RocketMQ封神之旅（一）-消息中间件需要解决的问题及RocketMQ发展历程

原创

gonghaiyu 于 2021-06-09 23:54:46 发布 40 收藏

分类专栏： RocketMQ

本文链接：https://blog.csdn.net/gonghaiyu/article/details/117756837

专业术语

 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。
 Push Consumer
Consumer的一种，应用通常向Consumer对象注册一个Listener接口，一旦收到消息，Consumer对象立 刻回调Listener接口方法。
 Pull Consumer
Consumer的一种，应用通常主动调用Consumer的拉消息方法从Broker拉消息，主动权由应用控制。
 Producer Group
一类Producer的集合名称，这类Producer通常发送一类消息，且发送逻辑一致。
 Consumer Group
一类Consumer的集合名称，这类Consumer通常消费一类消息，且消费逻辑一致。
 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。
 广播消费
一条消息被多个Consumer消费，即使这些Consumer属于同一个Consumer Group，消息也会被Consumer Group中的每个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。
在CORBA Notification规范中，消费方式都属于广播消费。
 集群消费
一个Consumer Group中的Consumer实例平均分摊消费消息。例如某个Topic有9条消息，其中一个Consumer Group有3个实例（可能是3个进程，或者3台机器），那么每个实例只消费其中的3条消息。
 顺序消息
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。
 普通顺序消息
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。
 严格顺序消息
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。
 Message Queue
在RocketMQ中，所有消息队列都是持久化，长度无限的数据结构，所谓长度无限是指队列中的每个存储单元都是定长，访问其中的存储单元使用Offset来访问，offset为java long类型，64位，理论上在100年内不会溢出，所以认为是长度无限，另外队列中只保存最近几天的数据，之前的数据会按照过期时间来删除。
也可以认为Message Queue是一个长度无限的数组，offset就是下标。

消息中间件需要解决的问题

本节阐述消息中间件通常需要解决哪些问题，在解决这些问题当中会遇到什么困难，RocketMQ是否可以解决，规范中如何定义这些问题。

Publish/Subscribe

发布订阅是消息中间件的最基本功能，也是相对于传统RPC通信而言。在此不再详述。

Message Priority

规范中描述的优先级是指在一个消息队列中，每条消息都有不同的优先级，一般用整数来描述，优先级高的消息先投递，如果消息完全在一个内存队列中，那么在投递前可以按照优先级排序，令优先级高的先投递。
由于RocketMQ所有消息都是持久化的，所以如果按照优先级来排序，开销会非常大，因此RocketMQ没有特意支持消息优先级，但是可以通过变通的方式实现类似功能，即单独配置一个优先级高的队列，和一个普通优先级的队列， 将不同优先级发送到不同队列即可。
对于优先级问题，可以归纳为2类

1. 只要达到优先级目的即可，不是严格意义上的优先级，通常将优先级划分为高、中、低，或者再多几个级别。每个优先级可以用不同的topic表示，发消息时，指定不同的topic来表示优先级，这种方式可以解决绝大部分的优先级问题，但是对业务的优先级精确性做了妥协。
2. 严格的优先级，优先级用整数表示，例如0～65535，这种优先级问题一般使用不同topic解决就非常不合适。如果要让MQ解决此问题，会对MQ的性能造成非常大的影响。这里要确保一点，业务上是否确实需要这种严格的优先级，如果将优先级压缩成几个，对业务的影响有多大？

Message Order

消息有序指的是一类消息消费时，能按照发送的顺序来消费。例如：一个订单产生了3条消息，分别是订单创建，订单付款，订单完成。消费时，要按照这个顺序消费才能有意义。但是同时订单之间是可以并行消费的。
RocketMQ可以严格的保证消息有序。

Message Filter

□ 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只支持按照简单的Message Tag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。

Message Persistence

几种持久化方式：
(1). 持久化到数据库，例如Mysql。
(2). 持久化到KV存储，例如levelDB、伯克利DB等KV存储系统。
(3). 文件记录形式持久化，例如Kafka，RocketMQ
(4). 对内存数据做一个持久化镜像，例如beanstalkd，VisiNotify
(1)、(2)、(3)三种持久化方式都具有将内存队列Buffer进行扩展的能力，(4)只是一个内存的镜像，作用是当Broker挂掉重启后仍然能将之前内存的数据恢复出来。
JMS与CORBA Notification规范没有明确说明如何持久化，但是持久化部分的性能直接决定了整个消息中间件的性能。

Message Reliablity

影响消息可靠性的几种情况：
(1). Broker正常关闭
(2). Broker异常Crash
(3). OS Crash
(4). 机器掉电，但是能立即恢复供电情况。
(5). 机器无法开机（可能是cpu、主板、内存等关键设备损坏）
(6). 磁盘设备损坏。
(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）。
(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。
RocketMQ从3.0版本开始支持同步双写。

Low Latency Messaging

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。

At least Once

是指每个消息必须投递一次
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。

Exactly Only Once

(1). 发送消息阶段，不允许发送重复的消息。
(2). 消费消息阶段，不允许消费重复的消息。
只有以上两个条件都满足情况下，才能认为消息是“Exactly Only 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。
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。

Broker的Buffer满了怎么办？

Broker的Buffer通常指的是Broker中一个队列的内存Buffer大小，这类Buffer通常大小有限，如果Buffer满了以后怎么办？
下面是CORBA Notification规范中处理方式：
(1). RejectNewEvents
拒绝新来的消息，向Producer返回RejectNewEvents错误码。
(2). 按照特定策略丢弃已有消息
a) AnyOrder - Any event may be discarded on overflow. This is the default setting for this property.
b) FifoOrder - The first event received will be the first discarded.
c) LifoOrder - The last event received will be the first discarded.
d) PriorityOrder - Events should be discarded in priority order, such that lower priority events will be discarded before higher priority events.
e) DeadlineOrder - Events should be discarded in the order of shortest expiry deadline first.
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。

回溯消费

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：
(1). 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。
(2). 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。
评估消息堆积能力主要有以下四点：
(1). 消息能堆积多少条，多少字节？即消息的堆积容量。
(2). 消息堆积后，发消息的吞吐量大小，是否会受堆积影响？
(3). 消息堆积后，正常消费的Consumer是否会受影响？
(4). 消息堆积后，访问堆积在磁盘的消息时，吞吐量有多大？

消息存储

消息中间件的一个核心实现是消息的存储，对消息存储一般有如下两个维度的考量：消息堆积能力和消息存储性能。RocketMQ追求消息存储的高性能，引入内存映射机制，所有主题的消息顺序存储在同一个文件中。同时为了避免消息无限在消息存储服务器中累积，引入了消息文件过期机制与文件存储空间报警机制。

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

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

1. 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。
2. 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。

RocketMQ产品发展历史

大约经历了三个主要版本迭代。

Metaq（Metamorphosis） 1.x

由开源社区killme2008维护，开源社区非常活跃。
https://github.com/killme2008/Metamorphosis

Metaq 2.x

于2012年10月份上线，在淘宝内部被广泛使用。

RocketMQ 3.x

基于公司内部开源共建原则， RocketMQ项目只维护核心功能，且去除了所有其他运行时依赖，核心功能最简化。每个BU的个性化需求都在RocketMQ项目之上进行深度定制。RocketMQ向其他BU提供的仅仅是Jar包，例如要定制一个Broker，那么只需要依赖rocketmq-broker这个jar包即可，可通过API进行交互，如果定制client，则依赖rocketmq-client这个jar包，对其提供的api进行再封装。
开源社区地址：
https://github.com/alibaba/RocketMQ
在RocketMQ项目基础上衍生的项目如下
 com.taobao.metaq v3.0 = RocketMQ + 淘宝个性化需求
为淘宝应用提供消息服务
 com.alipay.zpullmsg v1.0 = RocketMQ + 支付宝个性化需求
为支付宝应用提供消息服务
 com.alibaba.commonmq v1.0 = Notify + RocketMQ + B2B个性化需求
为B2B应用提供消息服务