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？

王争 · 设计模式之美

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的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

```
////////// Controller+VO(View Object) //////////
public class UserController {
  private UserService userService; //通过构造函数或者IOC框架注入

  public UserVo getUserById(Long userId) {
    UserBo userBo = userService.getUserById(userId);
    UserVo userVo = [...convert userBo to userVo...];
    return userVo;
  }
}

public class UserVo {//省略其他属性、get/set/construct方法
  private Long id;
  private String name;
  private String cellphone;
}

////////// Service+BO(Business Object) //////////
public class UserService {
  private UserRepository userRepository; //通过构造函数或者IOC框架注入

  public UserBo getUserById(Long userId) {
    UserEntity userEntity = userRepository.getUserById(userId);
    UserBo userBo = [...convert userEntity to userBo...];
    return userBo;
  }
}

public class UserBo {//省略其他属性、get/set/construct方法
  private Long id;
  private String name;
  private String cellphone;
}

////////// Repository+Entity //////////
public class UserRepository {
  public UserEntity getUserById(Long userId) { //... }
}

public class UserEntity {//省略其他属性、get/set/construct方法
  private Long id;
  private String name;
  private String cellphone;
}
```

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“五分钱”。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的开发模式，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的开发模式，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的开发模式，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的开发模式？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。

AI智能总结

本文探讨了贫血模型和充血模型在MVC架构中的应用及其对面向对象编程的影响。贫血模型将数据与操作分离，破坏了面向对象的封装特性，而充血模型则将数据和对应的业务逻辑封装到同一个类中，满足面向对象的封装特性。文章介绍了基于充血模型的领域驱动设计（DDD）开发模式，并探讨了其在微服务架构中的应用。同时，文章分析了基于贫血模型的传统开发模式受欢迎的原因，包括业务简单、充血模型设计难度大以及转型成本等因素。总的来说，本文通过对MVC架构、贫血模型和充血模型的解释，帮助读者更好地理解业务系统开发中常用的开发模式，以及贫血模型对面向对象编程风格的影响。文章重点强调了基于充血模型的DDD开发模式适用于复杂业务系统开发，需要在前期设计上投入更多时间和精力，以提高代码的复用性和可维护性。同时，课堂讨论部分提出了对比基于贫血模型和充血模型的优劣以及领域模型设计的问题。整体而言，本文对于读者快速了解贫血模型和充血模型在MVC架构中的应用及其对面向对象编程的影响提供了清晰的概览。

全部留言 (285)

最新　精选

倡印

2020-07-21

小时候妈妈说我贫血 ， 长大了才知道我真的贫血

作者回复: 你这是贫嘴，不是贫血

共 3 条评论>

69

lmdcx

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的局限性和适用性，看很多文章在讲一些理论时，总是恨不得把自己的理论（其实也算不得自己的）吹成银弹，态度上就让人很难接受。
我还是喜欢争哥的风格，逻辑很清晰，也很严谨，很务实。

关于老师的问题。
说句实话，我们就没有写过充血模型的代码。
我们会把 UserEntity、UserBo 混着用， UserBo 和 UserVo 之间转换时有时还会用 BeanUtils 之类的工具 copy 。
对于复杂的逻辑，我们就用复杂 SQL 或者 Service 中的代码解决。

不过我在翻一些框架时，比如 Java 的并发包时不可避免的需要梳理 Lock、Condition、Synchronizer 之间的关系。比如看 Spring IOC 时，也会需要梳理围绕着 Context 、 Factory 展开的很多类之间的关系。
就好像你要“混某个圈子”时，就不可避免的“拜码头”，认识一堆“七大姑八大姨”，然后你才能理解整个“圈子”里的关系和运转逻辑。
我也经常会有疑问， DDD 和面向对象究竟是什么关系，也会猜想：是不是面向对象主要关注“圈子”内的问题，而 DDD 主要关注“圈子”之间的问题？有没有高手可以回答一下。
（其实我最近一直都想订隔壁DDD的课，但是考虑到精力的问题，以及担心学不会，主要不是争哥讲O(∩_∩)O~，所以没下手）

作者回复: 哈哈，多谢认可，我写这篇文字的时候，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。

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

作者回复: 下一节课有的

9

DullBird
2019-11-27

1.目前基本都在接触贫血开发模型，充血的可能局部模块设计的时候，会把数据和方法组织到一个类里面去。但是DB的操作完全隔离。
这里有一个问题:充血模型的话，OOP的想法，应该是每个人(假设人是类，具体的人就是人这个类的实例化)管理自己的属性，比如我的主管。
这个时候有一个需求。批量修改人员的主管。那么充血模型是要遍历委托给每个具体的人自己去修改呢？还是提供一个service，直接批量操作DB。
2. entity,bo,vo我的做法是不合并，但是真的有贯穿三层的模型。那么就直接用一个。但是要单独分包。并且组内规范好这个包里面的东西都是有修改风险的。我个人倾向用麻烦换容错。毕竟软件的变化性比较大

作者回复: 1. 充血模型并不是哪都适用
2. 赞成你的看法

9

追风少年
2019-11-27

1. 以前做的项目都是基于贫血模型的，这次的话涉及风控业务，也是基于贫血模型，但是各种问题不断，正在考虑优化，这里刚好看到老师的文章，希望能有所借鉴。
2. Entity是ORM中数据库映射的实体类，BO是业务操作相关实体类，VO是视图层对应实体类。在简单情况下，这三个类可能是一样的，比方说你填写一个登陆注册的表单，此时前端传给后端接口的数据，一般就是VO，而通过业务层Service操作，加入创建时间，IP地址等，就转换成了BO，最后对应到数据层就转换为了Entity，也许一次注册可能需要写多个库，就会生成多个Entity。
有些复杂业务，还有DO,DTO，PO之类的概念，但是个人感觉很模糊，也不是很了解。这里希望老师能指点一下。

作者回复: DTO：data transfer object，是一种更抽象的概念，这种数据类型可以是贫血模型的，主要是用在接口之间传递数据。

其他的两个没听说过：《

共 2 条评论>

8

饭太司替可
2019-12-19

项目中用到了google的ProtocolBuffer，根据数据结构体生成的类模型只能包含数据，不能包含方法。这种情况也是贫血模型吧。

作者回复: 是的

7

安静
2019-11-27

要是有代码例子就好了。实操性会强很多。

作者回复: 一看就是没认真看文章，文章说了例子在下一节课中有的

4

阿卡牛
2019-11-27

最近在看消息队列的专栏，里面有提到Pulsar这个产品采用了存储与计算分离的设计。本质上和文中提到的数据与操作分离应该是一个意思吧？难道也是一种面向过程的设计

作者回复: 存储本身有自己的逻辑在那里面，不能单独的看做是数据。

共 2 条评论>

3

迁橘
2019-11-27

看完了，感觉热血沸腾，特别期待下一节课.
课堂讨论:

1, 自己所参与做的项目中都是典型的基于贫血模型开发模式.
2, 我是基本都用一个类的, 因为所做的系统业务想比较简单, 也就没必要, 还有些共用的属性字段会拿出来,用继承的方式. 基本项目都是这么过来的. 也没遇到啥问题, (大家勿笑哈)

从第一节课听到现在, 受益匪浅, 每节课都会听个3-5遍. 到现在, 基本能意识到自己在工作种存在的一些问题,以及需要提升进步的地方, 期待后面的课程....

作者回复: （づ￣3￣）づ╭❤～

3

十二差一点
2019-11-27

MVC是面向过程编程，是因为它违反了封装的特性，数据和逻辑操作分离开了，在controller进行相关数据逻辑操作，而model仅仅只是个数据层，没有任何操作。而MVVM是面向对象编程，因为它把数据和其相关逻辑操作封装在了viewModel，只暴露给外部相关方法，controller想要获取数据直接通过这些方法就行了，不用像MVC在controller层进行一堆逻辑操作，同时减轻了controller的代码，在viewModel也方便维护数据逻辑操作。不知道这样的理解对不对?

作者回复: 恩恩 可以这么理解

共 3 条评论>

3