

## 11 | 实战一（上）：业务开发常用的基于贫血模型的MVC架构违背OOP吗？

2019-11-27 王争

设计模式之美

[进入课程 >](#)



讲述：冯永吉

时长 15:52 大小 14.54M



在前面几节课中，我们学习了面向对象的一些理论知识，比如，面向对象四大特性、接口和抽象类、面向对象和面向过程编程风格、基于接口而非实现编程和多用组合少用继承设计思想等等。接下来，我们再用四节课的时间，通过两个更加贴近实战的项目来进一步学习，如何将这些理论应用到实际的软件开发中。

据我了解，大部分工程师都是做业务开发的，所以，今天我们讲的这个实战项目也是一个典型的业务系统开发案例。我们都知道，很多业务系统都是基于 MVC 三层架构来开发的。实际上，更确切点讲，这是一种基于贫血模型的 MVC 三层架构开发模式。

虽然这种开发模式已经成为标准的 Web 项目的开发模式，但它却违反了面向对象编程风格，是一种彻彻底底的面向过程的编程风格，因此而被有些人称为 **反模式 (anti-pattern)**。特别是**领域驱动设计** (Domain Driven Design, 简称 DDD) 盛行之后，这种基于贫血模型的传统开发模式就更加被人诟病。而基于充血模型的 DDD 开发模式越来越被人提倡。所以，我打算用两节课的时间，结合一个虚拟钱包系统的开发案例，带你彻底弄清楚这两种开发模式。

考虑到你有可能不太了解我刚刚提到的这几个概念，所以，在正式进入实战项目的讲解之前，我先带你搞清楚下面几个问题：

什么是贫血模型？什么是充血模型？

为什么说基于贫血模型的传统开发模式违反 OOP？

基于贫血模型的传统开发模式既然违反 OOP，那又为什么如此流行？

什么情况下我们应该考虑使用基于充血模型的 DDD 开发模式？

好了，让我们带着这些问题，正式开始今天的学习吧！

## 什么是基于贫血模型的传统开发模式？

我相信，对于大部分的后端开发工程师来说，MVC 三层架构都不会陌生。不过，为了统一我们之间对 MVC 的认识，我还是带你一块来回顾一下，什么是 MVC 三层架构。

MVC 三层架构中的 M 表示 Model，V 表示 View，C 表示 Controller。它将整个项目分为三层：展示层、逻辑层、数据层。MVC 三层开发架构是一个比较笼统的分层方式，落实到具体的开发层面，很多项目也并不会 100% 遵从 MVC 固定的分层方式，而是会根据具体的项目需求，做适当的调整。

比如，现在很多 Web 或者 App 项目都是前后端分离的，后端负责暴露接口给前端调用。这种情况下，我们一般就将后端项目分为 Repository 层、Service 层、Controller 层。其中，Repository 层负责数据访问，Service 层负责业务逻辑，Controller 层负责暴露接口。当然，这只是其中一种分层和命名方式。不同的项目、不同的团队，可能会对此有所调整。不过，万变不离其宗，只要是依赖数据库开发的 Web 项目，基本的分层思路都大差不差。

刚刚我们回顾了 MVC 三层开发架构。现在，我们再来看一下，什么是贫血模型？

实际上，你可能一直都在用贫血模型做开发，只是自己不知道而已。不夸张地讲，据我了解，目前几乎所有的业务后端系统，都是基于贫血模型的。我举一个简单的例子来给你解释一下。

 复制代码

```
1  //////////// Controller+VO(View Object) ////////////
2  public class UserController {
3      private UserService userService; // 通过构造函数或者 IOC 框架注入
4
5      public UserVo getUserById(Long userId) {
6          UserBo userBo = userService.getUserById(userId);
7          UserVo userVo = [...convert userBo to userVo...];
8          return userVo;
9      }
10 }
11
12 public class UserVo { // 省略其他属性、get/set/construct 方法
13     private Long id;
14     private String name;
15     private String cellphone;
16 }
17
18 //////////// Service+BO(Business Object) ////////////
19 public class UserService {
20     private UserRepository userRepository; // 通过构造函数或者 IOC 框架注入
21
22     public UserBo getUserById(Long userId) {
23         UserEntity userEntity = userRepository.getUserById(userId);
24         UserBo userBo = [...convert userEntity to userBo...];
25         return userBo;
26     }
27 }
28
29 public class UserBo { // 省略其他属性、get/set/construct 方法
30     private Long id;
31     private String name;
32     private String cellphone;
33 }
34
35 //////////// Repository+Entity ////////////
36 public class UserRepository {
37     public UserEntity getUserById(Long userId) { //... }
38 }
39
40 public class UserEntity { // 省略其他属性、get/set/construct 方法
41     private Long id;
```

```
42     private String name;
43     private String cellphone;
44 }
```

我们平时开发 Web 后端项目的时候，基本上都是这么组织代码的。其中，UserEntity 和 UserRepository 组成了数据访问层，UserBo 和 UserService 组成了业务逻辑层，UserVo 和 UserController 在这里属于接口层。

从代码中，我们可以发现，UserBo 是一个纯粹的数据结构，只包含数据，不包含任何业务逻辑。业务逻辑集中在 UserService 中。我们通过 UserService 来操作 UserBo。换句话说，Service 层的数据和业务逻辑，被分割为 BO 和 Service 两个类中。像 UserBo 这样，只包含数据，不包含业务逻辑的类，就叫作**贫血模型**（Anemic Domain Model）。同理，UserEntity、UserVo 都是基于贫血模型设计的。这种贫血模型将数据与操作分离，破坏了面向对象的封装特性，是一种典型的面向过程的编程风格。

## 什么是基于充血模型的 DDD 开发模式？

刚刚我们讲了基于贫血模型的传统的开发模式。现在我们再讲一下，另外一种最近更加被推崇的开发模式：基于充血模型的 DDD 开发模式。

### 首先，我们先来看一下，什么是充血模型？

在贫血模型中，数据和业务逻辑被分割到不同的类中。**充血模型**（Rich Domain Model）正好相反，数据和对应的业务逻辑被封装到同一个类中。因此，这种充血模型满足面向对象的封装特性，是典型的面向对象编程风格。

### 接下来，我们再来看一下，什么是领域驱动设计？

领域驱动设计，即 DDD，主要是用来指导如何解耦业务系统，划分业务模块，定义业务领域模型及其交互。领域驱动设计这个概念并不新颖，早在 2004 年就被提出了，到现在已经有十几年的历史了。不过，它被大众熟知，还是基于另一个概念的兴起，那就是微服务。

我们知道，除了监控、调用链追踪、API 网关等服务治理系统的开发之外，微服务还有另外一个更加重要的工作，那就是针对公司的业务，合理地做微服务拆分。而领域驱动设计恰好就是用来指导划分服务的。所以，微服务加速了领域驱动设计的盛行。



不过，我个人觉得，领域驱动设计有点儿类似敏捷开发、SOA、PAAS 等概念，听起来很高大上，但实际上只值“五毛钱”。即便你没有听说过领域驱动设计，对这个概念一无所知，只要你是在开发业务系统，也或多或少都在使用它。做好领域驱动设计的关键是，看你对自己所做业务的熟悉程度，而并不是对领域驱动设计这个概念本身的掌握程度。即便你对领域驱动搞得再清楚，但是对业务不熟悉，也并不一定能做出合理的领域设计。所以，不要把领域驱动设计当银弹，不要花太多的时间去过度地研究它。

实际上，基于充血模型的 DDD 开发模式实现的代码，也是按照 MVC 三层架构分层的。Controller 层还是负责暴露接口，Repository 层还是负责数据存取，Service 层负责核心业务逻辑。它跟基于贫血模型的传统开发模式的区别主要在 Service 层。

在基于贫血模型的传统开发模式中，Service 层包含 Service 类和 BO 类两部分，BO 是贫血模型，只包含数据，不包含具体的业务逻辑。业务逻辑集中在 Service 类中。在基于充血模型的 DDD 开发模式中，Service 层包含 Service 类和 Domain 类两部分。Domain 就相当于贫血模型中的 BO。不过，Domain 与 BO 的区别在于它是基于充血模型开发的，既包含数据，也包含业务逻辑。而 Service 类变得非常单薄。总结一下的话就是，基于贫血模型的传统开发模式，重 Service 轻 BO；基于充血模型的 DDD 开发模式，轻 Service 重 Domain。

基于充血模型的 DDD 设计模式的概念，今天我们只是简单地介绍了一下。在下一节课中，我会结合具体的项目，通过代码来给你展示，如何基于这种开发模式来开发一个系统。

## 为什么基于贫血模型的传统开发模式如此受欢迎？

前面我们讲过，基于贫血模型的传统开发模式，将数据与业务逻辑分离，违反了 OOP 的封装特性，实际上是一种面向过程的编程风格。但是，现在几乎所有的 Web 项目，都是基于这种贫血模型的开发模式，甚至连 Java Spring 框架的官方 demo，都是按照这种开发模式来编写的。

我们前面也讲过，面向过程编程风格有种种弊端，比如，数据和操作分离之后，数据本身的操作就不受限制了。任何代码都可以随意修改数据。既然基于贫血模型的这种传统开发模式是面向过程编程风格的，那它又为什么会被广大程序员所接受呢？关于这个问题，我总结了下面三点原因。

第一点原因是，大部分情况下，我们开发的系统业务可能都比较简单，简单到就是基于 SQL 的 CRUD 操作，所以，我们根本不需要动脑子精心设计充血模型，贫血模型就足以应付这种简单业务的开发工作。除此之外，因为业务比较简单，即便我们使用充血模型，那模型本身包含的业务逻辑也并不会很多，设计出来的领域模型也会比较单薄，跟贫血模型差不多，没有太大意义。

第二点原因是，充血模型的设计要比贫血模型更加有难度。因为充血模型是一种面向对象的编程风格。我们从一开始就要设计好针对数据要暴露哪些操作，定义哪些业务逻辑。而不是像贫血模型那样，我们只需要定义数据，之后有什么功能开发需求，我们就在 Service 层定义什么操作，不需要事先做太多设计。

第三点原因是，思维已固化，转型有成本。基于贫血模型的传统开发模式经历了这么多年，已经深得人心、习以为常。你随便问一个旁边的大龄同事，基本上他过往参与的所有 Web 项目应该都是基于这个开发模式的，而且也没有出过啥大问题。如果转向用充血模型、领域驱动设计，那势必有一定的学习成本、转型成本。很多人在没有遇到开发痛点的情况下，是不愿意做这件事情的。

## 什么项目应该考虑使用基于充血模型的 DDD 开发模式？

既然基于贫血模型的开发模式已经成为了一种约定俗成的开发习惯，那什么样的项目应该考虑使用基于充血模型的 DDD 开发模式呢？

刚刚我们讲到，基于贫血模型的传统开发模式，比较适合业务比较简单的系统开发。相对应的，基于充血模型的 DDD 开发模式，更适合业务复杂的系统开发。比如，包含各种利息计算模型、还款模型等复杂业务的金融系统。

你可能会有一些疑问，这两种开发模式，落实到代码层面，区别不就是一个将业务逻辑放到 Service 类中，一个将业务逻辑放到 Domain 领域模型中吗？为什么基于贫血模型的传统开发模式，就不能应对复杂业务系统的开发？而基于充血模型的 DDD 开发模式就可以呢？

实际上，除了我们能看到的代码层面的区别之外（一个业务逻辑放到 Service 层，一个放到领域模型中），还有一个非常重要的区别，那就是两种不同的开发模式会导致不同的开发流程。基于充血模型的 DDD 开发模式的开发流程，在应对复杂业务系统的开发的时候更加有优势。为什么这么说呢？我们先来回忆一下，我们平时基于贫血模型的传统开发模式，都是怎么实现一个功能需求的。

不夸张地讲，我们平时的开发，大部分都是 SQL 驱动（SQL-Driven）的开发模式。我们接到一个后端接口的开发需求的时候，就去看接口需要的数据对应到数据库中，需要哪张表或者哪几张表，然后思考如何编写 SQL 语句来获取数据。之后就是定义 Entity、BO、VO，然后模板式地往对应的 Repository、Service、Controller 类中添加代码。

业务逻辑包裹在一个大的 SQL 语句中，而 Service 层可以做的事情很少。SQL 都是针对特定的业务功能编写的，复用性差。当我要开发另一个业务功能的时候，只能重新写个满足新需求的 SQL 语句，这就可能导致各种长得差不多、区别很小的 SQL 语句满天飞。

所以，在这个过程中，很少有人会应用领域模型、OOP 的概念，也很少有代码复用意识。对于简单业务系统来说，这种开发方式问题不大。但对于复杂业务系统的开发来说，这样的开发方式会让代码越来越混乱，最终导致无法维护。

如果我们在项目中，应用基于充血模型的 DDD 的开发模式，那对应的开发流程就完全不一样了。在这种开发模式下，我们需要事先理清清楚所有的业务，定义领域模型所包含的属性和方法。领域模型相当于可复用的业务中间层。新功能需求的开发，都基于之前定义好的这些领域模型来完成。

我们知道，越复杂的系统，对代码的复用性、易维护性要求就越高，我们就越应该花更多的时间和精力在前期设计上。而基于充血模型的 DDD 开发模式，正好需要我们前期做大量的业务调研、领域模型设计，所以它更加适合这种复杂系统的开发。

## 重点回顾

今天的内容到此就讲完了，我们来一起回顾一下，你应该掌握的重点内容。

我们平时做 Web 项目的业务开发，大部分都是基于贫血模型的 MVC 三层架构，在专栏中我把它称为传统的开发模式。之所以称之为“传统”，是相对于新兴的基于充血模型的 DDD 开发模式来说的。基于贫血模型的传统开发模式，是典型的面向过程的编程风格。相反，基于充血模型的 DDD 开发模式，是典型的面向对象的编程风格。

不过，DDD 也并非银弹。对于业务不复杂的系统开发来说，基于贫血模型的传统开发模式简单够用，基于充血模型的 DDD 开发模式有点大材小用，无法发挥作用。相反，对于业务复杂的系统开发来说，基于充血模型的 DDD 开发模式，因为前期需要在设计上投入更多时


间和精力，来提高代码的复用性和可维护性，所以相比基于贫血模型的开发模式，更加有优势。

## 课堂讨论

今天课堂讨论的话题有两个。

1. 你做经历的项目中，有哪些是基于贫血模型的传统开发模式？有哪些是基于充血模型的 DDD 开发模式呢？请简单对比一下两者的优劣。
2. 对于我们举的例子中，UserEntity、UserBo、UserVo 包含的字段都差不多，是否可以合并为一个类呢？

欢迎在留言区写下你的答案，和同学一起交流和分享。如果有收获，也欢迎你把这篇文章分享给你的朋友。

点击参加小程序学习打卡 

## 8个月，攻克设计模式



扫一扫参与小程序打卡



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 10 | 理论七：为何说要多用组合少用继承？如何决定该用组合还是继承？

下一篇 12 | 实战一（下）：如何利用基于充血模型的DDD开发一个虚拟钱包系统？



## 精选留言 (126)

写留言



业余爱好者

2019-11-27

一直贫血而不自知

展开 ∨

3

51



乐

2019-11-27

## 为什么贫血模型盛行

下面几项自己都中过招（环境问题和个人问题）：

### 环境问题 ##...

展开 ∨

5

46



墨雨

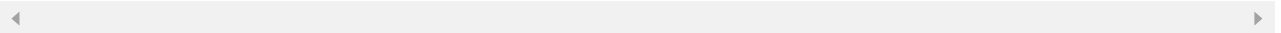
2019-11-27

老师，我平常做web开发都是，entity,dao,service,controller.对vo,bo不是很理解，也没有用到。有没有demo呢？

展开 ∨

作者回复: 我抽空写个demo，放到我的github上吧

<https://github.com/wangzheng0822>



6

14



ʘ(εʘ)

2019-11-27

个人感觉业务被贫血模型绑架的另一个原因是以前缓存nosql这些技术不成熟 刚毕业那会哪有什么redis，机器的内存也不多。都是公司堆在角落的旧机器。一些业务如果在domain里实现可能会hold住数据库中的大部分数据。所以业务上都需要翻译成sql的where和join来减少网络和内存的开销。功能都被sql抢了去，想充血也充不起来。现在随便开个项目不带个redis老板都会质疑一下。mysql的访问也都是能少就少，不行再多加几台云服务...

展开 ∨

2

13



有铭

2019-11-27

我个人认为，充血模型在web开发领域不流行的一个根本原因，在于互联网兴起后各种层出不穷的需求变动，以及短命的项目生存周期，充血模型应对复杂业务确实很有优势，但是这是建立在复杂业务本身其实相对稳定上，比如银行的业务，虽然复杂，但是其实很稳定。但是要是换在互联网，今天改需求明天改需求，甚至很多时候根本就是推倒了重来的需求，充血模型面对这种状态，根本是力不从心的

展开 ▾

💬 3

👍 12



李小四

2019-11-27

设计模式\_10

# 问题:

- 1. 做的Android项目更多，Android开发也是经历了MVC==>MVP(依然是一种MVC架构)==>MVVM的模式演进。类MVC模式比较多，在UI相关的开发中，只用过贫血模式(之前也尝试过使用充血模式，但考虑到不一致带来的成本就放弃了)；在UI无关的复杂服务...

展开 ▾

💬

👍 9



lizi

2019-11-27

沙发，不睡觉，听课。哈哈，加班好累，

展开 ▾

💬

👍 9



grey927

2019-11-27

能否用代码表达一下充血模型，其实还是不太理解

展开 ▾

作者回复: 下一节课有的



💬

👍 6



小晏子

2019-11-27

基本上经历过的web项目都是基于贫血模型开发模式的，entity，bo，vo不能放在一个类里，每个对象的应用场景不同，entity是映射数据库字段的，bo，vo适合业务和展示相关的，而且entity相对来讲变化不多，bo，vo可能会频繁变化，所以不适合放在同一个类里

展开 ▾



6



花儿少年

2019-11-27

有本书叫ddd 原则，模式与实践可以翻阅一下

首先要明白一点ddd适用范围，多数业务就是CRUD就可以搞定，理解起来也没有困难，为啥不继续用贫血模型。

充血模型就像老师说的需要精心设计，以应对变化，如果没有一个复杂的业务场景就根本用不到，或者说用起来很难受是一个似是而非的东西。...

展开 ▾



4



Lrwin

2019-11-27

我觉得代码架构和业务架构一样，只要将关注点分离就可以。

简单的系统，困难的不在于领域的拆分，而在于时间成本的控制。从软件工程角度考虑，时间，成本，范围三角理论可以进行分析、。

我们所说的复杂系统，更看重业务的复杂度，将复杂度降低的方法则是分而治之。这样可以降低复杂度。...

展开 ▾



3



Geek\_Zjy

2019-11-27

看到「领域驱动设计有点儿类似敏捷开发、SOA、PAAS 等概念，听起来很高大上，但实际上只值“五分钱”。」时，不知道引起了多少人的共鸣，O(∩\_∩)O~。做技术的本身就经常会遇到沟通问题，一些人还总喜欢“造概念”，唯恐别人听懂了，争哥这句话无疑说中了我们的心坎儿。

当然我这里也不是说 DDD 不好（看后面的争哥也没这个意思），但是每个理论都有自己...

展开 ▾

作者回复: 哈哈，多谢认可，我写这篇文章的时候，还害怕搞DDD的人会来骂我，看来是我多虑了。隔壁的DDD课程可以去学下，管它是不是我写的，看看他咋“吹”的也好。



3



梦倚栏杆

2019-11-27

第一个还没有太多的感受，还需要时间来练习感受

第二个是否合成一个各有优劣，可能还是和写代码人的功底有关：

拆开的优势：各层的防腐隔离，当前层的变化不影响其他层。

拆开的劣势：来一个迭代需求，比如需要加一下邮箱等，rd很有可能在三个类里各加一个字段，从上改到下，完全看不出隔离的优势，就看不到了一层层的类转换...

展开 ▾



3



杨树敏

2019-11-28

回想起来经历的互联网项目一直都是贫血模型，究其原因：

1.互联网项目多采用敏捷开发，需求多变，设计充血模型的性价比低；

2.互联网小产品业务场景相对简单，业务难点更多的出现在并发，性能上，在存储，内存成本急剧下降的当下，粗暴的横向扩展成为见效更快的选择。

展开 ▾



2



\_呱太\_

2019-11-28

小争哥，对设计模式这些概念都有了解，一直想找一些源码来看看顺便练下手，请问下有没有推荐的 C++ 比较经典的开源代码呢

展开 ▾



2



丶 Zero...

2019-11-28

我们公司的ERP系统使用的是贫血模型，支付系统由一个OOP的忠实践行者设计的，使用的是充血模型。两个系统我都有在维护，先说ERP。遇到新需求的时候，就像老师说的SQL驱动。从后往前返回数据。简单的需求还好，像一些复杂的模块。看着service一个方法动辄几百，甚至出现过上千行的。真心感觉改不动。在业务还比较简单的时候贫血模型还够用，但随着业务发展，service层越来越重。这时还不做封装抽象，系统真心不好维护。...

展开 ▾



2



Lonely绿豆蛙

2019-11-28

最近边看边重构自己的项目，感觉真的是从码农视角转到了架构师层次~



2



编程界的小学生

2019-11-28

- 1.一直贫血模型，却没有发现。
- 2.充血模型我有点没看懂，我粗糙的理解成把贫血模型的service里面的复杂业务逻辑都搬到了domain，然后service简单调用domain即可。那domain里不也是业务逻辑+bo.吗？还有复杂的sql无法复用问题，我一般写业务都是单表。然后放到service层调用多个dao去组装最终的数据出来。这样可以解决乱七八糟的sqlz吗？希望争哥解答下。...

展开 ∨



👍 2



**深度·仁**

2019-11-28

一拍大腿，靠，说到心坎里去了，各种细碎的sql，就为了解决某个小功能！业务熟悉，领域驱动设计就是屠龙刀，业务不熟悉，DDD也就值个半毛钱！茶不思，饭不想，期待后面的文章更新



👍 2



**默**

2019-11-27

我经历的所有的项目都是贫血模型，对于充血模型没尝试过，如果说业务数据与业务操作分离，那么在设计接口时如何将繁多的业务属性当成参数传入给接口方法呢？是说将所有属性的getter和setter直接生成在接口中吗？那么对于接口中使用这些零散的属性是怎样使用的呢？觉得二者确实还得从实际情况出发，看业务复杂度及对代码设计是否有要求，但往往都是为了习惯、速度而忽略了设计初心。...

展开 ∨

💬 1

👍 2