08 | 答疑解惑（一）：网关如何接收服务端的秒杀结果？

2019-08-08 李玥

你好，我是李玥。

我们的“消息队列高手课”专栏自从上线到现在，同学们的学习热情和参与度都非常高。每一节课都有很多同学留言评论，这些留言里有总结知识分享收获的，有提出精彩问题的，还有给自己加油打气立Flag的，竟然还有说老师长得像黄渤的。我又仔细去看了一下配图，还是真挺像的。下次老师和极客时间的设计师小姐姐说一样，让她们照着吴彦祖来P图。

同学们每一条的留言我都认真看过，大部分留言我都给出了回复。在基础篇的最后一节课，我来统一解答一下大家都比较关注的一些问题。

1. 网关如何接收服务端的秒杀结果？

在《01 | 为什么需要消息队列？》这节课里面，我们举了一个秒杀的例子，这个例子是用来说明消息队列是如何来实现异步处理的。课后很多同学留言提问，网关在发送消息之后，是如何来接收后端服务的秒杀结果，又如何来给APP返回响应的呢？

在解答这个问题之前，我需要先说一下，实际生产环境中的秒杀系统，远比我们举的这个例子复杂得多，实现方案也是多种多样的，不是说一定要按照我们这个例子的方式来实现。

在这个例子中，网关接收后端服务秒杀结果，实现的方式也不只一种，这里我给大家提供一个比较简单的方案。

比如说，用Java语言来举例子：

```
public class RequestHandler {

  // ID生成器
  @Inject
  private IdGenerator idGenerator;
  // 消息队列生产者
  @Inject
  private Producer producer;
  // 保存秒杀结果的Map
  @Inject
  private Map<Long, Result> results;

  // 保存mutex的Map
  private Map<Long, Object> mutexes = new ConcurrentHashMap<>();
  // 这个网关实例的ID
  @Inject
  private long myId;

  @Inject
  private long timeout;

  // 在这里处理APP的秒杀请求
  public Response onRequest(Request request) {
    // 获取一个进程内唯一的UUID作为请求id
    Long uuid = idGenerator.next();
    try {

      Message msg = composeMsg(request, uuid, myId);

      // 生成一个mutex，用于等待和通知
      Object mutex = new Object();
      mutexes.put(uuid, mutex)

      // 发消息
      producer.send(msg);
```

```
      // 等待后端处理
      synchronized(mutex) {
        mutex.wait(timeout);
      }

      // 查询秒杀结果
      Result result = results.remove(uuid);

      // 检查秒杀结果并返回响应
      if(null != result && result.success()){
        return Response.success();
      }

    } catch (Throwable ignored) {}
    finally {
      mutexes.remove(uuid);
    }
    // 返回秒杀失败
    return Response.fail();
  }

  // 在这里处理后端服务返回的秒杀结果
  public void onResult(Result result) {

    Object mutex = mutexes.get(result.uuid());
    if(null != mutex) { // 如果查询不到，说明已经超时了，丢弃result即可。
      // 登记秒杀结果
      results.put(result.uuid(), result);
      // 唤醒处理APP请求的线程
      synchronized(mutex) {
        mutex.notify();
      }
    }
  }
}
```

在这个方案中，网关在收到APP的秒杀请求后，直接给消息队列发消息。至于消息的内容，并不一定是APP请求的Request，只要包含足够的字段就行了，比如用户ID、设备ID、请求时间等等。另外，还需要包含这个请求的ID和网关的ID，这些后面我们会用到。

如果发送消息失败，可以直接给APP返回秒杀失败结果，成功发送消息之后，线程就阻塞等待秒杀结果。这里面不可能无限等待下去，需要设定一个等待的超时时间。

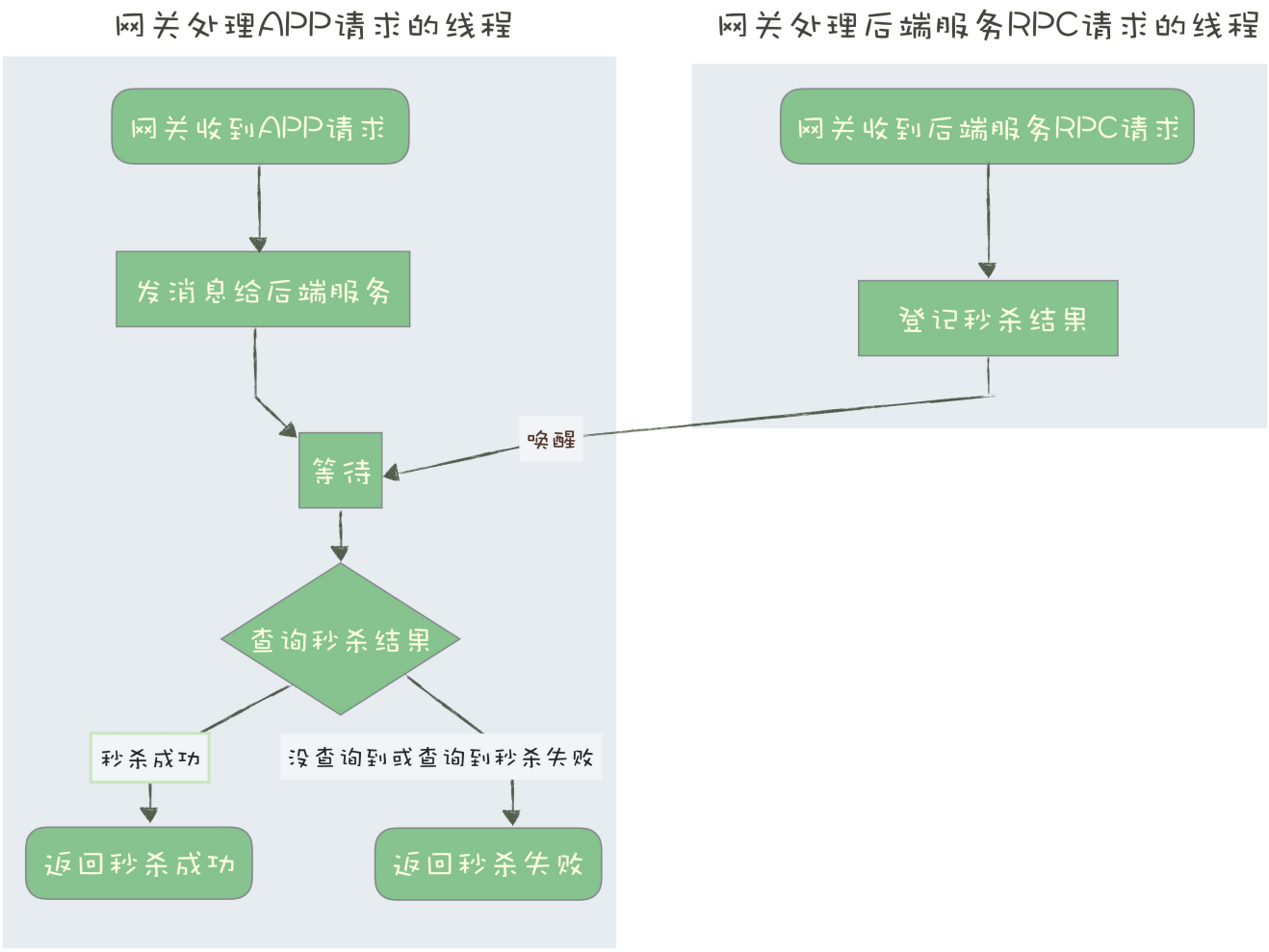
等待结束之后，去存放秒杀结果的Map中查询是否有返回的秒杀结果，如果有就构建Response，给APP返回秒杀结果，如果没有，按秒杀失败处理。

这是处理APP请求的线程，接下来我们来看一下，网关如何来接收从后端秒杀服务返回的秒杀结果。

我们可以选择用RPC的方式来返回秒杀结果，这里网关节点是RPC服务端，后端服务为客户端。之前网关发出去的消息中包含了网关的ID，后端服务可以通过这个网关ID来找到对应的网关实例，秒杀结果中需要包含请求ID，这个请求ID也是从消息中获取的。

网关收到后端服务的秒杀结果之后，用请求ID为Key，把这个结果保存到秒杀结果的Map中，然后通知对应的处理APP请求的线程，结束等待。我刚刚说过，处理APP请求的线程，在结束等待之后，会去秒杀的结果Map中查询这个结果，然后再给APP返回响应。

我把这个处理过程的流程图放在这里，便于你理解：



这个解决方案还不是一个性能最优的方案，处理APP请求的线程需要同步等待秒杀结果。后面课程中我们会专门来讲，如何使用异步方式来提升程序的性能。

2. 详解RocketMQ和Kafka的消息模型

我在看《03 | 消息模型：主题和队列有什么区别？》这节课的留言时发现，不少同学对RocketMQ和kafka的消息模型理解的还不是很透彻，这两个消息队列产品的消息模型是一样的，我在这里，再把这个模型相关的概念，通过一个例子详细地说一说。

假设有一个主题MyTopic，我们为主题创建5个队列，分布到2个Broker中。

Broker	主题	队列
Broker0	MyTopic	Q0，Q1
Broker1	MyTopic	Q2，Q3，Q4

先说消息生产这一端，假设我们有3个生产者实例：Produer0，Produer1和Producer2。

这3个生产者是如何对应到2个Broker的，又是如何对应到5个队列的呢？这个很简单，**不用对应，随便发**。每个生产者可以在5个队列中轮询发送，也可以随机选一个队列发送，或者只往某

个队列发送，这些都可以。比如Producer0要发5条消息，可以都发到队列Q0里面，也可以5个队列每个队列发一条。

然后说消费端，很多同学没有搞清楚消费组、消费者和队列这几个概念的对应关系。

每个消费组就是一份订阅，它要消费主题MyTopic下，所有队列的全部消息。注意，队列里的消息并不是消费掉就没有了，这里的“消费”，只是去队列里面读了消息，并没有删除，消费完这条消息还是在队列里面。

多个消费组在消费同一个主题时，消费组之间是互不影响的。比如我们有2个消费组：G0和G1。G0消费了哪些消息，G1是不知道的，也不用知道。G0消费过的消息，G1还可以消费。即使G0积压了很多消息，对G1来说也没有任何影响。

然后我们再说消费组的内部，一个消费组中可以包含多个消费者的实例。比如说消费组G1，包含了2个消费者C0和C1，那这2个消费者又是怎么和主题MyTopic的5个队列对应的呢？

由于消费确认机制的限制，这里面有一个原则是，在同一个消费组里面，每个队列只能被一个消费者实例占用。至于如何分配，这里面有很多策略，我就不展开说了。总之保证每个队列分配一个消费者就行了。比如，我们可以让消费者C0消费Q0，Q1和Q2，C1消费Q3和Q4，如果C0宕机了，会触发重新分配，这时候C1同时消费全部5个队列。

再强调一下，队列占用只是针对消费组内部来说的，对于其他的消费组来说是没有影响的。比如队列Q2被消费组G1的消费者C1占用了，对于消费组G2来说，是完全没有影响的，G2也可以分配它的消费者来占用和消费队列Q2。

最后说一下消费位置，每个消费组内部维护自己的一组消费位置，每个队列对应一个消费位置。消费位置在服务端保存，并且，**消费位置和消费者是没有关系的**。每个消费位置一般就是一个整数，记录这个消费组中，这个队列消费到哪个位置了，这个位置之前的消息都成功消费了，之后的消息都没有消费或者正在消费。

我把咱们这个例子的消费位置整理成下面的表格，便于你理解。

主题	消费组	Broker	队列	消费位置
MyTopic	G0	Broker0	Q0	234
MyTopic	G0	Broker0	Q1	23
MyTopic	G0	Broker1	Q2	87
MyTopic	G0	Broker1	Q3	0
MyTopic	G0	Broker1	Q4	888
MyTopic	G1	Broker0	Q0	4478
MyTopic	G1	Broker0	Q1	9832
MyTopic	G1	Broker1	Q2	7650
MyTopic	G1	Broker1	Q3	2346
MyTopic	G1	Broker1	Q4	3398

你可以看到，这个表格中并没有消费者这一列，也就是说消费者和消费位置是没有关系的。

3. 如何实现单个队列的并行消费？

下面说一下《03 | 消息模型：主题和队列有什么区别？》这节课的思考题：如果不要求严格顺序，如何实现单个队列的并行消费？关于这个问题，有很多的实现方式，在JMQ（京东自研的消息队列产品）中，它实现的思路是这样的。

比如说，队列中当前有10条消息，对应的编号是0-9，当前的消费位置是5。同时来了三个消费者来拉消息，把编号为5、6、7的消息分别给三个消费者，每人一条。过了一段时间，三个消费成功的响应都回来了，这时候就可以把消费位置更新为8了，这样就实现并行消费。

这是理想的情况。还有可能编号为6、7的消息响应回来了，编号5的消息响应一直回不来，怎么办？这个位置5就是一个消息空洞。为了避免位置5把这个队列卡住，可以先把消费位置5这条消息，复制到一个特殊重试队列中，然后依然把消费位置更新为8，继续消费。再有消费者来拉消息的时候，优先把重试队列中的那条消息给消费者就可以了。

这是并行消费的一种实现方式。需要注意的是，并行消费开销还是很大的，不应该作为一个常规的，提升消费并发的手段，如果消费慢需要增加消费者的并发数，还是需要扩容队列数。

4. 如何保证消息的严格顺序？

很多同学在留言中问，怎么来保证消息的严格顺序？我们多次提到过，主题层面是无法保证严格

顺序的，只有在队列上才能保证消息的严格顺序。

如果说，你的业务必须要求全局严格顺序，就只能把消息队列数配置成1，生产者和消费者也只能是一个实例，这样才能保证全局严格顺序。

大部分情况下，我们并不需要全局严格顺序，只要保证局部有序就可以满足要求了。比如，在传递账户流水记录的时候，只要保证每个账户的流水有序就可以了，不同账户之间的流水记录是不需要保证顺序的。

如果需要保证局部严格顺序，可以这样来实现。在发送端，我们使用账户ID作为Key，采用一致性哈希算法计算出队列编号，指定队列来发送消息。一致性哈希算法可以保证，相同Key的消息总是发送到同一个队列上，这样可以保证相同Key的消息是严格有序的。如果不考虑队列扩容，也可以用队列数量取模的简单方法来计算队列编号。

写在最后

在留言中，很多同学留言提出来，能不能讲一讲某个消息队列的某个功能具体如何配置。我的建议是，你先不要太关注功能、API和配置这些细节，在学习如何使用消息队列的过程中，要保持一定的高度来学习。

因为使用消息队列，大部分的难点在宏观架构层面，要解决这些难点，你需要掌握消息队列宏观层面上的实现原理和最佳实践，这样，无论你使用什么消息队列，都可以做到游刃有余。在选定了合适的消息队列产品，准备写代码之前，再去文档中查看这些细节都来得及。

所以，我们专栏的“基础篇”讲消息队列的使用，更多讲的是一些通用的原理。这节课是我们消息队列高手课“基础篇”的最后一节课，完整基础篇的学习后，意味着你已经是一个使用消息队列的小达人了。

在“进阶篇”中，我们将把学习重点从“如何使用”转为“如何实现”，在学习消息队列的实现技术时，你反而要专注到每一个技术点上，深入下去，把每个细节都要搞清楚、学透。课程的深度、难度也会逐步加强，当然你获得的经验值也会更多。

希望大家一如既往坚持学习，多思考，多练习，跟老师一起打怪升级，成为真正的高手。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言

大龄小学生

8

老师，一图胜千言，来点图吧。

2019-08-08

作者回复

你要的图来了。我在文中补充了一个流程图，便于同学们理解。

2019-08-09

linqw

8

学习完网关答疑篇，写下自己的理解和疑惑，老师有空帮忙看下哦

1、秒杀的理解:

APP--发送秒杀请求--》网关（也是RPC服务端，和配置中心保持长连接，比如nacos，将其路由和配置信息定时的发送给配置中心，配置中心对其进行管理，定时的清除宕机的网关路由信息，如超过一定时间没有接收到网关的心跳包）--》将其APP请求做一定的封装，增加网关id和网关实例中唯一的请求id发送给消息队列，为了保证消息不丢失，网关对其发送消息出现的异常进行处理，如超时异常，直接返回秒杀失败，网关发送消息的这个过程中可能涉及到分布式事务，使用消息队列的分布式事务进行处理，然后网关需要等待一段时间，等待秒杀服务端使用RPC调用网关实例的接收秒杀结果，为此创建一个新对象，将其请求id做为key，新对象做为value放入CurrentMap中，调用新对象的超时wait方法进行等待秒杀结果--发送封装的APP请求，包含网关id和请求id--》消息队列接收APP请求消息，为了保证消息不丢失，开启Sync_Flush参数将消息保存到磁盘，并且为了防止一台机器磁盘出问题，集群需要2台机器都有消息才确认请求--从消息队列中拉取消息--》秒杀服务端，为了低延迟执行风控、预占库存，拿到消息中网

关id，从本地路由中查询网关id的实例信息，如果获取不到调用网关实例时，需先从配置中心获取到网关的路由信息，秒杀服务端也需和配置中心保持长连接，定时的从配置中心拉取网关的路由信息，保存到本地，使用RPC调用网关实例的接收秒杀结果的方法，为了保证消息不丢失，先执行消费逻辑，再响应消息队列，如果根据网关id获取不到网关实例，或者确认消息队列超时或出现异常，秒杀服务端回滚事务，此过程也涉及到分布式事务，为了防止消费重复消息，接口的幂等性，将请求id和网关id做为唯一键。也为了防止消息积压，消息队列中的主题队列和消费组中的消费者一一对应，保证消息被快速消费。
2、秒杀异步，APP发送请求给网关，网关接收请求后将请求做一定的封装（包括请求id，网关id，账户id），然后发送到消息队列中，响应APP请求，无需等待后需的流程，然后秒杀成功以否直接返回，后续流程处理完使用短信的形式告知用户是否秒杀成功，不知道这样做法是否可行。
3、最近在撸rocketmq的源码，搞了namesrv、logging、logappend模块，想成为commiter，立个flag，等后续JMQ出来，撸其源码，也想成为commiter，道阻且长，持续进化。

2019-08-08

作者回复

我认真的看了同学的对于秒杀的理解，技术上都没什么问题。

从业务角度，老师有一些不同的看法。

对于秒杀这种场景，宏观上的设计应该是倾向于利用有限的资源处理短时间内海量的请求，保证服务不宕机。有少量请求处理出错（注意是后端错误，用户不可见）或消息丢失，是可以接受的。

毕竟秒杀拼的就是运气，某个用户秒杀请求在处理的时候丢失，和处理成功但没秒到，对于用户来说都是运气不好而已。

基于这样的设计理念，很多保证数据可靠性的做法都可以牺牲掉，用于换取系统更大的吞吐量比较划算。

2019-08-08

Abyssal

7

七夕不过节，继续学习 — 谁让我是单身狗呢

2019-08-08

滴流乱转小胖子

4

mq界，吴彦祖老师你好，感谢分享

2019-08-08

作者回复

谢谢你，蔡徐坤同学。

2019-08-08

A:春哥大魔王

2

看了几期，感觉分布式消息队列的设计方案和分布式存储系统的设计方案很类似，如果再加上事务处理，存储细节方案应该更像了

2019-08-08

作者回复

同学那不是像，消息队列就是个分布式存储系统。

2019-08-08

猿人谷 2

这篇答疑解惑，虽然简短，但绝对的诚意十足。希望多出这种答疑解惑的章节，毕竟评论区里很多留言的问题非常有代表性，对代表性的问题出这种答疑解惑的章节，学到的更多，也更能体现大家的参与度。

2019-08-08

业余草 2

相对来说，哪一个消息队列的学习成本较高？

2019-08-08

作者回复

我觉得是Kafka吧，功能足够复杂，而且老外写代码的脑回路和我们不大一样。

2019-08-08

Liam 1

秒杀这个案例中，超时之后不需要补偿机制吗，对于下游服务来说很可能以及成功了

2019-08-14

作者回复

这个案例中，你说的这种情况是有可能存在的。

是否需要补偿，也无所谓对错，总体效果是一样的。秒杀的目的就是从众多秒杀用户中公平的选择n个用户，补偿或不补偿，影响的只是这n个用户是谁的问题。

所以这是一个架构选择的问题。

我建议是不用补偿，按失败处理，锁定的库存超时未支付后会自动释放，好处是比较简单。

2019-08-14

微微一笑 1

老师好，看了秒杀的代码，有点疑问：
等待后端超过timeout设置的时间点，且没有秒杀结果，finally代码块中会remove掉这个请求id，并返回用户秒杀失败；若在remove之前，后端服务返回了秒杀结果并秒杀成功，在非常极致的情况下，会不会出现用户看到秒杀失败，系统却秒杀成功的情况发生呢？

2019-08-09

作者回复

你可以再看一下这课的文本，我补充了一个流程图。

对于你说的这个情况，是不会出现的。因为，后端服务返回的秒杀结果，只会存放在Map中，并不会直接返回APP。

给APP返回结果的，只能是处理APP请求的那个线程。

2019-08-09

sswzfly 1

我是消息队列小达人了

2019-08-09

大白先生 1

对于这句话我不是很理解，还请老师帮忙解答。“队列中当前有 10 条消息，对应的编号是 0-9，当前的消费位置是 5。同时来了三个消费者来拉消息，把编号为 5、6、7 的消息分别给三个消费者，每人一条。”一个消费队列在一个消费组内不是只有一个消费者可以消费消息么，怎么能够同时来拉消息。如果您说的三个消费者是三个消费组的话，每个消费组维护自己的消费位置都是5，那么消费的都会是编号5的消息呀。

2019-08-08

作者回复

我们不是在考虑如何来实现并行消费么？就不能再遵循现有的串行消费的限制了呀。

2019-08-08

Mark Yao 1

受益老师说的先宏观在微观深入，同时做事不需要整体有序，达到局部有序即可。

2019-08-08

明日 1

老师，关于事务消息的ACID那个问题没有提到，能不能找机会说下你的看法？个人的看法是没有实现隔离性，一致性只能保证最终一致，而原子操作和持久化可以通过各种手段实现。

2019-08-08

作者回复

严格的说，ACI都没实现，只有D实现了。

放宽点儿限制的话，或者考虑实际效果的话，A（原子性）绝大多数情况下还是可以保证的，即“要么都成功，要么都失败”。C（一致性）通过补偿，大部分情况下也可以保证最终一致。

2019-08-08

asdf100 0

订阅模型中，消息的发送方称为发布者（Publisher），消息的接收方称为订阅者（Subscriber），服务端存放消息的容器称为主题（Topic）。发布者将消息发送到主题中，订阅者在接收消息之前需要先“订阅主题”。“订阅”在这里既是一个动作，同时还可以认为是主题在消费时的一个逻辑副本，每份订阅中，订阅者都可以接收到主题的所有消息。

那么消息到底存储在主题里还是队列里？？？？

2019-08-22

作者回复

队列

2019-08-23

asdf100 0

消息一直在队列里不删除？

2019-08-22

作者回复

消息什么时候删除取决于消息队列的配置，比如Kafka默认就是超过多长时间后就自动删除了。

2019-08-23

asdf100 0

每个消费组就是一份订阅，它要消费主题 MyTopic 下，所有队列的全部消息。注意，队列里的消息并不是消费掉就没有了，这里的“消费”，只是去队列里面读了消息，并没有删除，消费完这条消息还是在队列里面。

理解：所有消息是保存在主题下面的队列里

2019-08-22

vi 0

李sir，一个迟来的学生，刚刚看到这里，对于rabbitmq来说，没有消费组队列，只有exchange转送到相应的对列中，要想提高并发，看到的方法可以设置多线程消息，好像变成了单个对列并行消息的模式，会不会也有文中所说的这个问题，rabbitmq还可以通过设置prefetch来缓存一定的数目，是不是就相当于增加每个消费者的队列数来解决并发的问题了

2019-08-21

作者回复

想要增加消费并发，可以考虑exchange把消息平均的分摊到多个消费队列中。

2019-08-22

godtrue 0

老师这节答疑真棒，越来越觉得老师像吴彦祖啦！
值得反复品味!
一图胜千言，如果之后老师的图上来了，那在我心目中老师就是吴彦祖

2019-08-21

木木木 0

关于有序性还有疑惑，即使采用了一致性hash，无论扩容还是缩容队列，对分配相邻队列的用户部分还是有影响的，难道要等这些队列消费完了，阻止生产者发消息吗？感觉不具有可操作性

2019-08-20

作者回复

扩容后只需要等一会儿，确保扩容之前的消息都消费完成了（不确定的话可以等久一点儿也没关系）再消费新分区的数据就可以了，生产不需要停。

因为一致性哈希可以保证单调性：如果已经有一些内容通过哈希分派到了相应的分区中，又有新的分区加入到系统中。哈希的结果应能够保证原有已分配的内容可以被映射到原有的或者新的分区中去，而不会被映射到旧的分区集合中的其他分区

2019-08-20

康师傅

0

如果一个topic中有多个消费者，但每个消费者可能只需要其中的一部分数据，一种可行的方案是消费者消费全量消息，然后自行过滤；另一种方式是生产者将这些消息进行分类，不同类别的消息分别对应不同的topic，但这样可能会出现N多的topic，topic太多是否又会出现随机io太多导致性能问题，另外对生产端的编码也不友好，每种消息都要感知发到哪个topic中

这种情况下应该如何取舍

2019-08-17

作者回复

可以使用RocketMQ的服务端过滤功能，正好可以满足你这个需求。

2019-08-18