

消息队列原理和选型：Kafka、RocketMQ、RabbitMQ 和 ActiveMQ

楼仔 2022-05-10 ◎ 5,134 ◎ 阅读14分钟

关注



大家好，我是楼仔！

之前写了一篇文章 [消息队列：从选型到原理，一文带你全部掌握](#)，干货真的很多，但是可读性很差，不想让这篇文章被埋没，周末特意花了 6 个小时重新整理。

消息队列中间件重要吗？面试必问问题之一，你说重不重要。我有时会问同事，为啥你用 RabbitMQ，不用 Kafka，或者 RocketMQ 呢，他给我的回答“因为公司用的就是这个，大家都这么用”，如果你去面试，直接就被 Pass，今天这篇文章，告诉你如何回答。

这篇文章，我重点突出消息队列选型，弱化每种队列内部的实现细节，精华提炼，可读性更强！

常用的消息队列主要这 4 种，分别为 Kafka、RabbitMQ、RocketMQ 和 ActiveMQ，主要介绍前三，不BB，上思维导图！

消息队列应用场景

消息队列

常用消息队列



消息队列对比&选型

消息队列对比

消息队列选型

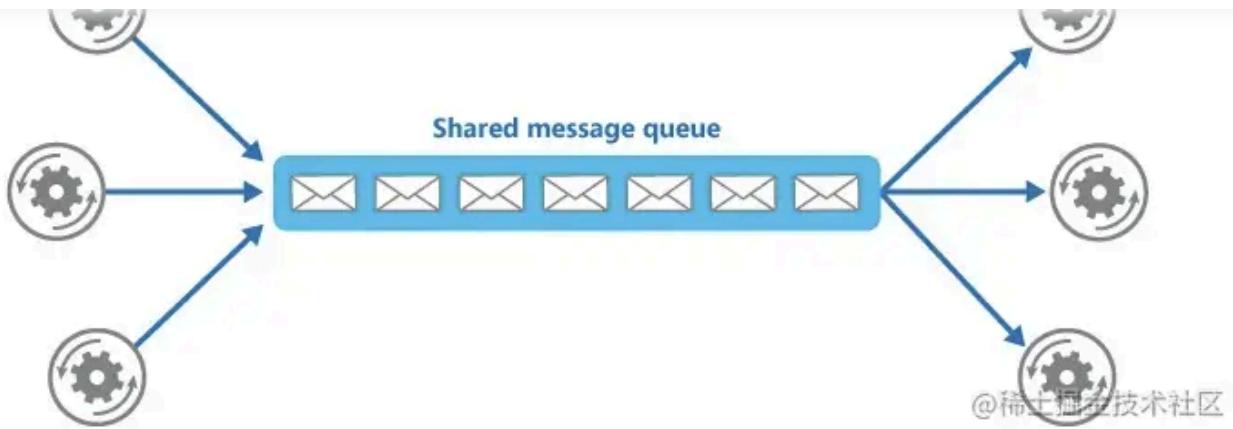
@稀土掘金技术社区

消息队列基础

什么是消息队列？

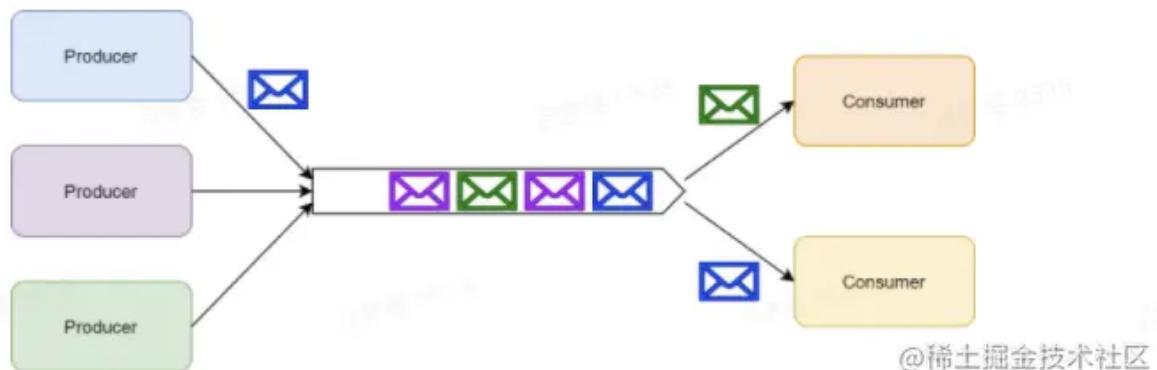
消息队列是在消息的传输过程中保存消息的容器，用于接收消息并以文件的方式存储，一个消息队列可以被一个也可以被多个消费者消费，包含以下 3 元素：

- Producer：消息生产者，负责产生和发送消息到 Broker；
- Broker：消息处理中心，负责消息存储、确认、重试等，一般其中会包含多个 Queue；
- Consumer：消息消费者，负责从 Broker 中获取消息，并进行相应处理。

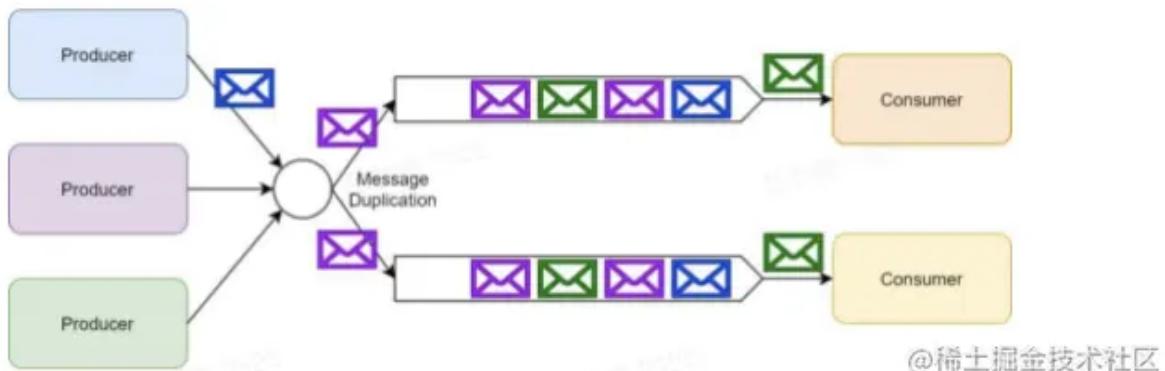


消息队列模式

- 点对点模式：多个生产者可以向同一个消息队列发送消息，一个具体的消息只能由一个消费者消费。



- 发布/订阅模式：单个消息可以被多个订阅者并发的获取和处理。



消息队列应用场景



- **异步处理**：消息队列本身是异步的，它允许接收者在消息发送很长时间后再取回消息。
- **流量削锋**：当上下游系统处理能力存在差距的时候，利用消息队列做一个通用的“载体”，在下游有能力处理的时候，再进行分发与处理。
- **日志处理**：日志处理是指将消息队列用在日志处理中，比如 Kafka 的应用，解决大量日志传输的问题。
- **消息通讯**：消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯，比如实现点对点消息队列，或者聊天室等。
- **消息广播**：如果没有消息队列，每当一个新的业务方接入，我们都要接入一次新接口。有了消息队列，我们只需要关心消息是否送达了队列，至于谁希望订阅，是下游的事情，无疑极大地减少了开发和联调的工作量。

常用消息队列

由于官方社区现在对 ActiveMQ 5.x 维护越来越少，较少在大规模吞吐的场景中使用，所以我们主要讲解 Kafka、RabbitMQ 和 RocketMQ。

Kafka

Apache Kafka 最初由 LinkedIn 公司基于独特的设计实现为一个分布式的提交日志系统，之后成为 Apache 项目的一部分，号称大数据的杀手锏，在数据采集、传输、存储的过程中发挥着举足轻重的作用。

它是一个分布式的，支持多分区、多副本，基于 Zookeeper 的分布式消息流平台，它同时也是一款开源的基于发布订阅模式的消息引擎系统。

重要概念

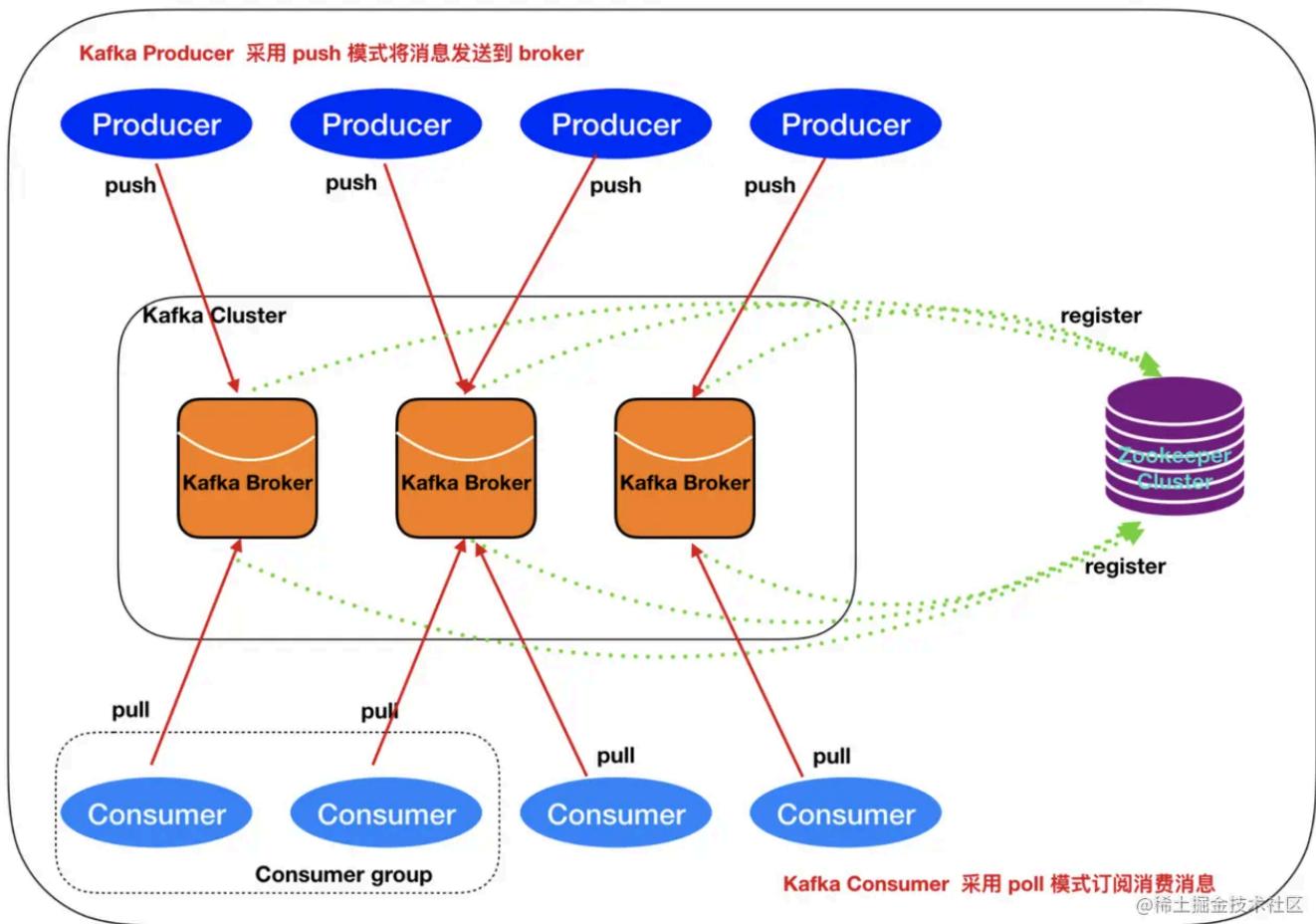
- **主题 (Topic)**：消息的种类称为主题，可以说一个主题代表了一类消息，相当于是对消息进行分类，主题就像是数据库中的表。
- **分区 (partition)**：主题可以被分为若干个分区，同一个主题中的分区可以不在一个机器上，有可能会部署在多个机器上，由此来实现 kafka 的伸缩性。
- **批次**：为了提高效率，消息会分批次写入 Kafka，批次就代指的是组消息。
- **消费者群组 (Consumer Group)**：消费者群组指的就是由一个或多个消费者组成的群体。

- **Broker 集群**: broker 集群由一个或多个 broker 组成。
- **重平衡 (Rebalance)** : 消费者组内某个消费者实例挂掉后，其他消费者实例自动重新分配订阅主题分区的过程。

Kafka 架构

一个典型的 Kafka 集群中包含 Producer、broker、Consumer Group、Zookeeper 集群。

Kafka 通过 Zookeeper 管理集群配置，选举 leader，以及在 Consumer Group 发生变化时进行 rebalance。Producer 使用 push 模式将消息发布到 broker，Consumer 使用 pull 模式从 broker 订阅并消费消息。

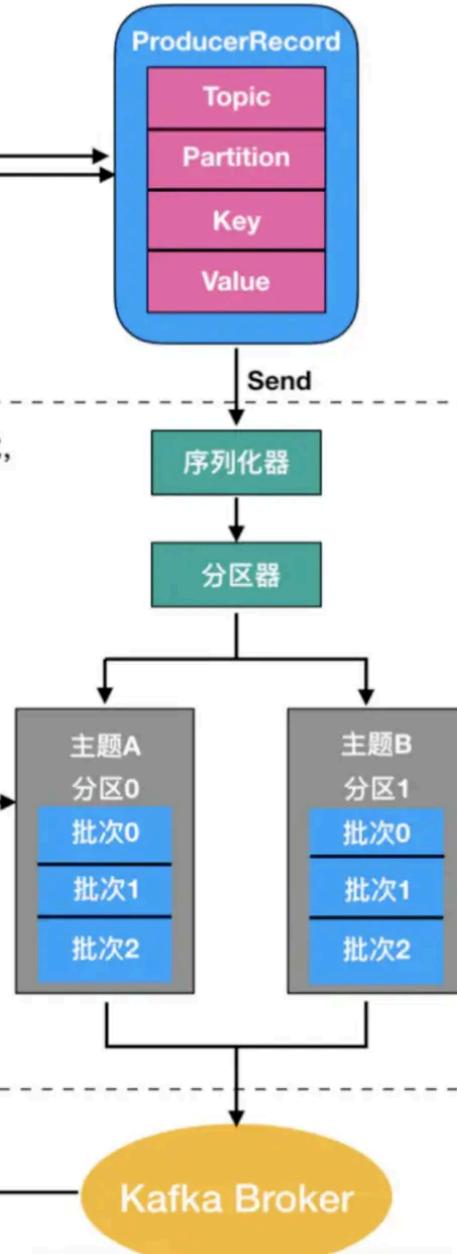


Kafka 工作原理

消息经过序列化后，通过不同的分区策略，找到对应的分区。

如果成功，
返回元数据

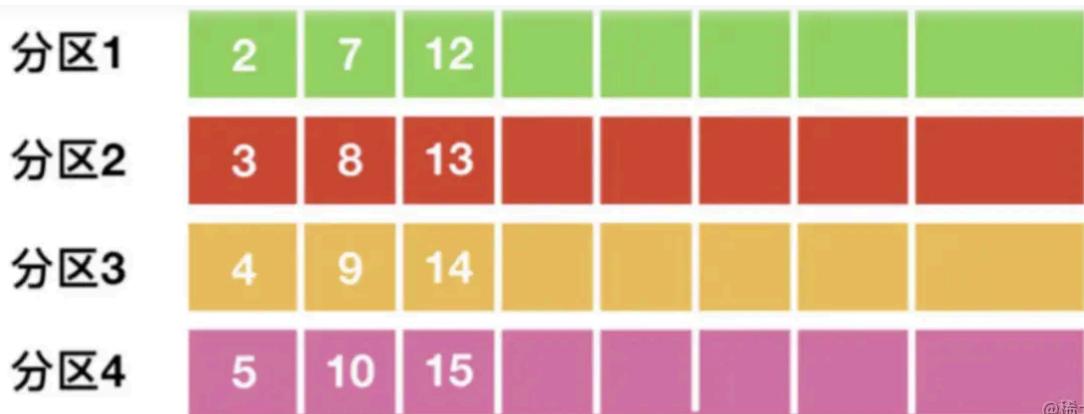
如果不能重试，
抛出异常



@稀土掘金技术社区

分区的策略包括顺序轮询、随机轮询和 key hash 这 3 种方式，那什么是分区呢？

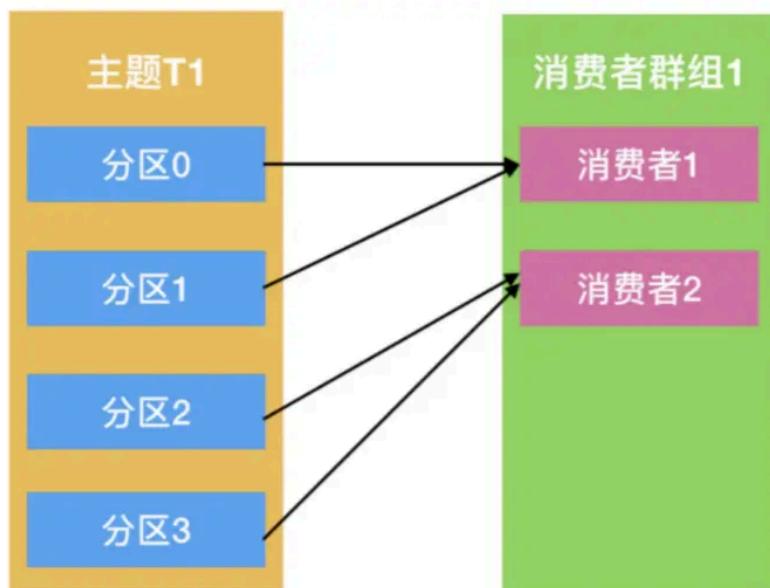
分区是 Kafka 读写数据的最小粒度，比如主题 A 有 15 条消息，有 5 个分区，如果采用顺序轮询的方式，15 条消息会顺序分配给这 5 个分区，后续消费的时候，也是按照分区粒度消费。



@稀土掘金技术社区

由于分区可以部署在多个不同的机器上，所以可以通过分区实现 Kafka 的伸缩性，比如主题 A 的 5 个分区，分别部署在 5 台机器上，如果下线一台，分区就变为 4。

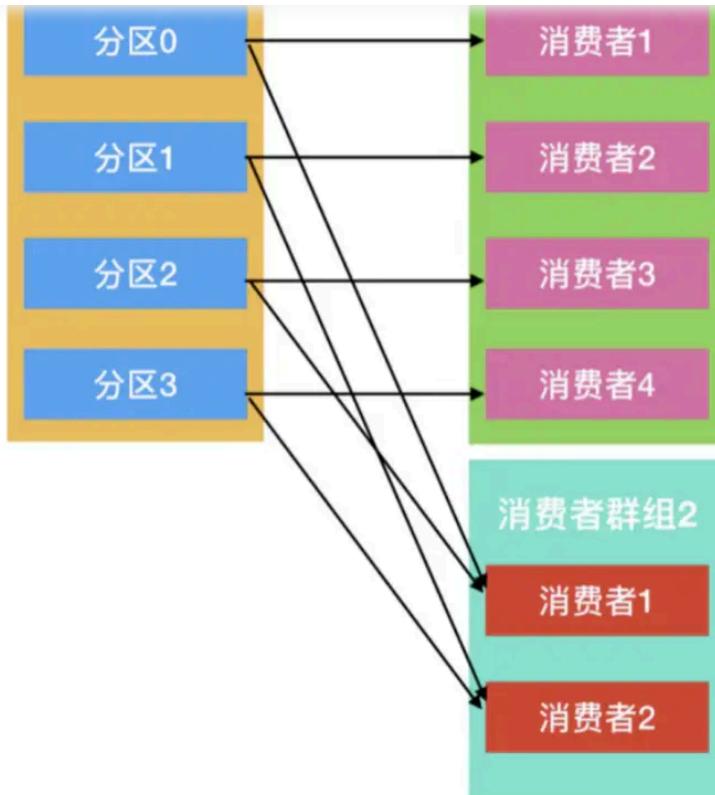
Kafka 消费是通过消费群组完成，同一个消费者群组，一个消费者可以消费多个分区，但是一个分区，只能被一个消费者消费。



@稀土掘金技术社区

如果消费者增加，会触发 **Rebalance**，也就是分区和消费者需要重新配对。

不同的消费群组互不干涉，比如下图的 2 个消费群组，可以分别消费这 4 个分区的消息，互不影响。



@稀土掘金技术社区

更多知识，详见 [《原理初探之 Kafka》](#)

RocketMQ

RocketMQ 是阿里开源的消息中间件，它是纯 Java 开发，具有高性能、高可靠、高实时、适合大规模分布式系统应用的特点。

RocketMQ 思路起源于 Kafka，但并不是 Kafka 的一个 Copy，它对消息的可靠传输及事务性做了优化，目前在阿里集团被广泛应用于交易、充值、流计算、消息推送、日志流式处理、binglog 分发等场景。

重要概念

- **Name 服务器 (NameServer)**：充当注册中心，类似 Kafka 中的 Zookeeper。
- **Broker**: 一个独立的 RocketMQ 服务器就被称为 broker，broker 接收来自生产者的消息，为消息设置偏移量。
- **主题 (Topic)**：消息的第一级类型，一条消息必须有一个 Topic。

- 分组 (Group)**：一个组可以订阅多个 Topic，包括生产者组 (Producer Group) 和消费者组 (Consumer Group)。
- 队列 (Queue)**：可以类比 Kafka 的分区 Partition。

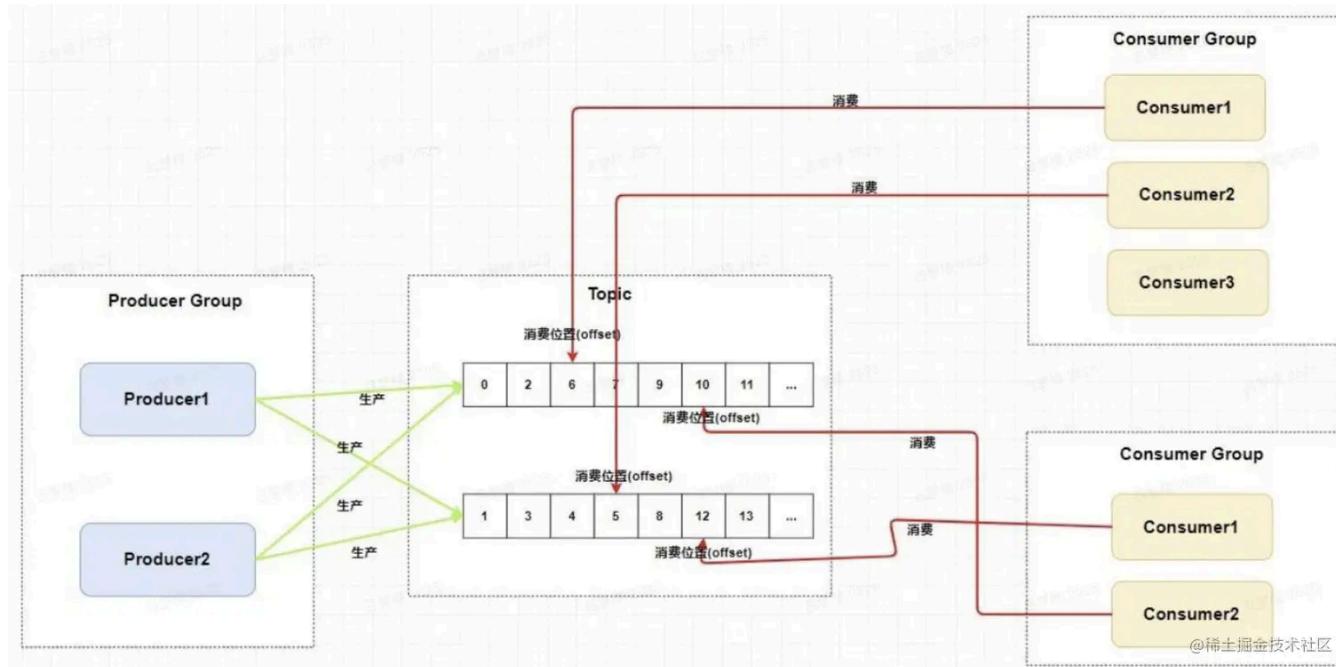
RocketMQ 工作原理

RocketMQ 中的消息模型就是按照主题模型所实现的，包括 Producer Group、Topic、Consumer Group 三个角色。

为了提高并发能力，一个 Topic 包含多个 Queue，生产者组根据主题将消息放入对应的 Topic，下图是采用轮询的方式找到里面的 Queue。

RocketMQ 中的消费群组和 Queue，可以类比 Kafka 中的消费群组和 Partition：不同的消费者组互不干扰，一个 Queue 只能被一个消费者消费，一个消费者可以消费多个 Queue。

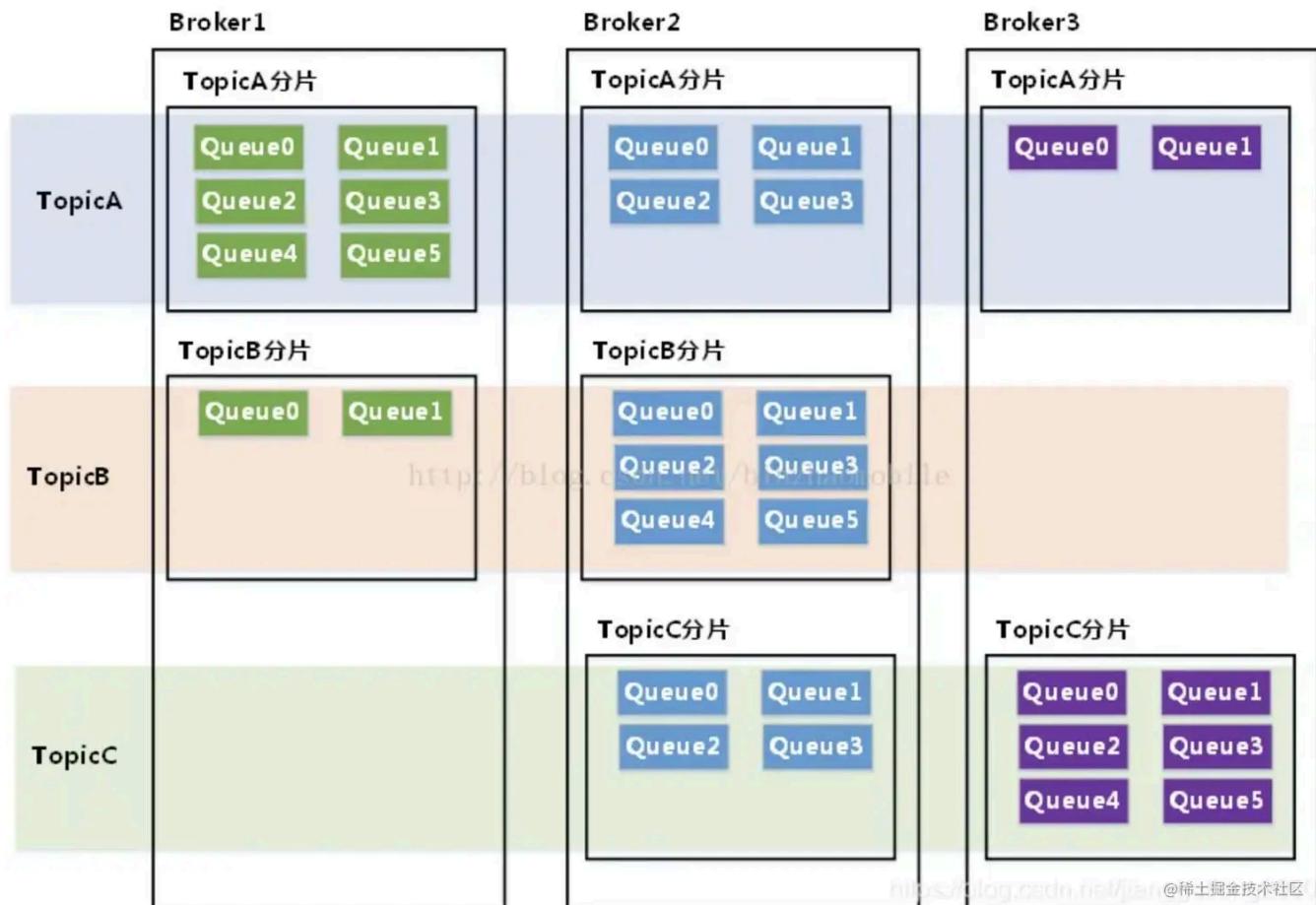
消费 Queue 的过程中，通过偏移量记录消费的位置。



RocketMQ 架构

RocketMQ 技术架构中有四大角色 NameServer、Broker、Producer 和 Consumer，下面主要介绍 Broker。

如果某个 Topic 消息量很大，应该给它多配几个 Queue，并且尽量多分布在不同 broker 上，以减轻某个 broker 的压力。Topic 消息量都比较均匀的情况下，如果某个 broker 上的队列越多，则该 broker 压力越大。



简单提一下，Broker 通过集群部署，并且提供了 master/slave 的结构，slave 定时从 master 同步数据（同步刷盘或者异步刷盘），如果 master 宕机，则 slave 提供消费服务，但是不能写入消息。

看到这里，大家应该可以发现，RocketMQ 的设计和 Kafka 真的很像！

更多知识，详见 [《原理初探之 RocketMQ》](#)

RabbitMQ

RabbitMQ 2007 年发布，是使用 Erlang 语言开发的开源消息队列系统，基于 AMQP 协议来实现。

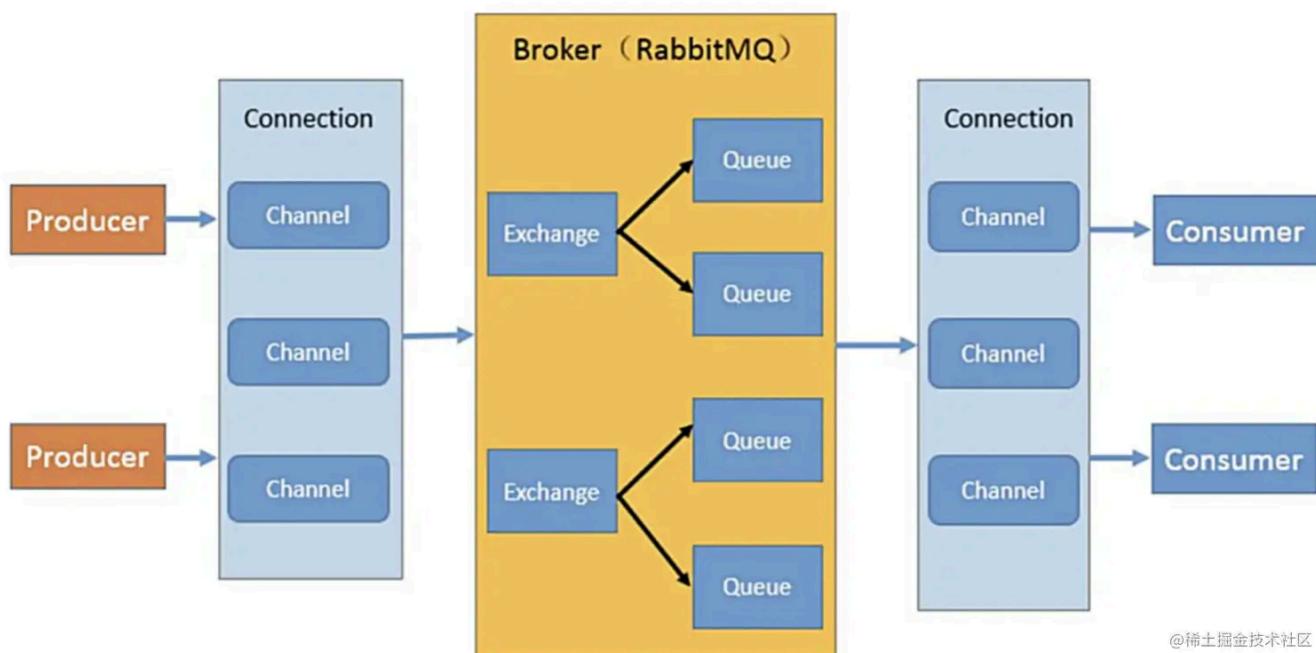
重要概念

- 信道（Channel）**：消息读写等操作在信道中进行，客户端可以建立多个信道，每个信道代表一个会话任务。
- 交换器（Exchange）**：接收消息，按照路由规则将消息路由到一个或者多个队列；如果路由不到，或者返回给生产者，或者直接丢弃。
- 路由键（RoutingKey）**：生产者将消息发送给交换器的时候，会发送一个 RoutingKey，用来指定路由规则，这样交换器就知道把消息发送到哪个队列。
- 绑定（Binding）**：交换器和消息队列之间的虚拟连接，绑定中可以包含一个或者多个 RoutingKey。

RabbitMQ 工作原理

AMQP 协议模型由三部分组成：生产者、消费者和服务端，执行流程如下：

1. 生产者是连接到 Server，建立一个连接，开启一个信道。
2. 生产者声明交换器和队列，设置相关属性，并通过路由键将交换器和队列进行绑定。
3. 消费者也需要进行建立连接，开启信道等操作，便于接收消息。
4. 生产者发送消息，发送到服务端中的虚拟主机。
5. 虚拟主机中的交换器根据路由键选择路由规则，发送到不同的消息队列中。
6. 订阅了消息队列的消费者就可以获取到消息，进行消费。



@稀土掘金技术社区

RabbitMQ 常用的交换器类型有 direct、topic、fanout、headers 四种，每种方法的详细介绍看这篇 [《入门RabbitMQ，这一篇绝对够！》](#)。

具体的使用方法，可以参考官网：

- 官网入口：www.rabbitmq.com/getstarted....

1 "Hello World!"

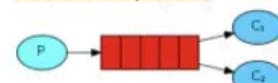
The simplest thing that does something



- [Python](#)
- [Java](#)
- [Ruby](#)
- [PHP](#)
- [C#](#)
- [JavaScript](#)
- [Go](#)
- [Elixir](#)
- [Objective-C](#)
- [Swift](#)
- [Spring AMQP](#)

2 Work queues

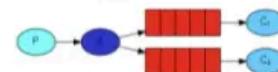
Distributing tasks among workers (the [competing consumers pattern](#))



- [Python](#)
- [Java](#)
- [Ruby](#)
- [PHP](#)
- [C#](#)
- [JavaScript](#)
- [Go](#)
- [Elixir](#)
- [Objective-C](#)
- [Swift](#)
- [Spring AMQP](#)

3 Publish/Subscribe

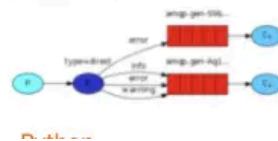
Sending messages to many consumers at once



- [Python](#)
- [Java](#)
- [Ruby](#)
- [PHP](#)
- [C#](#)
- [JavaScript](#)
- [Go](#)
- [Elixir](#)
- [Objective-C](#)
- [Swift](#)
- [Spring AMQP](#)

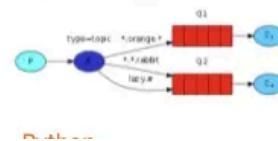
4 Routing

Receiving messages selectively



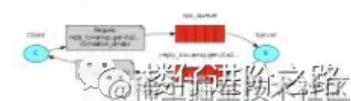
5 Topics

Receiving messages based on a pattern (topics)



6 RPC

[Request/reply pattern](#)
example



- [Python](#)

更多知识，详见 [《入门RabbitMQ，这一篇绝对够！》](#)

消息队列对比&选型

REQUEST-REPLY	支持	支持		
API完备性	高	高	高	高
多语言支持	支持，JAVA优先	语言无关	只支持JAVA	支持，java优先
单机吞吐量	万级	万级	万级	十万级
消息延迟		微秒级	毫秒级	毫秒级
可用性	高（主从）	高（主从）	非常高（分布式）	非常高（分布式）
消息丢失	低	低	理论上不会丢失	理论上不会丢失
消息重复		可控制		理论上会有重复
文档的完备性	高	高	高	高
提供快速入门	有	有	有	有
首次部署难度		低		中
社区活跃度	高	高	中	高
商业支持	无	无	阿里云	无
成熟度	成熟	成熟	比较成熟	成熟且领领域
特点	功能齐全，被大量开源项目使用	由于Erlang语言的并发能力，性能很好	各个环节分布式扩展设计，主从 HA；支持上万个队列；多种消费模式；性能很好	
支持协议	OpenWire、STOMP、REST、XMPP、AMQP	AMQP	自己定义的一套(社区提供JMS--不成熟)	
持久化	内存、文件、数据库	内存、文件	磁盘文件	
事务	支持	不支持	支持	
负载均衡	支持	支持	支持	
管理界面	一般	好	有web console实现	
部署方式	独立、嵌入	独立	独立	
评价	优点： 成熟的产品，已经在很多公司得到应用（非大规模场景）。有较多的文档。各种协议支持较好，有多重语言的成熟的客户端； 缺点： 根据其他用户反馈，会出莫名其妙的问题，切会丢失消息。其重心放到 activemq6.0 产品—apollo 上去了，目前社区不活跃，且对5.x 维护较少；Activemq 不适合用于上千个队列的应用场景。	优点： 由于erlang语言的特性，mq性能较好；管理界面较丰富，在互联网公司也有较大规模的应用；支持amqp系类，有多中语言且支持amqp的客户端可用； 缺点： erlang语言难度较大。集群不支持动态扩展。	优点： 模型简单，接口易用（JMS的接口很多场合并不太实用）。在阿里大规模应用。目前支付宝中的余额宝等新兴产品均使用 rocketmq 。集群规模大概在50台左右，单日处理消息上百亿；性能非常好，可以大量堆积消息在 broker 中，支持多种消费，包括集群消费、广播消费等。开发度较活跃，版本更新很快。 缺点： 产品较新文档比较缺乏。没有在mq核心中去实现JMS等接口，对已有系统而言不能兼容。阿里内部还有一套未开源的MQAPI，这一层API可以将上层应用和下层MQ的实现解耦（阿里内部有多个mq的实现，如notify、metaq1.x、metaq2.x、rocketmq等），使得下面mq可以很方便的进行切换和升级而对应用无任何影响。目前这一套东西未开源。	@稀土掘金技术社区

消息队列对比

Kafka

优点：

- 高吞吐、低延迟：** Kafka 最大的特点就是收发消息非常快，Kafka 每秒可以处理几十万条消息，它的最低延迟只有几毫秒；
- 高伸缩性：** 每个主题（topic）包含多个分区（partition），主题中的分区可以分布在不同的主机（broker）中；
- 高稳定性：** Kafka 是分布式的，一个数据多个副本，某个节点宕机，Kafka 集群能够正常工作；



- **消息有序：**通过控制能够保证所有消息被消费且仅被消费一次；
- **有优秀的第三方 Kafka Web 管理界面 Kafka-Manager，**在日志领域比较成熟，被多家公司和多个开源项目使用。

缺点：

- Kafka 单机超过 64 个队列/分区，Load 会发生明显的飙升现象，队列越多，load 越高，发送消息响应时间变长；
- **不支持消息路由，不支持延迟发送，不支持消息重试；**
- 社区更新较慢。

RocketMQ

优点：

- **高吞吐：**借鉴 Kafka 的设计，单一队列百万消息的堆积能力；
- **高伸缩性：**灵活的分布式横向扩展部署架构，整体架构其实和 kafka 很像；
- **高容错性：**通过ACK机制，保证消息一定能正常消费；
- **持久化、可回溯：**消息可以持久化到磁盘中，支持消息回溯；
- 消息有序：在一个队列中可靠的先进先出（FIFO）和严格的顺序传递；
- 支持发布/订阅和点对点消息模型，支持拉、推两种消息模式；
- 提供 docker 镜像用于隔离测试和云集群部署，提供配置、指标和监控等功能丰富的 Dashboard。

缺点：

- 不支持消息路由，支持的客户端语言不多，目前是 java 及 c++，其中 c++ 不成熟；
- 部分支持消息有序：需要将同一类的消息 hash 到同一个队列 Queue 中，才能支持消息的顺序，如果同一类消息散落到不同的 Queue 中，就不能支持消息的顺序。
- 社区活跃度一般。

RabbitMQ

优点：



- **支持消息路由：**RabbitMQ 可以通过不同的交换器支持不同种类的消息路由；
- **消息时序：**通过延时队列，可以指定消息的延时时间，过期时间TTL等；
- **支持容错处理：**通过交付重试和死信交换器（DLX）来处理消息处理故障；
- 提供了一个易用的用户界面，使得用户可以监控和管理消息 Broker；
- **社区活跃度高。**

缺点：

- **Erlang 开发，很难去看懂源码，不利于做二次开发和维护，基本职能依赖于开源社区的快速维护和修复 bug；**
- **RabbitMQ 吞吐量会低一些，这是因为他做的实现机制比较重；**
- 不支持消息有序、持久化不好、不支持消息回溯、伸缩性一般。

消息队列选型

- Kafka：追求高吞吐量，一开始的目的就是用于日志收集和传输，**适合产生大量数据的互联网服务的数据收集业务**，大型公司建议可以选用，**如果有日志采集功能，肯定是首选 kafka。**
- RocketMQ：**天生为金融互联网领域而生，对于可靠性要求很高的场景，尤其是电商里面的订单扣款，以及业务削峰，在大量交易涌入时，后端可能无法及时处理的情况。RocketMQ 在稳定性上可能更值得信赖，这些业务场景在阿里双 11 已经经历了多次考验，如果你的业务有上述并发场景，建议可以选择 RocketMQ。**
- RabbitMQ：结合 erlang 语言本身的并发优势，性能较好，社区活跃度也比较高，但是不利于做二次开发和维护，不过 RabbitMQ 的社区十分活跃，可以解决开发过程中遇到的 bug。**如果你的数据量没有那么大，小公司优先选择功能比较完备的 RabbitMQ。**
- ActiveMQ：官方社区现在对 ActiveMQ 5.x 维护越来越少，较少在大规模吞吐的场景中使用。

「每日推荐」

- **推荐文章：**[微服务网关：从对比到选型，由理论到实践](#)
- **推荐理由：**非常硬核的网关选型文章，转载阅读量 3W +，里面有我司自研的微服务网关。

尽信书则不如无书，因个人能力有限，难免有疏漏和错误之处，如发现 bug 或者有更好的建议，欢迎批评指正，不吝感激。

标签： 消息队列

评论 1

[登录 / 注册](#) 即可发布评论![最热](#) | [最新](#)

王功胜 软件开发工程师 @华为技术有限公司

2月前 点赞 评论

...

目录

[收起 ^](#)

消息队列基础

[什么是消息队列?](#)[消息队列模式](#)[消息队列应用场景](#)

常用消息队列

[Kafka](#)[重要概念](#)[Kafka 架构](#)[Kafka 工作原理](#)[RocketMQ](#)[重要概念](#)



探索稀土掘金



RabbitMQ

重要概念

RabbitMQ 工作原理

常用交换器

消息队列对比&选型

消息队列对比

Kafka

RocketMQ

RabbitMQ

消息队列选型

「每日推荐」

搜索建议

搜索关键词



消息队列原理及选型

消息队列工作原理对比以及选型

消息队列（三）常见消息队列介绍

不同消息队列的选型和对比

4 种消息队列，如何选型？

消息队列的消息队列与容器结合

消息队列的消息队列与容器结合

带你玩转消息队列和相关选型！

消息队列（一）为什么需要消息队列

为你推荐

新来个技术总监，把 RabbitMQ 讲的那叫一个透彻，佩服！

楼仔 | 1年前 | 20k | 296 | 29

RabbitMQ

4 种消息队列，如何选型？

楼仔 | 5月前 | 3.2k | 6 | 评论

消息队列

消息队列选型之 Kafka vs RabbitMQ



首页 ▾

探索稀土掘金



市小项目/月忘API↑↑↑大逃杀！

腾讯云开发者 | 1年前 | ⚡ 2.9k | ⬆ 9 | 🗣 评论

程序员

RabbitMQ和Kafka综合对比

唔贝 | 2年前 | ⚡ 2.3k | ⬆ 3 | 🗣 评论

Java 后端

消息队列简介及 RabbitMQ 的使用方法

somenzz | 2年前 | ⚡ 1.4k | ⬆ 点赞 | 🗣 评论

RabbitMQ

【进阶之路】消息队列——RabbitMQ原理（二）

南橘ryc | 4年前 | ⚡ 6.2k | ⬆ 28 | 🗣 评论

消息队列

RabbitMQ 入门教程与实战

青火_ | 2年前 | ⚡ 638 | ⬆ 点赞 | 🗣 评论

Rabbit...

得物技术消息中间件应用的常见问题与方案

得物技术 | 2年前 | ⚡ 1.5k | ⬆ 27 | 🗣 3

JavaScript

Kafka

wenxuan | 7月前 | ⚡ 186 | ⬆ 点赞 | 🗣 评论

Kafka Java

关于「消息队列」你应该知道的事

程序员狮子 | 2年前 | ⚡ 630 | ⬆ 2 | 🗣 2

后端

java消息队列基础和RabbitMQ相关概念

终有救赎 | 9月前 | ⚡ 268 | ⬆ 12 | 🗣 3

后端

高并发系统-消息队列-如何正确使用消息队列

技术驿站 | 4月前 | ⚡ 2.4k | ⬆ 12 | 🗣 1

后端 架构 Java

入门RabbitMQ，这一篇绝对够！

楼仔 | 2年前 | ⚡ 471 | ⬆ 2 | 🗣 评论

RabbitMQ 后端

浅谈消息模型

流年2020 | 3年前 | ⚡ 610 | ⬆ 3 | 🗣 评论

RocketMQ



首页 ▾

探索稀土掘金

