08 | 可复用架构案例（一）：如何设计一个基础服务?

2020-03-09 王庆友

架构实战案例解析 进入课程 >

讲述：王庆友
时长 16:09 大小 14.80M

你好，我是王庆友。

在上一讲中，我提到过，在架构设计中，要实现业务上的复用，一个比较可行的做法是，把各个基础业务封装成共享服务，供上层所有应用调用。所以今天，我就来和你聊一聊，如何从头开始，落地这样一个典型的共享服务。

我们知道，落地一个微服务其实并不困难，但要实现一个能够高度复用的共享服务并不容易，在落地过程中，经常会有一系列的问题困扰着我们。

> 我们事先对服务的边界没有进行很好的划分，结果在落地的过程中，大家反复争论具体功能的归属。

由于对业务的了解不够深入，我们要么设计不足，导致同一个服务有很多版本；要么服务过度设计，实现了一堆永远用不上的功能。

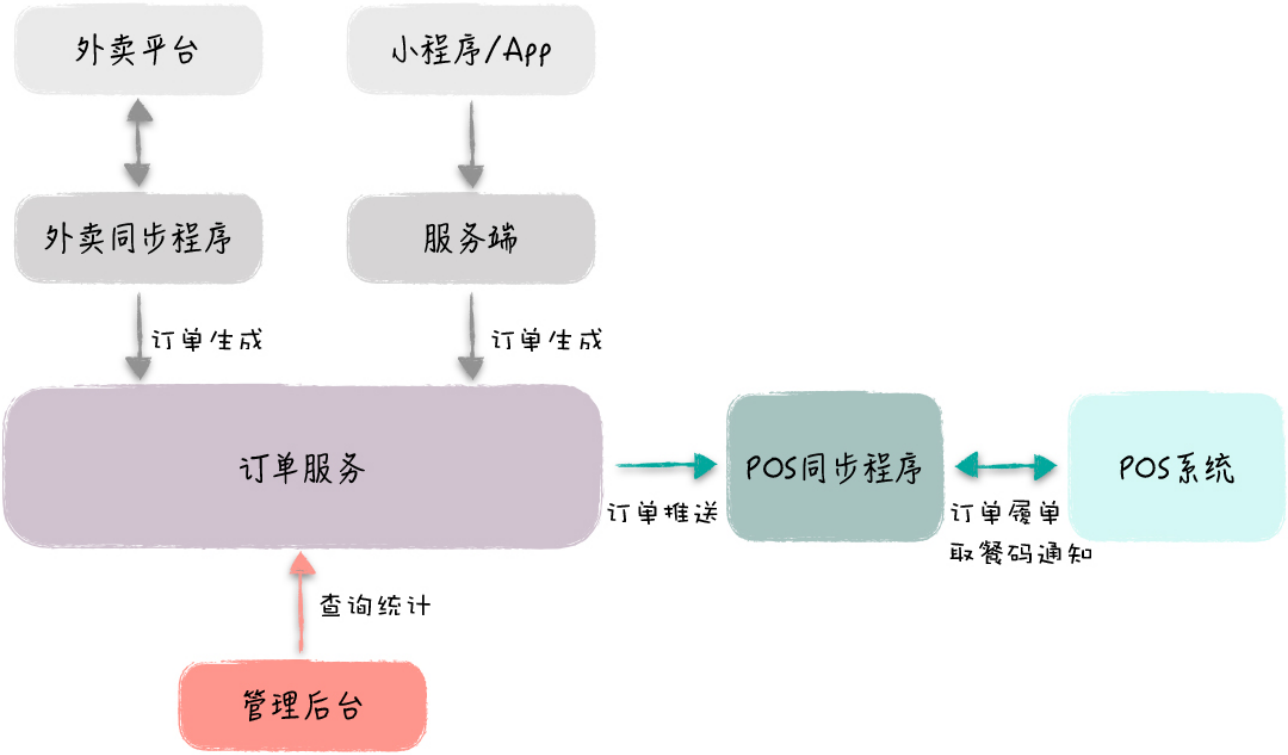
对于落地一个共享服务来说，服务边界的划分和功能的抽象设计是核心。服务边界确定了这个服务应该“做什么”，抽象设计确定了这个服务应该“怎么做”。

接下来，我就以一个**实际的订单服务例子**，为你详细讲解一下要如何重点解决这两个问题。这样你可以通过具体的案例，去深入地理解如何落地共享服务，实现业务能力的复用。

订单业务架构

不同企业的订单业务是不一样的，所以这里我先介绍下这个订单的业务场景。

这是个 O2O（Online To Offline，线上到线下）的交易业务，订单的来源有两个，一个是自有小程序或 App 过来的订单，还有一个是外卖平台过来的订单，然后这些线上的订单会同步到门店的收银系统进行接单和进一步处理。这里我放了一张订单的业务架构图，你可以到文稿中看下：



在这里，订单服务是和 4 个应用直接打交道的：

小程序服务端调用订单服务落地自有线上订单；

外卖同步程序接收三方外卖平台的订单，然后调用订单服务落地订单；

POS 同步程序通过订单服务拉取订单，并推送给商户内部的收银系统；

最后还有一个**订单管理后台**，通过订单服务查询和修改订单。

OK，接下来，我们就具体看下，如何从头开始落地这个订单服务。

订单服务边界划分

首先，我们要确定这个服务的边界，这是进行服务内部设计的前提。划分边界时，你需要对相关的业务场景有充分了解，并且在一定程度上，能够预测潜在的需求。在上一讲，我也和你分享了划分边界一些比较实用的原则和做法，你可以对照学习一下。

根据业务场景的分析，这个订单服务需要负责三个方面的功能。

基本信息管理

首先是订单基本信息管理，主要提供订单基础信息的增删改查功能，包括下单用户、下单商品、收货人、收货地址、收货时间、堂食或外卖、订单状态、取餐码等。

另外，你需要注意的是，这里有多个下单渠道，除了通用的订单信息，每个渠道还有特定的渠道相关信息，比如堂食的订单要有取餐码、外卖的订单要有收货人和收货地址等等，这个都需要在我们的数据模型里给出定义。

订单优惠管理

然后是订单优惠管理功能，这对应的是订单的小票信息，从最开始的商品金额，到最后需要用户实际支付的金额，中间会有一系列的折扣和减免，这些都是属于订单信息的一部分。这些信息我们需要展示给用户看，如果后续要进行订单成本的分摊，也需要用到它。

订单生命周期管理

最后是订单的生命周期管理功能，主要负责管理订单的状态变化。我们知道，从不同下单渠道过来的订单，它的状态变化过程是不一样的；不同行业的订单，它的状态变化过程也是不

同的，所以**订单服务的状态要做到通用**，能够支持各种可能的状态定义和状态转换过程。这个也是订单服务设计的难点，我在后面会重点介绍。

好了，现在我们已经给出了订单服务的功能。**为了更好地定义边界，在实践中，你还需要澄清哪些功能不属于服务**，这样可以避免后续的很多争论。所以在这里，我会进一步给出订单服务不包括的功能，你在划分自己的服务边界时最好也能够明确给出。

第一，作为基础服务，订单服务不主动调用其他服务。

比如说，你想了解订单的用户详情、商品详情等等，这应该由上层应用通过调用相应的服务来实现，然后和订单信息组装在一起，而不是在订单服务内部直接调用其他服务，否则会导致基础服务之间相互依赖，职责模糊。

如果说这个信息整合的场景非常通用，我们可以创建一个在基础服务之上的聚合服务来实现，把订单信息、用户信息、商品信息整合在一起。

第二，订单服务不负责和第三方系统的集成。

在这里，订单需要在我们的订单服务和三方外卖平台，以及收银系统之间进行同步，这些同步功能都是针对第三方系统定制的，不具有通用性。而我们的订单服务作为基础服务，需要具备通用性，因此这些和外部系统对接的功能不会在订单服务的内部实现，而是由额外的同步程序实现。

> 小提示：这些同步程序可以主动调用订单服务，然后再和第三方对接，如果想实时获取订单信息的变化，同步程序可以订阅订单服务的消息通知，第一时间了解订单变化。

第三，订单服务不提供优惠计算或成本分摊逻辑。

订单服务不负责具体的优惠计算，只提供优惠结果的存储和查询，用于还原订单的费用组成。优惠的具体计算过程一般由专门的促销系统负责，成本的分摊一般由后续的财务系统负责。这个我们在上一讲中已经说过，这里就不详细解释了。

最后，该服务不提供履单详情，不负责详细物流信息的存储。

比如说，订单已经发送至上海、订单已经到达某某快递站等等这些信息，订单服务不负责提供这些详细信息，这些都是属于后续履单系统的职责。订单服务可以存储一些外部系统的单据号码，比如配送单号，这样能方便上层应用通过订单记录和配送系统进行关联，获取配送的详细信息。但订单服务只负责存储，不负责数据的进一步解释。

到这里，你可以看到，通过从正反两个方面说明订单服务的职责，我们就得到了一个边界很清晰、职责很聚焦的订单服务边界，所有人对它的职责认识是一致的，尽可能地避免了后续的争论。

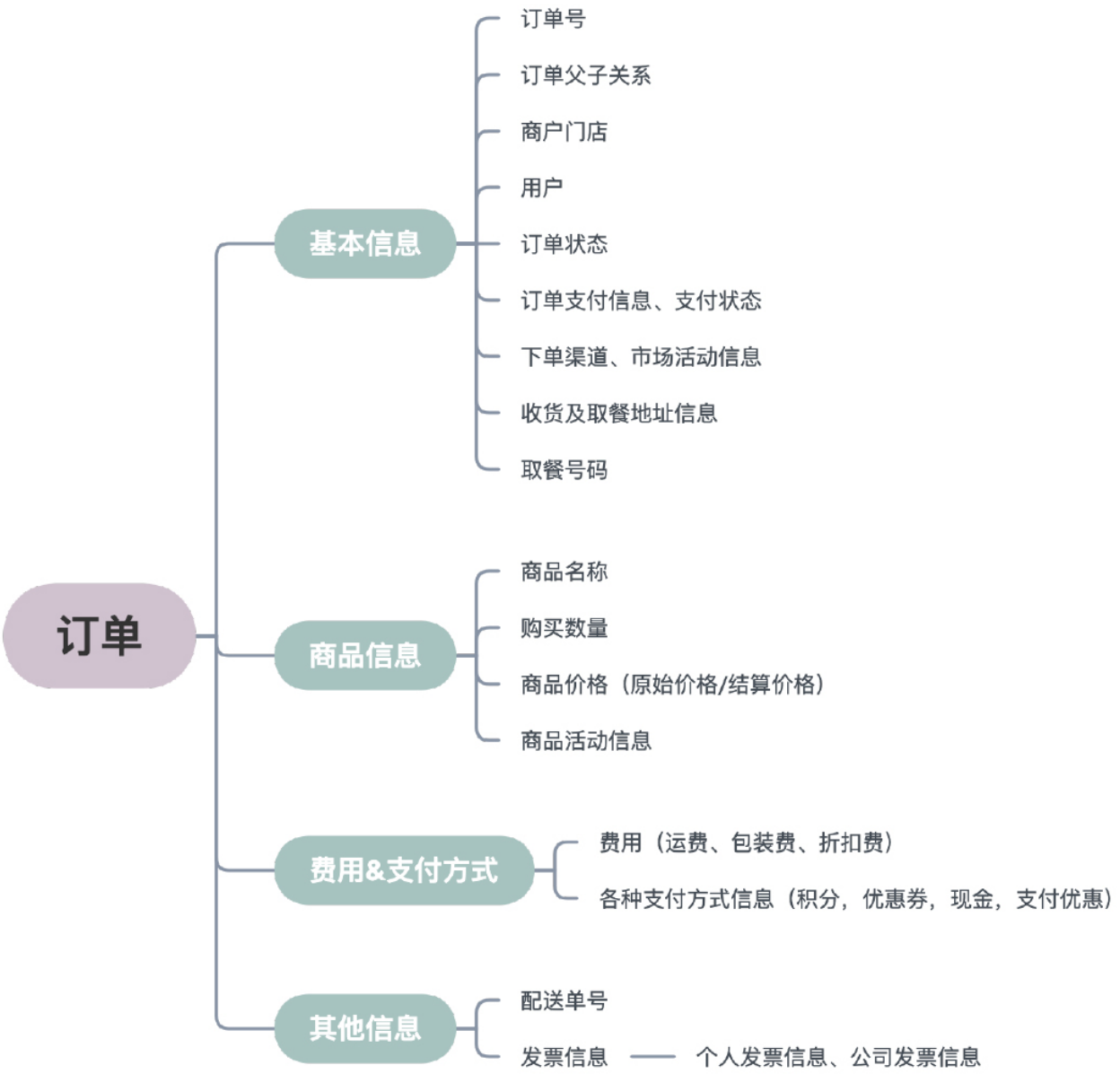
订单服务内部设计

好，确定了这个**订单服务要做什么**之后，接下来，我们要解决的就是**服务内部怎么做**的问题了。

作为共享服务，我们要保证订单服务功能上的通用性，就需要同时对内部数据模型和外部接口进行良好的抽象设计。

订单状态通用化

对于数据模型来说，订单要存储哪些信息，已经比较明确了，具体你可以看下这个图。



但对于如何管理订单的状态，情况就比较复杂了。

我们知道，如果针对一个具体的项目，无论它的订单状态有多么的复杂，我们都可以事先精确地定义出来。**但不同的行业甚至不同的企业，他们对于订单状态管理都是不一样的，订单服务作为一个共享服务，它必须要满足不同项目的订单状态管理。**所以对于如何解决这个问题，这里我有两个思路供你参考。

一个是开放订单状态定义。

在这里，订单服务事先不限定订单有哪些状态，每个项目都可以自己定义有哪些订单状态。服务的调用方可以在接口里传递任意的状态值；订单服务只负责保存状态数据，不负责解释具体的状态，也不负责任何的规则校验，它允许订单从一个状态转换为其他任意的状态。

这样的设计，在理论上可以满足各种状态的定义，满足各种状态之间的变化，但这样做其实有很大的问题。在这里，订单状态是完全由外部负责管理的，上层应用的负担会很重，不但要负责定义有哪些状态，而且还要维护状态的转换规则，一不小心，订单可能从状态 A 非法地变成状态 B，导致业务出问题。

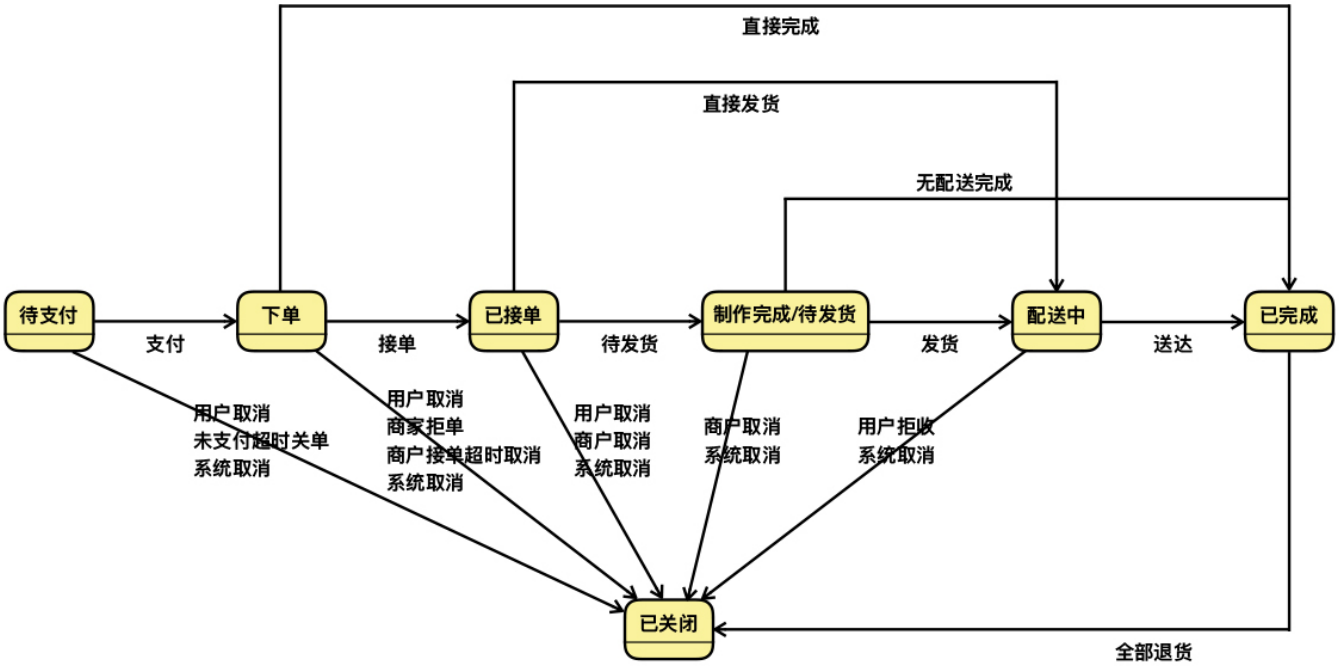
另外一个是应用和服务共同管理状态。

对于订单状态管理，应用和服务各自承担一部分职责，我们看下具体如何实现。

我们知道，无论订单的状态变化是如何的复杂，我们总是可以定义一个订单有哪些基本的状态，包括这些基本状态之间是如何变化的。比如，订单一开始都是用户下单后待支付，支付完成后变成一个有效的订单，然后由商家进行接单，制作完成后进行发货配送等等，订单最终的状态要么是完成，要么是取消。

这些订单的基本状态，我们称之为“主状态”，它们由订单服务负责定义，包括这些主状态之间的转换规则，比如已完成的订单不能变为已取消的订单。主状态的数量是比较有限的，状态之间的变化关系也是比较明确的。

这个主状态，我们对大量现有的业务场景进行总结和抽象，是完全可以定义出来的。在这个订单服务例子里，我们定义了如下图所示的订单状态机，包括有哪些主状态，以及它们的转化关系。



订单除了“主状态”，还有“子状态”。

比如，一个订单处于配送中，实际情况可能是“仓库已发货”，“货已到配送站”，或者是“快递员正在送货中”等等，那么在这些情况中，订单的主状态都是“配送中”，它的子状态就是细化的这几种情况。**子状态有哪些具体的取值，不同的项目是不一样的，这个就开放给各个应用来定义。**

所以，订单服务数据模型里有**两个字段**，其中的主状态由订单服务负责管理，包括主状态之间的变化规则；而子状态由上层应用来定义，管理子状态的变化规则，比如一个配送中的订单，它的子状态可以由“仓库已发货”，变为“快递员正在送货中”。

现在，我们就可以总结下这两种订单状态的设计思路。

第一种方案，我们不对订单状态进行管理，而是把订单的状态作为一个简单的属性存储，只支持订单状态简单的增删改查功能。我们知道，订单状态是订单业务规则的核心体现，这样的订单服务是没有灵魂的，也失去了大部分业务复用的价值。

第二种方案，应用和服务共同管理订单的状态，订单服务抓大放小，通过主状态管理把控住了订单的核心业务规则，同时把子状态开放给应用进行管理，为具体的业务场景提供了灵活性。通过主状态和子状态的结合，订单服务就满足了不同行业、不同企业的订单状态管理需求。

订单服务接口定义

说完了订单的状态管理，接下来，我们从调用方怎么使用服务的角度，来看下订单服务外部接口是如何设计的。

外部系统和服务的交互有**两种方式**，包括同步的服务接口调用和异步的消息通知。

首先是同步的服务接口调用。

为了方便外部调用方，我们在服务接口命名时，一定要规范和统一，接口名字要能够望文生义，方便调用者快速找到所需要的接口。并且，我们还要提供接口具体的请求和响应样例帮助说明。

具体的接口设计规范，我就不具体展开了，每个公司都要有明确的规范要求，这里我就说下常见的查询接口是如何设计的。

一个订单有很多字段，每次调用方要查询的信息可能都不相同，不同字段之间的组合方式有很多，我们不可能一一支持。

那么，我们怎么设计查询接口，来满足各种场景需求呢? 一般来说，我们可以根据返回字段数量的不同，提供三个不同粒度的查询接口来满足多样化的需求。

第一个是**粗粒度接口**，只返回订单最基本的 7-8 个字段，比如订单编号、订单状态、订单金额、下单用户、下单时间等等；第二个是**中粒度接口**，返回订单比较常用的十几个字段；第三个是**细粒度接口**，返回订单的详细信息。

这样，不同的查询需求，就可以根据要返回信息的详细程度，来选择合适的接口，通过这种方式，我们兼顾了要定义的接口数量和查询的性能。

其次是异步的消息通知。

订单服务除了提供同步的接口调用，还针对每次订单信息的变化，提供异步的消息通知，感兴趣的外部系统可以通过接收消息，第一时间感知订单的变化。

按照消息详细程度的不同，订单消息可以分为“胖消息”和“瘦消息”。

顾名思义，**胖消息**包含了尽可能多的字段，但**传输效率低**；**瘦消息**只包含最基本的字段，**传输效率高**。如果外部系统需要更多的信息，它们可以通过进一步调用订单服务的接口来获取。

在这个订单服务的例子里，如果是订单状态的变化，我们只需提供订单号、变化前后的状态即可，因此主要以瘦消息为主；如果是新订单的创建，由于订单的字段比较多，所以使用胖消息，避免外部系统进一步调用订单服务接口。你在实践中，可以根据实际情况，在消息的数据量和消费者处理消息的复杂度之间做平衡。

前面我们说了，订单服务不会主动调用外部系统的接口，这里的异步消息通知，就可以很好地保证外部系统及时感知订单的任何变化，同时避免订单服务和外部系统直接耦合。

总结

要想打造一个可高度复用的共享服务，你需要掌握最核心的两点：**清晰的边界划分、内部的抽象设计。**

今天，我通过一个实际的订单服务例子，帮助你理解如何清晰地定义服务的边界，以及如何通过抽象设计保证服务的通用性。你在实践中，一定要深入分析业务场景，识别真正的挑战在哪里，避免设计的简单化或过度复杂化。

通过今天的讲解，相信你在前一篇理论内容的基础上，对如何打造一个共享服务有了更深入的体会，希望你在工作中能不断地去实践，真正掌握这些技能。

最后，给你留一道思考题：在落地共享服务的时候，你碰到过哪些挑战，都是怎么解决的？

欢迎你在留言区与大家分享你的答案，如果你在学习和实践的过程中，有什么问题或者思考，也欢迎给我留言，我们一起讨论。感谢阅读，我们下期再见。

上一篇　07 | 可复用架构：如何实现高层次的复用？

精选留言 (5)

写留言

Jxin 置顶

2020-03-09

1.碰巧也是做订单系统的。
2.目前接管的订单系统是一个开发了四五年的单体订单系统。
3.没有边界划分，以往的实现，基本就是开发将产品的业务逻辑翻译成代码。现象就是部分拆单策略非常复杂，所有本该外部系统承接的业务逻辑都由订单组完成。这就导致你想维护订单组，支付，营销，会员，商品你都要懂业务。（起步简单，可能这些现在的分...

展开

作者回复: 感觉你这个不仅仅是订单服务，而包含了订单之上打的oms系统，如果一开始各个基础业务划分的好，系统整体就很好调整，下一讲说的是现有系统的服务化改造，也许对你有用

1

探索无止境

2020-03-09

可惜只有22讲，每一讲都有收获

展开

2

中国合伙人

2020-03-09

我想问一下，订单模型不一样，我们经常通过扩展字段存储差异化数据。那在哪解析扩展字段？基础服务负责解析这些差异字段吗？这里要怎么设计不同业务类型订单查询？

作者回复: 如果扩展字段是通用的，订单服务会把这部分逻辑落进去，如果是某个项目专用，订单服务支持落数据就可以

1

Jeff.Smile

2020-03-09

老师好，订单服务在异步通知应用的时候使用的技术我觉得不一定非要mq吧，只要应用把自己的通知url告知订单服务，订单服务负责在信息变动时进行推送就可以了，这种更简单一些，不过可能可用性不是很高而已。

作者回复: 你这种说的是url回调，这也是一种方式，在支付里用的比较多，收到第三方平台支付成功后，支付服务回调应用系统提供的URL完成通知。
这种方式比同步调用耦合性低，比消息通知耦合性高一些，并且调用不成功，要重试，服务要做的事情要更多一些。
如果针对不特定的接收者，消息通知更合适，解耦更彻底一些。

2 1

tt
2020-03-09

可复用的两个点：清晰的边界划分和抽象的内部设计。

1、对于边界划分。在服务的设计之处，总是会现有一个头脑风暴、各方需求裹挟的发散过程，然后随着设计的进行，必需经过收敛，所以我觉得界定服务不做什么更有实践意义。同时，服务功能越单一，越利于复用。...
展开

1