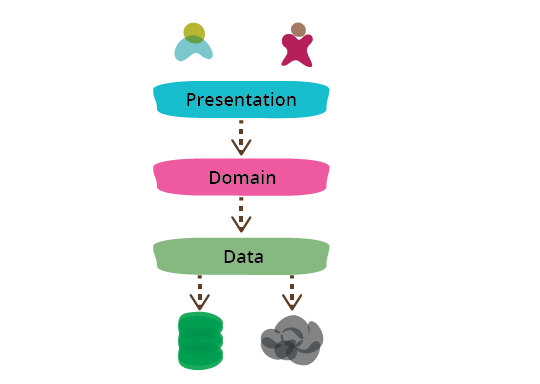
<https://martinfowler.com/bliki/PresentationDomainDataLayering.html>

表示域数据分层

对信息丰富的程序进行模块化的最常见方法之一是将其分为三个广泛的层:表示层(UI)、域逻辑(又名业务逻辑)和数据访问层。因此，您经常会看到web应用程序被分为web层(了解如何处理HTTP请求和呈现HTML)、业务逻辑层(包含验证和计算)和数据访问层(整理如何管理数据库或远程服务中的持久数据)。



总的来说，我发现这是许多应用程序的一种有效的模块化形式，也是我经常使用和鼓励的一种形式。它最大的优点(对我来说)是可以减少我的注意力范围，允许我相对独立地思考这三个主题。当我在处理领域逻辑代码时，我基本上可以忽略UI，将任何与数据源的交互视为一组抽象函数，这些函数提供我需要的数据，并根据我的需要更新数据。当我在处理数据访问层时，我关注的是将数据处理到表单requi中的细节

这种范围的缩小并不意味着编程有任何顺序——我通常发现我需要在层之间迭代。我可能会根据我对UX的最初理解来构建数据和领域层，但在优化UX时，我需要改变领域，这就需要改变数据层。但即使有这种跨层迭代，我发现在进行更改时一次只关注一个层更容易。这类似于你在重构的两个角色中切换思维模式。

模块化的另一个原因是允许我替换模块的不同实现。这种分离允许我在相同的域逻辑之上构建多个表示，而不需要复制它。多个演示文稿可能是一个web应用程序中的独立页面，一个web应用程序加上移动原生应用程序，一个用于脚本编写的API，甚至是一个老式的命令行界面。对数据源进行模块化，使我能够优雅地处理数据库技术中的变化，或者支持那些可能在很少注意到的情况下发生变化的持久性服务。然而，我不得不提一下

模块化也支持可测试性，这自然吸引了我这个自我测试代码的超级粉丝。模块边界暴露了接缝，这对测试有很好的启示作用。UI代码通常很难测试，所以最好将尽可能多的逻辑放入一个易于测试的域层，而不必通过UI[1]来访问程序。数据访问通常是缓慢和笨拙的，因此在数据层周围使用testdouble通常会使域逻辑测试更容易和响应性更好。

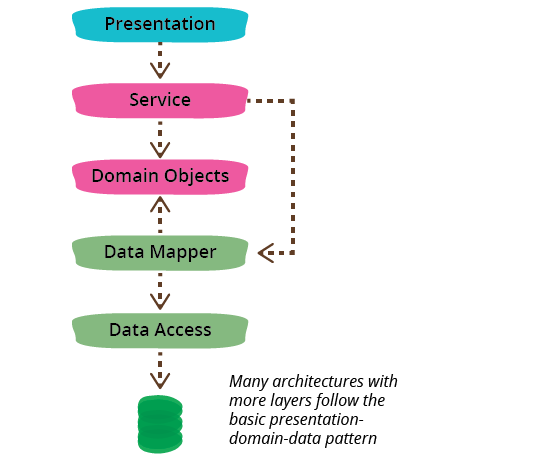
虽然可替换性和可测试性当然是这种分层的好处，但我必须强调，即使没有上述任何一个原因，我仍然会像这样划分层。注意力原因范围的缩小本身就足够了。

在讨论这个问题时，我们可以将其视为一个模式(表示-领域-数据)，也可以将其分为两个模式(表示-领域和领域-数据)。这两种观点都很有用——我认为表示领域数据是表示领域和领域数据的组合。

我认为这些层是模块的一种形式，这是我用来描述如何将软件聚集成相对独立的部分的通用词。这与代码的对应关系取决于我们所处的编程环境。通常，最低层次是某种形式的子程序或函数。面向对象的语言有一个集合函数和数据结构的类的概念。大多数语言都有某种形式的更高级别，称为包或名称空间，通常可以形成层次结构。模块可能对应于单独的可部署单元:库或服务，但是t

分层可以发生在这些级别中的任何一个。一个小程序可能只是将层的单独函数放入不同的文件中。较大的系统可能有对应于名称空间的层，每个层中有许多类。

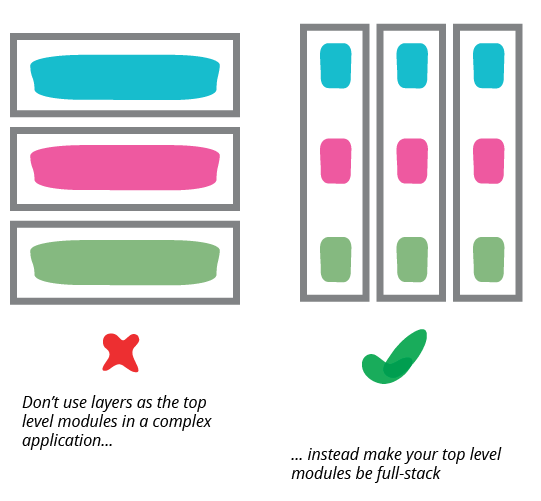
我在这里提到了三层，但是架构通常有三层以上。一种常见的变体是在域和表示之间放置一个服务层，或者使用表示模型之类的东西将表示层拆分为单独的层。我没有发现更多的层打破了基本模式，因为核心分离仍然存在。



依赖关系通常从上到下贯穿整个层堆栈:表示依赖于域，域又依赖于数据源。一种常见的变体是通过在域和数据源层之间引入映射器来安排事物，使域不依赖于它的数据源。这种方法通常被称为六边形架构。

这些层是逻辑层，而不是物理层。我可以在我的笔记本电脑上运行所有三层，我可以在服务器上有数据库的桌面中运行表示和域模型，我可以在浏览器中使用富客户端分割表示，在服务器上使用后台前端。在这种情况下，我将BFF视为一个表示层，因为它专注于支持特定的表示选项。

尽管表示-领域-数据分离是一种常见的方法，但它只能在相对较小的粒度上应用。随着应用程序的发展，每一层都可能变得非常复杂，需要进一步模块化。当这种情况发生时，使用表示域数据作为更高级别的模块通常不是最好的。框架通常鼓励您使用视图-模型-数据之类的顶级命名空间;这对于较小的系统来说是可以的，但是一旦这些层变得太大，你应该把你的顶层划分为面向领域的模块



我见过的这种分层将组织引入歧途的一种常见方式是按照这些层分离开发团队的反模式。这看起来很吸引人，因为前端和后端开发需要不同的框架(甚至语言)，这使得开发人员很容易专攻其中一种。将那些具有共同技能的人放在一起可以支持技能共享，并允许组织将团队视为单一的、描述良好的工作类型的提供者。同样，将所有数据库专家放在一起符合数据库和模式的通用集中化。但这些层之间丰富的相互作用需要它们之间的频繁交换。当你在同一个团队中有可以随意合作的专家时，这并不是很难，但团队边界会增加相当大的摩擦，同时也会降低个人开发重要性的动机

延伸阅读

我在其他地方从许多不同的角度写过关于这种分离的文章。这种分层驱动EAA的P结构，那本书的第一章更多地讨论了这种分层。在那本书中，我并没有把这个分层本身作为一种模式，而是用分离的表示和PresentationDomainSeparation来玩弄这个领域。

关于为什么表示域数据不应该是更大系统中的最高级别模块的更多信息，请查看Simon Brown的写作和演讲。我也同意他的观点，即软件架构应该嵌入到代码中。

我和同事Badri Janakiraman就六边形建筑的本质进行了一次有趣的讨论。上下文主要是围绕使用Ruby on Rails的应用程序，但当您可能考虑这种方法时，很多思考也适用于其他情况。