面试真题

【美团】用RocketMQ主要是用来解决什么问题的？

RocketMQ 是一种分布式消息中间件，主要功能是实现高性能、高吞吐量的消息传递和流处理。

主要应用场景：

1）异步通信：当系统中有大量的请求需要异步处理时，使用 RocketMQ 可以解耦系统，提升性能。例如，用户注册后发送欢迎邮件等操作可以通过消息队列异步处理。

2）服务解耦：通过使用消息队列，各个服务之间可以通过消息进行通信，降低耦合，提高系统的可扩展性和稳定性。

3）削峰填谷：在高并发系统中，短时间内的高并发请求可能导致系统崩溃，通过消息队列可以进行流量削峰填谷，保护系统。

4）日志传输：分布式系统中各个节点产生的日志，可以通过消息队列进行集中采集和传输。

5）分布式事务：RocketMQ 提供了分布式事务的支持，在分布式系统中可以保证事务的一致性。

【美团】RocketMQ什么情况下会出现重复消费的问题？

重复消费问题出现的情况：

1. **没有及时确认**：默认情况下，RocketMQ使用"消费确认"机制，消费者需要手动确认消息是否处理成功。如果消费者在一定时间内未成功消费消息，RocketMQ会认为该消息未成功处理并重新投递。若消息处理成功但消费者没有及时确认，也会导致该消息被重新消费。
2. **没能成功确认**：在消费者与RocketMQ之间发生网络故障的情况下，消息可能未能成功确认，RocketMQ会再次投递该消息，造成重复消费。
3. **消费失败**：如果消费者出现异常导致消费失败，RocketMQ会尝试将消息重新投递。消费者处理失败的消息会被重复消费，直到达到最大重试次数。
4. **消费进度丢失**：在某些情况下，如果消费者的消费进度（偏移量）丢失或错误，RocketMQ无法正确判断已消费的消息，可能导致重复消费。

【美团】RocketMQ如何避免重复消费？

在 RocketMQ 中，通过以下几种方法可以确保消息不会重复消费：

1）幂等性：确保消费者的处理逻辑是幂等的，即对于相同的消息重复处理多次，结果应保持一致。

2）消息状态记录：在消费前记录消息状态，如果记录存在则不重复处理。常见的实现方式是使用数据库记录消费的消息ID。

3）消息重试机制：配置合理的消息重试机制，限制重试次数，避免消息反复被消费。

【美团】Kafka和RocketMQ有什么区别，实现细节上有了解吗

RocketMQ 和 Kafka 都是流行的分布式消息队列系统。

1. 数据模型：
   1. Kafka 主要是日志模型，适用于高吞吐量的数据流处理。
   2. RocketMQ 的消息模型更为多样，包括点对点、发布订阅和顺序消息。
2. 存储机制：
   1. Kafka 使用的是磁盘顺序写，主要依赖文件系统的缓存机制；
   2. RocketMQ 存储有自己的机制，更像数据库，有索引能快速定位消息。
3. 高可用机制：
   1. Kafka 有 ISR（in-sync replicas）机制来保障数据的高可用性
   2. RocketMQ 有强一致性协议及消息重发机制。
4. 消费模式和顺序：
   1. Kafka 基于分区偏移量来进行消费，并且有严格的顺序保证。
   2. RocketMQ 数据消费模式灵活，支持普通消费与顺序消费。
5. 吞吐延迟：
   1. Kafka 高吞吐
   2. RocketMQ低延迟
6. 适用场景：
   1. Kafka适用于高吞吐量的日志收集系统
   2. RocketMQ适合低延迟、高可靠的电商支付、金融交易等。

【美团】【腾讯】Kafka怎么处理消息积压？

消息积压最粗粒度的原因：

1. 消费变慢了：可以通过**扩容消费端的实例数来提升总体的消费能力**。
2. 发送变快了：如果短时间内没有足够的服务器资源进行扩容，没办法的办法是，将**系统降级**，通过关闭一些不重要的业务，减少发送方发送的数据量，最低限度让系统还能正常运转，服务一些重要业务。

**1. 水平分区、并行消费：**Kafka通过分区（Partition）将消息数据水平拆分，多个消费者可以并行消费不同分区的数据，从而提高消息的消费能力，减少单个消费者的积压。

**2. 消费者控制pull速率：**Kafka使用**pull模式**（消费者主动拉取消息）。如果消费者消费能力不足，消费者可以控制拉取消息的速率，防止积压过多。

**3. 消费者组共同消费：**Kafka允许多个消费者组成一个**消费者组**，不同的消费者组可以独立消费相同的消息，而同一个消费者组内部的消费者则共同消费分区的数据，分担负载，避免单个消费者积压。

**4. 消息过期与日志删除：**Kafka有消息保留机制，**保留策略**可以根据时间（如7天）、消息大小（如10GB）等进行配置，当达到保留条件时，旧消息会被自动删除，防止过期消息导致积压。

【美团】怎么保证Kafka处理消息是有序且不重复的？

Kafka 通过分区机制（Partition）和消息键（Message Key）来保证消息的顺序性。在 Kafka 中，每个 Topic 可以分为多个分区，每个分区内的消息都是有序的。因此，**Kafka 提供了有限度的顺序性保证，具体来说**：

1）在同一个分区内，消息是有序的。

2）靠消息键将相关消息分配到同一分区，可以保证这些消息在同一分区内依然有序。

在哪些场景下消息的顺序性是十分关键？

1）金融交易系统：交易指令必须按正确的顺序执行，例如银行的转账操作。

2）日志聚合：日志事件需要按发生的时间顺序进行处理，以便准确地重现事件顺序。

3）库存管理系统：商品的出入库操作必须按照操作顺序执行，否则会造成库存记录的混乱。

4）流媒体服务：视频或者音频流的帧数据需要按照播放顺序发送，否则会影响用户的观看体验。

【美团】用过RPC框架吗？介绍一下dubbo。

Dubbo 是阿里巴巴开源的一个高性能Java RPC（远程过程调用）框架，适用于构建分布式服务体系。它主要解决了服务治理过程中关于服务调用远程化、服务注册与发现、负载均衡、流量控制等问题。

例如，在一个电子商务平台中，购物车服务、订单服务、用户服务等各式各样的服务之间会有相互调用。使用Dubbo可以简化这些服务之间的调用，统一管理服务注册和发现，同时利用其高性能的通信能力和负载均衡策略，保障高并发场景下的服务稳定和性能。

Dubbo的核心功能：

1）远程通信：Dubbo 提供了基于多种协议（如Dubbo协议、HTTP协议等）的远程调用能力，使得不同服务之间可以通过网络进行通信。

2）服务注册与发现：Dubbo 采用了分布式服务注册中心机制（如Zookeeper），服务提供者会把自身服务信息注册到中心，消费者从中心获取服务提供者的信息进行调用，从而实现服务注册和发现。

3）负载均衡：通过多种负载均衡策略（如随机、轮询、一致性哈希），Dubbo能够在多实例之间合理分配流量，提升系统的容错和伸缩性。

4）高可用：包括服务降级、失败重试等机制，在服务不可用或异常时，提高系统的整体稳定性。

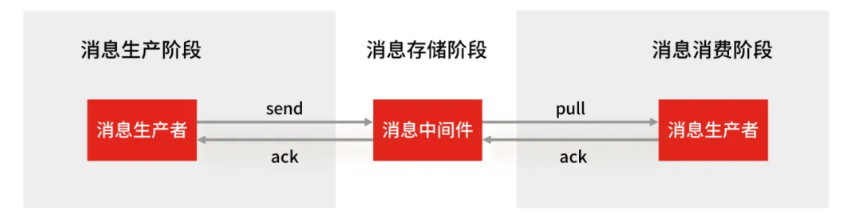
【腾讯】用的什么消息队列，消息队列怎么选型的？

项目用的是 RocketMQ 消息队列。选择RocketMQ的原因是：

* **开发语言优势**。RocketMQ 使用 Java 语言开发，比起使用 Erlang 开发的 RabbitMQ 来说，有着更容易上手的阅读体验和受众。在遇到 RocketMQ 较为底层的问题时，大部分熟悉 Java 的同学都可以深入阅读其源码，分析、排查问题。
* **社区氛围活跃**。RocketMQ 是阿里巴巴开源且内部在大量使用的消息队列，说明 RocketMQ 是的确经得起残酷的生产环境考验的，并且能够针对线上环境复杂的需求场景提供相应的解决方案。
* **特性丰富**。例如顺序消息、事务消息、消息过滤、定时消息、顺序消息、事务消息、消息过滤、定时消息。

【腾讯】RocketMQ 消息可靠性怎么保证？

使用一个消息队列，其实就分为三大块：**生产者、中间件、消费者**，所以要保证消息可靠性就是保证三个环节都不能丢失数据。



* **消息生产阶段**：生产者会不会丢消息，取决于生产者对于异常情况的处理是否合理。从消息被生产出来，然后提交给 MQ 的过程中，只要能正常收到 （ MQ 中间件） 的 ack 确认响应，就表示发送成功，所以只要处理好返回值和异常，如果返回异常则进行消息重发，那么这个阶段是不会出现消息丢失的。
* **消息存储阶段**：RabbitMQ 或 Kafka 这类专业的队列中间件，在使用时是部署一个集群，生产者在发布消息时，队列中间件通常会写「多个节点」，也就是有多个副本，这样一来，即便其中一个节点挂了，也能保证集群的数据不丢失。
* **消息消费阶段**：消费者接收消息+消息处理之后，才回复 ack 的话，那么消息阶段的消息不会丢失。不能收到消息就回 ack，否则可能消息处理中途挂掉了，消息就丢失了。

【腾讯】Kafka 和 RocketMQ 消息确认机制有什么不同？

Kafka的消息确认机制有三种：0，1，-1：

* **ACK=0**：这是最不可靠的模式。生产者在发送消息后不会等待来自服务器的确认。这意味着消息可能会在发送之后丢失，而生产者将无法知道它是否成功到达服务器。
* **ACK=1**：这是默认模式，也是一种折衷方式。在这种模式下，生产者会在消息发送后等待来自分区领导者（leader）的确认，但不会等待所有副本（replicas）的确认。这意味着只要消息被写入分区领导者，生产者就会收到确认。如果分区领导者成功写入消息，但在同步到所有副本之前宕机，消息可能会丢失。
* **ACK=-1**：这是最可靠的模式。在这种模式下，生产者会在消息发送后等待所有副本的确认。只有在所有副本都成功写入消息后，生产者才会收到确认。这确保了消息的可靠性，但会导致更长的延迟。

RocketMQ 提供了三种消息发送方式：同步发送、异步发送和单向发送：

* **同步发送**：是指消息发送方发出一条消息后，会在收到服务端同步响应之后才发下一条消息的通讯方式。应用场景非常广泛，例如重要通知邮件、报名短信通知、营销短信系统等。
* **异步发送**：是指发送方发出一条消息后，不等服务端返回响应，接着发送下一条消息的通讯方式，但是需要实现异步发送回调接口（SendCallback）。消息发送方在发送了一条消息后，不需要等待服务端响应即可发送第二条消息。发送方通过回调接口接收服务端响应，并处理响应结果。适用于链路耗时较长，对响应时间较为敏感的业务场景，例如，视频上传后通知启动转码服务，转码完成后通知推送转码结果等。
* **单向发送**：发送方只负责发送消息，不等待服务端返回响应且没有回调函数触发，即只发送请求不等待应答。此方式发送消息的过程耗时非常短，一般在微秒级别。适用于某些耗时非常短，但对可靠性要求并不高的场景，例如日志收集。

【腾讯】Kafka 和 RocketMQ 的 broker 架构有什么区别

* Kafka 的 broker 架构：采用了分布式的设计，**每个 Kafka broker 是一个独立的服务实例**，负责存储和处理一部分消息数据。Kafka 的 topic 被分区存储在不同的 broker 上，实现了水平扩展和高可用性。
* RocketMQ 的 broker 架构：是分布式的，但是**每个 RocketMQ broker 有主从之分**，一个主节点和多个从节点组成一个 broker 集群。主节点负责消息的写入和消费者的拉取，从节点负责消息的复制和消费者的负载均衡。

面向简历

Nginx

Docker及其相关组件

容器化技术

Devops