（五）基于RocketMQ/Demo项目的测试和原理说明

> 简介：
>
> 文章一、二、三为RocketMQ的初步了解、搭建环境工作，文章四基于myeclipse搭建Demo测试项目
>
> 本文的内容基于之前的Demo项目和环境进行测试工作，探究说明RocketMQ的工作原理
>
> 作者：Nick

术语解释

Name Server：

- 无状态节点，可集群部署，节点之间无任何信息同步（Broker与每个namesrv连接，可以保证信息同步性）
- NameServer用来保存活跃的 broker 列表，包括 Master 和 Slave。
- NameServer用来保存所有 topic 和该 topic 所有队列的列表。
- Namesrv只是Broker的服务注册与发现，具体的消息收发，都依靠Broker。

Broker

- 消息中转角色，负责存储消息，转发消息。
- 拥有Master、slave（主备）的概念，主备有同步双写、异步复制功能来保持数据同步。标识：Master的BrokerId 为 0 ，Slave的BrokerId 非0。
- Broker 与 Name Server 集群中的所有节点建立长连接，定时（心跳）注册 Topic 信息到所有 Name Server。

Producer

- 消息发送者，消息通过topic来标示

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

- 有两种实现方式，一种push Consumer，第二种是 pull Consumer。

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。
- 两种模式：广播消费和集群消费

广播消费（Broadcasting）

- 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。broadcasting

集群消费（Clustering）

- 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。

Topic

- 消息的逻辑管理单位。

Queue(Message Queue/Topic 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，具有了水平扩展的能力
- Queue可以理解为Topic的实际存储单元，服务器创建Topic时会指定ReadQueueNum和WriteQueueNum。即某Producer发送了40条关于TopicA的消息，如果TopicA下有4个queue，则每个queue均分存储10条消息
- 在 RocketMQ 中，Queue都是持久化，长度近乎无限的数据结构，访问其中的存储单元使用 Offset 来访问，offset 为 java long 类型64 位，另外队列中只保存最近几天（默认三天）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
- 详细可以参看RocketMQ数据存储结构，如Commit log、offset等概念

工作原理的问题及测试

Q1：producer发送消息时，如果有多个Queue，每个Queue是随机存储消息的吗？

测试结果：发送消息的顺序是随机的（黑盒），Topic下有4个queue，发送4条消息，分别发送到queue0-3-2-0中，queue1没有消息，不是按照queue0-1-2-3的顺序进行依次发送。此外，大量发送消息，每个queue的消息数量为均分（只在个数极小下才能看出这种随机的个数差别）

结论：顺序是随机的，但总体上每个queue的消息数量均分

Q2：一个Topic只能对应一个producer ground？

测试结果：指定一个topic对应2个producer ground，两个producer groud分别发送该topic消息，consumer读到了两组的信息。

结论：一个topic可以对应多个producer ground，另：一个producer ground可以指定多个topic

附注：producer ground内的Producer 的发送消息应保持类别相同，逻辑一致。所以producer ground指定多topic的场景，应当保持组内Producer的topic逻辑一致

Q3：Broker关闭自动创建Topic、消费订阅组功能时，Producer和Consumer是否只能从指定的Topic列表中进行消息收发？

测试3.1：关闭 autoCreateTopicEnable 后启动broker，producer发送未知的topic失败

测试3.2：关闭 autoCreateSubscriptionGroup 后启动broker，producer发送已知的topic成功，但是consumer无法读取到消息，订阅该consumer后成功读取到消息

结论：

- broker控制topic的增删，发送未知topic的producer会报错。
- broker控制topic的订阅，未指定的订阅组内consumer无法拉取到该topic下的信息。

Q4：ConsumerGroup、Broker授权订阅组、Topic的关系?

结论：

- Broker创建（允许）ConsumerGroup（订阅组），ConsumerGroup内的Consumer实例控制Topic的订阅，即：Broker间接控制了topic的订阅权限
- ConsumerGroup内的Consumer实例可以订阅多个Topic

Q5：ConsumerGroup内的Consumer能否订阅消费不同的Topic？

测试：ConsumerA和ConsumerB使用相同的ConsumerGroup Name，但订阅消费不同的TopicA和TopicB。TopicA、B分别发送20条消息，ConsumerA、B只能接收10条，均有一半消息丢失。

结论：同一ConsumerGroup下的Consumer订阅不同的Topic，消费时会造成一半的消息丢失

附注：ConsumerGroup同producer ground相似，可订阅多个Topic，但需保持Consumer订阅关系一致

Q6：一个Consumer Queue只能被一个Consumer实例消费？

测试结果：集群模式下，TopicA设置了6个Consumer Queue，ConsumerGroup1（已为订阅组）下启动了8个Consumer，都已订阅TopicA。Producer发送60条TopicA消息，ConsumerGroup1下只有6个Consumer消费了信息，每个10条。剩余两个Consumer无消费行为。

结论：一个Consumer Queue只能被一个Consumer实例消费

附注：comsumer节点的数量>topic中queue的数量，多出来的comsumer会闲置。生产中应该尽量让consumer group下的consumer的数目和topic的queue的数目一致或成倍数关系。这样每个consumer消费的queue的数量总是一样的，每个consumer服务器的压力才会差不多

Q7：一个Topic只能在一个Broker上吗？

结论：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

附注：

- 每个topic代表一类消息，多个topic就可以对消息进行归类与隔离，不同的消费者只对某些个topic感兴趣。
- Kafka和RocketMQ都是磁盘消息队列的模式，对于同一个消费组，一个分区只支持一个消费线程来消费消息，生产环境中，一个topic会设置成多分区的模式，支持多个消费者。

Q8：Producer发送消息是如何得知发到哪个broker的

- 每个应用在收发消息之前，一般会调用一次producer.start()/consumer.start()做一些初始化工作，其中包括：创建需要的实例对象，如MQClientInstance；设置定时任务，如从Nameserver中定时更新本地的Topic route info，发送心跳信息到所有的 broker，动态调整线程池的大小，把当前producer加入到指定的组中等等。
- 客户端会缓存路由信息TopicPublishInfo, 同时定期从NameServer取Topic路由信息，每个Broker与NameServer集群中的所有节点建立长连接，定时注册Topic信息到所有的NameServer。Producer在发送消息的时候会去查询本地的topicPublishInfoTable（一个ConcurrentHashMap），如果没有命中的话就会询问NameServer得到路由信息 (RequestCode=GET_ROUTEINTO_BY_TOPIC) 如果nameserver中也没有查询到（表示该主题的消息第一次发送），那么将会发送一个default的topic进行路由查询。

Q9：Producer在发送消息时是如何指定topic的？发送不存在的topic怎么办？

- 发送消息需要指定消息所属的Topic。
- Producer首先根据Topic去Name Server获取Topic的队列信息，如果Topic在Name Server中不存在，Name Server为使用系统默认的Topic消息队列返回给生产者（前提：Broker允许自动创建Topic）。
- Producer根据Name Server返回的消息队列更新本地的Topic信息表。然后从Topic信息表中选择一条消息队列（Queue）进行通信。Producer根据Queue所在的Broker名称去Name Server查找该Broker的地址信息。最后与该Broker建立连接，将消息发送给该Broker，该Queue中。

Q10：Topic和Queue的关系和区别？

- 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，需要指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。

Q11：Broker的如何管理消息的（消费进度管理）？

- RocketMQ的broker端，不负责推送消息，无论消费者是否消费消息，都将消息存储起来。谁要消费消息，就向broker发请求获取消息，消费记录由consumer来维护。
- RocketMQ提供了两种存储方式来保留消费记录：一种是保留在consumer所在的服务器上；另一种是保存在broker服务器上。用户还可以自己实现相应的消费进度存储接口。
- 默认情况下，采用集群消费，会将记录保存在broker端；而采用广播消费则会将消费记录保存在本地。

Q12：RocketMQ高可用性是如何保证的？掉了一台主机后，如何保持消息的正常读写？

高可用：通过测试当队列服务器出现故障或者网络故障时，客户端可以自动连接到其他队列，保持业务的不间断。

- 单主无高可用性而言，实际生产中多数采用模式：多Master无slave。
- 个人理解：单点故障后，消息不会丢失，确保消息的顺序性和准确性。producer继续发送信息，会转移到其他主上，其他主上无此topic时，会重新分配一个topic queue给producer发送消息（或主节点都已添加相同的topic）。保证了高可用性。

Q13：Broker的部署模式？

- 单Master无Slave（脆弱）
- 单Master多Slave（单点故障就瘫，开源版无主备切换功能）
- 多Master无Slave（无单点故障，线上生产常用模式。多个broker实例构成了集群，某主挂掉后，这个主上未被消费的消息，暂时不能被消费了）
- 多Master多Slave（无单点故障，每个主都有一个backup的slave，HA高可用，但有些浪费）
    - 采用同步双写时，master挂掉，可以从slave处消费消息。
    - 采用异步复制时，因延时可能存在部分未写入slave的消息丢失。

附注：当采用多Master模式时，Master与Master之间是不需要知道彼此的，这样的设计直接降低了Broker实现的复查性，Master只做好自己的事情（比如和Slave进行数据同步）即可，这样，在分布式环境中，某台Master宕机或上线，不会对其他Master造成任何影响。

Q14： Consumer如何订阅消息，依据是什么（topic）？主备节点都订阅，找谁优先读？

- consumer通过topic来订阅消息，不同的topic指向不同的broker地址，然后push/pull方式去订阅拉取消息。
- 消费端会通过RebalanceService线程，10秒钟做一次基于topic下的所有队列负载：
  - 遍历Consumer下的所有topic，然后根据topic订阅所有的消息
  - 获取同一topic和Consumer Group下的所有Consumer
  - 根据具体的分配策略来分配消费队列，分配的策略包含：平均分配、消费端配置等
- 主备节点的读取优先规则可以在 broker中配置。

Q15：客户端与NameServer的连接关系？

- 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

Q16：什么是异步复制，什么是同步读写？

异步复制

- 异步复制的实现思路非常简单，Slave 启动一个线程，不断从 Master 拉取 Commit Log 中的数据，然后在异步 build 出 Consume Queue 数据结构。整个实现过程基本同 Mysql 主从同步类似
- 每个 Master 配置一个 Slave，有多对 Master-Slave，HA 采用异步复制方式，主备有短暂消息延迟，毫秒级。
- 优点：即使磁盘损坏，消息丢失的非常少，且消息实时性不会受影响，因为 Master 宕机后，消费者仍然可以从 Slave 消费，此过程对应用透明。不需要人工干预。性能同多 Master 模式几乎一样。
- 缺点：Master 宕机，磁盘损坏情况，会丢失少量消息

同步双写

- 每个 Master 配置一个 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：数据与服务都无单点，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，大约低 10%左右，发送单个消息的 RT 会略高。3.0支持同步双写。

Q17：Message reliablity/消息的可靠性？

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。

总结：(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(5)、(6)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故障。

Q18：Borker如何存储消息的？

1. topic与broker是多对多的关系，一个topic可以做分区配置的，使得可以分散为队列交付给多个btoker。分区的设计采用的是偏移量做法。
2. Broker是把消息持久化到磁盘文件的，同步刷盘就是写入后才告知producer成功；异步刷盘是收到消息后就告知producer成功了，之后异步地将消息从内存(PageCache)写入到磁盘上。(注意此处涉及到磁盘写入速度是否大于网卡速度的问题，应用的不好可能造成消息堆积)
3. 磁盘空间达到85%之后，不再接收消息，会打印日志并且告知producer发送消息失败。
4. 持久化采取的是ext4文件系统，存储的数据结构见RocketMQ_userguide.pdf

Q19：Broker的Buffer满了怎么办？

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。

Q20：Consumer进行消费时，消息可靠性如何保证？

- consumer与所有关联的broker保持长连接(包括主备)，每隔30s发送心跳，可以通过heartbeatBrokerInterval配置。
- broker每隔10s扫描连接，发现2分钟内没有心跳，则关闭连接，并通知该consumer group内其他实例，过来继续消费该topic。

因为是长连接，consumer挂掉会即时发生上述动作。所以consumer集群的情况下，消费是可靠的。

此外consumer可以同时订阅主备两种角色，master故障之后仍可以从slave读取信息，保证消息的顺利消费。