# 消息中间件技术选型

## 消息中间件常见使用场景

消息队列是分布式系统中重要的中间件，在高性能、高可用、低耦合等系统架构中扮演着重要作用。分布式系统可以借助消息队列的能力，轻松实现以下功能：

1. 异步

在分布式微服务的复杂处理流程中，通过MQ消息机制，把部分同步处理变成异步处理，从而降低整体交易链路的延迟，提高系统的响应速度和吞吐量。

1. 解耦

将不同的服务之间通过MQ消息传递进行解耦，减少服务间直接调用的依赖影响，从而提高系统稳定性及可扩展性，降低系统设计和实现的复杂度。

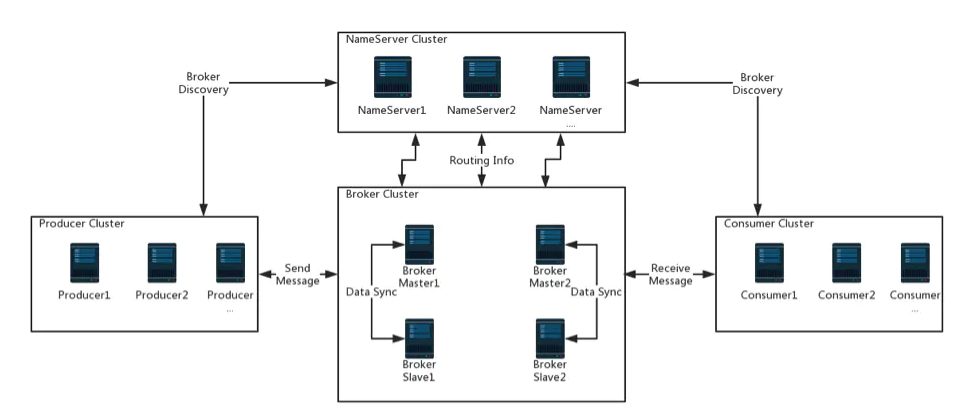
1. 削峰

在海量用户的金融系统里，存在热点交易时段，此时在极短的时间内产生了大量的并发业务处理请求，可能会击穿系统的处理能力上限导致宕机。而通过MQ进行业务请求的处理，可利用其允许消息堆积的特性，实现削峰填谷，应对突发的流量冲击，并以系统可以接受的速度进行后续的业务处理，从而保护系统稳定性和可用性。

## 消息中间件主流产品

1. **RocketMQ消息中间件**

1）架构图



1. 各组件介绍

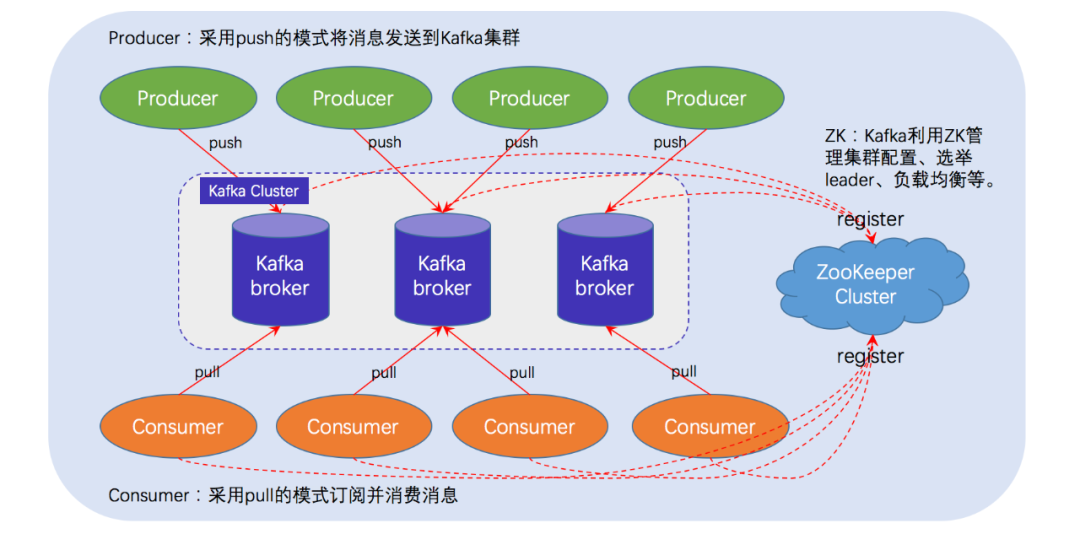
Apache RocketMQ 自诞生以来，因其架构简单、业务功能丰富、具备极强可扩展性等特点被众多企业开发者以及云厂商广泛采用。历经十余年的大规模场景打磨，RocketMQ 已经成为业内共识的金融级可靠业务消息首选方案，被广泛应用于互联网、大数据、移动互联网、物联网等领域的业务场景；

RocketMQ是阿里开源的消息中间件，它是一个开源的分布式消息传递和流式数据平台，总共有四大部分：NameServer，Broker，Producer，Consumer；

NameServer主要用来管理brokers以及路由信息。broker服务器启动时会注册到NameServer上，并且两者之间保持心跳监测机制，以此来保证NameServer知道broker的存活状态。而且，每一台NameServer都存有全部的broker集群信息和生产者/消费者客户端的请求信息。

Broker负责管理消息存储分发，主从数据同步，为消息建立索引，提供消息查询等能力。

1. **Kafka消息中间件**
2. 架构图



1. 组件介绍

一个Kafka集群由多个Broker和一个ZooKeeper集群组成，Broker作为Kafka节点的[服务器](https://cloud.tencent.com/product/cvm?from=20065&from_column=20065" \t "https://cloud.tencent.com/developer/article/_blank)。同一个消息主题Topic可以由多个分区Partition组成，分区物理存储在Broker上。负载均衡考虑，同一个Topic的多个分区存储在多个不同的Broker上，为了提高可靠性，每个分区在不同的Broker会存在副本。

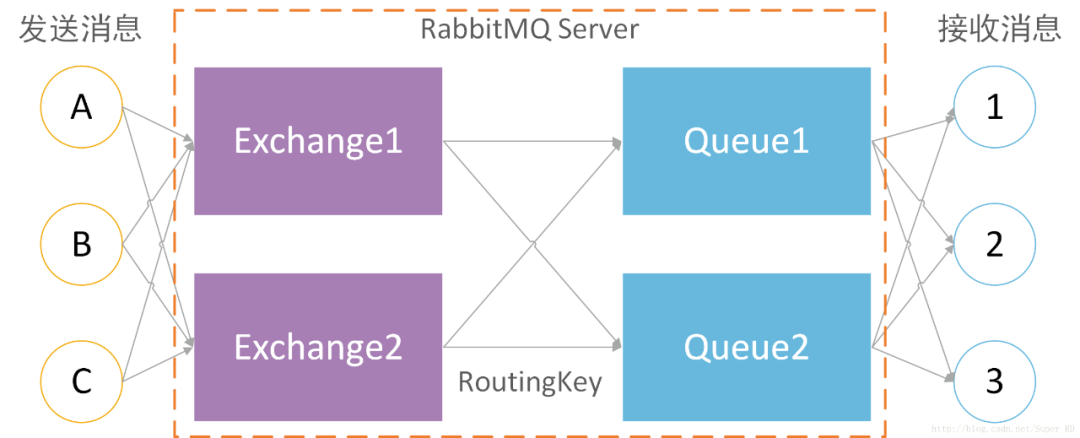
ZooKeeper是一个分布式开源的应用程序协调服务，可以实现统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等工作。Kafka里的ZooKeeper主要有一下几个作用：

Broker注册，当有Broker故障的时候能及时感知。

Topic注册，维护Topic各分区的各副本所在的Broker节点， 以及对应leader/follower的角色。

Consumer注册，维护消费者组的offset以及消费者与分区 的对应关系，实现负载均衡。

1. **RabbitMQ消息中间件**
2. 架构



1. 组件介绍

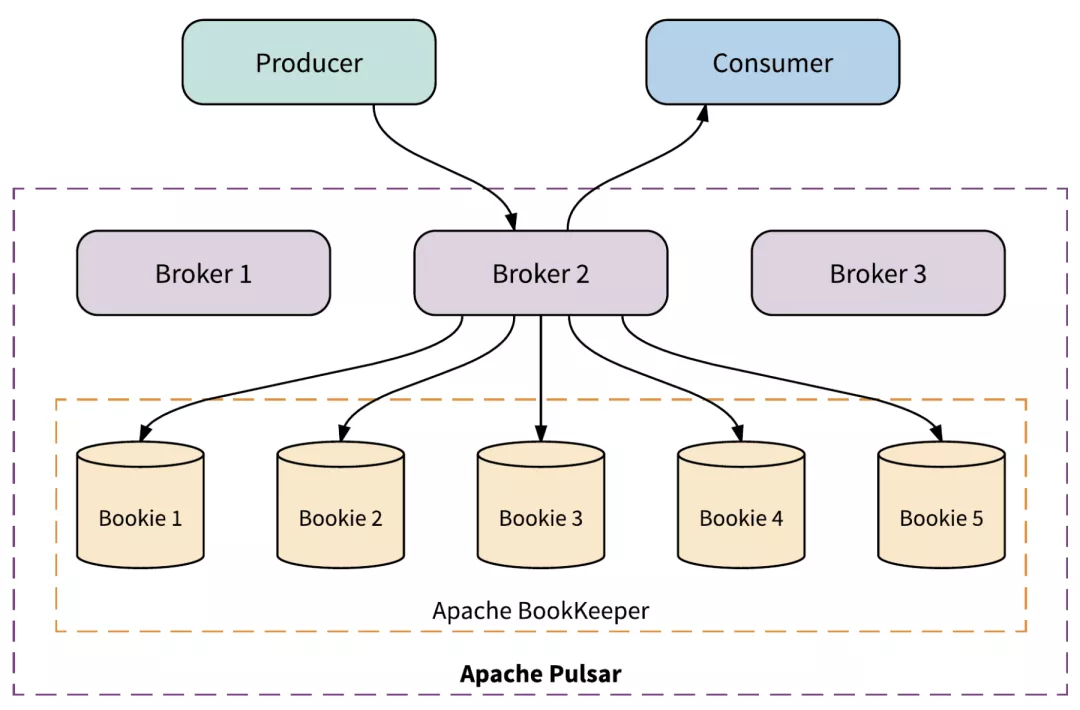
RabbitMQ基于AMQP协议来实现，主要由Exchange和 Queue两部分组成，然后通过RoutingKey关联起来，消息投递 到Exchange然后通过Queue接收

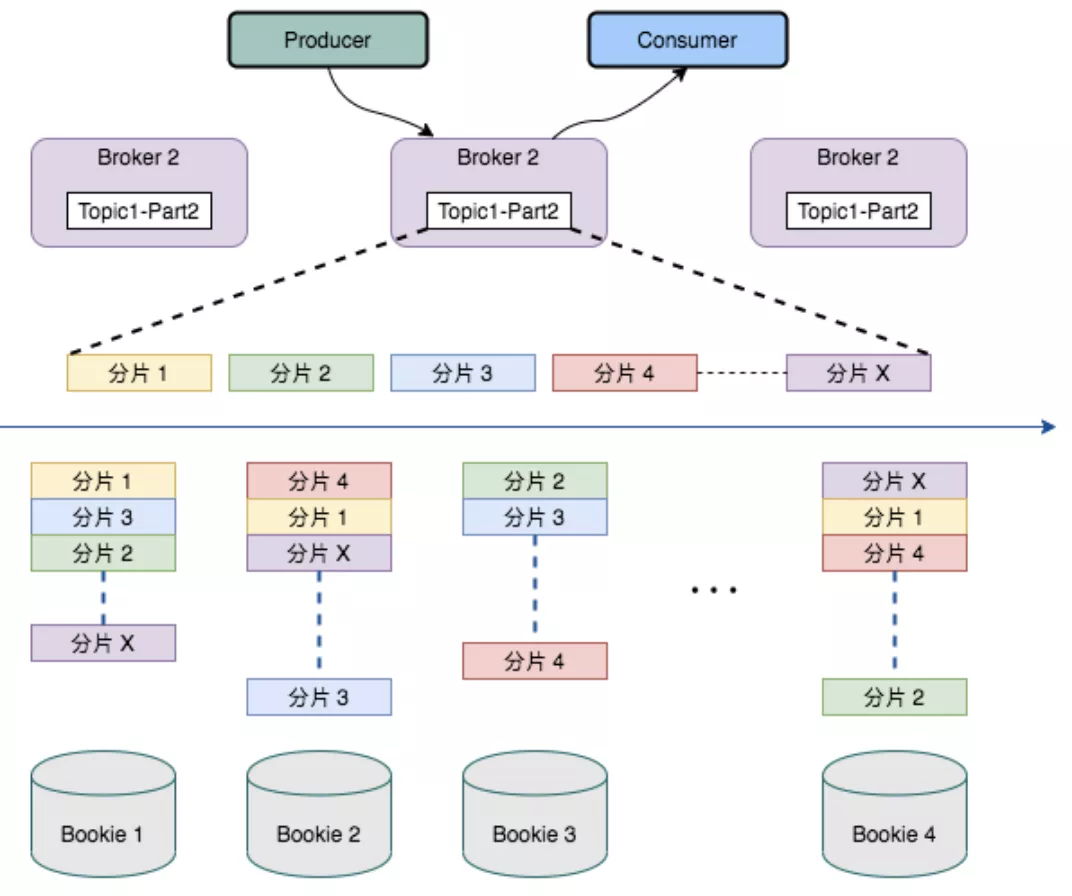
Exchange ：交换机，用来接收生产者发送的消息并将这些 消息路由给服务器中的队列。

Queue ：消息队列，用来保存消息直到发送给消费者。它 是消息的容器。一个消息可投入一个或多个队列

Banding ：绑定关系，用于消息队列和交换机之间的关联。 通过路由键（Routing Key）将交换机和消息队列关联起来。

1. **Pulsar消息中间件**
2. 架构





1. 组件介绍

Pulsar有三个重要的组件，Broker、BookKeeper和ZooKeeper，Broker是无状态服务，客户端需要连接到Broker上进行消息的传递。BookKeeper与ZooKeeper是有状态服务。BookKeeper的节点叫Bookie，负责存储消息和游标，ZooKeeper存储Broker和Bookie的元数据。Pulsar以这种架构，实现存储和计算分离，Broker负责计算，Bookie负责有状态存储。

Pulsar的多层架构影响了存储数据的方式。Pulsar将Topic分区划分为分片（Segment），然后将这些分片存储在Apache BookKeeper的存储节点上，以提高性能、可伸缩性和可用性。Pulsar的分布式日志以分片为中心，借助扩展日志存储（通过Apache BookKeeper）实现，内置分层存储支持，因此分片可以均匀地分布在存储节点上。由于与任一给定Topic相关的数据都不会与特定存储节点进行捆绑，因此很容易替换存储节点或缩扩容。另外，集群中最小或最慢的节点也不会成为存储或带宽的短板。

## 各消息中间件选型对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | RocketMQ | Kafaka | Pulsar | RabbitMQ |
| 开发语言 | Java | Scala | Java | Erlang |
| 高可用 | 主从架构 | 分布式架构 | 分布式架构 | 主从架构 |
| 事务消息 | 支持 | 支持 | 支持 | 支持 |
| 集群扩容 | 增加节点 | 增加节点，通过复制数据均衡 | 增加节点，通过新增分片均衡 | 增加节点 |
| 单机吞吐量 | 高（十万级） | 高（十万级） | 高（十万级） | 低（万级） |
| 消息延迟性 | 毫秒 | 毫秒 | 毫秒 | 微秒 |
| 持久化 | 支持 | 支持 | 支持 | 支持，性能差 |
| 消息回溯 | 支持 | 支持 | 支持 | 不支持 |
| 延迟队列 | 支持 | 不支持 | 支持 | 支持 |
| 死信队列 | 支持 | 不支持 | 支持 | 支持 |
| 优先级队列 | 不支持 | 不支持 | 不支持 | 支持 |
| 消费推拉模式 | 长轮询pull | 长轮询pull | push | push |
| 消息顺序性 | 通过加锁保证消息发送到一个队列，单个消费者线程消费来保证有序 | 分区有序 | 独占订阅模式保证消息有序 | 单线程发送单线程消费来保证 |
| 支持topic数量 | 万级 | 千级 | 万级 | 百级 |
| 元数据保存 | 基于轻量级namesever（AP） | 基于zk(cp)/新版本不需要zk | 基于zk(cp) | 本地保存 |
| 技术成熟度 | 成熟 | 成熟 | 接近成熟 | 较为成熟 |
| 社区活跃度 | 活跃 | 活跃 | 活跃 | 不活跃 |
| 银行案例 | 邮储银行，平安银行，民生银行，网商银行等 | 各银行的大数据平台消息传输场景 | 中国银行小范围使用了腾讯TDMQ-pulsar |  |

## 消息中间件选型建议

综合以上信息来看：

1）RabbitMQ：作为第一代MQ的代表，在性能和消息持久化堆积方面表现较差，并且社区不活跃，使用erlang语言开发，有问题时难以分析定位和维护扩展。不建议使用。

2）Pulsar：作为比较新的MQ模型，使用Java开发实现，目前的成熟度和在金融行业的应用还处于早期阶段，建议观望和储备技术，但是暂时不引入到生产系统。

3）Kafka：国外使用最多的MQ，并且在大数据平台领域是当前的事实标准。一般来说对于延迟不敏感、批量型、Topic 数量可控、对于消息丢失不敏感的场景，Kafka有非常高的吞吐性能，所以非常适合用于做不同平台之间的数据抽取、日志聚合。对于业务交易场景，反而不是那么合适。同时由于开发语言是scala，所以定制维护成本不低。

4）RocketMQ：RocketMQ是目前国内做业务交易中用于异步消息处理的最广泛消息中间件。就技术成熟度而言，在经历阿里双十一数万亿洪峰、微众银行、民生银行、蚂蚁金服、平安、字节跳动、快手、美团、京东、网易等各种行业大厂的考验后，稳定性非常高。基于Java 开发的，代码也非常稳定、有条理，各个版本之间除了功能有差异之外，Api 、传输协议几乎没有太多变化，对于升级而言也更加方便。RocketMQ 社区也比较活跃，社区文档非常丰富和完善，原理剖析视频、文档也非常多，非常易于学习和使用。RocketMQ 适用于金融转账消息、订单状态变更消息等场景。对于海量数据的问题，一般地横向扩容完全可以解决。对于金融级应用要求的一致性、高可用，跨IDC机房部署等，也都有很好的支持。

**建议采用RocketMQ作为本平台的消息中间件。**