

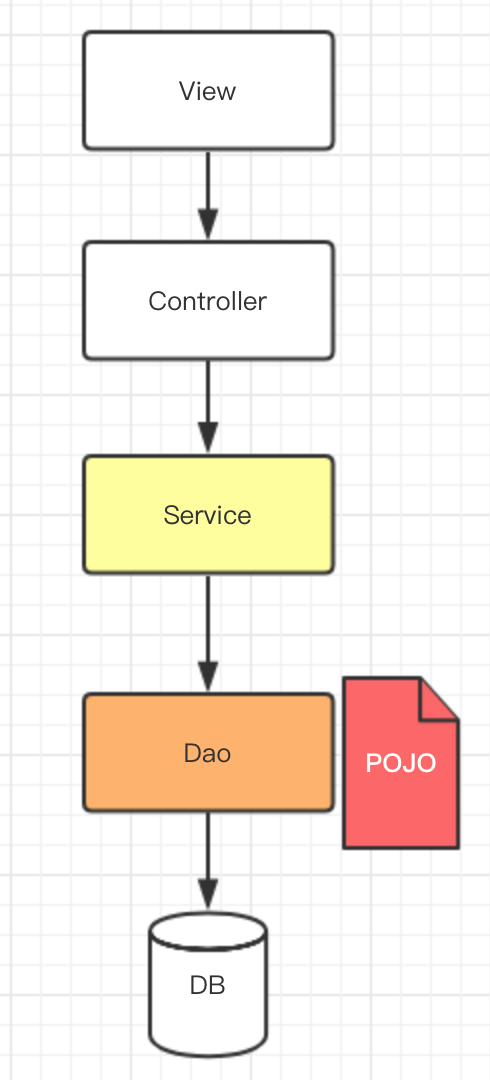
## 背景

随着产品的不断迭代和演化，业务逻辑变得越来越复杂，我们的系统也越来越冗杂。各个模块之间彼此关联，甚至到后期连相应的开发者都很难说清模块的具体功能和意图到底是啥

在维护代码时，将会导致牵一发而动全身

一般他们的解决方案是不断的重构系统，让系统的设计随着业务成长也进行不断的演进。通过重构出一些独立的类来存放某些通用的逻辑解决混乱问题，但是我们很难给它一个业务上的含义，只能以技术纬度进行描述，那么带来的问题就是其他人接手这块代码的时候不知道这个的含义或者只能通过修改通用逻辑来达到某些需求.

在传统项目中，数据库是整个项目的根本，数据模型出来以后后续的开发都是围绕着数据展开:



这其中存在的问题如下：

* Service层很重，所有逻辑处理基本都放在service层。
* POJO作为 Service 层非常重要的一个实体，会因为不同场景的需求做不同的变化和组合，就会造成 POJO 的几种不同模型(失血、贫血、充血)，以此用来形容领域模型太胖或者太瘦。
* 随着业务变得复杂以后，包括数据结构的变化，那么各个模块就需要进行修改，原本清晰的系统经过不断的演 化变得复杂、冗余、耦合度高，后果就非常严重。

如果没有了数据存储，那么我们的领域模型就得基于程序本身来设计。那这个就是 DDD 需要去考虑的问题

与Transaction Script风格的架构相比:



可以看出，两者的差异并不是太大（对于Façade来说，它是一种可选设施，如果系统架构中省略Façade，则DTO与领域对象的互换工作可在service中进行），这也从则面说明推行领域驱动设计的关键并不在架构上，而在于整个团队在分析、设计和开发上没有自始至终地以领域模型为核心开展工作，以面向对象的思想进行设计和编程

两个类组件上有质的区别，一个是领域对象，一个是Service。领域驱动设计的架构核心目标是要创建一个富领域模型，其典型特征是它的领域对象具有丰富的业务方法用以处理业务逻辑，而Transaction Script风格的领域对象则仅仅是数据的载体，没有业务方法，这种领域也被称作“贫血的领域对象”(Anemic Domain Objects)。在Service方面，领域驱动设计的架构里Service是非常“薄“的一层，其并不负责处理业务逻辑，而在TransactionScript风格的架构里，Service是处理业务逻辑的主要场所，因而往往非常厚重。

## 一般步骤

1. 根据需求建立一个初步的领域模型，识别出一些明显的领域概念以及它们的关联，关联可以暂时没有方向但需要有（1：1，1：N，M：N）这些关系；可以用文字精确的没有歧义的描述出每个领域概念的涵义以及包含的主要信息；
2. 分析主要的软件应用程序功能，识别出主要的应用层的类；这样有助于及早发现哪些是应用层的职责，哪些是领域层的职责；
3. 进一步分析领域模型，识别出哪些是实体，哪些是值对象，哪些是领域服务；
4. 分析关联，通过对业务的更深入分析以及各种软件设计原则及性能方面的权衡，明确关联的方向或者去掉一些不需要的关联；
5. 找出聚合边界及聚合根，这是一件很有难度的事情；因为你在分析的过程中往往会碰到很多模棱两可的难以清晰判断的选择问题，所以，需要我们平时一些分析经验的积累才能找出正确的聚合根；
6. 为聚合根配备仓储，一般情况下是为一个聚合分配一个仓储，此时只要设计好仓储的接口即可；
7. 走查场景，确定我们设计的领域模型能够有效地解决业务需求；
8. 考虑如何创建领域实体或值对象，是通过工厂还是直接通过构造函数；
9. 停下来重构模型。寻找模型中觉得有些疑问或者是蹩脚的地方，比如思考一些对象应该通过关联导航得到还是应该从仓储获取？聚合设计的是否正确？考虑模型的性能怎样，等等；

领域建模是一个不断重构，持续完善模型的过程，大家会在讨论中将变化的部分反映到模型中，从而是模型不断细化并朝正确的方向走。领域建模是领域专家、设计人员、开发人员之间沟通交流的过程，是大家工作和思考问题的基础。

面向对象与面向过程最大区别就是：分解。边界的分解。从需求到最后实现都贯穿

**源**

1. <https://www.cnblogs.com/ealenxie/p/9559781.html> [一个微服务+DDD(领域驱动设计)的代码结构示例]
2. <https://blog.csdn.net/bluishglc/article/details/6681253> [领域驱动设计(Domain Driven Design)参考架构详解]
3. <https://www.cnblogs.com/netfocus/archive/2011/10/10/2204949.html> [DDD领域驱动设计基本理论知识总结]
4. <https://www.cnblogs.com/hafiz/p/9388334.html> [可以落地的DDD到底长什么样?]