

04 | 可扩展架构案例（一）：电商平台架构是如何演变的？

2020-02-28 王庆友

架构实战案例解析

[进入课程 >](#)



讲述：王庆友

时长 18:31 大小 14.84M



你好，我是王庆友。

上一讲中，我们介绍了如何打造一个可扩展的架构。今天，我就针对最近十几年电商平台的架构变化过程，来具体说明下，为了支持业务的快速发展，架构是如何一步步演进的。

从 2003 年淘宝上线开始，国内电商平台经历了高速的发展，在这个过程中，系统遇到了很多的挑战，比如说：

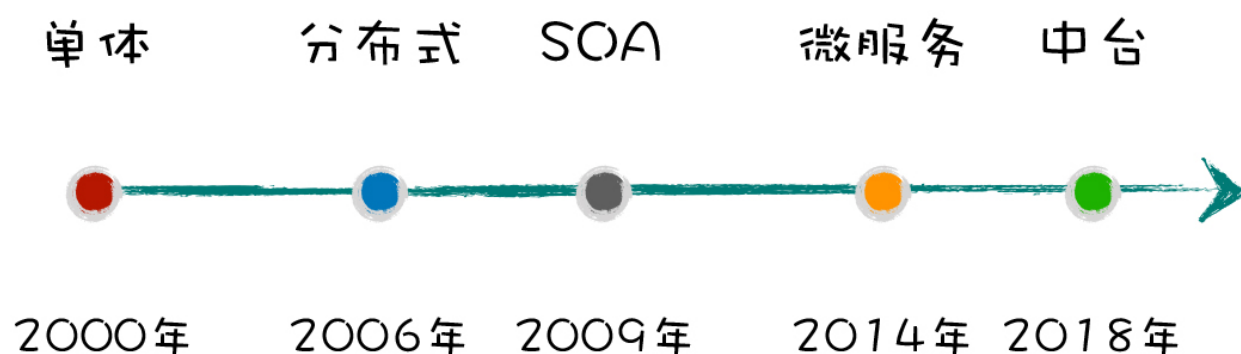


如何针对当前的业务现状，选择合适的架构呢？

如何在业务发展过程中，升级改造架构，并保证系统的平滑过渡呢？

接下来，我会结合自己的工作实践，和你一起探讨架构的演变历程，你可以从中了解到各种架构的优缺点和适用性，然后在实际工作中选择合适的架构。

这里，我总结了国内电商平台架构发展的大致过程，你可以结合图片参考下。



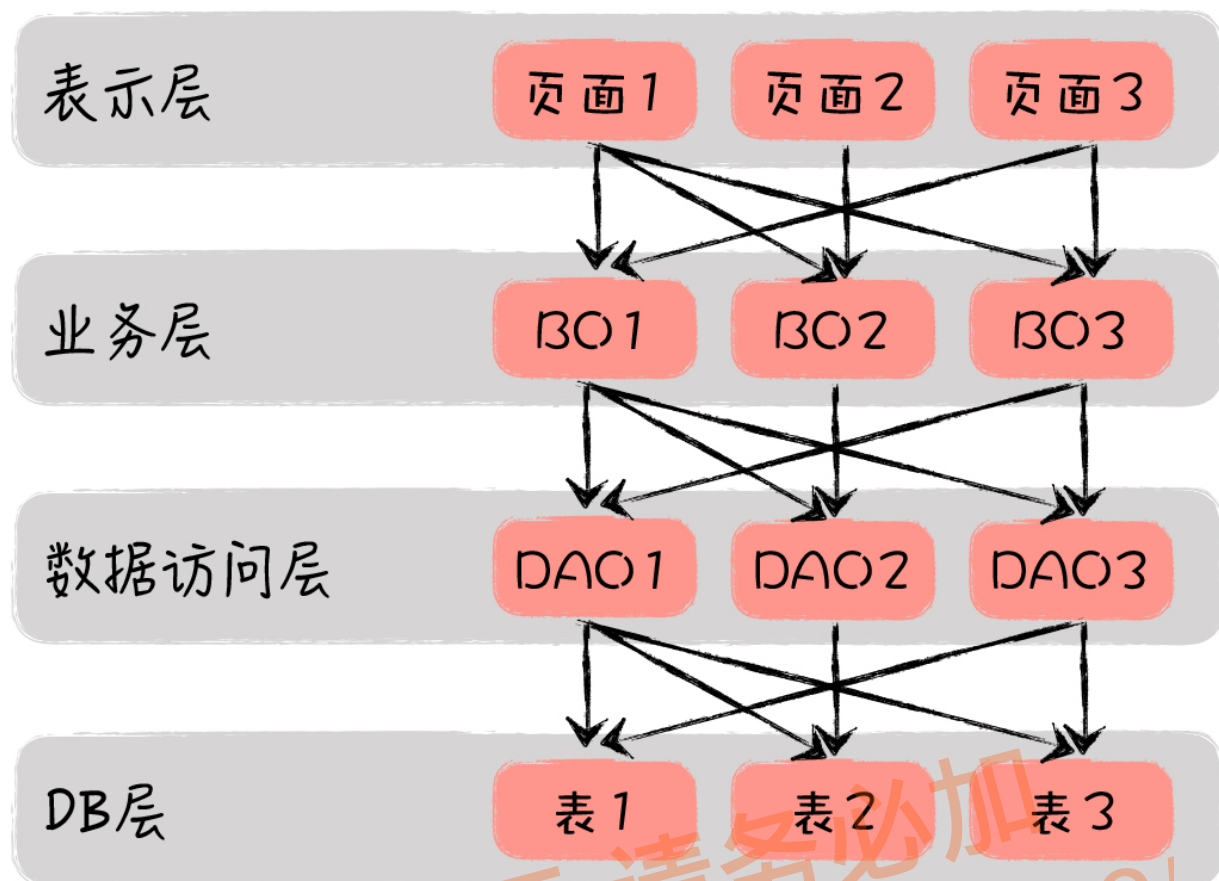
电商平台架构发展过程

我们可以看到，从最初的单体架构到最新的中台架构，架构的可扩展性越来越强，这些都是系统不断适应业务复杂化的结果。下面，我就结合电商业务的变化，按照顺序和你介绍下各个架构。因为篇幅的原因，对于中台架构，我会放在后面的文章里重点介绍。

单体架构

在单体架构中，只有一个应用，所有代码跑在一个进程，所有的表放在一个 DB 里。第一代电商平台都是单体架构，比如说淘宝，在最初的 3 年，它的系统就是一个巨大的单体应用。

单体应用内部一般采用分层结构，从上到下，一般分为表示层、业务层、数据访问层、DB 层。表示层负责用户体验，业务层负责业务逻辑，数据访问层负责 DB 的数据存取。



我们可以看到，各个层的职责，正好对应业务处理的不同阶段，所以，单体架构在水平方向上，通过层次化的划分，降低了业务的深度复杂性（所谓的业务深度，指的是业务流程从开始到结束的长度）。

不过在垂直方向上，单体应用缺乏清晰的边界，上下层模块之间是多对多的网状依赖关系，比如业务层的某个模块（上图中 BO1），可能调用数据访问层的所有模块（DAO1~3），同样的道理，数据访问层的某个模块，也可能被业务层的所有业务模块给调用。

所以，单体架构中的模块只是在逻辑上独立，并没有在物理上严格分开，导致系统在落地时，模块的职责和边界划分比较随意，相应地，模块之间的依赖关系也比较模糊。所以，在单体架构中，模块结构是否合理，很大程度上依赖于开发者的个人水平。

在电商发展的初期，业务并不复杂，比如前台的首页、搜索页、详情页、结算页等，页面的功能都比较简单，可以放在一个应用里处理，这样，使用单体架构就可以快速落地系统。但当业务开始变得复杂时，每个页面都发展为一个独立的业务体系，比如说首页，它原先展示相对固定的内容，现在发展为一个动态的千人千面系统。

这样一来，业务的复杂度急剧上升，模块的数量也大幅度增加了，我们就很难在单体架构里，通过构建一个清晰的模块体系来支持系统的扩展。而且，所有代码放在一个代码库里管理，如果多团队并行开发的话，很容易发生代码冲突，这样也难以满足系统的快速扩展。

举个例子，07 年的时候，eBay 网站总体上也是一个单体应用，它的核心工程有数百万行代码，由于代码合并和编译非常复杂，他们甚至有专门的团队负责代码合并，有专门的团队负责编译脚本开发，另外还有一套复杂的火车模型，来协调不同团队之间的并行开发和上线。

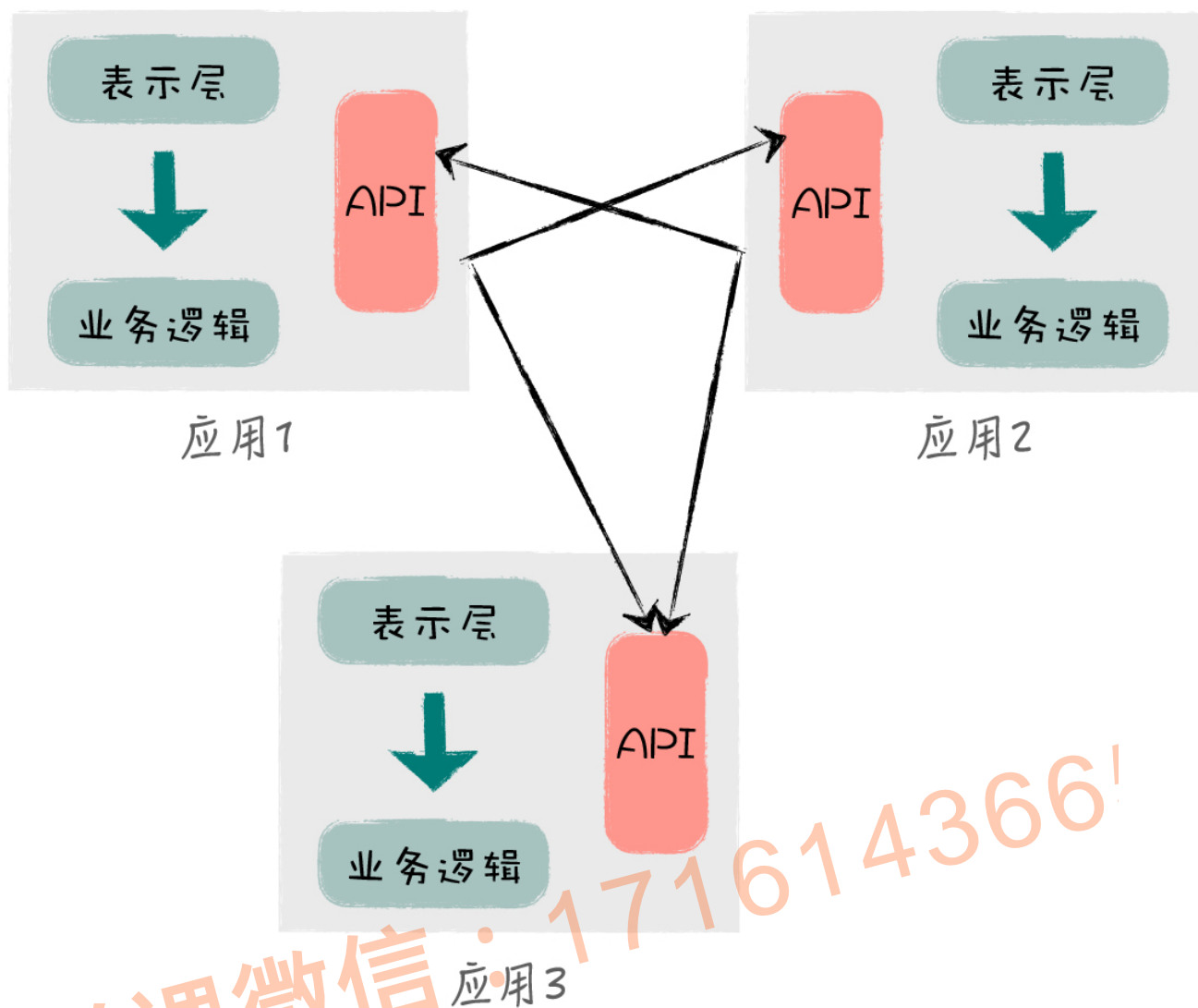
所以，当业务系统的体量变大时，单体架构的弊端就充分暴露出来了，我们就需要对系统进行有效的拆分，比如把首页、搜索页、详情页、结算页拆成一个个独立的应用，分别进行管理。于是，分布式架构就应运而生了。

分布式架构

分布式架构，简单来说就是系统由多个独立的应用组成，它们互相协作，成为一个整体。

分布式架构包括了多个应用，每个应用分别负责不同的业务线，当一个应用需要另一个应用的功能时，会通过 API 接口进行调用。在分布式架构中，API 接口属于应用的一部分，它和表示层共享底层的业务逻辑，你可以认为，API 相当于应用在实现本身业务的基础上，开了个小窗口，给外部使用。

关于分布式的具体架构，你可以参考下图：



你可以看到，分布式架构在单体应用的基础上，进一步对系统按照业务线，进行了业务广度上的切分（所谓业务广度，指的是不同业务线的数量），这样就把一个大系统的业务复杂度，分割成多个小业务的复杂度，从而降低了整体的复杂度。通过拆分后，各个应用之间的耦合度低，就可以很好地支持多团队的并行开发。

但分布式架构也有局限性，作为应用的开发者，除了要满足自身业务的需求之外，同时还需要考虑外部业务的需求，这两部分经常会打架。比如，由于自身业务的需求，引起底层的业务逻辑修改，这时会同时影响 API 接口功能，导致其他业务受影响；同样的道理，外部业务需求过来，需要 API 接口做调整，即使不影响底层业务逻辑，也会导致整个应用重新部署，影响自身业务的稳定性。

另外，在分布式架构下，每个应用都是从头到尾，自搭一套完整的体系，导致业务之间重复造轮子，造成资源浪费。举个例子，在 2008 年，淘宝还没有开始服务化改造之前，不同业务线的用户、商品、订单逻辑非常类似，导致了整个系统有超过 1/3 的核心代码重复。

所以，你可以发现，**分布式架构适用于业务相关性低、耦合少的业务系统**。举个例子，企业内部的管理系统，分别服务于不同的职能部门，比如财务系统和 HR 系统，就比较适合按照分布式架构去落地。

但在电商场景下，业务都是围绕交易展开的，各个页面（应用）都需要和商品、用户、订单、库存打交道，对于这样业务相互依赖、应用之间需要紧密协作的场景，在系统架构方面，是否有更好的手段，可以更高效地集成这些应用呢？

答案是有的，SOA 架构就可以有效地解决这个问题。接下来，我们就具体了解下。

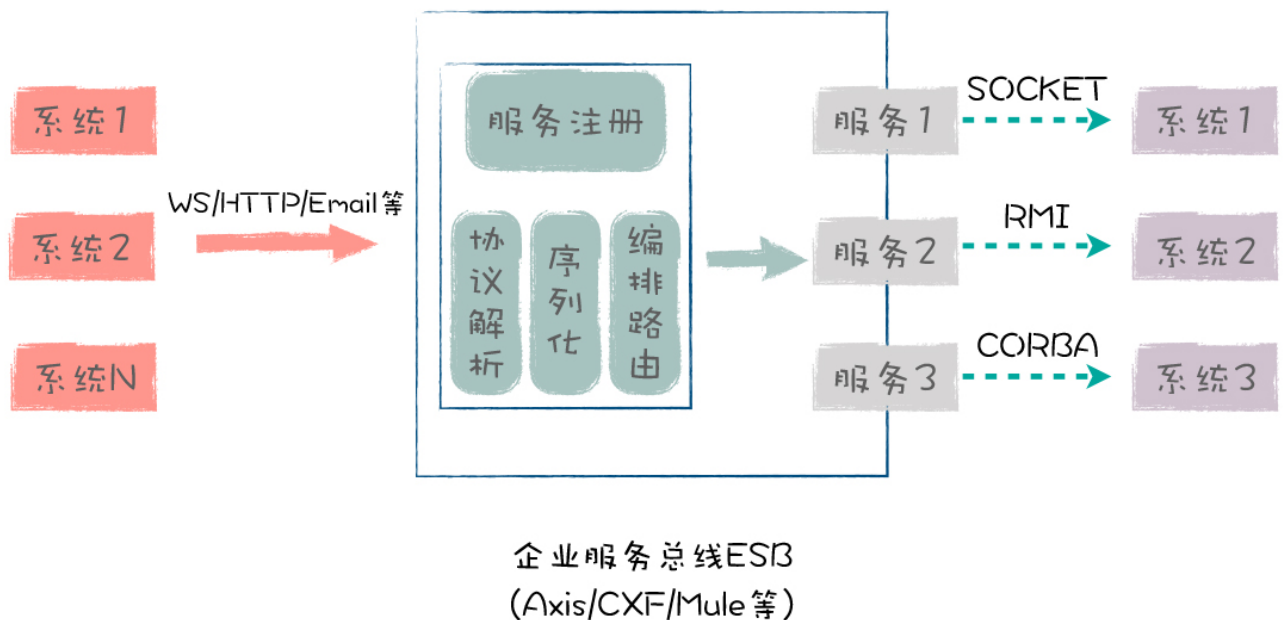
SOA 架构

SOA 架构（Service Oriented Architecture）是一种面向服务的架构，它的发展经历了两个阶段：传统的 SOA 架构，它解决的是企业内部大量异构系统集成的问题；新的 SOA 架构，它解决的是系统重复建设的问题。下面我就来和你详细介绍一下。

从 2000 年开始，很多传统企业进入了信息化建设高潮，先后采购了很多系统，比如 ERP、OA、CRM 等等。这些系统都是由不同的供应商提供的，落地后，就形成了很多的信息孤岛。随着业务的发展，企业需要打通这些不同的系统，那么问题来了，这些系统使用不同的技术，事先也没有提供开放接口给外部使用，那我们如何才能有效地集成这些系统呢？

解决的办法是，每个系统首先把外部需要的能力，封装为一个个粗粒度的接口，打包成一个独立的服务；然后，外部系统通过这个服务访问系统内部，解决不同系统相互集成的问题。经过这样的改造，系统最后就变成了一个面向服务的 SOA 架构。

这就是一个传统的 SOA 架构，如下图所示：



你可以看到，在 SOA 架构中，每个服务都对应一个现有的系统，所有这些服务都部署在一个中心化的平台上，我们称之为企业服务总线 ESB（Enterprise Service Bus），ESB 负责管理所有调用过程的技术复杂性，包括服务的注册和路由、各种通信协议的支持等等。

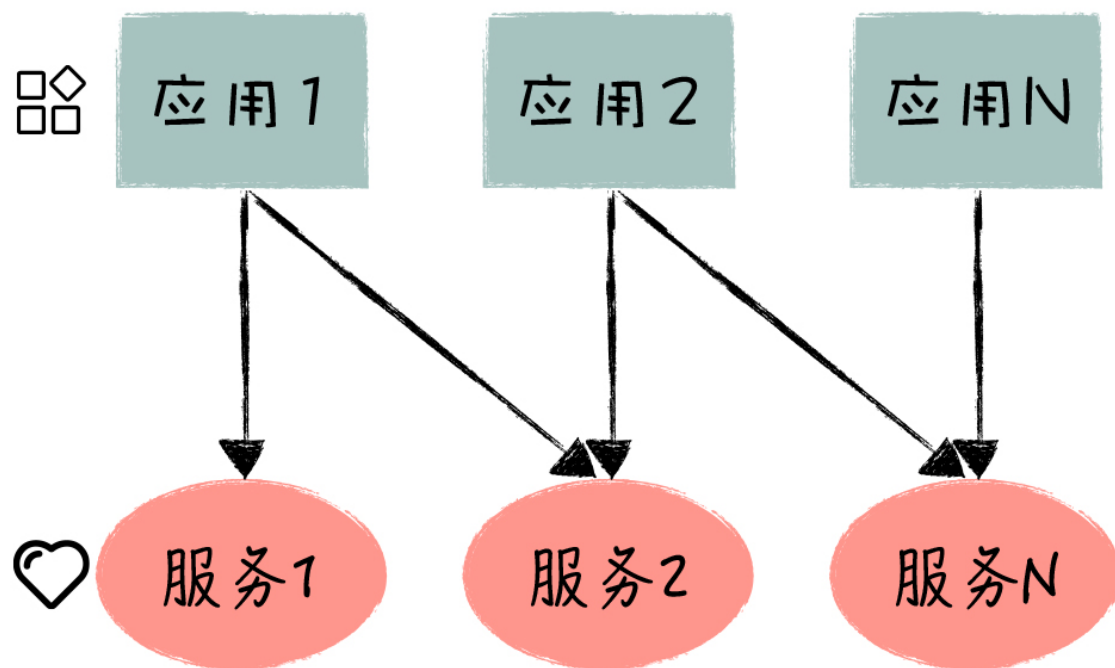
比如说，09 年的时候，eBay 就基于 Axis 2，开发了自己的 SOA 框架，让各个系统通过提供标准的服务，来满足外部调用需求。比如后台搜索系统，本身是 C++ 开发的，但是它通过提供 Java 服务，封装常见的搜索功能，就方便了其他系统（大多是 Java）和搜索系统进行集成。

以上讲的是传统 SOA 架构，它主要用于解决遗留系统的集成问题。而新的 SOA 架构，它利用服务共享的思想，解决系统的重复开发问题。

举个淘宝的例子，淘宝的系统基本是自建的，系统相互打通的问题不大。但经过一段时间的自然生长，系统重复建设的问题很突出，前面也提到，有超过 1/3 的核心代码重复。针对这种情况，我们就可以通过服务化手段，把通用的逻辑和数据从各个业务系统里抽取出来，封装成独立的服务，提供给所有业务进行共享。

基于这个思路，淘宝花了 2~3 年时间，先后落地了用户、商品、订单、库存、店铺、营销等服务，搭建了共享服务体系。通过共享，淘宝不仅提升了开发效率和质量，也加强了系统的扩展能力。

新的 SOA 架构如下图所示：



新SOA架构图

所以我们可以看到，相对于分布式架构，SOA 架构给系统的扩展带来了一系列的好处：

首先，它通过服务化思想，提供更好的业务封装性，并通过标准技术，能更友好地对外输出业务能力；

其次，SOA 服务不依附于某个具体应用，它可以独立地部署和扩展，这样避免了直接影响现有的系统；

最后，服务通过封装通用的业务逻辑，可以供所有应用共享，解决了重复造轮子的问题。

不过，虽然 SOA 服务化的思想很好，但在系统实现上比较重，落地比较困难。**那有没有更轻量级的架构，使得系统各个部分更容易构建和相互协作呢？**

这时候，微服务架构便悄悄地登场了。

微服务架构

关于微服务，大家都不陌生，但究竟什么是微服务，每个人的理解可能都不一样。接下来，我就基于自己的服务化实践，和你分享我的看法。

微服务概念的提出，一开始是用来和单体架构做区分的。我们知道，单体架构和分布式架构，实际上都是围绕一个大的业务线来构建应用，当业务变得复杂时，就无法做到模块边界和依赖关系的清晰划分，模块局部的调整往往会导致系统整体的调整，使得系统很难扩展。

而微服务围绕更小的业务单元构建独立的应用。

比如说，一个飞机航班预订系统，我们可以把它划分为预订航班、时间表查询、计算票价、分配座位等几个小应用（微服务）来落地。那么经过划分后，每个小应用都比较简单，只关注于一个业务功能即可。

这里要注意的是，每个微服务，都是负责端到端的业务，包括前端的 UI 展现部分和后端业务逻辑。微服务的团队成员可能包括产品、开发、测试、运维等人员，由这个小团队负责应用的整个生命周期管理。

因此，从一定程度上说，微服务叫做微应用，或者说微产品，更合适一点，你也可以认为微服务架构是拆分得更细的分布式架构。

另外，微服务强调围绕业务，进行清晰的业务和数据边界划分，并通过良好定义的接口输出业务能力，这和 SOA 架构里的服务有点类似。**但两者不同的地方在于，微服务是去中心化的，不需要 SOA 架构中 ESB 的集中管理方式。**

一方面，**微服务强调所谓的哑管道**，即客户端可以通过 HTTP 等简单的技术手段，访问微服务，避免重的通信协议和数据编码支持。另一方面，**微服务强调智能终端**，所有的业务逻辑包含在微服务内部，不需要额外的中间层提供业务规则处理。

这样子，微服务提供方可以自由地选择语言和工具，来落地微服务，服务的部署和维护上也更灵活，从这个意义上来说，你也可以认为微服务是轻量级的 SOA 服务。

所以说，微服务兼有应用和服务的特征，你可以把微服务理解为：

微服务 = 小应用 + 小服务。

以上就是微服务架构设计的初衷，但在实践中，我们更多地把微服务当做一个小服务，而不是一个端到端的小应用，那么为什么会这样呢？这里有几个原因。

首先，我们很难把一个大系统，按照端到端业务的方式，拆分为一个个应用；而拆分为服务是比较灵活的，我们可以把系统核心的业务逻辑和数据封装成服务，其它部分还是以应用的方式落地。另一方面，微服务要求团队人员跨多个职能，构建独立的小团队，来负责服务完整的生命周期，这就需把现有的职能团队打散后重组，这种人员组织的调整实际上也很难落地。

我们可以看到，微服务强调围绕端到端的小业务功能，通过组建跨职能的团队，来进行落地，这只是一种理想化的做法。所以，**在实践中，我们往往弱化微服务的小应用定位，然后扩大化微服务小服务的定位，我们不再强调端到端的业务封装，而是可以有各种类型的微服务。**

比如说，封装底层基础业务的是共享微服务，封装流程的是聚合微服务，封装具体业务场景的服务端是应用微服务，封装基础中间件（如 Redis 缓存、消息推送）的是系统微服务。当然，这些服务在具体落地时，我们还是采取去中心化的机制，使用轻量级的通讯框架，最后把它们打造成一个个技术上轻量级的、功能职责上细分的微服务。

所以，基于这样的思路，微服务就很容易构建，同时，也像水电煤一样，容易被我们使用。然后，我们在这个基础上组装微服务，像搭积木一样搭建系统，这样的系统更具弹性，更容易扩展。

值得注意的是，我们需要对服务依赖关系进行有效的管理，打造一个有序的微服务体系。否则的话，东一个服务，西一个服务，这样会让系统变得碎片化，难以维护和扩展。

所以我这里也放了一张图，来帮助你理解，一个有序的层次化微服务体系大致是什么样子的。

业务服务

搜索服务

下单服务

采购服务

团购服务

共享服务

积分
服务

抵用券
服务

发票
服务

促销
服务

评论
服务

产品
服务

用户
服务

订单
服务

库存
价格

支付
服务

系统服务

短信
服务

邮件
服务

缓存
服务

消息
服务

存储
服务

总结

今天，我与你分享了电商平台架构的发展过程，从单体架构到分布式架构，再到 SOA 架构和微服务架构，每种架构都针对前一种架构的缺点做了改进，架构的扩展性也变得越来越好，可以满足更高的业务复杂性要求。

但值得注意的是，每种架构都有两面性，既有优点，又有缺点，在实际系统中，这些架构也都是并存的。**架构没有最好，只有最合适的。**我们做架构设计时，一定要根据当前业务的特点，选择合适的架构。

通过今天的分享，相信你对架构的扩展性有了更深入的理解，也能够根据公司的业务现状，进行更合理的架构选型了。

最后，给你留个思考题：现在人人都在落地微服务，你在这方面有什么经验和教训吗？

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。感谢收听，我们下期再见。

点击参与 

20年架构老兵邀你一起 打卡，带你进阶资深架构师



扫一扫参与小程序话题



新版升级：点击「 请朋友读」，20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 03 | 可扩展架构：如何打造一个善变的柔性系统？

下一篇 05 | 可扩展架构案例（二）：App服务端架构是如何升级的？


精选留言 (8)

 写留言



知莫问 置顶

2020-02-29

分布式系统解决的是早期自建系统之间的通信，SOA解决的是企业采购的系统之间的通信，是两个不同的应用方向；分布式系统是各个系统内部提供统一接口，SOA是在系统外部重新封装一套统一接口。但是，令人疑惑的是，这些系统都是不同公司使用不同语言开发的，要怎么才能把它们封装起来呢？这就需要使用十八般武艺了。有些系统自己提供 WebService 接口或者 HTTP 接口，这个好封装。有些系统并没有提供任何接口，那就可...
展开 

作者回复: 十几年前，两个术语很流行，EAI (enterprise application intergratioin) 和EII(enterprise information intergratioin), 都是为了各个系统打通。

现在互联网企业盖过传统企业，应用都是自己开发，集成没问题，更多考虑系统扩展和复用，能够快。



tt

2020-02-28

架构的演进，真是一个螺旋式上升的过程。

本文的第一张图和最后一张图，从图的拓扑结构上，没有本质的区别。但：

1、图中的连线已经从进程内的函数调用，变为了通过轻量级的通讯框架的服务间的调用...

展开 ∨



孙同学

孙同学

2020-02-28

<https://www.processon.com/view/link/5e51378ce4b0c037b5f9d1e3> 看了两遍整理好的，没有实际经验，感觉理解的很浅。

展开 ∨



Jeff.Smile

2020-02-28

“但在电商场景下，业务都是围绕交易展开的，各个页面（应用）都需要和商品、用户、订单、库存打交道，对于这样业务相互依赖、应用之间需要紧密协作的场景，在系统架构方面，是否有更好的手段，可以更高效地集成这些应用呢？

答案是有的，SOA 架构就可以有效地解决这个问题。”

展开 ∨

作者回复: 通过api可以互通，但在业务关系紧密的情况下。每个应用都会有部分的商品，用户，订单，库存逻辑，职责碎片化，应用之间耦合紧密，相互影响。如果用服务化的思路把商品，用户等单独封装起来，然后供各个业务共享，开发效率和质量都更好。

SOA架构也是分布式的一种，当然微服务也是分布式。



翌

2020-03-01

"架构没有最好，只有最合适"，这才是精髓啊，感觉很多企业动不动就要上微服务，也不管自己的体量是否需要微服务，有些小的公司服务体量也小，那么其实单体应用就可以，完全能撑得起业务，如果强行上微服务，那么往往就浪费了很多人力成本，技术成本，还

达不到好的效果，所以我的理解是要看微服务是否适合自己，能否帮我们解决最根本的问题，再去衡量如何上微服务，怎么上微服务。

展开 ▾



李

2020-03-01

端对端这个词怎么理解？是指一条线闭环吗？

展开 ▾

作者回复: 一个业务处理的起点到终点



Rory

2020-02-28

已经反复读了这些章节几遍了，也尝试用这些原则和方法分析现有产品，很有收获。希望更新再快一些）：

展开 ▾



探索无止境

2020-02-28

我理解的微服务其实就是分布式架构的一种，只是有两点改进：

- 1，服务粒度拆分更细，方便上层应用复用
- 2，服务间的通信协议更简单便捷，比如http方式

