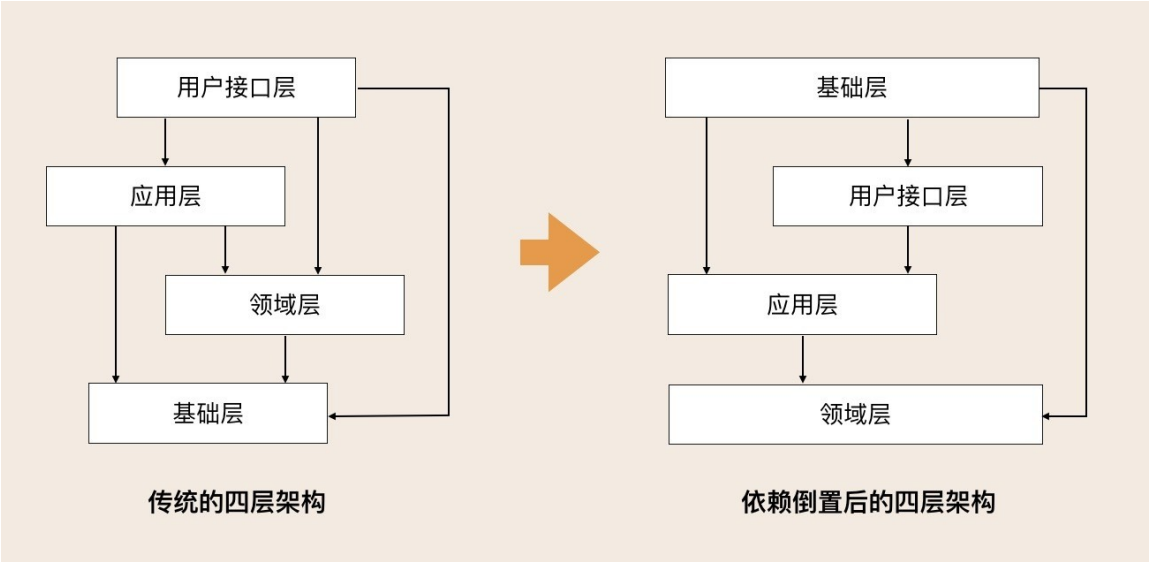
微服务架构模型有好多种，例如整洁架构、CQRS 和六边形架构等等。每种架构模式虽然提出的时代和背景不同，但其核心理念都是为了设计出“高内聚低耦合”的架构，轻松实现架构演进。而 DDD 分层架构的出现，使架构边界变得越来越清晰，它在微服务架构模型中，占有非常重要的位置。

那 DDD 分层架构到底长什么样？DDD 分层架构如何推动架构演进？我们该怎么转向 DDD 分层架构？这就是我们这一讲重点要解决的问题。

什么是 DDD 分层架构？

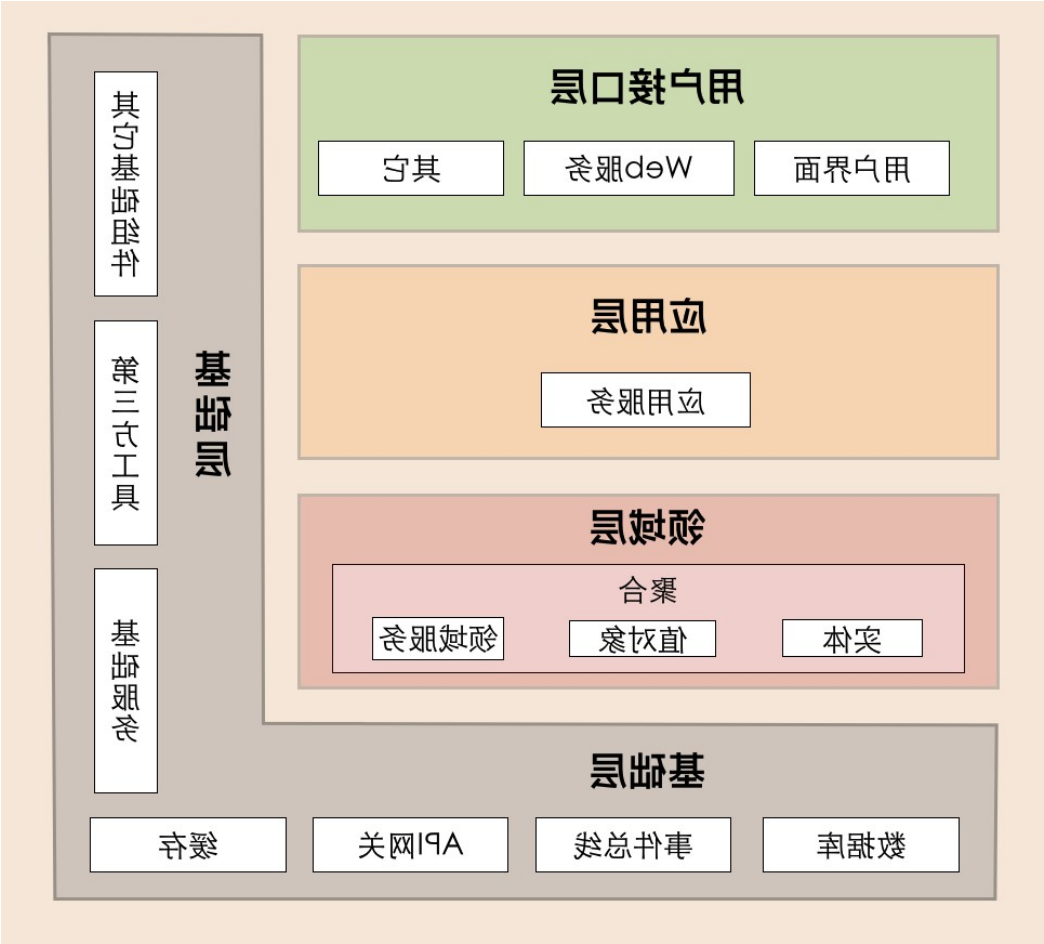
DDD 的分层架构在不断发展。最早是传统的四层架构；后来四层架构有了进一步的优化，实现了各层对基础层的解耦；再后来领域层和应用层之间增加了上下文环境（Context）层，五层架构（DCI）就此形成了。



我们看一下上面这张图，在最早的传统四层架构中，基础层是被其它层依赖的，它位于最核心的位置，那按照分层架构的思想，它应该就是核心，但实际上领域层才是软件的核心，所以这种依赖是有问题的。后来我们采用了依赖倒置

（Dependency inversion principle,DIP）的设计，优化了传统的四层架构，实现了各层对基础层的解耦。

我们今天讲的 DDD 分层架构就是优化后的四层架构。在下面这张图中，从上到下依次是：用户接口层、应用层、领域层和基础层。那 DDD 各层的主要职责是什么呢？下面我来逐一介绍一下。



1. **用户接口层**

用户接口层负责向用户显示信息和解释用户指令。这里的用户可能是：用户、程序、自动化测试和批处理脚本等等。

2. **应用层**

应用层是很薄的一层，理论上不应该有业务规则或逻辑，主要面向用例和流程相关的操作。但应用层又位于领域层之上，因为领域层包含多个聚合，所以它可以协调多个聚合的服务和领域对象完成服务编排和组合，协作完成业务操作。

此外，应用层也是微服务之间交互的通道，它可以调用其它微服务的应用服务，完成微服务之间的服务组合和编排。

这里我要提醒你一下：在设计和开发时，不要将本该放在领域层的业务逻辑放到应用层中实现。因为庞大的应用层会使领域模型失焦，时间一长你的微服务就会演化为传统的三层架构，业务逻辑会变得混乱。

另外，应用服务是在应用层的，它负责服务的组合、编排和转发，负责处理业务用例的执行顺序以及结果的拼装，以粗粒度的服务通过 API 网关向前端发布。还有，应用服务还可以进行安全认证、权限校验、事务控制、发送或订阅领域事件等。

3. **领域层**

领域层的作用是实现企业核心业务逻辑，通过各种校验手段保证业务的正确性。领域层主要体现领域模型的业务能力，它用来表达业务概念、业务状态和业务规则。

领域层包含聚合根、实体、值对象、领域服务等领域模型中的领域对象。

这里我要特别解释一下其中几个领域对象的关系，以便你在设计领域层的时候能更加清楚。首先，领域模型的业务逻辑主要是由实体和领域服务来实现的，其中实体会采用充血模型来实现所有与之相关的业务功能。其次，你要知道，实体和领域服务在实现业务逻辑上不是同级的，当领域中的某些功能，单一实体（或者值对象）不能实现时，领域服务就会出马，它可以组合聚合内的多个实体（或者值对象），实现复杂的业务逻辑。

4. **基础层**

基础层是贯穿所有层的，它的作用就是为其它各层提供通用的技术和基础服务，包括第三方工具、驱动、消息中间件、网关、文件、缓存以及数据库等。比较常见的功能还是提供数据库持久化。

基础层包含基础服务，它采用依赖倒置设计，封装基础资源服务，实现应用层、领域层与基础层的解耦，降低外部资源变化对应用的影响。

比如说，在传统架构设计中，由于上层应用对数据库的强耦合，很多公司在架构演进中最担忧的可能就是换数据库了，因为一旦更换数据库，就可能需要重写大部分的代码，这对应用来说是致命的。那采用依赖倒置的设计以后，应用层就可以通过解耦来保持独立的核心业务逻辑。当数据库变更时，我们只需要更换数据库基础服务就可以了，这样就将资源变更对应用的影响降到了最低。

DDD 分层架构最重要的原则是什么？

在《实现领域驱动设计》一书中，DDD 分层架构有一个重要的原则：每层只

能与位于其下方的层发生耦合。

而架构根据耦合的紧密程度又可以分为两种：严格分层架构和松散分层架构。优化后的 DDD 分层架构模型就属于严格分层架构，任何层只能对位于其直接下方的层产生依赖。而传统的 DDD 分层架构则属于松散分层架构，它允许某层与其任意下方的层发生依赖。

那我们怎么选呢？综合我的经验，为了服务的可管理，我建议你采用严格分层架构。

在严格分层架构中，领域服务只能被应用服务调用，而应用服务只能被用户接口层调用，服务是逐层对外封装或组合的，依赖关系清晰。而在松散分层架构中，领域服务可以同时被应用层或用户接口层调用，服务的依赖关系比较复杂且难管理，甚至容易使核心业务逻辑外泄。

试想下，如果领域层中的某个服务发生了重大变更，那该如何通知所有调用方同步调整和升级呢？但在严格分层架构中，你只需要逐层通知上层服务就可以了。

DDD 分层架构如何推动架构演进？

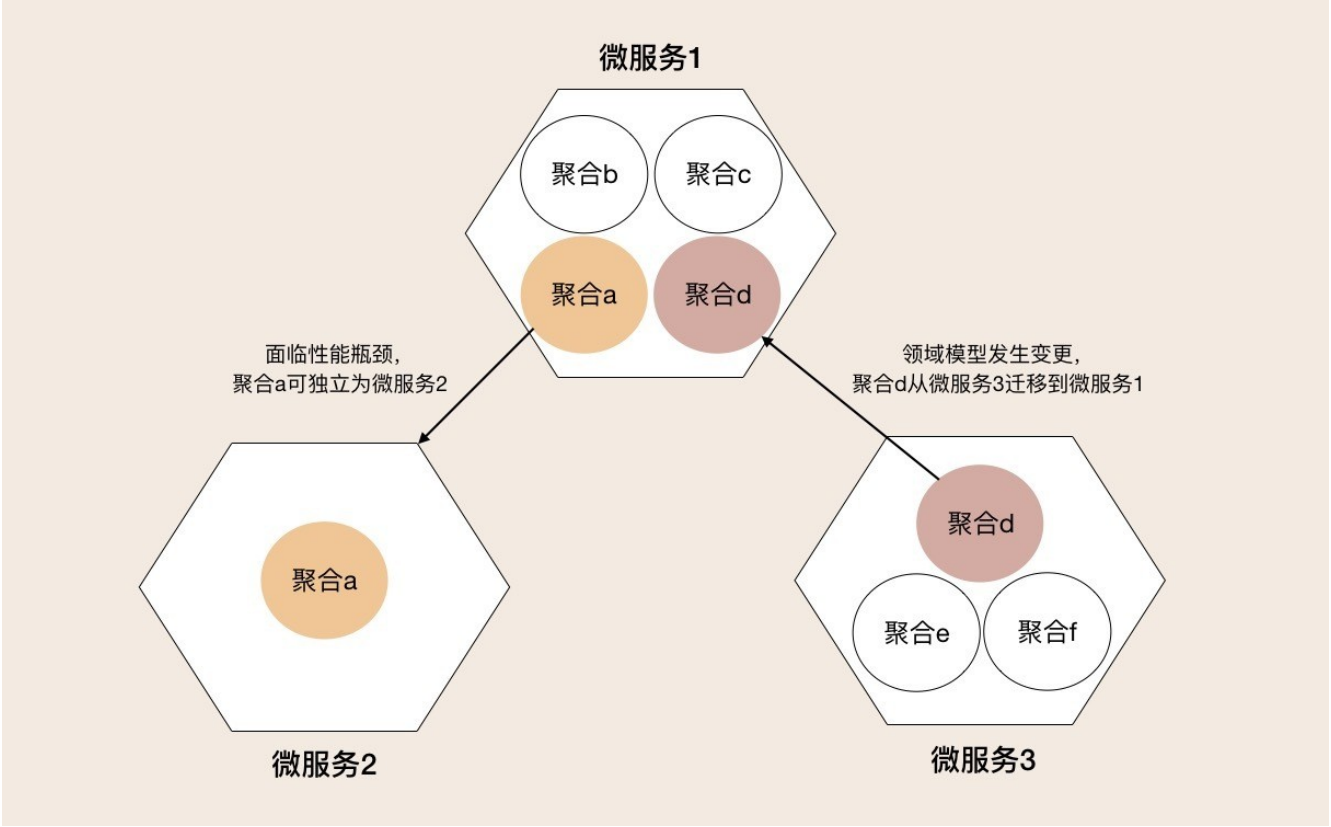
领域模型不是一成不变的，因为业务的变化会影响领域模型，而领域模型的变化则会影响微服务的功能和边界。那我们该如何实现领域模型和微服务的同步演进呢？

1. **微服务架构的演进**

通过基础篇的讲解，我们知道：领域模型中对象的层次从内到外依次是：值对象、实体、聚合和限界上下文。

实体或值对象的简单变更，一般不会让领域模型和微服务发生大的变化。但聚合的重组或拆分却可以。这是因为聚合内业务功能内聚，能独立完成特定的业务逻辑。那聚合的重组或拆分，势必就会引起业务模块和系统功能的变化了。

这里我们可以以聚合为基础单元，完成领域模型和微服务架构的演进。聚合可以作为一个整体，在不同的领域模型之间重组或者拆分，或者直接将一个聚合独立为微服务。



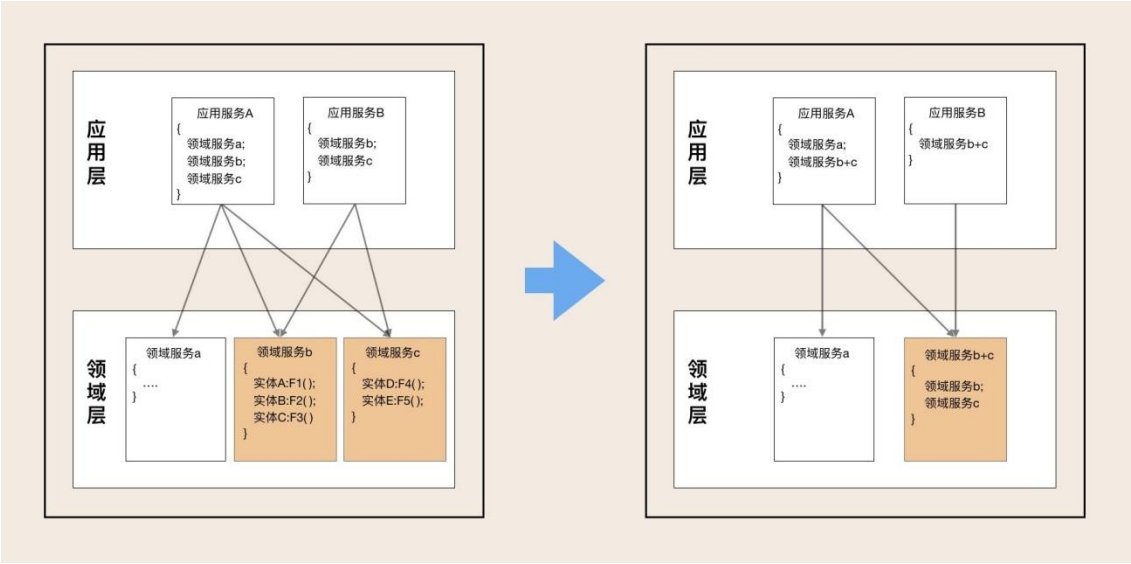
我们结合上图，以微服务 1 为例，讲解下微服务架构的演进过程：

- 当你发现微服务 1 中聚合 a 的功能经常被高频访问，以致拖累整个微服务 1 的性能时，我们可以把聚合 a 的代码，从微服务 1 中剥离出来，独立为微服务 2。这样微服务 2 就可轻松应对高性能场景。
- 在业务发展到一定程度以后，你会发现微服务 3 的领域模型有了变化，聚合 d 会更适合放到微服务 1 的领域模型中。这时你就可以将聚合 d 的代码整体搬迁到微服务 1 中。如果你在设计时已经定义好了聚合之间的代码边界，这个过程不会太复杂，也不会花太多时间。
- 最后我们发现，在经历模型和架构演进后，微服务 1 已经从最初包含聚合 a、b、c，演进为包含聚合 b、c、d 的新领域模型和微服务了。

你看，好的聚合和代码模型的边界设计，可以让你快速应对业务变化，轻松实现领域模型和微服务架构的演进。你可能还会想，那怎么实现聚合代码快速重组呢？别急，后面实战篇会详细讲解，这里我们先感知下大的实现流程。

2. **微服务内服务的演进**

在微服务内部，实体的方法被领域服务组合和封装，领域服务又被应用服务组合和封装。在服务逐层组合和封装的过程中，你会发现这样一个有趣的现象。



我们看下上面这张图。在服务设计时，你并不一定能完整预测有哪些下层服务会被多少个上层服务组装，因此领域层通常只提供一些原子服务，比如领域服务 a、b、c。但随着系统功能增强和外部接入越来越多，应用服务会不断丰富。有一天你会发现领域服务 b 和 c 同时多次被多个应用服务调用了，执行顺序也基本一致。这时你可以考虑将 b 和 c 合并，再将应用服务中 b、c 的功能下沉到领域层，演进为新的领域服务（b+c）。这样既减少了服务的数量，也减轻了上层服务组合和编排的复杂度。

你看，这就是服务演进的过程，它是随着你的系统发展的，最后你会发现你的领域模型会越来越精炼，越来越能适应需求的快速变化。

三层架构如何演进到 DDD 分层架构？

综合前面的讲解，相信 DDD 分层架构的优势，你心里也有个谱了。我们不妨总结一下最最重要两点。

首先，由于层间松耦合，我们可以专注于本层的设计，而不必关心其它层，也

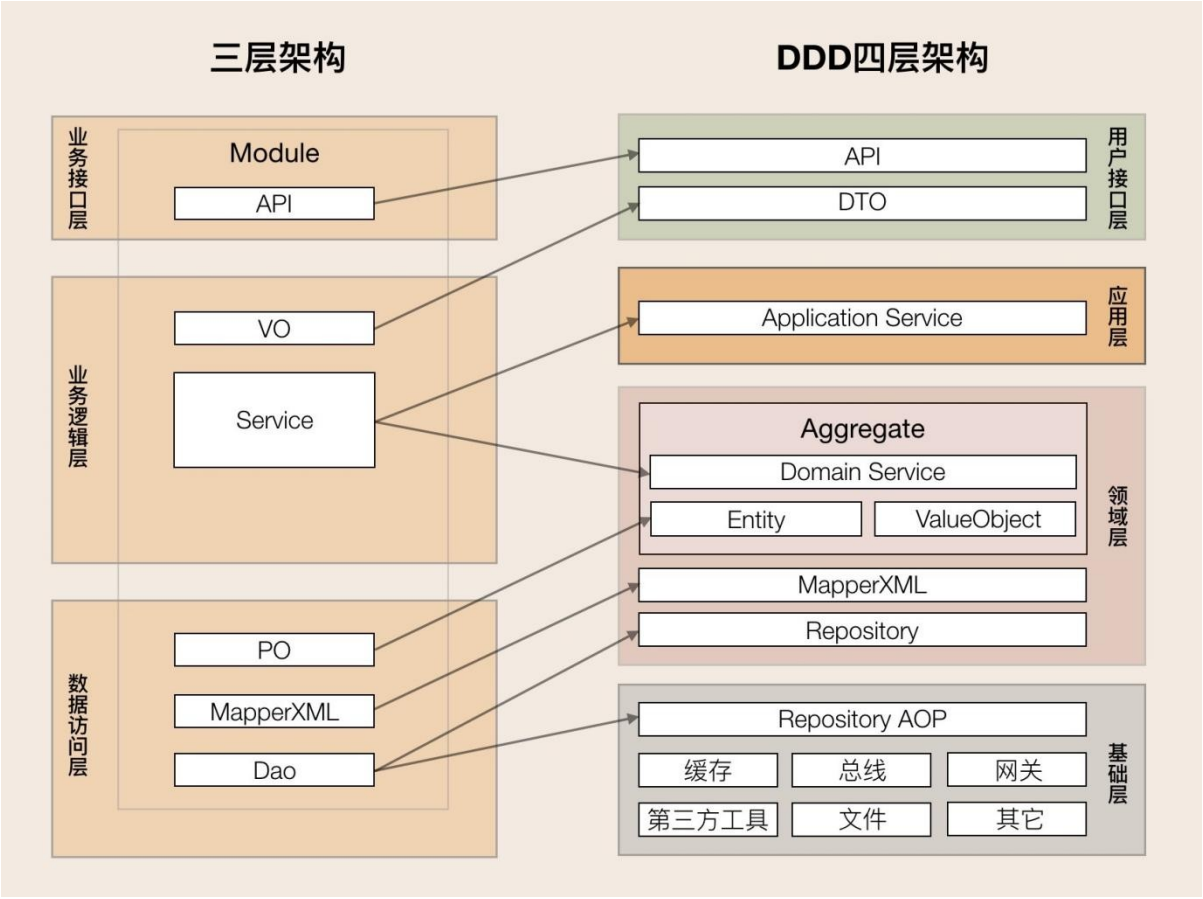
不必担心自己的设计会影响其它层。可以说，DDD 成功地降低了层与层之间的依赖。

其次，分层架构使得程序结构变得清晰，升级和维护更加容易。我们修改某层代码时，只要本层的接口参数不变，其它层可以不必修改。即使本层的接口发生变化，也只影响相邻的上层，修改工作量小且错误可以控制，不会带来意外的风险。

那我们该怎样转向 DDD 分层架构呢？不妨看看下面这个过程。

传统企业应用大多是单体架构，而单体架构则大多是三层架构。三层架构解决了程序内代码间调用复杂、代码职责不清的问题，但这种分层是逻辑概念，在物理上它是中心化的集中式架构，并不适合分布式微服务架构。

DDD 分层架构中的要素其实和三层架构类似，只是在 DDD 分层架构中，这些要素被重新归类，重新划分了层，确定了层与层之间的交互规则和职责边界。



我们看一下上面这张图，分析一下从三层架构向 DDD 分层架构演进的过程。

首先，你要清楚，三层架构向 DDD 分层架构演进，主要发生在业务逻辑层和数据访问层。

DDD 分层架构在用户接口层引入了 DTO，给前端提供了更多的可使用数据和更高的展示灵活性。

DDD 分层架构对三层架构的业务逻辑层进行了更清晰的划分，改善了三层架构核心业务逻辑混乱，代码改动相互影响大的情况。DDD 分层架构将业务逻辑层的服务拆分到了应用层和领域层。应用层快速响应前端的变化，领域层实现领域模型的能力。

另外一个重要的变化发生在数据访问层和基础层之间。三层架构数据访问采用DAO方式；DDD分层架构的数据库等基础资源访问，采用了仓储（Repository）设计模式，通过依赖倒置实现各层对基础资源的解耦。

仓储又分为两部分：仓储接口和仓储实现。仓储接口放在领域层中，仓储实现放在基础层。原来三层架构通用的第三方工具包、驱动、Common、Utility、Config 等通用的公共的资源类统一放到了基础层。

最后，我想说，传统三层架构向 DDD 分层架构的演进，体现的正是领域驱动设计思想的演进。希望你也感受到了，并尝试将其应用在自己的架构设计中。