01-为什么需要消息队列？

你好，我是李玥。今天我们来讲讲为什么需要消息队列，消息队列主要解决的是什么问题。

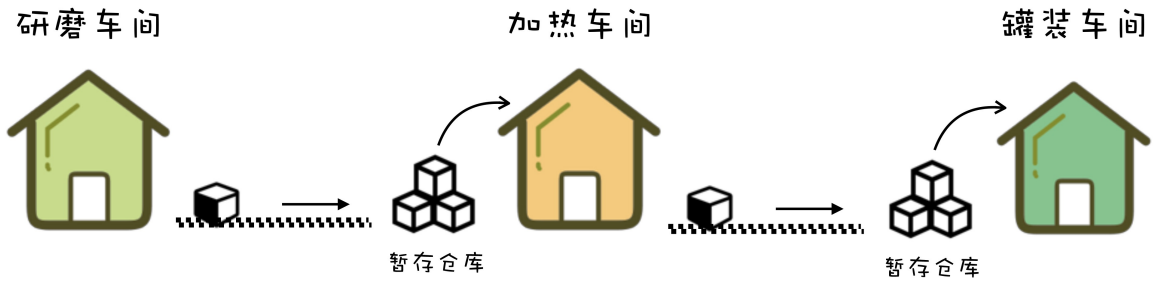
消息队列是最古老的中间件之一，从系统之间有通信需求开始，就自然产生了消息队列。但是给消息队列下一个准确的定义却不太容易。我们知道，消息队列的主要功能就是收发消息，但是它的作用不仅仅只是解决应用之间的通信问题这么简单。

我们举个例子说明一下消息队列的作用。话说小袁是一家巧克力作坊的老板，生产出美味的巧克力需要三道工序：首先将可可豆磨成可可粉，然后将可可粉加热并加入糖变成巧克力浆，最后将巧克力浆灌入模具，撒上坚果碎，冷却后就是成品巧克力了。

最开始的时候，每次研磨出一桶可可粉后，工人就会把这桶可可粉送到加工巧克力浆的工人手上，然后再回来加工下一桶可可粉。小袁很快就发现，其实工人可以不用自己运送半成品，于是他在每道工序之间都增加了一组传送带，研磨工人只要把研磨好的可可粉放到传送带上，就可以去加工下一桶可可粉了。 传送带解决了上下游工序之间的“通信”问题。

传送带上线后确实提高了生产效率，但也带来了新的问题：每道工序的生产速度并不相同。在巧克力浆车间，一桶可可粉传送过来时，工人可能正在加工上一批可可粉，没有时间接收。不同工序的工人们必须协调好什么时间往传送带上放置半成品，如果出现上下游工序加工速度不一致的情况，上下游工人之间必须互相等待，确保不会出现传送带上的半成品无人接收的情况。

为了解决这个问题，小袁在每组传送的下游带配备了一个暂存半成品的仓库，这样上游工人就不用等待下游工人有空，任何时间都可以把加工完成的半成品丢到传送带上，无法接收的货物被暂存在仓库中，下游工人可以随时来取。传送带配备的仓库实际上起到了“通信”过程中“缓存”的作用。



传送带解决了半成品运输问题，仓库可以暂存一些半成品，解决了上下游生产速度不一致的问题，小袁在不知不觉中实现了一个巧克力工厂版的消息队列。

哪些问题适合使用消息队列来解决？

接下来我们说一下日常开发中，哪些问题适合使用消息队列解决。

1. 异步处理

大多数程序员在面试中，应该都问过或被问过一个经典却没有标准答案的问题：如何设计一个秒杀系统？这个问题可以有一百个版本的合理答案，但大多数答案中都离不开消息队列。

秒杀系统需要解决的核心问题是，如何利用有限的服务器资源，尽可能多地处理短时间内的海量请求。我们

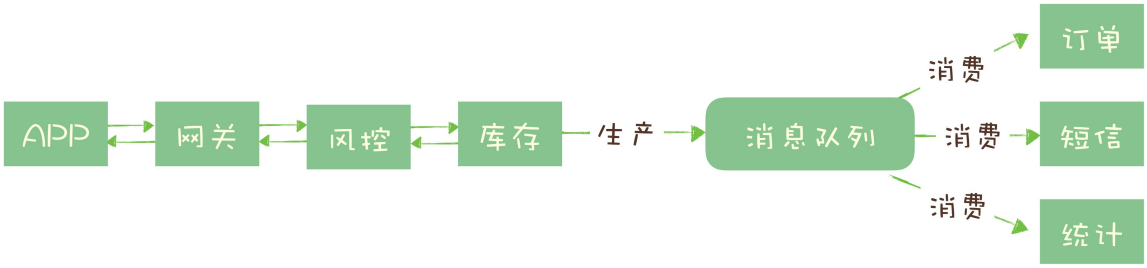
知道，处理一个秒杀请求包含了很多步骤，例如：

- 风险控制；
- 库存锁定；
- 生成订单；
- 短信通知；
- 更新统计数据。

如果没有任何优化，正常的处理流程是：App将请求发送给网关，依次调用上述5个流程，然后将结果返回给APP。

对于对于这5个步骤来说，能否决定秒杀成功，实际上只有风险控制和库存锁定这2个步骤。只要用户的秒杀请求通过风险控制，并在服务端完成库存锁定，就可以给用户返回秒杀结果了，对于后续的生成订单、短信通知和更新统计数据等步骤，并不一定要在秒杀请求中处理完成。

所以当服务端完成前面2个步骤，确定本次请求的秒杀结果后，就可以马上给用户返回响应，然后把请求的数据放入消息队列中，由消息队列异步地进行后续的操作。



处理一个秒杀请求，从5个步骤减少为2个步骤，这样不仅响应速度更快，并且在秒杀期间，我们可以把大量的服务器资源用来处理秒杀请求。秒杀结束后再把资源用于处理后面的步骤，充分利用有限的服务器资源处理更多的秒杀请求。

可以看到，在这个场景中，消息队列被用于实现服务的异步处理。这样做的好处是：

- 可以更快地返回结果；
- 减少等待，自然实现了步骤之间的并发，提升系统总体的性能。

2. 流量控制

继续说我们的秒杀系统，我们已经使用消息队列实现了部分工作的异步处理，但我们还面临一个问题：如何避免过多的请求压垮我们的秒杀系统？

一个设计健壮的程序有自我保护的能力，也就是说，它应该可以在海量的请求下，还能在自身能力范围内尽可能多地处理请求，拒绝处理不了的请求并且保证自身运行正常。不幸的是，现实中很多程序并没有那么“健壮”，而直接拒绝请求返回错误对于用户来说也是不怎么好的体验。

因此，我们需要设计一套足够健壮的架构来将后端的服务保护起来。**我们的设计思路是，使用消息队列隔离**

网关和后端服务，以达到流量控制和保护后端服务的目的。

加入消息队列后，整个秒杀流程变为：

1. 网关在收到请求后，将请求放入请求消息队列；
2. 后端服务从请求消息队列中获取APP请求，完成后续秒杀处理过程，然后返回结果。

秒杀开始后，当短时间内大量的秒杀请求到达网关时，不会直接冲击到后端的秒杀服务，而是先堆积在消息队列中，后端服务按照自己的最大处理能力，从消息队列中消费请求进行处理。

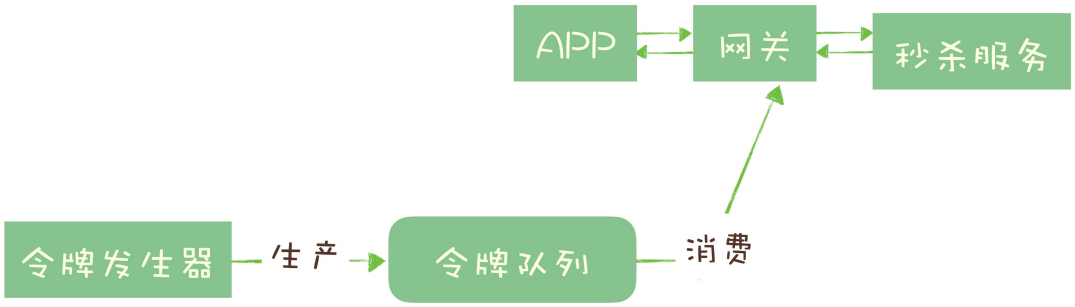
对于超时的请求可以直接丢弃，APP将超时无响应的请求处理为秒杀失败即可。运维人员还可以随时增加秒杀服务的实例数量进行水平扩容，而不用对系统的其他部分做任何更改。

这种设计的优点是：能根据下游的处理能力自动调节流量，达到“削峰填谷”的作用。但这样做同样是有代价的：

- 增加了系统调用链环节，导致总体的响应增加了系统调用链环节，导致总体的响应时延变长。
- 上下游系统都要将同步调用改为异步消息，增加了系统的复杂度。

那还有没有更简单一点儿的流量控制方法呢？如果我们能预估出秒杀服务的处理能力，就可以用消息队列实现一个令牌桶，更简单地进行流量控制。

令牌桶控制流量的原理是：单位时间内只发放固定数量的令牌到令牌桶中，规定服务在处理请求之前必须先从令牌桶中拿出一个令牌，如果令牌桶中没有令牌，则拒绝请求。这样就保证单位时间内，能处理的请求不超过发放令牌的数量，起到了流量控制的作用。



实现的方式也很简单，不需要破坏原有的调用链，只要网关在处理APP请求时增加一个获取令牌的逻辑。

令牌桶可以简单地用一个有固定容量的消息队列加一个“令牌发生器”来实现：令牌发生器按照预估的处理能力，匀速生产令牌并放入令牌队列（如果队列满了则丢弃令牌），网关在收到请求时去令牌队列消费一个令牌，获取到令牌则继续调用后端秒杀服务，如果获取不到令牌则直接返回秒杀失败。

以上是常用的使用消息队列两种进行流量控制的设计方法，你可以根据各自的优缺点和不同的适用场景进行合理选择。

3. 服务解耦

消息队列的另外一个作用，就是实现系统应用之间的解耦。再举一个电商的例子来说明解耦的作用和必要性。

我们知道订单是电商系统中比较核心的数据，当一个新订单创建时：

1. 支付系统需要发起支付流程；
2. 风控系统需要审核订单的合法性；
3. 客服系统需要给用户发短信告知用户；
4. 经营分析系统需要更新统计数据；
5. ……

这些订单下游的系统都需要实时获得订单数据。随着业务不断发展，这些订单下游系统不断的增加，不断变化，并且每个系统可能只需要订单数据的一个子集，负责订单服务的开发团队不得不花费很大的精力，应对不断增加变化的下游系统，不停地修改调试订单系统与这些下游系统的接口。任何一个下游系统接口变更，都需要订单模块重新进行一次上线，对于一个电商的核心服务来说，这几乎是不可接受的。

所有的电商都选择用消息队列来解决类似的系统耦合过于紧密的问题。引入消息队列后，订单服务在订单变化时发送一条消息到消息队列的一个主题Order中，所有下游系统都订阅主题Order，这样每个下游系统都可以获得一份实时完整的订单数据。

无论增加、减少下游系统或是下游系统需求如何变化，订单服务都无需做任何更改，实现了订单服务与下游服务的解耦。

小结

以上就是消息队列最常被使用的三种场景：异步处理、流量控制和服务解耦。当然，消息队列的适用范围不仅仅局限于这些场景，还有包括：

- 作为发布/订阅系统实现一个微服务级系统间的观察者模式；
- 连接流计算任务和数据；
- 用于将消息广播给大量接收者。

简单的说，我们在单体应用里面需要用队列解决的问题，在分布式系统中大多都可以用消息队列来解决。

同时我们也要认识到，消息队列也有它自身的一些问题和局限性，包括：

- 引入消息队列带来的延迟问题；
- 增加了系统的复杂度；
- 可能产生数据不一致的问题。

所以我们说没有最好的架构，只有最适合的架构，根据目标业务的特点和自身条件选择合适的架构，才是体

现一个架构师功力的地方。

思考题

在你工作或学习涉及到的系统中，哪些问题可以通过引入消息队列来解决？对于系统中已经使用消息队列，可以对应到这一讲中提到的哪个场景？如果没有可以对应的场景，那这个消息队列在系统中起到的是什么作用？解决了什么问题？是否又带来了什么新的问题？欢迎在留言区写下你的想法。

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：

- 白小白 2019-07-23 08:00:43
  现在用的消息队列主要是做数据异步传输用的，之前也做过多个系统之间的解耦。看到用消息队列做秒杀系统，忽然想到之前只想过用redis去做，利用redis去做了流量的把控。不过细想想，这种情况下的redis和文章中的令牌桶很像…… [2赞]

  作者回复2019-07-23 09:48:49
  是的，令牌桶可以用消息队列实现，也可以用Redis实现，你也可以写一个简单的令牌桶服务，原理是一样的。

- QQ怪 2019-07-23 08:05:33
  滴滴滴，打卡 [1赞]

  作者回复2019-07-23 09:49:48
  打卡已收到，记得每节课都来打卡。

- mhswordman 2019-07-23 07:10:07
  生产项目中用到了kafka，
  1 异部的处理交易：提高用户请求的响应速度，同时也提升了用户的体验感。
  2 削峰 ： 保护服务器的一种方式，用户的请求放到kafka中，交易服务根据自己服务器的消费能力来消费交易数据。

3 项目的解耦：交易服务和后续的服务之间是通过Kafka进行交付，当一个服务为多个服务提供数据的时候，可以通过MQ进行交换来解耦服务间的耦合。 [1赞]

作者回复2019-07-23 09:44:36
总结的很赞！

- 我知道了嗯 2019-07-23 06:31:44
打卡，滴滴滴 [1赞]

作者回复2019-07-23 09:43:09
记得每节课来打卡。

- 流氓无产者 2019-07-23 06:30:32
修改数据库做数据同步也可以用 [1赞]

作者回复2019-07-23 09:42:54
是的，很多公司会用消息队列来做异构数据库之间的数据同步，但是一定要注意顺序问题。像MySQL Binlog这种，是要求严格有序的，否则会出现问题。

- 江南皮革厂研发中心保安队长 2019-07-23 10:05:03
工作中用过的应该是在云平台上的rabbitmq，对虚拟机生命周期管理的调度和对接云管平台的工单（创建、迁移、扩容等）服务动作，这个不像电商系统，对请求处理和实时响应的要求没有那么高，主要是在数据一致性和通信性能上有一定要求。

- Jxin 2019-07-23 09:50:27
1.拆单失败的延时重拆，死信告警。2.消峰和解耦也用到。

问题：控制topic消费线程也能限流，不一定要引入令牌桶，要弄令牌桶，其实走redis更好一点。

- 广训 2019-07-23 09:31:05
服务解耦用的最多。尤其是订单状态变化，需要通知各种下游系统和重新推送最新动态给外部商家。不过消息队列也没那么容易使用，特别是topic的设计，如果前期设计缺陷不及时补救，后期只能无限制的增加topic，直到成为灾难。我现在的项目就是如此，后继者只能增加新的topic来适应新业务，有的地方甚至保留各种双写，不丢弃旧业务，也解决新业务。所以说架构选型和设计，即使是糟粕，也会延续很长一段时间，给所有后继者都带来更大的工作量和维护成本。

- 吕 2019-07-23 08:57:45
目前项目用到了rabbitmq和kafka，rabbitmq用于用户系统通知和相关用户画像模型的处理，进行个性化推荐，同时利用redis和mq，进行文章简单信息的缓存，减少后台数据库的压力，kafka主要是负责爬虫的文章存储，然后各个微服务进行文章内容的订阅

作者回复2019-07-23 09:58:37
有没有想过要统一成一种消息队列呢？

- kk 2019-07-23 08:40:29
打卡，滴滴

- Jaising 2019-07-23 08:38:28
所在教育项目没有高实时性要求，有两个地方用到了消息队列：
1.大量数据定时入库（从数仓到mongo）

2.较大文件的下载
按照李老师总结的，是异步处理和服务解耦
知其然，更要知其所以然，期待后续课程😄

作者回复2019-07-23 09:55:32
记得坚持学习

- 老王的老李头 2019-07-23 08:37:01
小袁是个有故事的朋友

作者回复2019-07-23 09:54:49
小袁：“我不是，我没有，你们别乱说！”

- 夏天 2019-07-23 08:22:14
第一天打卡

作者回复2019-07-23 09:53:29
坚持，加油。

- 盛 2019-07-23 08:11:34
服务解耦：其实就是后续将要面对的问题。
数据库压力偏大：虽然性能已经通过之前的丁奇的课获得了不少思路和处理方式，确实让数据库的查询时间提升到之前的20-30%。
数据量进一步再次增长后期压力太大了：redis已经用了，可是还是不够，希望能够通过消息队列预先为6-12个月后的大数据量做些准备，以应当后期。

作者回复2019-07-23 09:53:11
架构可以随着业务的发展持续的演进，相信问题都可以解决。

- 张学磊 2019-07-23 07:52:04
目前在工作中使用的场景是工作流向业务系统发审批结果消息，业务系统接入工作流进行流程审批，工作流中任何一个节点的推动都向业务系统所订阅的主题发审批结果消息，业务系统收到消息后进行状态更新及后续操作。比场景使用消息队列的目的是进行系统解耦和异步调用。

作者回复2019-07-23 09:46:40
赞总结！

- Tx爱上360 2019-07-23 02:11:12
打卡，滴

作者回复2019-07-23 09:41:30
记得每节课都来打卡。