

24 | 消息顺序：保证消息有序，一个 topic 只能有一个 partition 吗？

2023-8-11 邓明



你好，我是大明。我们今天接着学习消息队列的新主题——有序消息。

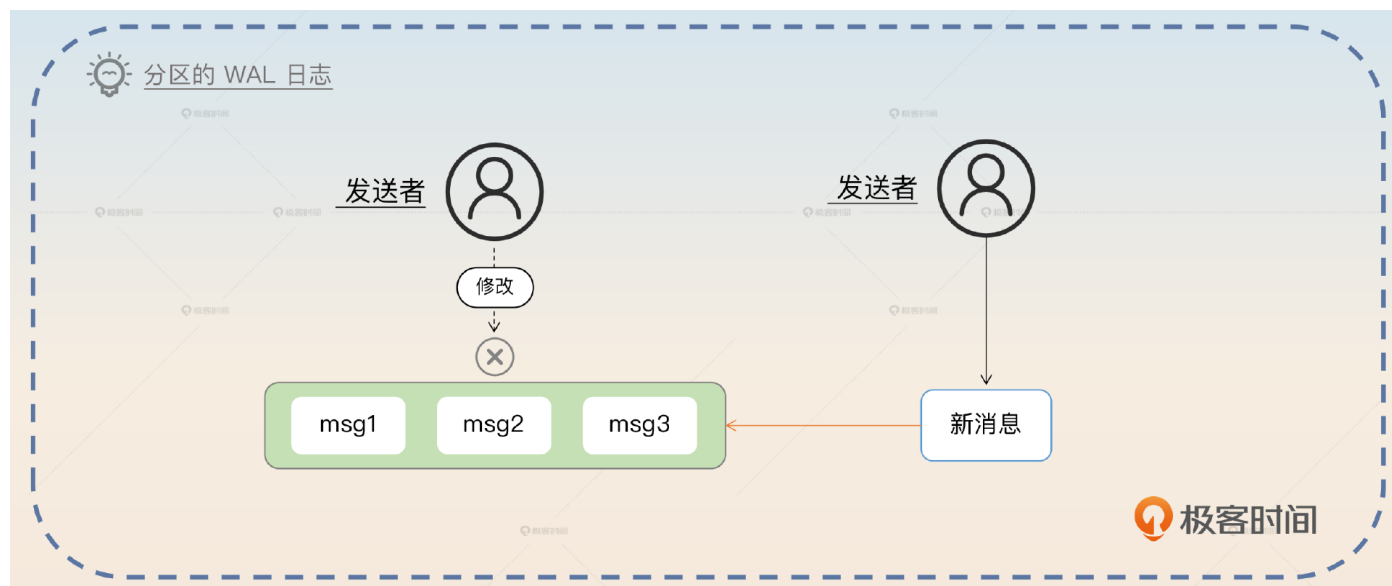
在消息队列的相关的面试里面，有序消息和消息不丢失、消息重复消费是三个并列的面试热点，同时在实践中也很容易遇到要求使用有序消息的场景。但是大部分人在面试的时候，无法深入透彻地讨论这个问题。大多数时候，只能说出 topic 只能有一个分区这种最简单的方案。当面试官追问这种方案有什么缺陷的时候就开始答不上来了。

所以今天我就带你深入了解有序消息的方方面面，深挖解决方案的缺陷以及对应的改进策略。

消息在分区上的组织方式

在 Kafka 中，消息是以分区为单位进行存储的。分区是逻辑上的概念，用于对消息进行水平划分和并行处理。每个 topic 都可以被划分为一个或多个分区，每个分区都是一个**有序、不可变**的消息日志。

Kafka 使用 WAL (write-ahead-log) 日志来存储消息。每个分区都有一个对应的日志文件，新的消息会被追加到文件的末尾，而已经加入日志里的消息，就不会再被修改了。



每个消息在分区日志里都有一个唯一的偏移量 (offset) ，用来标识消息在分区里的位置。

Kafka 保证同一分区内的消息顺序，但不保证不同分区之间的顺序。

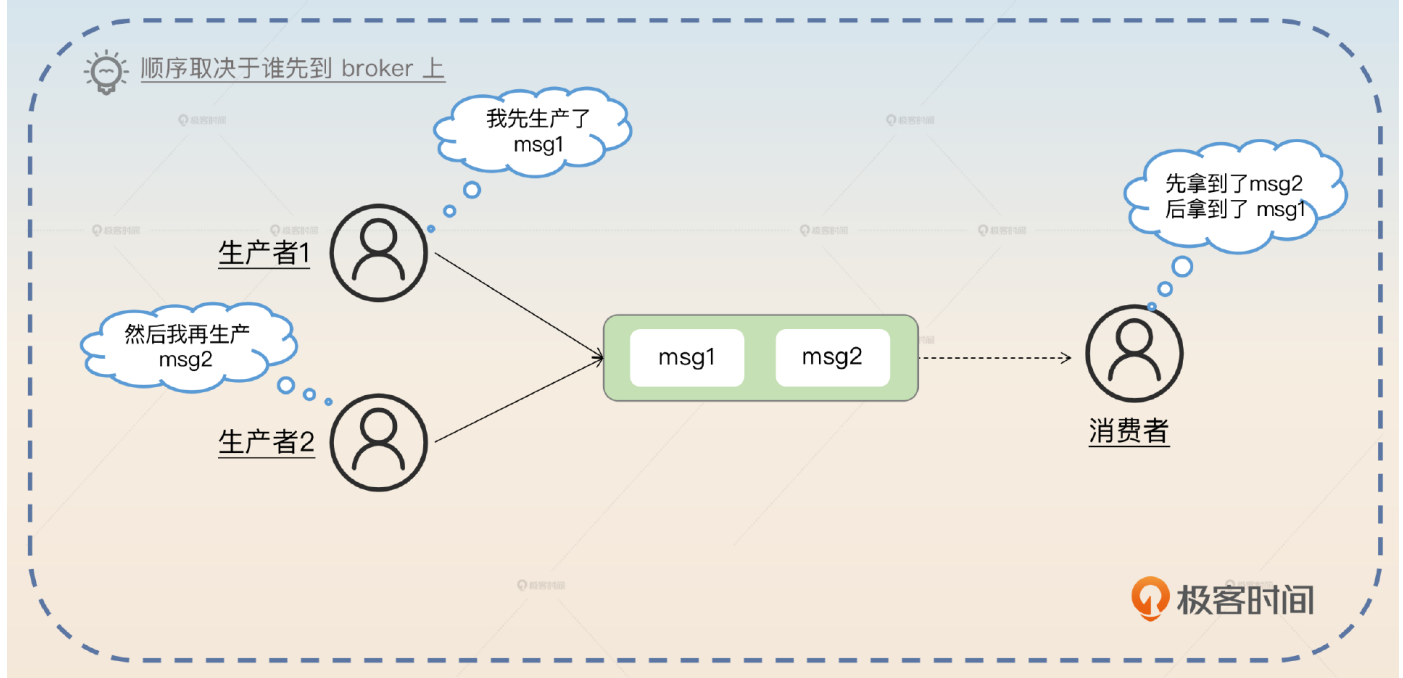
而 Kafka 本身暴露了对应的接口，也就是说你可以显式地指定消息要发送到哪个分区，也可以显式地指定消费哪个分区的数据。

什么是有序消息？

在消息队列里面，有序消息是指消费者消费某个 topic 消息的顺序，和生产者生产消息的顺序一模一样，它也叫做顺序消息。

前面你应该注意到了，Kafka 并不能保证不同分区之间的顺序。也就是说，如果业务上有先后顺序的消息被发送到不同的分区上，那么你难以确定哪一个消息会先被消费。

你需要注意一点语义上的差别，这里说生产者生产消息的顺序，不是指你创建出来消息那个实例的先后顺序，而是指broker收到的顺序。比如说你有两个生产者，一个生产者先生产了 msg1，另外一个生产者生产了 msg2，但是 msg2 先发到了broker上。那么实际上我们认为 msg2 是先于 msg1 的。



如果你要求的是两个生产者，一个生产者一定要先于另外一个生产者发送一条消息，那么这实际上已经超出了消息队列要解决的问题的范畴了，它属于在分布式环境下如何协调不同节点按照先后顺序执行一定的步骤这个问题范畴。有序消息强调的是某个 topic 内，而不是跨 topic 的有序消息。

跨 topic 的有序消息

不过有些时候可能因为业务特征或者历史问题，业务会要求你在不同的 topic 之间也保证消息是有序的。比如说 msg1 先发送到了 topic_a 上，msg2 被发送到了 topic_b 上。但是在业务层面上，要求 msg1 一定要先于 msg2 消费。

这种场景在事件驱动的架构中更加常见。在复杂的事件驱动架构下，我们可能会倾向于使用不同的 topic 来代表不同的事件，那么就会遇到要求在不同的 topics 下消息依旧需要保持有序的问题。

这一类的问题是不能依赖于消息队列来解决的。要想支持这种跨 topic 的有序消息，一定要引入一个协调者，这个协调者负责把消息重组为有序消息。比如说，如果 msg2 先到了，但是 msg1 还没出来，那么这个协调者要有办法让 msg2 的消费者 B 停下来，暂时不消费 msg2。而在 msg1 来了之后，唤醒消费者 A 消费 msg1，并且在消费完 msg1 之后要再唤醒消费者 B 处理 msg2。

应当说，这是一个全新的话题，面试基本上不会出现这一类问题，你心中有个基本的概念就可以了。

面试准备

在准备面试的时候，你需要了解一下公司内部使用有序消息的情况。

在什么业务场景下，你需要用到有序消息？

你是如何解决有序消息这个问题的？用的是哪种方案？

如果你用的是单分区解决方案，那么有没有消息积压问题？如果有，你是怎么解决的？

如果你用的是多分区解决方案，那么有没有分区负载不均衡的问题？如果有，你是怎么解决的？

你可以在简历介绍技能的部分或者项目部分点出你解决过消息有序性的问题。基本上你只要介绍到了，面试官就肯定会问你这方面的问题。

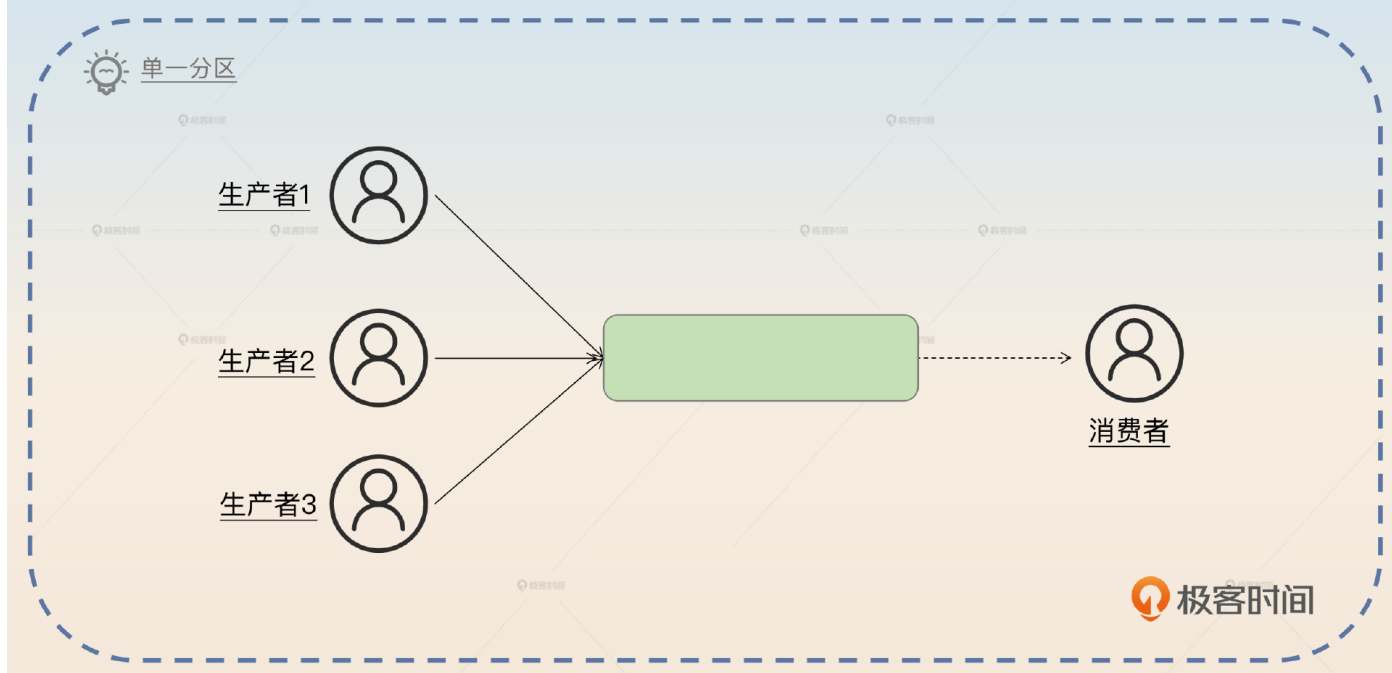
如果你自己在实践中真的用了全局有序的单分区解决方案，但是业务层面上其实并不要求全局有序，那么你可以尝试使用一下多分区解决方案。这样一来，你就会对两个方案有非常深入的理解，也能把握更多细节。

除此之外，如果面试官问到了这些问题，你也可以把话题引导到有序消息这个话题上。

1. 增加分区的问题，后面的多分区方案专门讨论了增加分区可能带来的消息失序的问题。
2. Redis 的槽和槽分配。
3. 负载均衡，你记得回答一致性哈希，然后把话题引导到利用一致性哈希来解决多分区数据分布不均匀的问题。
4. 消息积压的问题，你可以把话题引导到单分区方案和多分区方案上。

基本思路

我在前置知识里面已经说了 Kafka 分区是如何存储数据的。所以你很容易就猜到第一个解决方案，就是每一个 topic 只使用一个分区，也就是所谓的单分区解决方案。



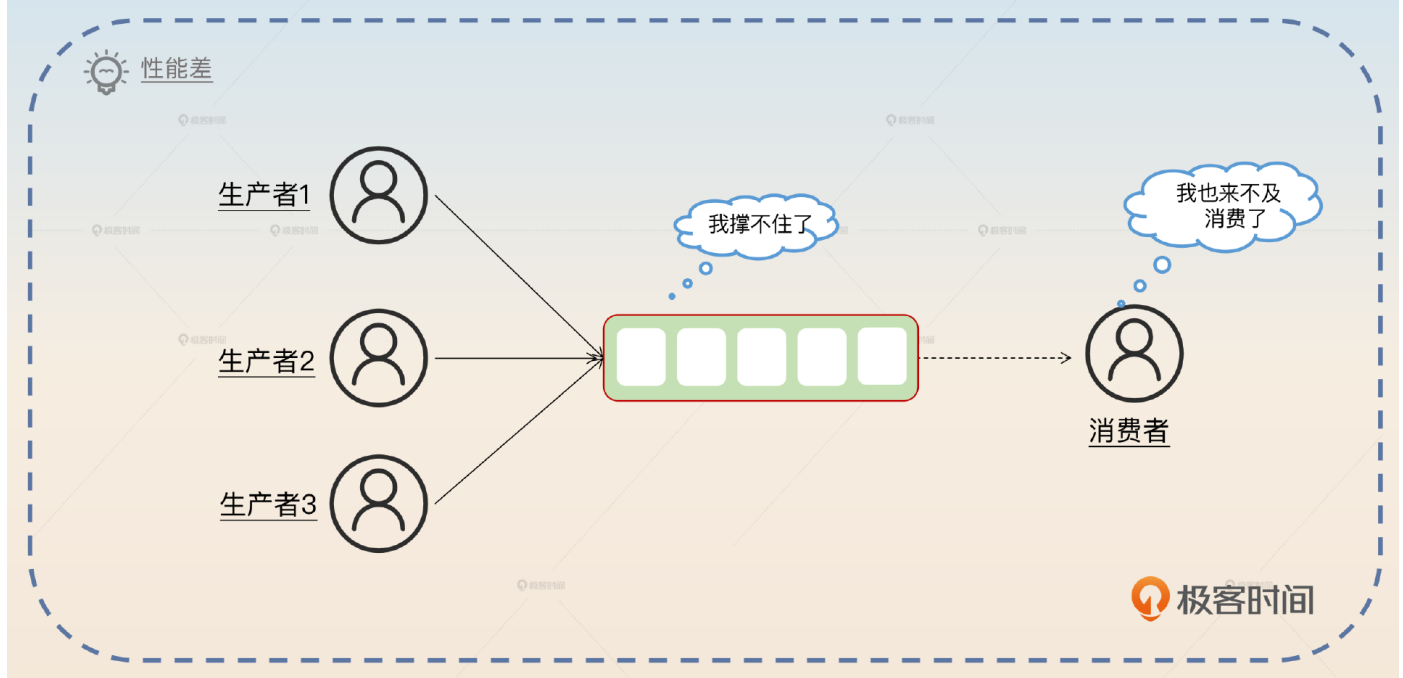
所以你可以简单介绍这个方案。

要保证消息有序，最简单的做法就是让特定的 topic 只有一个分区。这样所有的消息都发到同一个分区上，那么自然就是有序的。

这个方案的优点是简单，并且是全局有序。但是这个方案有一个很严重的问题：性能太差。因为你一个 topic 只有一个分区，它就没办法支撑高并发。

所以你可以在回答中补充这一点，关键词是**性能差**。

这种只有一个分区的方案性能差，没办法支撑高并发。对于生产端来说，所有的消息都在一个分区上，也同时意味着所有的消息都发送到了同一个broker上，这个服务器很可能撑不住压力；对于消费端来说，只有一个分区，那么就只能有一个消费者消费，很容易出现消息积压的问题。



既然你说了这个点，自然就是为了引出改进方案。不过改进方案都有一个基本的前提，消息需要保持有序，但不是全局有序，而是同一个业务内有序。

如果业务要求的是全局有序，那就没什么优化手段，只能是换用更加强大的机器。但是你设想一下真实的业务场景，大部分业务强调的有序是全局有序吗？并不是，而是**业务内有序**。比如说在下单的场景下，会产生创建订单消息和完成支付消息。业务上只会要求同一个订单的创建订单消息应该优先于完成支付消息，但是不会要求订单 A 的创建消息需要先于订单 B 的支付消息。

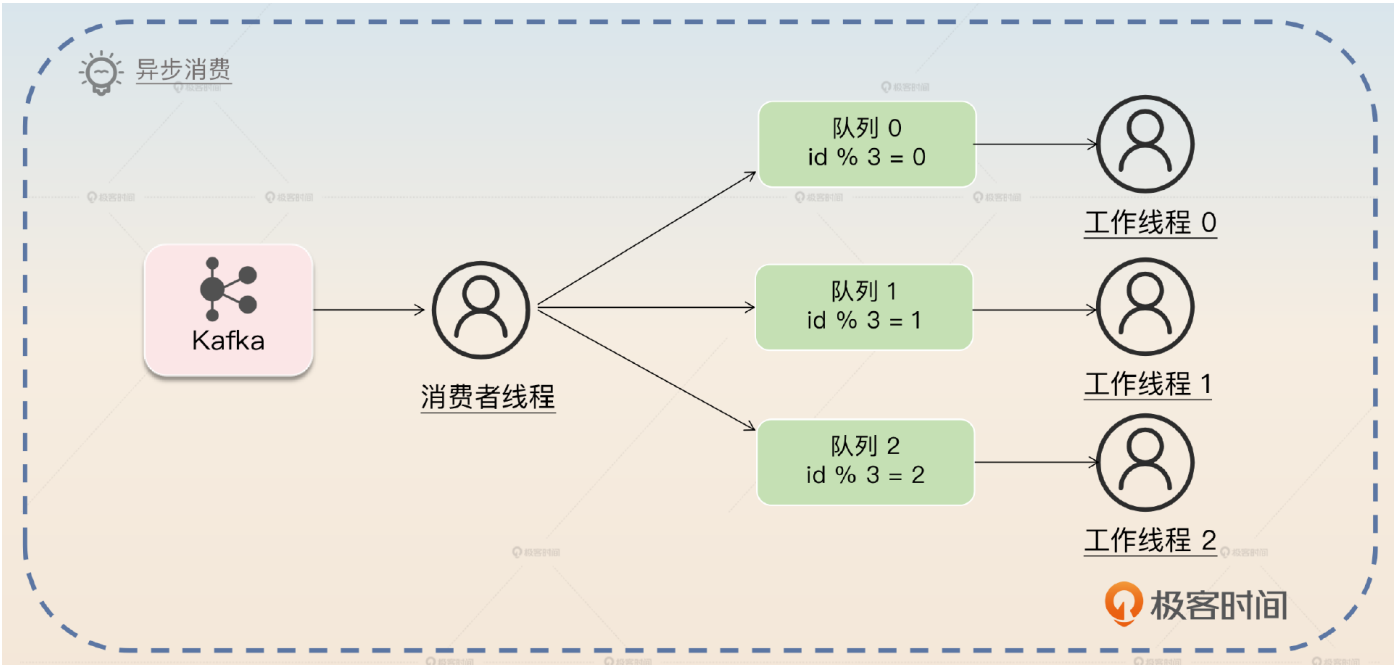
所以你可以先暂时澄清一下同一个业务内有序的概念，然后引出优化方案。

如果我们要求的是全局有序，那除了换更加强大的机器就没别的办法了。而事实上，大部分的业务场景要求的都不是全局有序，而是业务内有序。例如要求同一个订单创建订单的消息应该先于完成支付消息，但是不会要求不同订单之间的消息是有序的。在这种场景下，还有两个改进方案可以考虑。一个是异步消费，还有一个是多分区方案。

异步消费

在单分区方案里，最容易遇到的问题就是消息积压，因为你只有一个消费者。在遇到消息积压的情况下，你可以考虑异步消费。但是这里的异步消费和上节课讲的比起来，有点不一样，这里你要关注一个问题：如何维持顺序？

这里说的顺序，就是同一个业务内的顺序。所以我们可以考虑在异步消费时，当消费者从队列取出来消息之后，把同一个业务的消息丢到同一个队列里。你可以看一下整体的架构图。



图中有一个消费者负责从 Kafka 里拉取消息，然后发送到基于内存的队列里面。在发送的时候，它会根据 id 做一个简单的取余操作来筛选队列。然后工作线程从队列里面拿取消息，真正执行业务逻辑。

这个例子用的是 id，但是你可以根据自己的业务来选择其他字段，你选择的这些字段就被叫做业务特征。

你抓住关键字**异步消费**介绍这个方案。

第一个方案是异步消费，这个方案和解决消息积压的异步消费方案差不多，但是要做一点改进。消费者线程从 Kafka 里获取消息，然后转发到内存队列里面。在转发的时候，要把同一个业务的消息转发到同一个队列里面。一般来说可以根据业务特征字段计算一个哈希值，比如说直接使用业务 id 作为哈希值。利用这个哈希值除以工作线程数量，然后取余数，得到对应的内存队列。

这种做法的缺陷就是存在消息未消费的问题。也就是消费线程取出来了，转发到队列之后，工作线程还没来得及处理，消费者整体就宕机了，那么这些消息就存在丢失的可能。

最后提到了一个消息未处理的问题，我们上节课已经聊过了，那么你就可以灵活地把话题引导到消息积压的问题上去。

如果你仔细观察，你就会发现，既然我可以用内存队列，我干嘛不直接使用多个分区呢？这也就是我们的第二个改进方案：多分区。

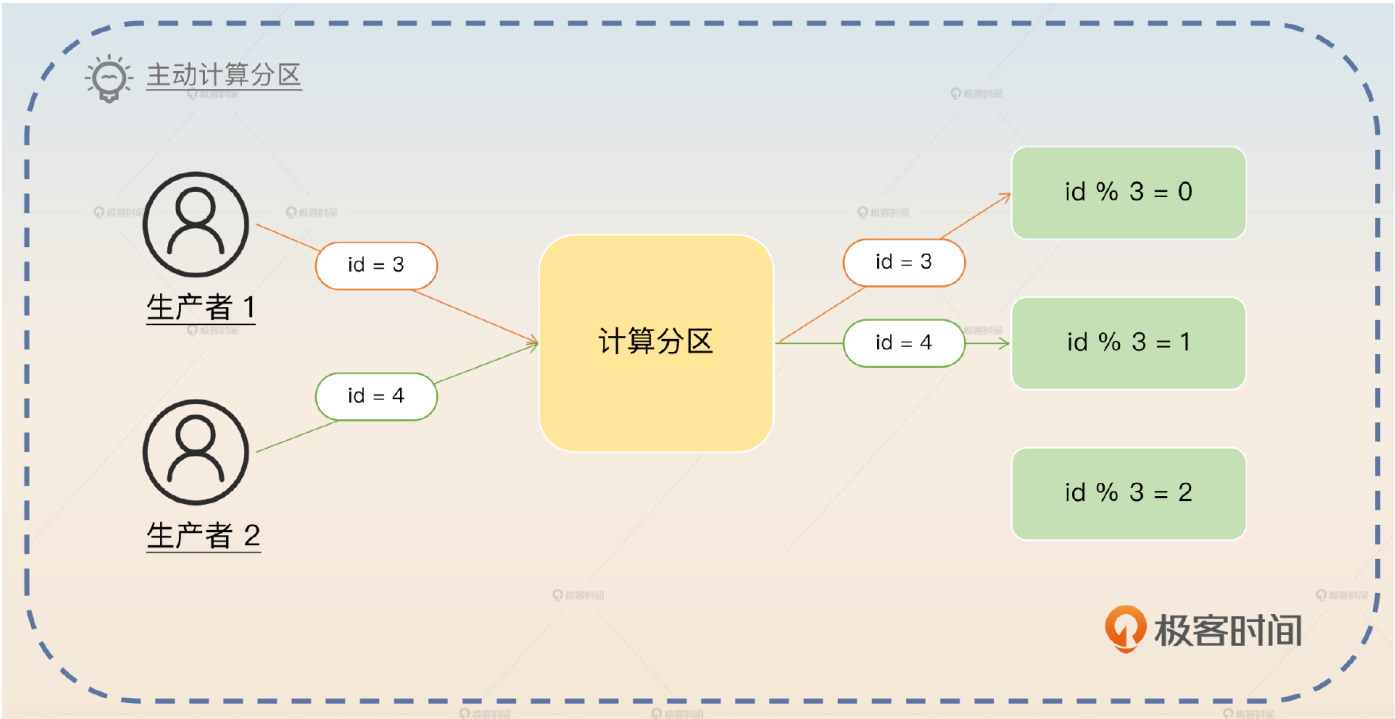
亮点方案：多分区

这个方案在面试的时候也有一些人能回答出来，但是很少有人能够深入讨论这个方案的潜在问题和对应的解决方案，所以这也就是你拉开差距的地方。

第二个方案和第一个方案差不多，就是把内存队列换成了多个分区。原本同一个业务的消息发送到同一个队列，这里是同一个业务的消息要发送到同一个分区。

第二个方案就是直接扩展为使用多个分区，只需要确保同一个业务的消息发送到同一个分区就可以保证同一个业务的消息是有序的。

面试官自然就会追问，怎么保证同一个业务的消息必然发送到同一个分区呢？做法也很简单，只需要生产者在发送消息的时候，根据业务特征，比如说业务 ID 计算出目标分区，在发送的时候显式地指定分区就可以了。



这里你抓住关键词**计算分区**回答。

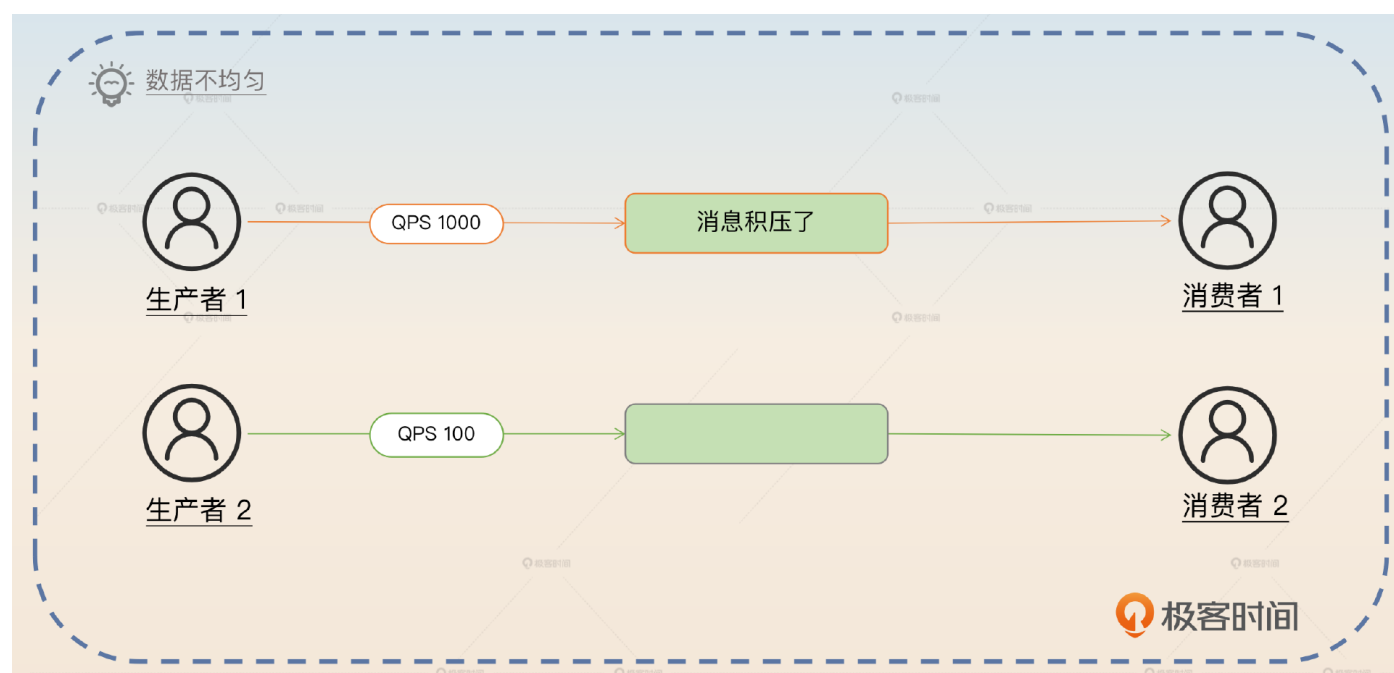
要想确保同一个业务的消息都发送到同一个分区，那么只需要发送者自己根据业务特征，直接计算出来一个目标分区。比如说最简单的策略就是根据业务 ID 对分区数量取余，余数就是目标分区。

就像我之前所说的，所有的方案都有优缺点，这个方案也不例外。它的优点是足够简单，业务方需要做的改动很小。但是缺点有两个，一个是数据不均匀，另一个是增加分区可能导致消息失序。

我们先来看第一个缺点。

数据不均匀

这个缺点很容易理解，因为发送方要按照业务特征来选择分区，自然就容易导致一些分区有很多数据，而另外一些分区数据很少。而如果某个分区有很多数据的话，消费者来不及消费也是正常的事情。



数据不均匀一般是业务造成的。在我们的方案里面，分区是根据业务特征来选择的，那么自然有一些分区有很多数据，有一些分区数据很少。比如说万一我们不小心把热点用户的消息都发到了同一个分区里面，那么这个分区的 QPS 就会很高，消费者也不一定来得及消费，就可能引起消息积压。

怎么解决这个问题呢？

要想解决这个问题，可以通过改进计算目标分区的方式来解决，比如说采用类似于 Redis 中槽和槽分配的机制，又或者说一致性哈希算法，基本上就能解决这个问题了。

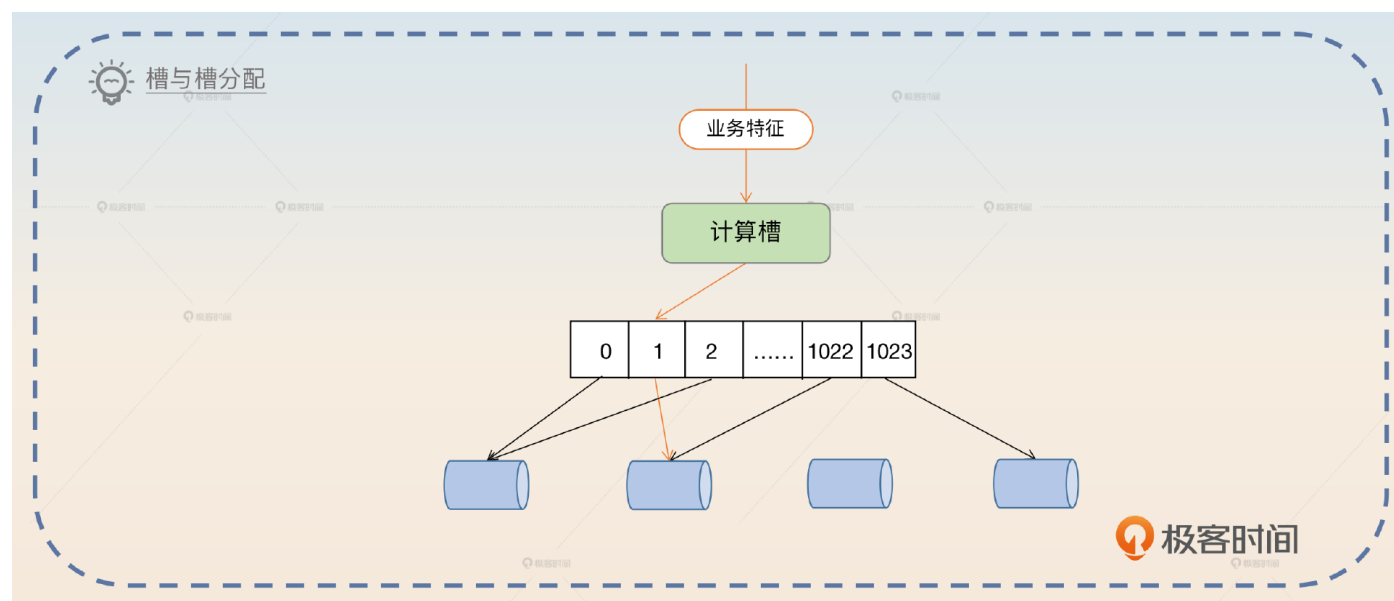
这两个解决思路，我建议你不要等面试官追问，而是自己直接接着说，因为到这里你讨论的深度大概率已经超出了一般面试官的经验范畴了，所以他们可能想不到该怎么问你。

槽与槽分配

这种解决方案的基本思路就是借鉴 Redis 的数据分布方案。你可以根据业务特征计算一个哈希值，然后映射到槽上。这里，你可以参考 Redis，使用 16384 个槽，不过如果业务体量不是那种几百万 QPS 的，你用 1024 个槽就可以。

你再通过指定不同的槽把数据分配到不同的分区上，这个时候你可以根据你的业务特征合理分配槽，从而保证分区之间数据分布是均匀的。

整个架构如图：



你抓住关键词**槽**回答。

第一种思路是借鉴 Redis 的槽与槽分配方案。不过 Redis 使用了 16384 个槽，一般的业务用不上那么多槽，所以可以考虑用 1024 个槽。根据业务的特征来计算一个哈希值，再除以 1024 取余就可以得到所在的槽。再根据不同槽的数据多少，合理地把槽分配到不同的分区。最好把槽和分区的绑定关系做成

动态的，也就是说我可以随时调整槽到分区的映射关系，保证所有的分区负载都是均匀的。

最后我们提到的动态调整槽与分区的绑定关系，可以借助于配置中心来完成。比如说最开始你把槽 1 绑定到分区 2 上，后面在运行的时候你发现分区 2 数据太多，就把槽 1 重新绑定到了分区 3 上。

当然，因为这个回答的核心是借鉴了 Redis，那么就有可能把话题引导到 Redis 那里。不过你也不用担心，课程后面你会了解到和 Redis 有关的内容。

但是这个过程还是有问题的，可能会引起消息失序，和增加新的分区指向了同一个问题。所以你可以参考增加分区引起消息失序部分来回答。接下来，我们再来看看一致性哈希是怎么解决问题的。

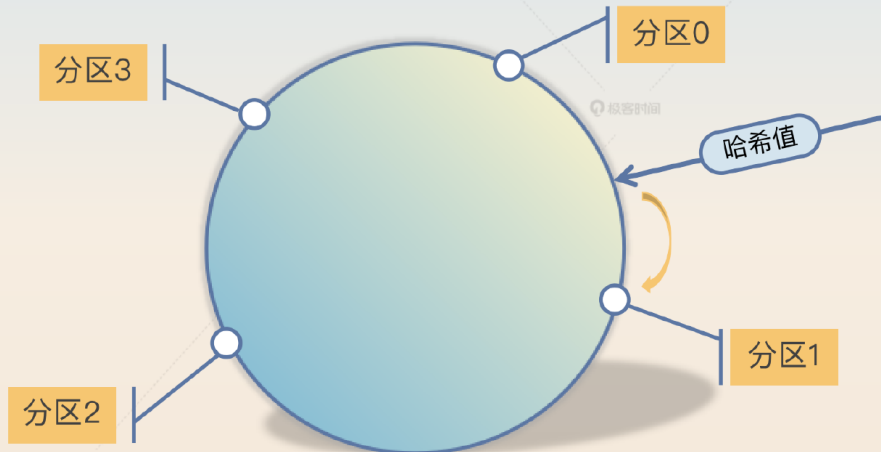
一致性哈希

你已经接触过一致性哈希这个算法很多次了。那么在这里使用哈希一致性算法，就是把分区分散在哈希环上。你根据业务特征计算出一个哈希值之后，根据哈希值在这个环上找到合适的分区，然后把消息发送过去。

你只要通过调整不同分区之间的间隔，就能控制分区上的数据分布。



调整不同分区之间的间隔



极客时间

另外一种思路是使用一致性哈希算法来筛选分区。首先要根据数据分布的整体情况，把分区分布在哈希环上，确保每一个分区上的数据分布大体上是均匀的。如果一部分哈希值上数据较多，就多插入几个分区节点。然后根据业务特征计算一个哈希值，从哈希环上找到对应的分区。

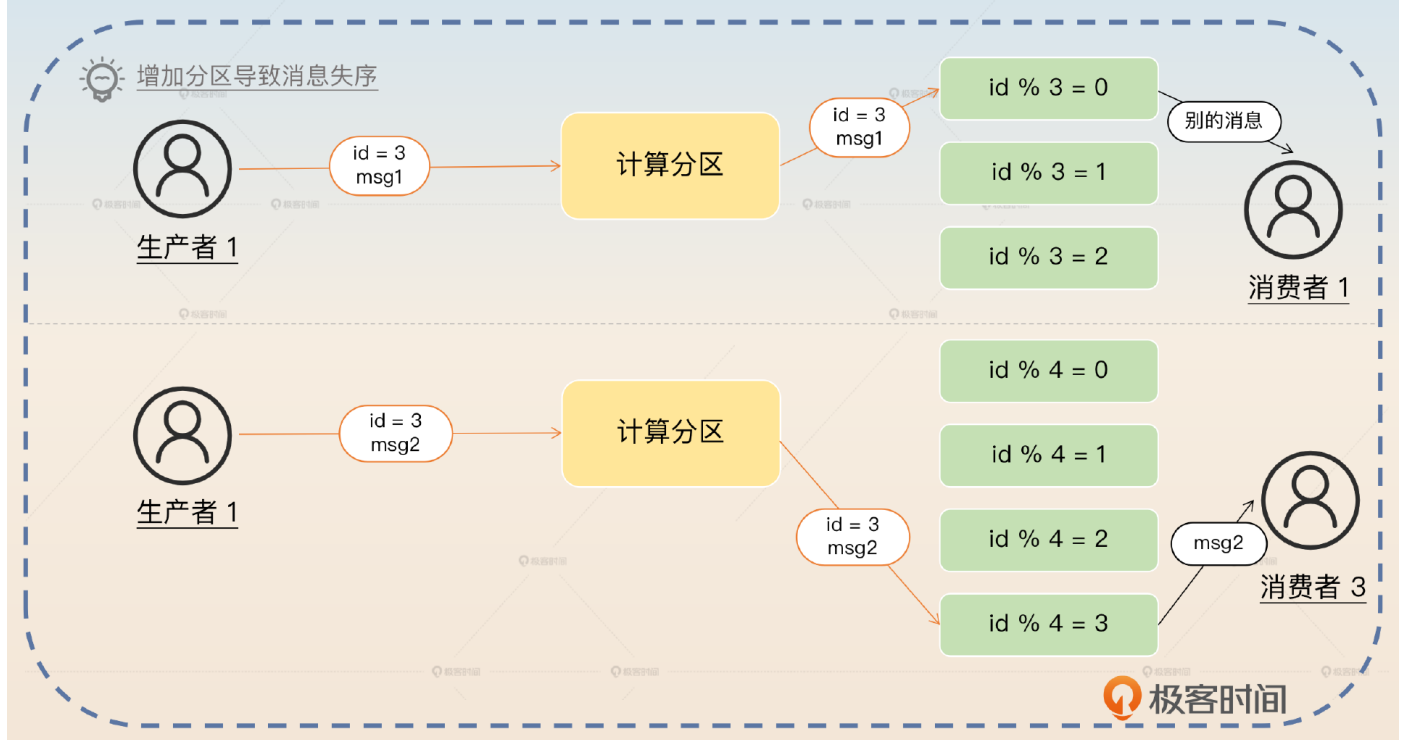
最后你可以进一步总结，升华一下。

这种槽分配和一致性哈希算法非常适合解决数据或者流量分布不均匀的问题，因为我们总是可以通过手工调整槽的映射关系或者哈希环上节点的分布来保证数据或者流量在每一个节点上的分布大体是均匀的。

到这里，我们已经彻底解决了数据不均匀的问题，接下来就要讨论增加分区引起消息失序的问题了。

增加分区引起消息失序

你可以结合下面这张图来理解。



在面试的时候，你可以先介绍这个缺点，关键词是**增加分区会引起消息失序**，同时补充一个例子。

它还有另外一个缺点，就是如果中间有增加新的分区，那么就可能引起消息失序。比如说最开始 id 为 3 的订单消息 msg1 发到分区 0 上，但是这时候很不幸分区 0 上积攒了很多消息，所以 msg1 迟迟得不到消费。

紧接着我们扩容，增加了一个新的分区。如果这时候来了一个消息 msg2，那么它会被转发到分区 3 上。分区 3 上面没有积攒什么数据，所以消费者 3 直接就消费了这个消息。

这时候我们发现，本来 msg1 应该先于 msg2 被消费。而增加分区之后 msg2 反而被先消费了。这就是一个典型的消息失序场景。

那么针对这个缺点我们也可以进一步提出解决方案。这个消息失序的场景解决起来倒也很简单，就是新增加了分区之后，这些新分区的消费者先等一段时间，比如说三分钟，确保同一个业务在其他分区上的消息已经被消费了。

要解决这个问题也很容易。对于新加入的分区，可以暂停消费一段时间。比如说在前面的例子中，如果我们估算 msg1 会在一分钟之内被消费，那么新加入的分区的消费者可以在三分钟后才开始消费。那么大概率 msg1 就会先于 msg2

消费。不过这种等待的解决方式并不能解决根本问题，只能说是很大程度上缓解了问题。但是本身增加分区也是一个很不常见的操作，再叠加消息失序的概率也很低，所以我們也可以通过监控发现这种失序场景，然后再手工修复一下就可以了。

到了这一步，常规的方案和升级的多分区版本已经聊得差不多了。最后我再来教你一个面试的小技巧。

基于优化的面试思路

刚刚我们从单分区讨论到多分区，就是一个关于性能优化的非常好的案例，我们如何通过这个案例来让我们的能力更加突出呢？这里可以利用我们之前学过的一个面试小技巧，关键词就是**优化 + 前后对比**。

最开始我进公司的时候就遇到了一个 Kafka 的线上故障。我司有一个业务需要用到有序消息，所以最开始的设计就是对应的 topic 只有一个分区，从而保证了消息有序。

可是随着业务增长，一个分区很快就遇到了性能瓶颈。只有一个分区，也就意味着只有一个消费者，所以在业务增长之后，就开始出现了消息积压。另外一方面，这个分区所在的broker的负载也明显比其他服务器要大，偶尔也会有一些性能抖动的问题。

后来我仔细看了我们的业务，实际上，我们的业务要求的不是全局有序，而是业务内有序。

换句话说来说，不一定非得用一个分区，而是可以考虑使用多个分区。所以我就给这个 topic 增加了几个分区，同时也增加了消费者。优化完之后，到目前为止，还没有出现过消息积压的问题。

这里你应该注意到，从单个分区增加到多个分区，还是会出现前面说的消息失序的问题。我建议你主动提起这个问题，展开聊一聊你的应对方案。

当然，为了避免在单分区增加到多分区的时候，出现消息失序的问题，我用了一个很简单的方案，就是对应的消费者在启动之后，并没有立刻消费，而是停

顿了三分钟，从而避免了潜在的消息失序问题。

注意，这里的停顿三分钟的前提是你要先把积压的消息消费掉。如果积压的消息还需要三十分钟，那你这里就至少要停顿三十分钟。

面试思路总结

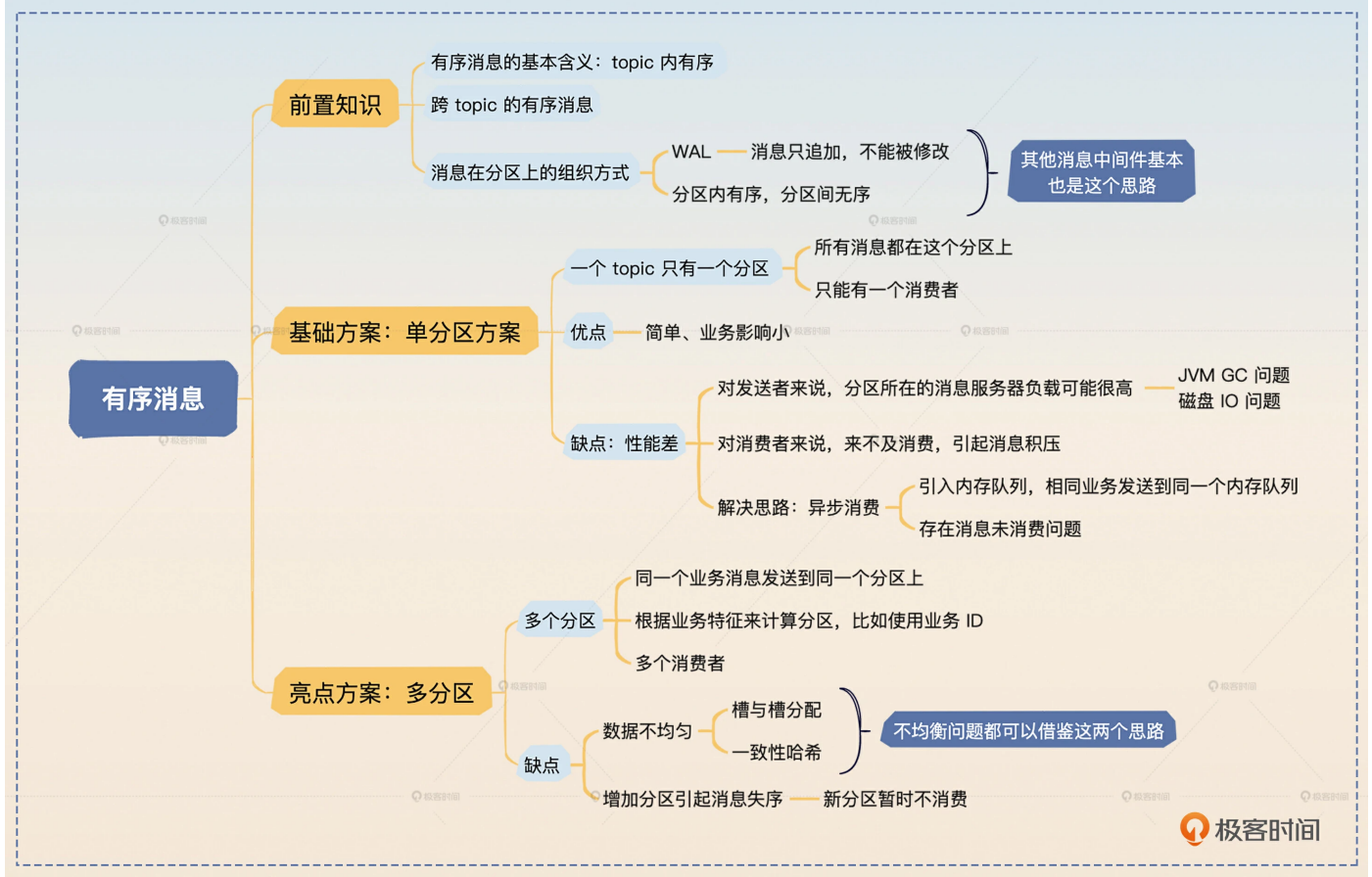
这节课我们了解了有序消息的基本概念，还有一个看起来有点接近但是完全不同的话题：跨 topic 的有序消息。其中有一个关键问题，就是消息在分区上是如何组织的，这里你记住 **WAL 和分区内有序**这两个关键词就可以了。

在这个基础上，我们讨论了最常规的方案，也就是保证**全局有序的单分区方案**，并且进一步解释了如果在单分区上消息积压了，可以通过**异步消费**来解决问题。

然后我们重点讨论了**多分区方案**，这里你要记住多分区方案的两个缺陷，还有应对策略。

1. 数据不均匀。对应的解决方案是借鉴 Redis 的槽与槽分配的方案和一致性哈希方案。
2. 增加分区引起的消息失序。要解决这个问题也很简单，就是新增加的分区暂时不要消费，确保在别的分区上的消息已经被消费完了再消费。

RabbitMQ 和 RocketMQ 里面消息有序性的解法也是差不多的。不同的是，在 RabbitMQ 里面使用的是 queue，因为 RabbitMQ 没有分区概念。而 RocketMQ 里面内置了有序消息的功能，底层原理也基本相似。



思考题

最后请你来思考两个问题。

你觉得在多分区方案里面，如果某个分区消息积压了就启用异步消费，这种解决思路你觉得怎么样？

我在最后总结了一个槽分配和一致性哈希非常适合用于解决数据或者流量分布不均匀问题的观点，你还在什么地方见过类似的解决思路？

欢迎你把你的答案分享在评论区，也欢迎你把这节课的内容分享给需要的朋友，我们下节课再见！

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 23 | 延迟消息：怎么在 Kafka 上支持延迟消息？

下一篇 25 | 消息积压：业务突然增长，导致消息消费不过来怎么办？

