07 | 可复用架构：如何实现高层次的复用？

2020-03-06 王庆友

架构实战案例解析

进入课程 >

讲述：王庆友

时长 14:01　大小 12.85M

你好，我是王庆友。在前面几讲中，我们讨论了如何打造一个可扩展的架构，相信你对架构的可扩展有了一定的了解，而架构还有一个非常重要的目标，那就是可复用。所以从今天开始，我就来和你聊一聊，如何打造可复用的架构。

作为开发人员，你对复用这个概念一定不陌生。在开发过程中，我们把系统中通用的代码逻辑抽取出来，变成公共方法或公共类，然后在多个地方调用，这就是最简单的技术上的复用。

但一开始，我们不会过多地考虑复用，当一个新项目过来，我们会选择最直接的方式来实现，结果往往是欲速而不达，比如说：

好不容易搞定了一个项目，接着又有新的类似项目过来，我们又要从头再来；

项目的代码是定制的，项目结束后，系统维护的噩梦刚刚开始。

如果项目缺乏沉淀，每个项目都是全新的开始，出现这些情况，一点都不意外。而要想解决这个问题，我们一开始就要考虑系统的复用性。

复用，它可以让我们站在巨人的肩膀上，基于现有的成果，快速落地一个新系统。

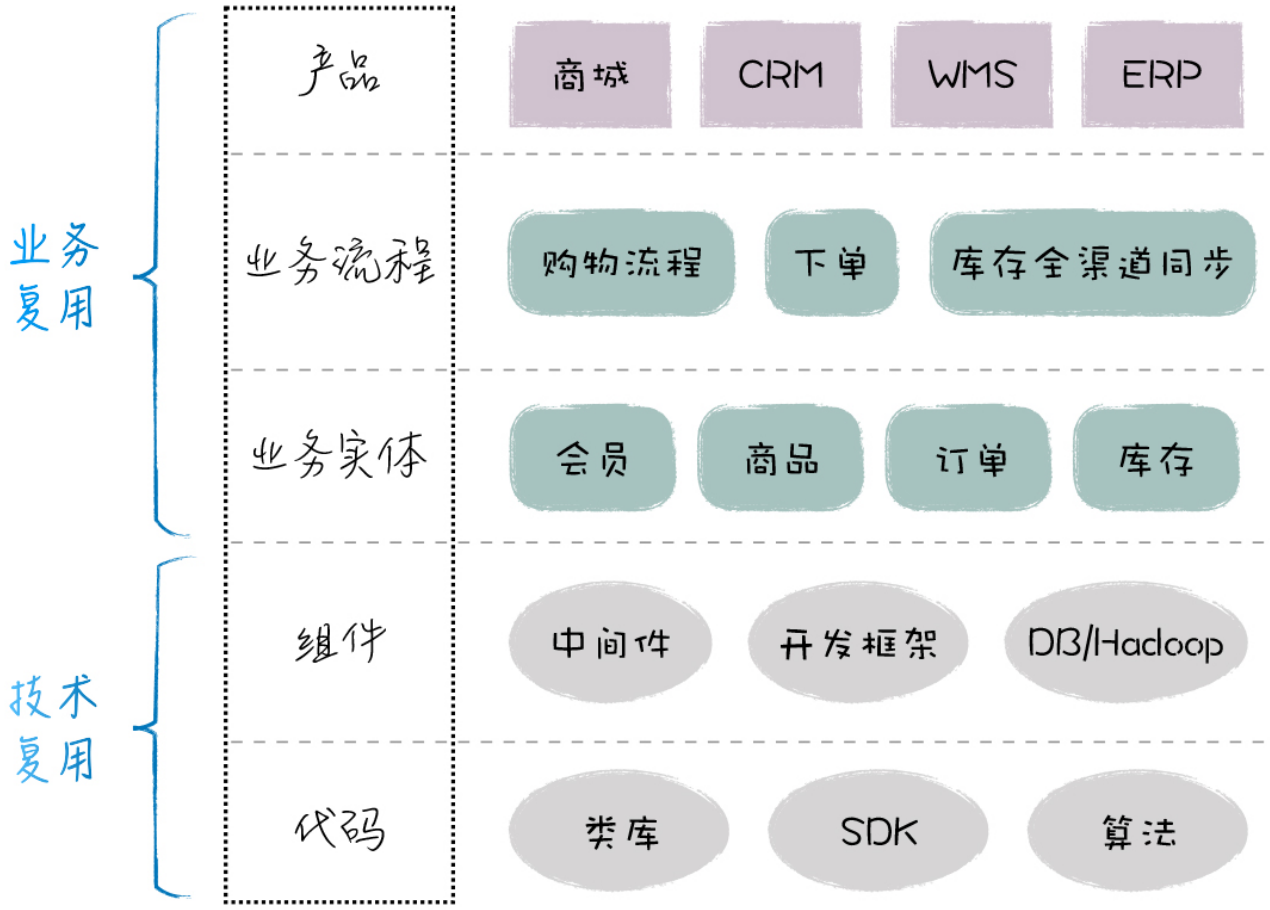
那么，我们在做架构设计时，如何实现系统的高可复用呢？

今天，我就针对复用这个话题，首先和你介绍一下，复用具体都有哪些形式；然后，我会针对最有价值的业务复用，带你了解如何划分服务的边界，让你能够在工作中，设计一个可以高度复用的系统。

复用的分类

复用有多种形式，它可以分为技术复用和业务复用两大类。**技术复用**包括代码复用和技术组件复用；**业务复用**包括业务实体复用、业务流程复用和产品复用。

从复用的程度来看，从高到低，我们可以依次划分为产品复用 > 业务流程复用 > 业务实体复用 > 组件复用 > 代码复用。



接下来，我就按照复用度从低到高，对这些复用方式进行一一分析，帮助你更好地理解架构的可复用性。

技术复用

首先是**代码级复用**，这部分应该是你最熟悉的了。这里包括你自己打包的类库，第三方提供的 SDK，还有各种算法封装等。我们的代码可以直接调用它们，物理上也和我们的应用打包在一起，运行在同一个进程里。代码级复用是最低层次的复用，你可以把它当作你自己源代码的一部分。

再往上，是**技术组件复用**。这些组件有我们自己封装的，更多的是大量开源的中间件，比如 Redis、MQ、Dubbo 等；组件也包括各种开发框架，比如 Spring Cloud。这些基础组件技术复杂度很高，它们的存在，极大地简化了我们的开发工作。

值得注意的是，代码级复用和技术组件复用都属于工具层面，它们的好处是在很多地方都可以用，但和业务场景隔得有点远，不直接对应业务功能，因此复用的价值相对比较低。

业务复用

我们知道，系统最终是为业务而服务的，如果能够实现直接的业务复用，那系统开发的效率就更高。在前面的课程中，我们讨论架构的演进过程时，很多地方谈到了业务能力的复用，比如说，微服务强调单个业务实体的封装和复用，而中台进一步实现了企业级业务能力的复用。

所以接下来，我们就从比较简单的业务实体复用开始说起。

业务实体复用针对细分的业务领域，比如订单、商品、用户等领域。它对各个业务领域的数据和业务规则进行封装，将它变成上层应用系统可以直接使用的业务组件。

业务流程的复用针对的是业务场景，它可以把多个业务实体串起来，完成一个端到端的任务。比如说，下单流程需要访问会员、商品、订单、库存等多个业务，如果我们把这些调用逻辑封装为一个下单流程服务，那下单页面就可以调用这个流程服务来完成下单，而不需要去深入了解下单的具体过程。相比单个的业务实体复用，业务流程的复用程度更高，业务价值也更大。

最高层次的复用是对整个系统的复用，比如说一个 SaaS 系统（Software-as-a-Service），它在内部做了各种通用化设计，允许我们通过各种参数配置，得到我们想要的功能；或者说一个 PaaS（Platform-as-a-Service）平台，它会提供可编程的插件化支持，允许我们“嵌入”外部代码，实现想要的功能。

这种产品级的复用，它的复用程度无疑是最高的。这样的系统，在落地的时候，它无需核心的开发团队进行开发，只由外围的实施团队负责就可以了，这样，一个项目的上线就能简化为一次快速的实施，不但上线周期短，系统也更稳定。

当然，实现这样的复用，难度也是很大的，你既要对所在行业的业务有很全面的理解，又要有很强的抽象设计能力。这类系统中，比较典型的有 Salesforce 的 CRM 系统和 SAP 的 ERP 系统。

现在，我们先对复用做个总结。**从技术复用到业务复用，越往上，复用程度越高，复用产生的价值也越大，但实现起来也越复杂，它能复用的场景就越有限。**在实际工作中，技术层面上的复用相对比较简单，我们对这部分的认知也最多，而且由于开源的普及，现在有丰富的中间件让我们选择，我们可以基于它们，逐步构建适合自己的技术体系。

但如果我们能进一步打造业务中间件，并在这个基础上，形成业务平台，这样，我们就能实现更高的业务级复用，可以更高效地支持系统的快速落地。

而在实现业务组件化和平台化的过程中，首要的问题就是基础服务边界的划分。边界划分决定了服务的粒度和职责，在实际工作中，也是非常困扰我们和有争议的地方。

接下来，我就针对基础服务边界的划分，和你分享我自己在项目开发的过程中，总结的一些实用的原则和做法。

基础服务边界划分

服务边界划分要解决“我是谁”的问题，它实现了服务和周边环境的清晰切割。

我们都知道，服务包含了业务数据和业务规则，并提供接口给外部访问，其中，接口是服务的对外视图，它封装了服务的业务数据和规则。

所以从边界划分的角度来看，我们就是要确定哪些数据属于这个服务，哪些接口功能由这个服务提供。这里，我总结了 3 个基础服务边界划分的原则，供你设计时做参考。

首先，是服务的完整性原则

你在划分服务的边界时，需要确保服务内部数据的完整性。

举个例子，一个商品服务的数据模型，不仅要有商品基本信息，比如商品名称、价格、分类、图片、描述等；还需要包含商品的扩展信息，如商品的各种属性、商品标签等；最后还要包含各种复杂商品类型的定义，比如组合商品、套餐商品、多规格商品等。

另外，你还要保证服务功能的完整性。对于服务使用者来说，他们是以业务的角度看服务，而不是纯粹的数据角度。比如一个套餐商品，在服务内部，它是多个单品的复杂组合，但从服务调用者的角度来看，它就是一个商品。

那现在问题来了，对于套餐的价格，商品服务是给出一个最终价格呢？还是给出各个单品的价格，然后让调用方自己算最终价格呢？我们知道，套餐的价格不是各个单品价格累加的结果，它包含了一定的优惠，如果它的价格由服务调用方来算，这会导致商品的部分业务规则游离于服务外面，破坏了商品服务的功能完整性。

在实践中，有些服务只是存储基础数据，然后提供简单的增删改查功能，这样一来，服务只是一个简单的 DAO，变成了数据访问通道。这样的服务，它的价值就很有限，也容易被服务调用方质疑。因此，我们要尽可能在服务内部封装完整的业务规则，对外提供完整的业务语义，最大程度地简化服务的使用。

所以，当你在划分服务边界时，要保证服务数据完整、功能全面，这样才能支撑一个完整的业务领域。

其次，是服务的一致性原则

也就是说，服务的数据和职责要一致，谁拥有信息，谁就负责提供相应的功能。

服务内部的业务逻辑要尽量依赖内部数据，而不是接口输入的数据，否则会造成数据和业务规则的脱节（一个在外面，一个在里面），如果服务对外部的依赖性很强，就无法提供稳定的能力了。

很多时候，我们对一个功能到底划分到哪个服务，有很大的争议。这时，我们可以结合这个功能所依赖的数据来判断，如果功能所需要的大部分数据都存储在 A 服务里，那当然由 A 服务来提供接口比较合适，这样接口输入的数据比较少，不但简化了服务对外部的依赖，同时也降低了接口调用的成本。

给你举个例子，在订单小票上，我们经常能看到一些优惠信息，比如说商品原价是多少，其中因为满减优惠了多少，因为商品特价减免了多少。这个优惠计算的结果是订单的一部分，毫无疑问，它需要保存在订单服务里。

但这个订单的优惠计算过程，却不是由订单服务来负责，而是由独立的促销服务负责的。因为优惠计算所需要的优惠规则是在促销服务里定义的，促销服务可以在内部拿到所有的优惠规则，然后完成整个优惠计算。

否则，如果是由订单服务负责优惠计算，订单服务的调用者就需要在接口中提供完整的促销规则，不但调用成本高，而且外部促销规则的改变会影响订单服务的内部实现。

所以在这里，促销服务负责促销规则的维护，以及对应的优惠计算功能；订单服务负责优惠结果数据落地，以及后续的查询功能。这样，每个服务存储的数据和对外提供的功能是一致的。

最后一个，是正交原则

既然是基础服务，它们就处于调用链的底层，服务之间不会有任何的调用关系，也就是说基础服务相互之间是正交的。比如说会员服务和商品服务，它们代表不同维度的基础业务域，彼此之间不会有调用关系。

正交还有另外一种情况：服务之间有数据的依赖关系，但没有接口的调用关系。

比如说，订单明细里包含商品 ID 信息，但订单服务内部不会调用商品服务来获取商品详情。如果页面需要展示订单的商品详情，针对这个具体的业务场景，我们可以在上层的聚合服务里，通过聚合订单服务和商品服务来实现。

总结

可复用是架构设计的一个重要目标，今天我们对复用进行了梳理，包括复用有哪些形式，以及它们有哪些价值，相信你现在对复用已经有了一个整体的认识。**业务上的复用比纯粹的技术复用有更高的价值，我们要尽量往这个方向上靠。**

在实践中，落地基础服务是实现业务复用的有效方式，而基础服务边界的划分，它有科学的成分，但更多的是一种艺术，这里我提供了几个实用的划分原则，你可以在工作中结合实际情况，灵活地运用它们。

在专栏的下一讲，我会通过一个具体的订单服务例子，来帮助你更好地落地基础服务。

最后，给你留一道思考题：我们在落地服务时，有时会冗余存储其它服务的数据，你对这个有什么看法呢？

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　06 | 可扩展架构案例（三）：你真的需要一个中台吗?

下一篇　08 | 可复用架构案例（一）：如何设计一个基础服务?

精选留言 (12)

写留言

孙同学
2020-03-06

https://www.processon.com/view/link/5e51378ce4b0c037b5f9d1e3 冗余一定程度的数据可以简化上层的业务聚合调用，不过在存储数据时也会增加相应的复杂度，这应该是个辨证关系，看常用业务流程而定吧。不知道理解的对不对

展开

4

Middleware
2020-03-06

冗余一些数据，字段，我认为适当的场景是很有必要的。检索数据方便，查询效率也会提高

1

Din
2020-03-06

冗余其他服务的数据时比较常见的做法，这样可以减少和其他系统数据的交互，提高服务性能。不过数据一旦冗余，就会带来数据一致性的问题。

展开

1

每天晒白牙
2020-03-06

冗余数据有时也是可以的，就像电商中有买家查数据的需求，也有卖家查数据的需求，在做分表时为了提升性能会做冗余

展开

1

嬰
2020-03-08

冗余存储其他服务的数据好处是查询效率高，单个服务就能满足业务需求，无序聚合查询其他服务，但是也要注意冗余的度，要综合考虑到数据一致性和性能来做决定。

dowannado
2020-03-07

20200307 高复用 代码 技术组件 业务实体 业务流程 业务 完整性 一致性 正交 基础服务边界划分原则等

小伟
2020-03-07

在持久化数据中还是尽量少冗余其他服务的数据，因为维护一致性的开销较大。
如果某些服务数据是热点且变化频率不高，则可使用外部缓存提升性能，如redis。
如果热点数据变化大且一致性要求强，还是每次去调服务接口吧。

展开

卫江
2020-03-07

之所以冗余数据，说明该服务需要这些数据，但是又不想因为获取该冗余的数据而对其他的服务产生依赖而违背了我们的正交原则，但是就会有一致性问题，多个服务维护同一份数据的问题，如果这一块的冗余可以通过聚合层来避免，把相关的这一块的逻辑放在上层的聚合层来减少底层的冗余。

展开

Alex
2020-03-07

我们在落地服务时，有时会冗余存储其它服务的数据。典型的空间换时间的做法。对于特定业务需求避免了服务间的联合查询，简化实现难度减少时间消耗。但要注意冗余数据带来的一致性问题。

tt
2020-03-06

冗余数据是为了业务实现的方便和效率吧，应该不能无条件的否决，但如果后续无法控制住这些数据的生长，可能会带来问题。

老师的课太赞了，篇篇精华，信息量大，又都重点明确。

...
展开

探索无止境
2020-03-06

冗余数据的作用，我认为是为了提高查询性能，老师的课非常清晰到位，如果再附上一些代码就完美了

Jeff.Smile
2020-03-06

要点总结:
业务上的复用比纯粹的技术复用有更高的价值，我们要尽量往这个方向上靠。
在实践中，落地基础服务是实现业务复用的有效方式，而基础服务边界的划分，它有科学的成分，但更多的是一种艺术。一般有:完整性、一致性、正交性。
思考题:...
展开