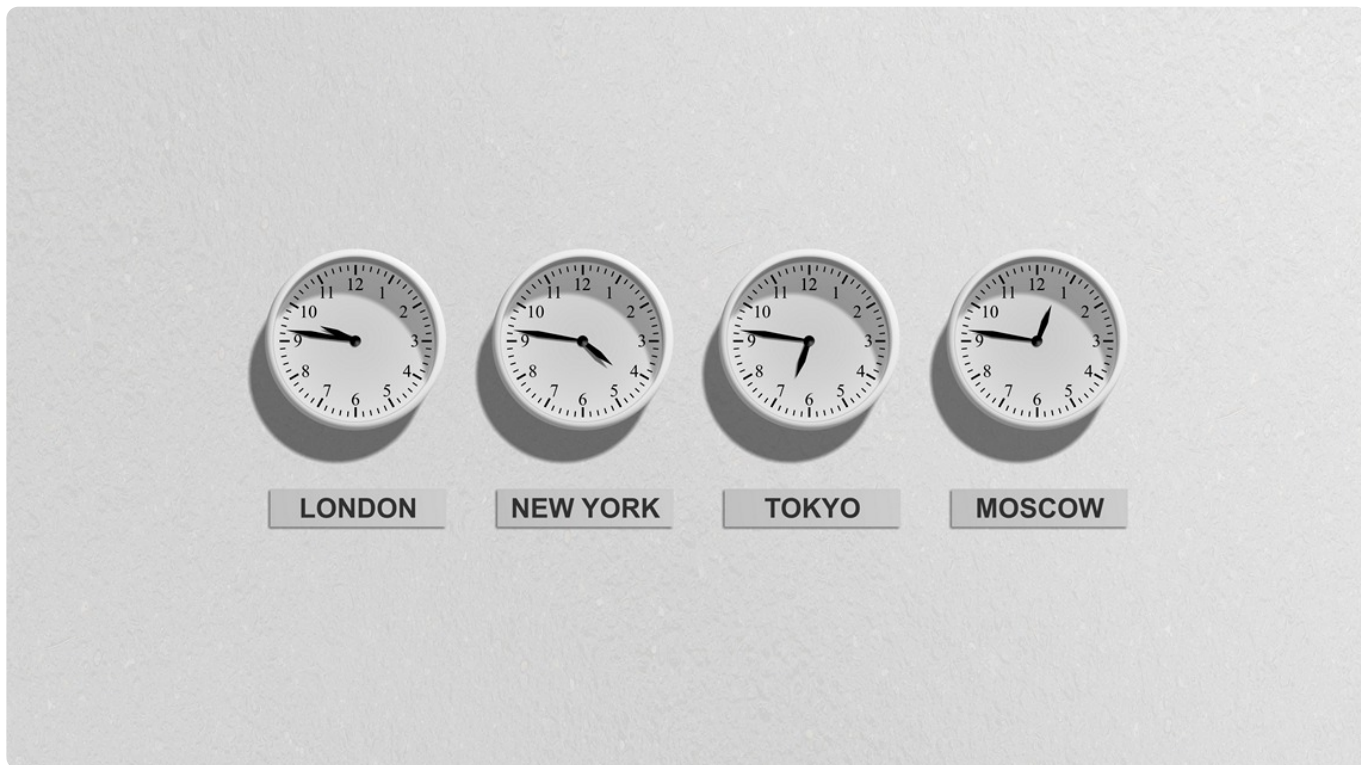


## 43 | 弹力设计篇之“异步通讯设计”

2018-02-27 陈皓

左耳听风

[进入课程 >](#)



讲述：柴巍

时长 11:16 大小 5.16M



前面所说的隔离设计通常都需要对系统做解耦设计，而把一个单体系统解耦，不单单是把业务功能拆分出来，正如上面所说，拆分完后还会面对很多的问题。其中一个重要的问题就是这些系统间的通讯。

通讯一般来说分同步和异步两种。同步通讯就像打电话，需要实时响应，而异步通讯就像发邮件，不需要马上回复。各有千秋，我们很难说谁比谁好。但是在面对超高吐吞量的场景下，异步处理就比同步处理有比较大的优势了，这就好像一个人不可能同时接打很多电话，但是他可以同时接收很多的电子邮件一样。

同步调用虽然让系统间只耦合于接口，而且实时性也会比异步调用要高，但是我們也需要知道同步调用会带来如下几个问题。

同步调用需要被调用方的吞吐不低于调用方的吞吐。否则会导致被调用方因为性能不足而拖死调用方。换句话说，整个同步调用链的性能会由最慢的那个服务所决定。

同步调用会导致调用方一直在等待被调用方完成，如果一层接一层地同步调用下去，所有的参与方会有相同的等待时间。这会非常消耗调用方的资源。因为调用方需要保存现场（Context）等待远端返回，所以对于并发比较高的场景来说，这样的等待可能会极度消耗资源。

同步调用只能是一对一的，很难做到一对多。

同步调用最不好是，如果被调用方有问题，那么其调用方就会跟着出问题，于是会出现多米诺骨牌效应，故障一下就蔓延开来。

所以，异步通讯相对于同步通讯来说，除了可以增加系统的吞吐量之外，最大的一个好处是其可以让服务间的解耦更为彻底，系统的调用方和被调用方可以按照自己的速率而不是步调一致，从而可以更好地保护系统，让系统更有弹力。

异步通讯通常来说有三种方式。

## 异步通讯的三种方式

### 请求响应式

在这种情况下，发送方（sender）会直接请求接收方（receiver），被请求方接收到请求后，直接返回——收到请求，正在处理。

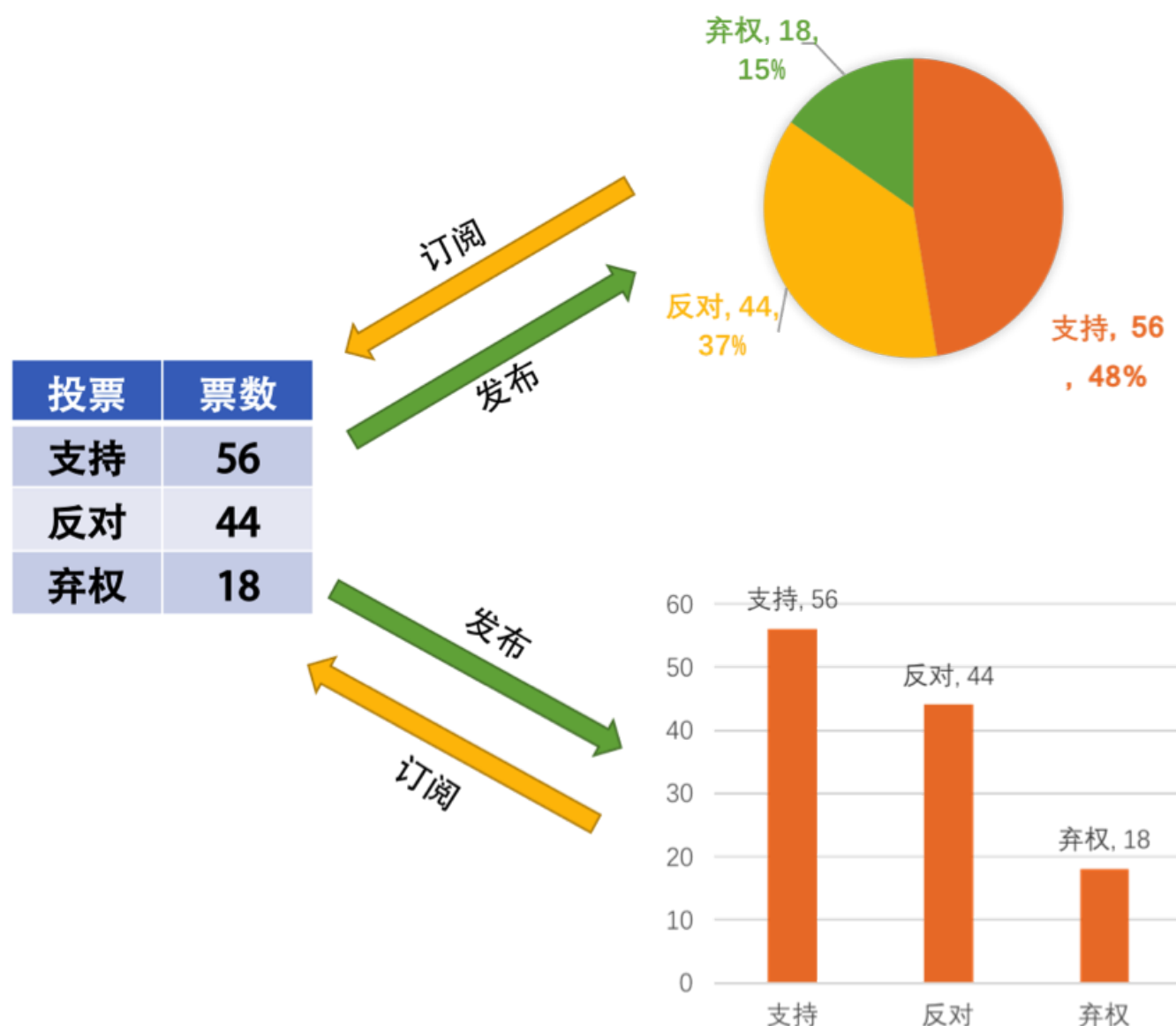
对于返回结果，有两种方法，一种是发送方时不时地去轮询一下，问一下干没干完。另一种方式是发送方注册一个回调方法，也就是接收方处理完后回调请求方。这种架构模型在以前的网上支付中比较常见，页面先从商家跳转到支付宝或银行，商家会把回调的 URL 传给支付页面，支付完后，再跳转回商家的 URL。

很明显，这种情况下还是有一定耦合的。是发送方依赖于接收方，并且要把自己的回调发送给接收方，处理完后回调。

### 通过订阅的方式

这种情况下，接收方（receiver）会来订阅发送方（sender）的消息，发送方会把相关的消息或数据放到接收方所订阅的队列中，而接收方会从队列中获取数据。

这种方式下，发送方并不关心订阅方的处理结果，它只是告诉订阅方有事要干，收完消息后给个 ACK 就好了，你干成啥样我不关心。这个方式常用于像 MVC ( Model-View-Control ) 这样的设计模式下，如下图所示。



这就好像下订单的时候，一旦用户支付完成了，就需要把这个事件通知给订单处理以及物流，订单处理变更状态，物流服务需要从仓库服务分配相应的库存并准备配送，后续这些处理的结果无需告诉支付服务。

为什么要做成这样？好了，重点来了！前面那种请求响应的方式就像函数调用一样，这种方式有数据有状态的往来（也就是说需要有请求数据、返回数据，服务里面还可能保存调用的状态），所以服务是有状态的。如果我们把服务的状态给去掉（通过第三方的状态服务来保证），那么服务间的依赖就只有事件了。

你知道，分布式系统的服务设计是需要向无状态服务（Stateless）努力的，这其中有太多的好处，无状态意味着你可以非常方便地运维。所以，事件通讯成为了异步通讯中最重要的一个设计模式。

就上面支付的那个例子，商家这边只需要订阅一个支付完成的事件，这个事件带一个订单号，而不需要让支付方知道自己的回调 URL，这样的异步是不是更干净一些？

但是，在这种方式下，接收方需要向发送方订阅事件，所以是接收方依赖于发送方。这种方式还是有一定的耦合。

## 通过 Broker 的方式

所谓 Broker，就是一个中间人，发送方（sender）和接收方（receiver）都互相看不到对方，它们看得到的是一个 Broker，发送方向 Broker 发送消息，接收方向 Broker 订阅消息。如下图所示。



这是完全的解耦。所有的服务都不需要相互依赖，而是依赖于一个中间件 Broker。这个 Broker 是一个像数据总线一样的东西，所有的服务要接收数据和发送数据都发到这个总线上，这个总线就像协议一样，让服务间的通讯变得标准和可控。

在 Broker 这种模式下，发送方的服务和接收方的服务最大程度地解耦。但是所有人都依赖于一个总线，所以这个总线就需要有如下的特性：

- 必须是高可用的，因为它成了整个系统的关键；

- 必须是高性能而且是可以水平扩展的；

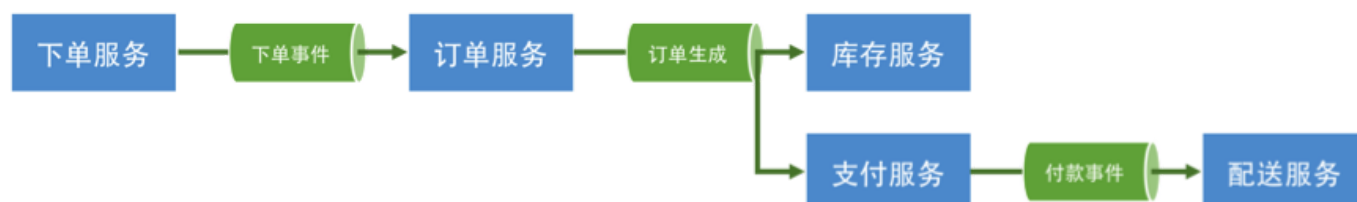
- 必须是可以持久化不丢数据的。

要做到这三条还是比较难的。当然，好在现在开源软件或云平台上 Broker 的软件是非常成熟的，所以节省了我们很多的精力。

## 事件驱动设计

上述的第二种和第三种方式就是比较著名的事件驱动架构（EDA – Event Driven Architecture）。正如前面所说，事件驱动最好是使用 Broker 方式，服务间通过交换消息来完成交流和整个流程的驱动。

如下图所示，这是一个订单处理流程。下单服务通知订单服务有订单要处理，而订单服务生成订单后发出通知，库存服务和支付服务得到通知后，一边是占住库存，另一边是让用户支付，等待用户支付完成后通知配送服务进行商品配送。



每个服务都是“自包含”的。所谓“自包含”也就是没有和别人产生依赖。而要把整个流程给串联起来，我们需要一系列的“消息通道（Channel）”。各个服务做完自己的事后，发出相应的事件，而又有一些服务在订阅着某些事件来联动。

事件驱动方式的好处至少有五个。

服务间的依赖没有了，服务间是平等的，每个服务都是高度可重用并可被替换的。

服务的开发、测试、运维，以及故障处理都是高度隔离的。

服务间通过事件关联，所以服务间是不会相互 block 的。

在服务间增加一些 Adapter（如日志、认证、版本、限流、降级、熔断等）相当容易。

服务间的吞吐也被解开了，各个服务可以按照自己的处理速度处理。

我们知道任何设计都有好有不好的方式。事件驱动的架构也会有一些不好的地方。

业务流程不再那么明显和好管理。整个架构变得比较复杂。解决这个问题需要有一些可视化的工具来呈现整体业务流程。

事件可能会乱序。这会带来非常 Bug 的事。解决这个问题需要很好地管理一个状态机的控制。

事务处理变得复杂。需要使用两阶段提交来做强一致性，或是退缩到最终一致性。

## 异步通讯的设计重点

首先，我们需要知道，为什么要异步通讯。

异步通讯最重要的是解耦服务间的依赖。最佳解耦的方式是通过 Broker 的机制。

解耦的目的是让各个服务的隔离性更好，这样不会出现“一倒倒一片”的故障。

异步通讯的架构可以获得更大的吞吐量，而且各个服务间的性能不受干扰相对独立。

利用 Broker 或队列的方式还可以达到把抖动的吞吐量变成均匀的吞吐量，这就是所谓的“削峰”，这对后端系统是个不错的保护。

服务相对独立，在部署、扩容和运维上都可以做到独立不受其他服务的干扰。

但我们需要知道这样的方式带来的问题，所以在设计成异步通信的时候需要注意如下事宜。

用于异步通讯的中间件 Broker 成为了关键，需要设计成高可用不丢消息的。另外，因为是分布式的，所以可能很难保证消息的顺序，因此你的设计最好不依赖于消息的顺序。

异步通讯会导致业务处理流程不那么直观，因为像接力一样，所以在 Broker 上需要有相关的服务消息跟踪机制，否则出现问题后不容易调试。

因为服务间只通过消息交互，所以业务状态最好由一个总控方来管理，这个总控方维护一个业务流程的状态变迁逻辑，以便系统发生故障后知道业务处理到了哪一步，从而可以在故障清除后继续处理。

这样的设计常见于银行的对账程序，银行系统会有大量的外部系统通讯，比如跨行的交易、跨企业的交易，等等。所以，为了保证整体数据的一致性，或是避免漏处理及处理错的交易，需要有对账系统，这其实就是那个总控，这也是为什么银行有的交易是 T+1（隔天结算），就是因为要对个账，确保数据是对的。

消息传递中，可能有的业务逻辑会有像 TCP 协议那样的 send 和 ACK 机制。比如：A 服务发出一个消息之后，开始等待处理方的 ACK，如果等不到的话，就需要做重传。此时，需要处理方有幂等的处理，即同一件消息无论收到多少次都只处理一次。

## 小结

好了，我们来总结一下今天分享的主要内容。首先，同步调用有四个问题：影响吞吐量、消耗系统资源、只能一对一，以及有多米诺骨牌效应。于是，我们想用异步调用来避免该问题。

异步调用有三种方式：请求响应、直接订阅和中间人订阅。最后，我介绍了事件驱动设计的特点和异步通讯设计的重点。下篇文章中，我们讲述幂等性设计。希望对你有帮助。

也欢迎你分享一下你在分布式服务的设计中，哪些情况下使用异步通讯？是怎样设计的？又有哪些情况使用同步通讯？

文末给出了《分布式系统设计模式》系列文章的目录，希望你能在这个列表里找到自己感兴趣的内容。

## 弹力设计篇

[认识故障和弹力设计](#)

[隔离设计 Bulkheads](#)

[异步通讯设计 Asynchronous](#)

[幂等性设计 Idempotency](#)

[服务的状态 State](#)

[补偿事务 Compensating Transaction](#)

[重试设计 Retry](#)

[熔断设计 Circuit Breaker](#)

[限流设计 Throttle](#)

[降级设计 degradation](#)

[弹力设计总结](#)

## 管理设计篇

[分布式锁 Distributed Lock](#)

[配置中心 Configuration Management](#)

[边车模式 Sidecar](#)

[服务网格 Service Mesh](#)

[网关模式 Gateway](#)

[部署升级策略](#)



性能设计篇

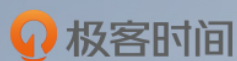
[缓存 Cache](#)

[异步处理 Asynchronous](#)

[数据库扩展](#)

[秒杀 Flash Sales](#)

[边缘计算 Edge Computing](#)



# 左耳朵耗子

全年独家专栏《左耳听风》

20000 名程序员的练级攻略

陈皓 资深技术专家  
骨灰级程序员



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 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 42 | 弹力设计篇之“隔离设计”

下一篇 44 | 弹力设计篇之“幂等性设计”

## 精选留言 (19)

写留言



wgy

2018-02-28

10

陈老师能不能推荐几个开源的Broker软件？



展开 ▾



**Bing**

2018-05-11

👍 7

进程内，使用EventBus，进程外使用MQ。现在业务难点就是在消息的顺序性上 😊



**幻想**

2018-02-27

👍 4

见过的，写异步最好的文章了。总结的太好了。不过有些业务呢，像下完订单并支付后，用消息通知的方式，立刻流单也不是很好。一方面可能要等到某个时机才流单，尤其是大促时，用户取消订单的很多。另外，也想在高峰期优先全部资源搞下单和支付处理，不搞其他，等高峰小一些的时候，才处理售后的一些业务。

展开 ▾



**名贤集**

2018-03-01

👍 3

是否能推荐个靠谱的事件框架

展开 ▾



**涛哥迷妹**

2018-07-28

👍 2

事件可能会乱序请问这个如何保证需要有序处理业务的场景，状态机主要做什么？



**小沫**

2018-03-13

👍 2

broker方式 是否可以理解成 消息中间件方式  
发送方为 消息生产者 将消息发送到 Q 中  
接收方为 消息消费者 将消息从Q中取出



**Freezer**

2018-03-01

👍 2

干货

展开 ∨



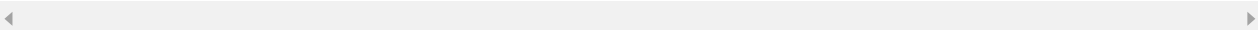
朱海峰

2018-03-21

👍 1

这篇无意中讲了微服务的东西，在go-micro一直猜broker是啥作用

作者回复: 是有意的，嘿嘿



华煜

2018-02-27

👍 1

我们现在大部的异步都是通过请求响应的方式完成的，在http的接口下感觉耦合还不明显，现在工公司推了rpc，在rpc接口下，耦合很明显的，感觉可能用订阅的方式比较好，但这请求响应的方式应该可靠性更好一些，而且设计简单一些，不涉及到丢消息（中间件自身问题或网络原因等等），因为每一次丢消息都可能带来比较高的运维成本，耗子叔怎么看，至于无状态，在这个场景下的坑能简单说下么，没经历过。所以感觉不到重要性。

展开 ∨



sprzhing

2019-05-11

👍

有哪些broker呢？

展开 ∨



Geek\_9ed47...

2019-04-16

👍

broker可以理解成消息队列吧，用的比较多的是kafka，个人觉得需要注意在消费消息的时候业务失败了要做好相应的处理，要不然会出现数据不一致。

展开 ∨



godtrue

2019-01-25

👍

消息的接受有序性确实不好控制，不过可以通过发送消息的时间戳来感知一下。我们的订单全程跟踪消息的有序性就是这么弄的。

展开 ▾



恒修

2018-11-11



下单成功，支付成功，库存失败需要回退用户钱  
这个场景流程扣库存和支付是一个串行流程



阿康

2018-09-18



事件驱动 发布订阅模型只适用于内部系统，对外的服务还是只能用 请求响应式的异步通信方式，这样理解没问题吧？

展开 ▾



403

2018-07-30



我也想知道消息的顺序性怎么保证

展开 ▾

作者回复: 分布式环境下，非常之难。



王磊

2018-06-10



不太理解为什么说'请求响应式'是有状态的，我理解请求方可以把请求发送给任意的接收方，任意的接收方都可以独立的完成处理，这种情况下可以方便的scale-up接收方的实例个数。还有说这种状态下是有数据，状态传输的，但即使第二种和第三种的消息里也是有数据的。我理解到请求响应式还只能是1对1，而第二种可以1对多，这是我理解到的区别。请解惑。

展开 ▾



来

2018-05-20



目前系统中的异步通信主要是采用消息中间件，消息中间件采用的异步方式为broker方式。目前系统中异步通信的使用场景如下：

- 1、削峰上游系统调用的压力，借助mq先将订单接下来，然后根据系统自身的处理能力来处理请求
  - 2、当某一动作、事件完成后，将消息广播出去，其它业务系统监听此消息，然后做响应...
- 展开 ∨
- 



kingeaster...

2018-03-29



gokit 或go micro 的rpc机制是否适合大文件的流式上传或下载，目前想把手头的服务rpc话，但是涉及到文件的上传或下载不知道怎么处理

展开 ∨



kingeaster...

2018-03-29



gokit 或go micro 的rpc机制是否适合大文件的流式上传或下载，目前想把手头的服务rpc话，但是涉及到文件的上传或下载不知道怎么处理

展开 ∨