RocketMQ的架构设计、关键特性与应用场景

题目标签

学习时长：20分钟

题目难度：中等

知识点标签：RocketMQ

题目描述

RocketMQ的架构设计、关键特性与应用场景

1. 面试题分析

根据题目要求我们可以知道：

- RocketMQ的简介
- RocketMQ的演进
- RocketMQ的架构设计
- RocketMQ的关键特性
- RocketMQ的应用场景

> 分析需要全面并且有深度

容易被忽略的坑

- 分析片面
- 没有深入

2. RocketMQ的简介

RocketMQ一个纯java、分布式、队列模型的开源消息中间件，前身是MetaQ，是阿里研发的一个队列模型的消息中间件，后开源给apache基金会成为了apache的顶级开源项目，具有高性能、高可靠、高实时、分布式特点。

3. RocketMQ的演进

RocketMQ一共前后经历了三代演进：

1.第一代，推模式

数据存储采用关系型数据库，典型代表包括Notify、Napoli。

2.第二代，拉模式

自研的专有消息存储，在日志处理方面参考Kafka，典型代表MetaQ。

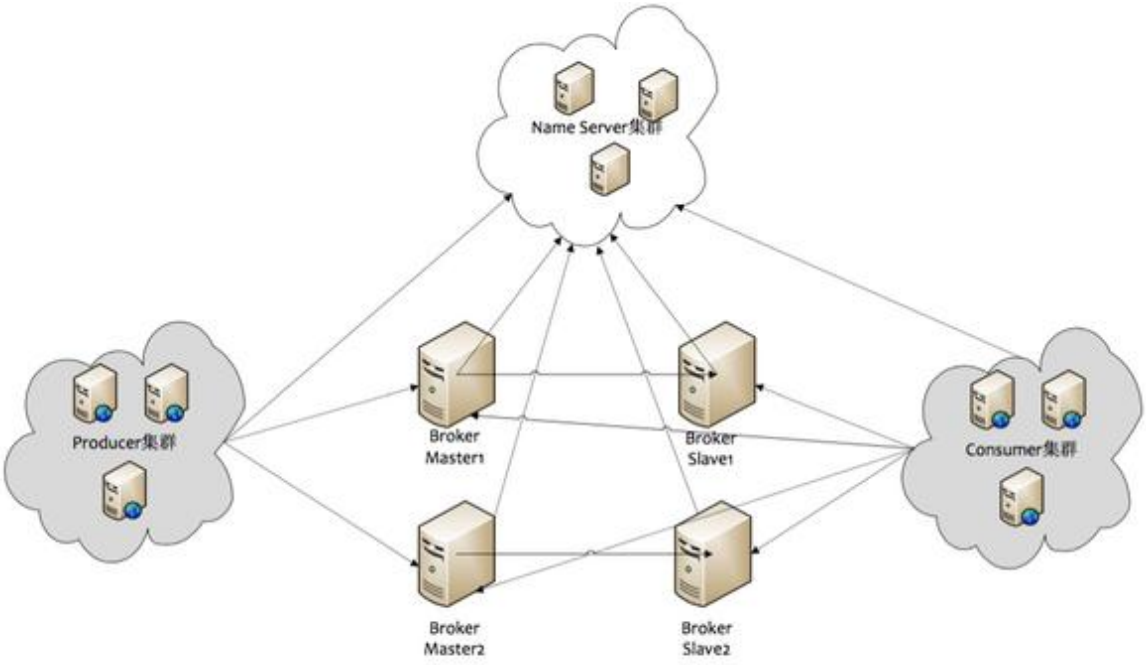
3.第三代，以拉模式为主，兼有推模式

低延迟消息引擎RocketMQ，在二代功能特性的基础上，为电商金融领域添加了可靠重试、基于文件存储的分布式事务等特性。使用在了阿里大量的应用上，典型如双11场景，具有万亿级消息流转。

4. RocketMQ的架构设计

1.RocketMQ的核心组件

RocketMQ主要由NameServer、Broker、Producer以及Consumer四部分构成。



1）NameServer：主要负责对于源数据的管理，包括了对于Topic和路由信息的管理。

NameServer是一个功能齐全的服务器，其角色类似Dubbo中的Zookeeper，但NameServer与Zookeeper相比更轻量。主要是因为每个NameServer节点互相之间是独立的，没有任何信息交互。

> 备注：下面的消息类型有Topic的介绍。

2） 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。

- Producer由用户进行分布式部署，消息由Producer通过多种负载均衡模式发送到Broker集群，发送低延时，支持快速失败。

3 ）Broker

消息中转角色，负责存储消息，转发消息。

- Broker是具体提供业务的服务器，单个Broker节点与所有的NameServer节点保持长连接及心跳，并会定时将Topic信息注册到NameServer，顺带一提底层的通信和连接都是基于Netty实现的。
- Broker负责消息存储，以Topic为纬度支持轻量级的队列，单机可以支撑上万队列规模，支持消息

推拉模型。

- 官网上有数据显示：具有上亿级消息堆积能力，同时可严格保证消息的有序性。

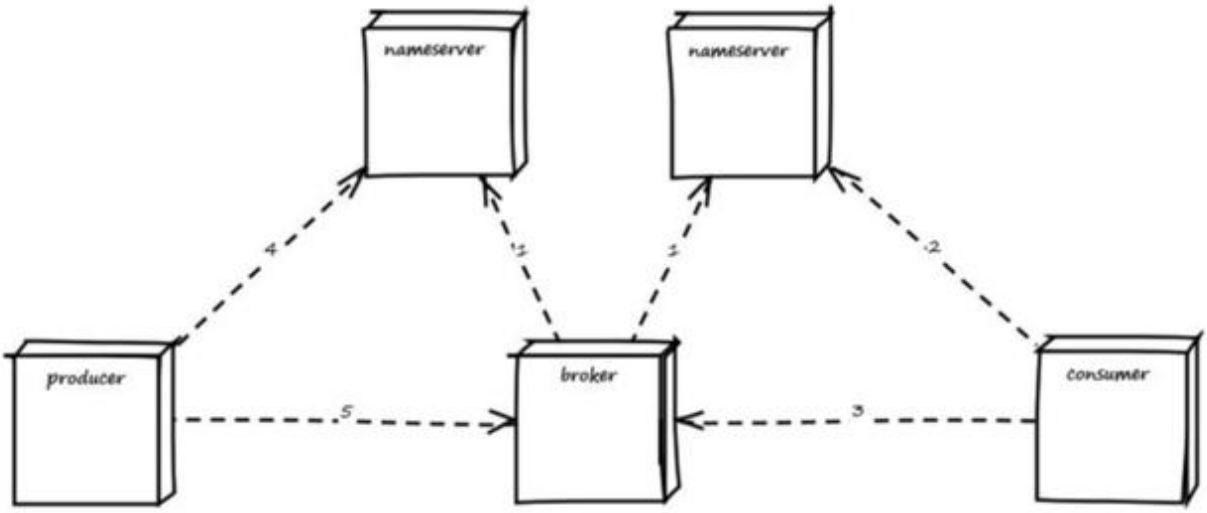
4）Consumer

消息消费者，负责消费消息，一般是后台系统负责异步消费。

- Consumer也由用户部署，支持PUSH和PULL两种消费模式，支持集群消费和广播消息，提供实时的消息订阅机制。

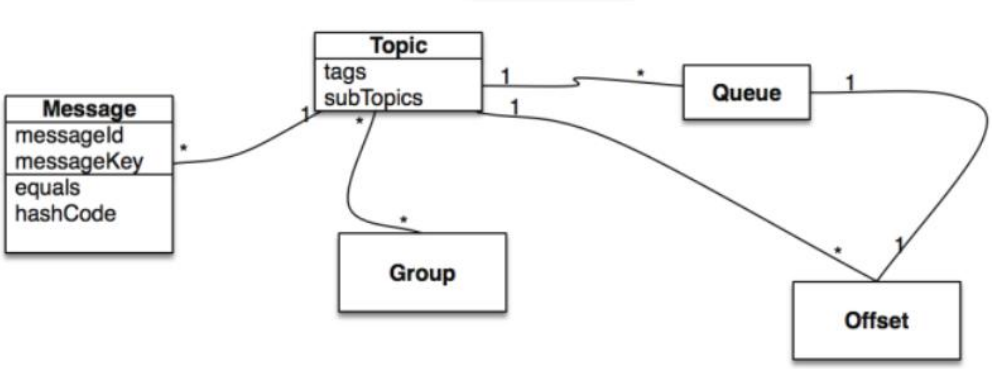
5）大致流程

Broker在启动的时候会去向NameServer注册并且定时发送心跳，Producer在启动的时候会到NameServer上去拉取Topic所属的Broker具体地址，然后向具体的Broker发送消息。具体如下图：



2.RocketMQ的消息领域模型

主要分为Message、Topic、Queue、Offset以及Group这几部分。



1）Topic

Topic表示消息的第一级类型，比如一个电商系统的消息可以分为：交易消息、物流消息等。一条消息必须有一个Topic。

最细粒度的订阅单位，一个Group可以订阅多个Topic的消息。

2）Tag

Tag表示消息的第二级类型，比如交易消息又可以分为：交易创建消息，交易完成消息等。RocketMQ提供2级消息分类，方便灵活控制。

3)Group

组，一个组可以订阅多个Topic。

4）Message Queue

消息的物理管理单位。一个Topic下可以有多个Queue，Queue的引入使得消息的存储可以分布式集群化，具有了水平扩展能力。

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。

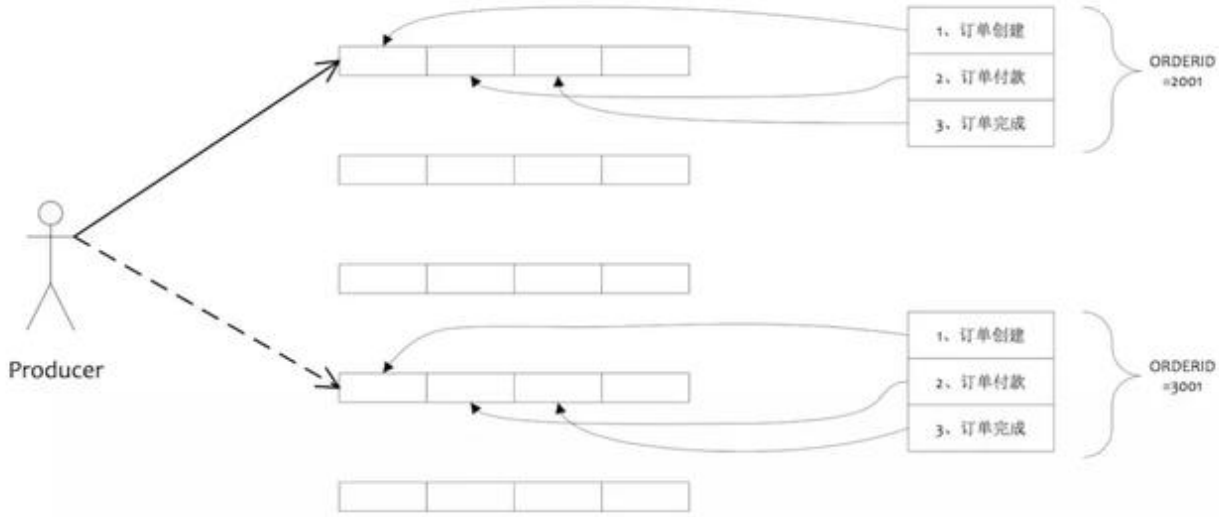
也可以认为 Message Queue 是一个长度无限的数组，Offset 就是下标。

5. RocketMQ的关键特性

1.消息的顺序

消息的顺序指的是消息消费时，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有意义。但同时订单之间又是可以并行消费的。

RocketMQ是通过将“相同ID的消息发送到同一个队列，而一个队列的消息只由一个消费者处理“来实现顺序消息。如下图：



这样对于同一个订单的创建、付款和完成消息，确保按照这一顺序被发送和消费。

2.消息重复

1）消息重复的原因

消息领域有一个对消息投递的QoS定义，分为：

- 最多一次（At most once）
- 至少一次（At least once）
- 仅一次（ Exactly once）

> QoS：Quality of Service，服务质量

几乎所有的MQ产品都声称自己做到了At least once。既然是至少一次，那避免不了消息重复，尤其是在分布式网络环境下。比如：网络原因闪断，ACK返回失败等等故障，确认信息没有传送到消息队列，导致消息队列不知道自己已经消费过该消息了，再次将该消息分发给其他的消费者。

不同的消息队列发送的确认信息形式不同,例如RabbitMQ是发送一个ACK确认消息，RocketMQ是返回一个CONSUME_SUCCESS成功标志，kafka实际上有个offset的概念。

RocketMQ没有内置消息去重的解决方案，最新版本是否支持还需确认。

2）消息去重

1）去重原则：使用业务端逻辑保持幂等性

幂等性：就是用户对于同一操作发起的一次请求或者多次请求的结果是一致的，不会因为多次点击而产生了副作用，数据库的结果都是唯一的，不可变的。

只要保持幂等性，不管来多少条重复消息，最后处理的结果都一样，需要业务端来实现。

2）**去重策略**：保证每条消息都有唯一编号**(比如唯一流水号)**，且保证消息处理成功与去重表的日志同时出现。

建立一个消息表，拿到这个消息做数据库的insert操作。给这个消息做一个唯一主键（primary key）或者唯一约束，那么就算出现重复消费的情况，就会导致主键冲突，那么就不再处理这条消息。

6. RocketMQ的应用场景

1.削峰填谷

比如如秒杀等大型活动时会带来较高的流量脉冲，如果没做相应的保护，将导致系统超负荷甚至崩溃。如果因限制太过导致请求大量失败而影响用户体验，可以利用MQ 超高性能的消息处理能力来解决。

2.异步解耦

通过上、下游业务系统的松耦合设计，比如：交易系统的下游子系统（如积分等）出现不可用甚至宕机，都不会影响到核心交易系统的正常运转。

3.顺序消息

与FIFO原理类似，MQ提供的顺序消息即保证消息的先进先出，可以应用于交易系统中的订单创建、支付、退款等流程。

4.分布式事务消息

比如阿里的交易系统、支付红包等场景需要确保数据的最终一致性，需要引入 MQ 的分布式事务，既实现了系统之间的解耦，又可以保证最终的数据一致性。

将大事务拆分成小事务，减少系统间的交互，既高效又可靠。再利用MQ 的可靠传输与多副本技术确保消息不丢，At-Least-Once 特性来最终确保数据的最终一致性。

7. 扩展内容

- 分布式消息Kafka的原理、基础架构、使用场景
- Kafka、RocketMQ、RabbitMQ的优劣势比较
- 详解RPC远程调用和消息队列MQ的区别