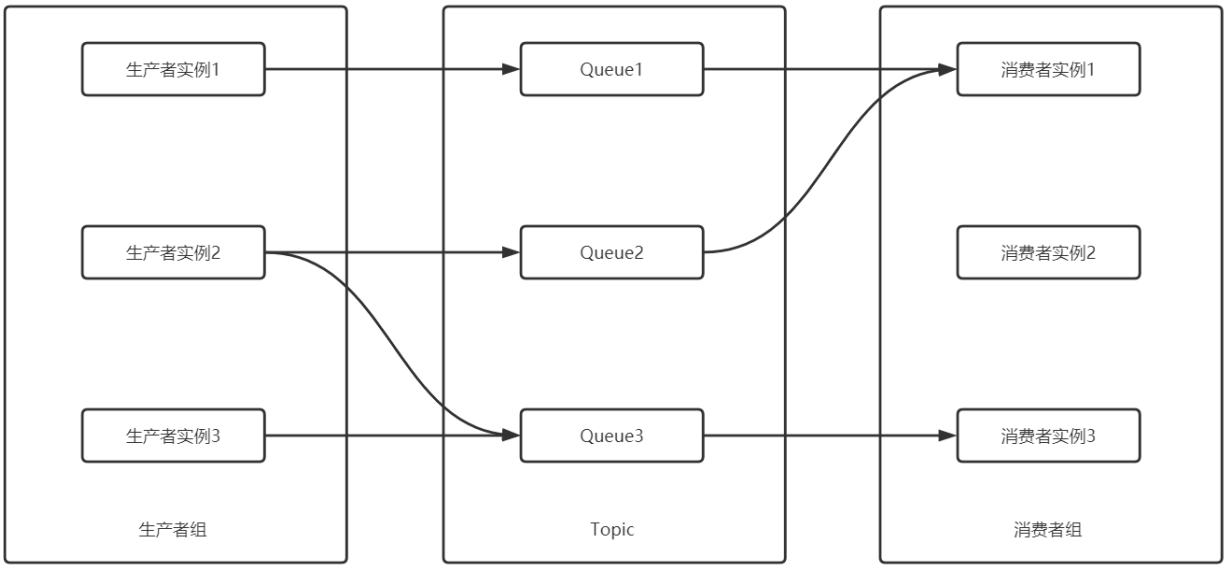
消费者概述

几个关键概念



- 消费者组：一个逻辑概念，在使用消费者时需要指定一个组名。一个消费者组可以订阅多个Topic。
- 消费者实例：一个消费者组程序部署了多个进程，每个进程都可以称为一个消费者实例。
- 订阅关系：一个消费者组订阅一个 Topic 的某一个 Tag，这种记录被称为订阅关系。

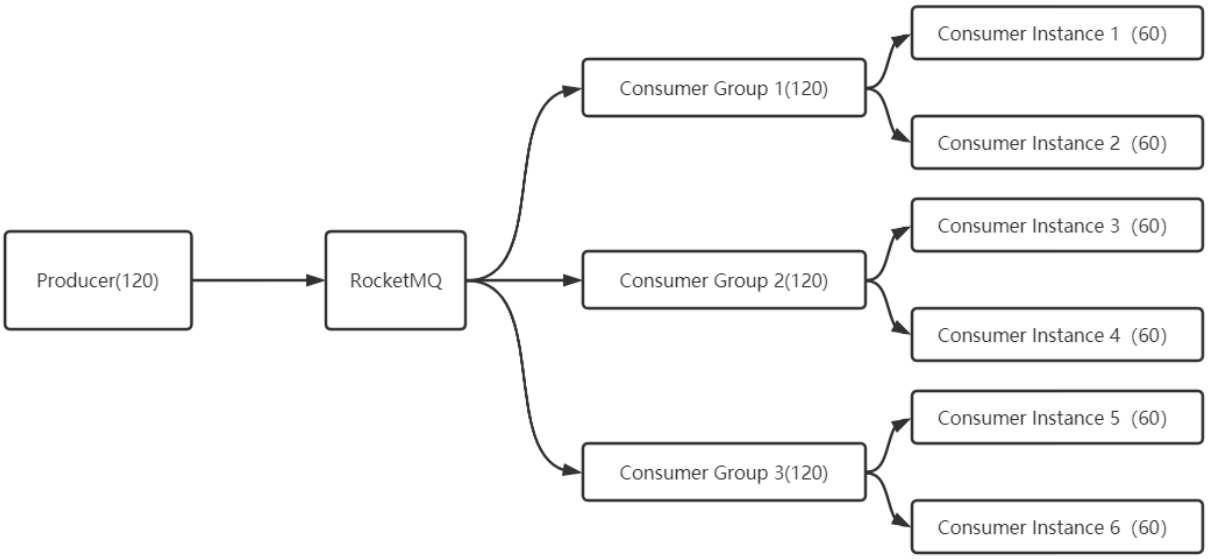
PS：RocketMQ规定消费订阅关系（消费者组名-Topic-Tag）必须一致，一定要重视这个问题，一个消费者组中的实例订阅的Topic和Tag必须完全一致，否则就是订阅关系不一致。订阅关系不一致会导致消费消息紊乱。

消费模式

RocketMQ目前支持**集群消费模式**和**广播消费模式**，其中集群消费模式使用最为广泛。

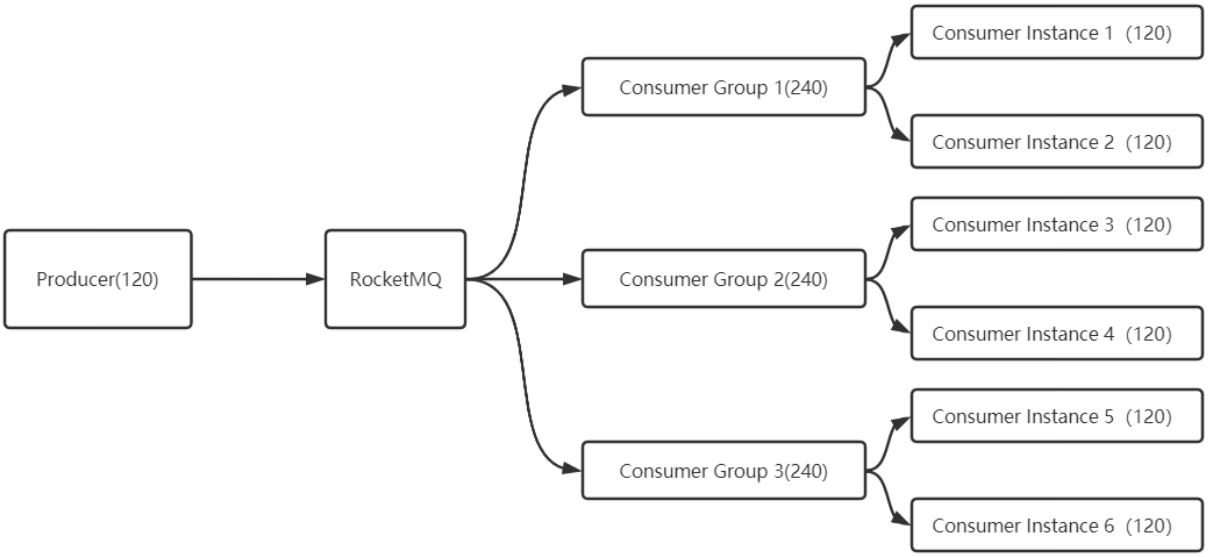
集群消费模式

在同一个消费者组中的消费者实例，是负载均衡（策略可以配置）地消费Topic中的消息，假如有一个生产者（Producer）发送了 120 条消息，其所属的 Topic 有 3 个消费者（Consumer）组，每个消费者组设置为集群消费，分别有2个消费者实例。



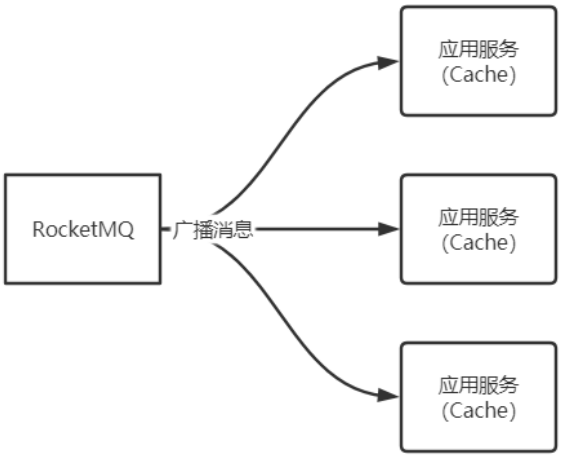
Consumer Group 1的两个实例分别负载均衡地消费60条消息。由此我们可以得出使用负载均衡策略时，每个**消费者实例消费消息数=生产消息数/消费者实例数**，在本例中是60=120/2。目前大部分场景都适合集群消费模式，RocketMQ 的消费模式**默认是集群消费**。比如异步通信、削峰等对消息没有顺序要求的场景都适合集群消费。因为**集群模式的消费进度是保存在Broker端的**，所以即使应用崩溃，消费进度也不会出错。

广播消费模式



广播消费，顾名思义全部的消息都是广播分发，即消费者组中的全部消费者实例将消费整个 Topic 的全部消息。比如，有一个生产者生产了 120 条消息，其所属的 Topic 有 3个消费者组，每个消费者组设置为广播消费，分别有两个消费者实例，Consumer Group 1中的消费者1和消费者2分别消费120条消息。整个消费者组收到消息120×2=240条。由此我们可以得出广播消费时，每个**消费者实例的消费消息数=生产者生产的消息数**，整个消费者组中所有实例消费消息数=每个消费者实例消费消息数×消费者实例数，本例中是240=120×2。

广播消费比较适合各个消费者实例都需要通知的场景，比如刷新应用服务器中的缓存



生产者发一个刷新缓存的广播消息，消费者组如果设置为广播消费，那么每个应用服务中的消费者都可以消费这个消息，也都能刷新缓存。

广播消费的消费进度保存在客户端机器的文件中。如果文件弄丢了，那么消费进度就丢失了，可**能会导致部分消息没有消费**。

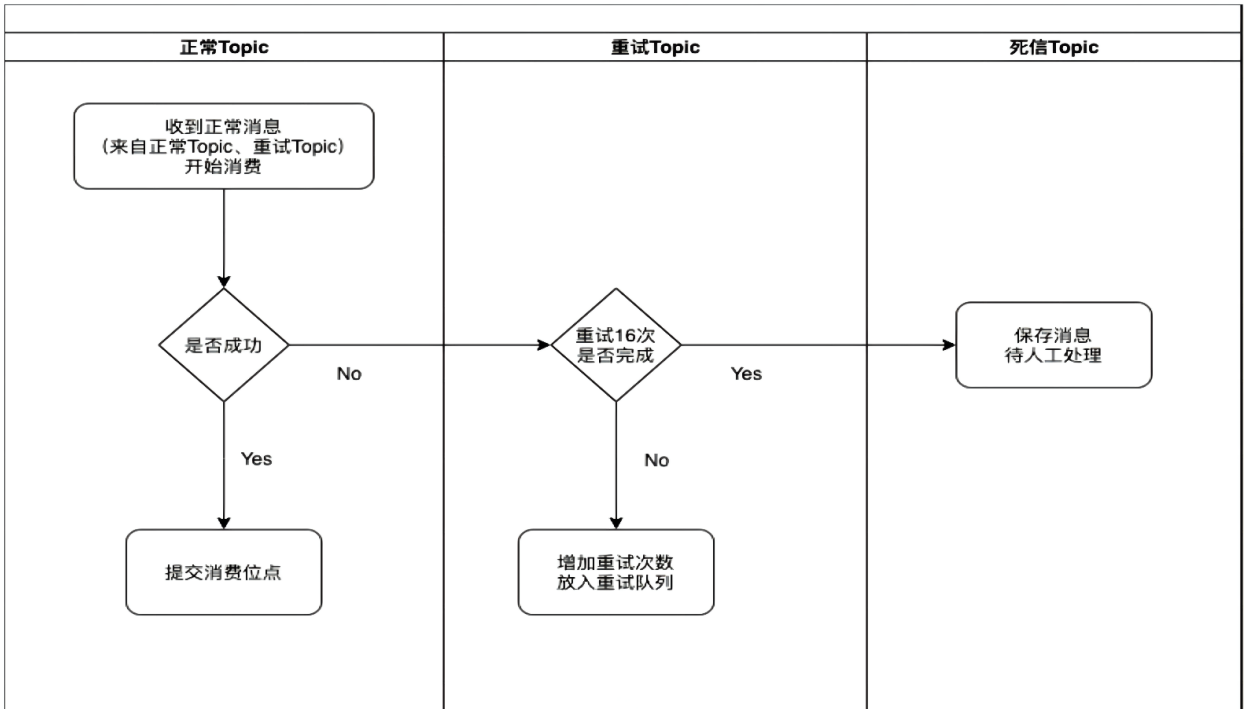
可靠消费

RocketMQ是一种十分可靠的消息队列中间件，消费侧通过**重试-死信机制**、**Rebalance机制**等多种机制保证消费的可靠性。

重试-死信机制

我们假设有一个场景，在消费消息时由于网络不稳定导致一条消息消费失败。此时是让生产者重新手动发消息呢，还是自己做数据补偿？RocketMQ 告诉你，消费不是一锤子买卖。

横向看，RocketMQ的消费过程分为 3个阶段：正常消费、重试消费和死信。在引进了正常Topic、重试队列、死信队列后，消费过程的可靠性提高了。



- 正常Topic：正常消费者订阅的Topic名字。

- 重试Topic：如果由于各种意外导致消息消费失败，那么该消息会自动被保存到重试Topic中，格式为“%RETRY%消费者组”，在订阅的时候会自动订阅这个重试Topic。

进入重试队列的消息有16次重试机会，每次都会按照一定的时间间隔进行。RocketMQ 认为消费不是一锤子买卖，可能由于各种偶然因素导致正常消费失败，只要正常消费或者重试消费中有一次消费成功，就算消费成功。

第几次重试	与上次重试的间隔时间	第几次重试	与上次重试的间隔时间
1	10 秒	9	7 分钟
2	30 秒	10	8 分钟
3	1 分钟	11	9 分钟
4	2 分钟	12	10 分钟
5	3 分钟	13	20 分钟
6	4 分钟	14	30 分钟
7	5 分钟	15	1 小时
8	6 分钟	16	2 小时

死信Topic：死信Topic名字格式为“%DLQ%消费者组名”。如果正常消费1次失败，重试16次失败，那么消息会被保存到死信Topic中，进入死信Topic的消息不能被再次消费。RocketMQ认为，如果17次机会都失败了，说明生产者发送消息的格式发生了变化，或者消费服务出现了问题，需要人工介入处理。

Rebalance机制

Rebalance（重平衡）机制，用于在发生Broker掉线、Topic扩容和缩容、消费者扩容和缩容等变化时，自动感知并调整自身消费，以尽量减少甚至避免消息没有被消费。

两种消息获取方式

DefaultMQPullConsumer

该消费者使用时需要用户主动从 Broker 中 Pull 消息和消费消息，提交消费位点。用户主动Pull消息，自主管理位点，可以灵活地掌控消费进度和消费速度，适合流计算、消费特别耗时等特殊的消费场景。缺点也显而易见，需要从代码层面精准地控制消费，对开发人员有一定要求。在 RocketMQ 中 org.apache.rocketmq.client.consumer.DefaultMQPullConsumer是默认的Pull消费者实现类。

DefaultMQPushConsumer

由Broker主动向消费者推送最新的消息。代码接入非常简单，适合大部分业务场景。缺点是灵活度差，在了解其消费原理后，排查消费问题方可简单快捷。在 RocketMQ 中

org.apache.rocketmq.client.consumer.DefaultMQPushConsumer是默认的Push消费者实现类。

消费方式/对比项	Pull	Push	备　注
是否需要主动拉取	理解分区后，需要主动拉取各个分区消息	自动	Pull 消息灵活；Push 使用更简单
位点管理	用户自行管理或者主动提交给 Broker 管理	Broker 管理	Pull 自主管理位点，消费灵活；Push 位点交由 Broker 管理
Topic 路由变更是否影响消费	否	否	Pull 模式需要编码实现路由感知；Push 模式自动执行 Rebalance，以适应路由变更