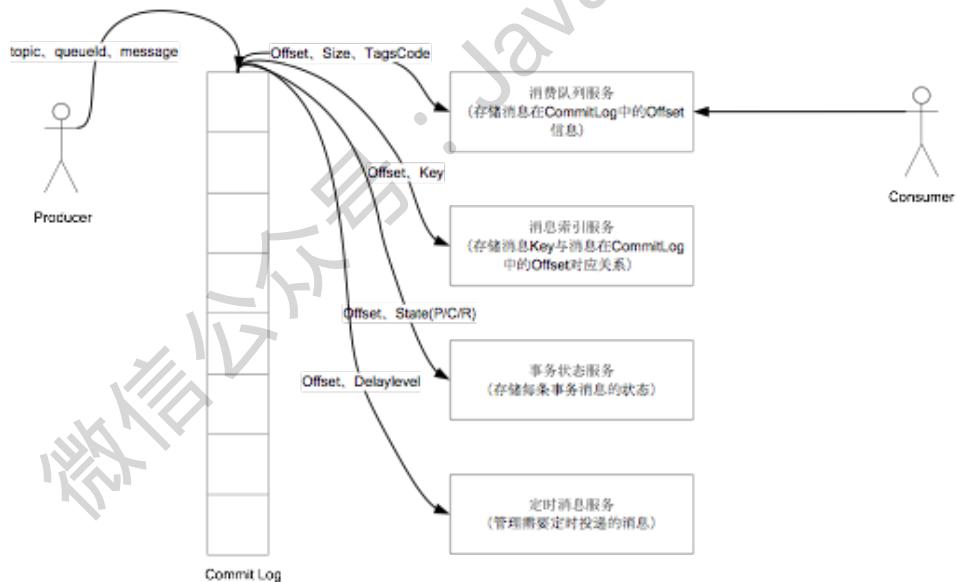
1. 标识一类Producer

1. 可以通过运维工具查询这个发送消息应用下有多个Producer实例
2. 发送分布式事务消息时，如果Producer中途意外宕机，Broker会主动回调Producer Group内的任意一台机器来确认事务状态。

2. Consumer Group

用来表示一个消费消息应用，一个Consumer Group下包含多个Consumer实例，可以是多台机器，也可以是多个进程，或者是一个进程的多个Consumer对象。一个Consumer Group下的多个Consumer以均摊方式消费消息，如果设置为广播方式，那么这个Consumer Group下的每个实例都消费全量数据。

14. RocketMq数据存储结构



如上图所示，RocketMQ采取了一种数据与索引分离的存储方法。有效降低文件资源、IO资源，内存资源的损耗。即便是阿里这种海量数据，高并发场景也能够有效降低端到端延迟，并具备较强的横向扩展能力。