

## 08 | MVC架构解析：模型（Model）篇

2019-09-27 四火

全栈工程师修炼指南

[进入课程 >](#)



讲述：四火

时长 17:04 大小 15.63M



你好，我是四火。

在上一讲中，我们了解了 MVC 这个老而弥坚的架构模式，而从这一讲开始，连同第 09、10 讲共计 3 篇，我将分别展开介绍 MVC 三大部分内容。今天我要讲的就是第一部分——模型（Model）。

### 概念

首先我们要了解的是，我们总在谈“模型”，那到底什么是模型？

简单说来，**模型就是当我们使用软件去解决真实世界中各种实际问题的时候，对那些我们关心的实际事物的抽象和简化。**比如我们在软件系统中设计“人”这个事物类型的时候，通常

只会考虑姓名、性别和年龄等一些系统用得着的必要属性，而不会把性格、血型和生辰八字等我们不关心的东西放进去。

更进一步讲，我们会谈领域模型（Domain Model）。“领域”两个字显然给出了抽象和简化的范围，不同的软件系统所属的领域是不同的，比如金融软件、医疗软件和社交软件等等。如今领域模型的概念包含了比其原本范围定义以外更多的内容，**我们会更关注这个领域范围内各个模型实体之间的关系。**

MVC 中的“模型”，说的是“模型层”，它正是由上述的领域模型来实现的，可是当我们讲这一层的时候，它包含了模型上承载的实实在在的业务数据，还有不同数据间的关联关系。因此，**我们在谈模型层的时候，有时候会更关心领域模型这一抽象概念本身，有时候则会更关心数据本身。**

## 贫血模型和充血模型

第一次听到“贫血模型”“充血模型”这两个词的时候，可能你就会问了，什么玩意儿？领域模型还有贫血和充血之说？


其实，这两兄弟是 Martin Fowler 造出来的概念。要了解它们，得先知道这里讲的“血”是什么。

这里的“血”，就是逻辑。它既包括我们最关心的业务逻辑，也包含非业务逻辑。因此，**贫血模型（Anemic Domain Model），意味着模型实体在设计和实现上，不包含或包含很少的逻辑。**通常这种情况下，逻辑是被挪了出去，由其它单独的一层代码（比如这层代码是“Service”）来完成。

严格说起来，贫血模型不是面向对象的，因为对象需要数据和逻辑的结合，这也是贫血模型反对者的重要观点之一。如果主要逻辑在 Service 里面，这一层对外暴露的接口也在 Service 上，那么事实上它就变成了面向“服务”的了，而模型实体，实际只扮演了 Service API 交互入参出参的角色，或者从本质上说，它只是遵循了一定封装规则的容器而已。

**这时的模型实体，不包含逻辑，但包含状态，而逻辑被解耦到了无状态 Service 中。**既然没有了状态，Service 中的方法，就成为过程式代码的了。请注意，不完全面向对象并不代表它一定“不好”，事实上，在互联网应用设计中，贫血模型和充血模式都有很多成功的使用案例，且非常常见。


比如这样的一个名为 Book 的类：

 复制代码

```
1 public class Book {
2     private int id;
3     private boolean onLoan;
4
5     public int getId() {
6         return this.id;
7     }
8     public void setId(int id) {
9         this.id = id;
10    }
11    public boolean isOnLoan() {
12        return this.onLoan;
13    }
14    public void setOnLoan(boolean onLoan) {
15        this.onLoan = onLoan;
16    }
17 }
```

你可以看到，它并没有任何实质上的逻辑在里面，方法也只有简单的 getters 和 setters 等属性获取和设置方法，它扮演的角色基本只是一个用作封装的容器。

那么真正的逻辑，特别是业务逻辑在哪里呢？有这样一个 Service：


 复制代码

```
1 public class BookService {
2     public Book lendOut(int bookId, int userId, Date date) { ... }
3 }
```

这个 lendOut 方法表示将书从图书馆借出，因此它需要接收图书 id 和 用户 id。在实现中，可能需要校验参数，可能需要查询数据库，可能需要将从数据源获得的原始数据装配到返回对象中，可能需要应用过滤条件，这里的内容，就是逻辑。

现在，我们再来了解一下充血模型（Rich Domain Model）。**在充血模型的设计中，领域模型实体就是有血有肉的了，既包含数据，也包含逻辑，具备了更高程度的完备性和自恰**

性，并且，充血模型的设计才是真正面向对象的。在这种设计下，我们看不到 XXXService 这样的类了，而是通过操纵有状态的模型实体类，就可以达到数据变更的目的。

 复制代码

```
1 public class Book {
2     private int id;
3     private boolean onLoan;
4     public void lendOut(User user, Date date) { ... }
5     ... // 省略属性的获取和设置方法
6 }
```

在这种方式下，lendOut 方法不再需要传入 bookId，因为它就在 Book 对象里面存着呢；也不再需要传出 Book 对象作为返回值，因为状态的改变直接反映在 Book 对象内部了，即 onLoan 会变成 true。

也就是说，Book 的行为和数据完完全全被封装的方法控制起来了，中间不会存在不应该出现的不一致状态，因为任何改变状态的行为只能通过 Book 的特定方法来进行，而它是可以被设计者严格把控的。

而在贫血模型中就做不到这一点，一是因为数据和行为分散在两处，二是为了在 Service 中能组装模型，模型实体中本不该对用户开放的接口会被迫暴露出来，于是整个过程中就会存在状态不一致的可能。

但是请注意，**无论是充血模型还是贫血模型，它和 Model 层做到何种程度的解耦往往没有太大关系**。比如说这个 lendOut 方法，在某些设计中，它可以拆分出去。对于贫血模型来说，它并非完全属于 BookService，可以拿到新建立的“借书关系”的服务中去，比如：

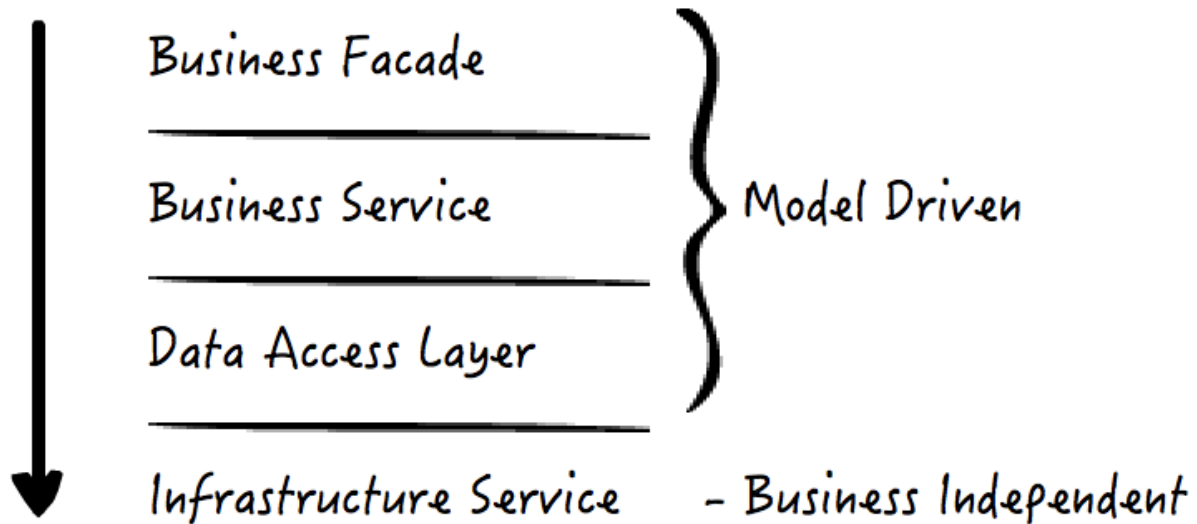
 复制代码

```
1 public class LoanService {
2     public Loan add(int bookId, int userId, Date date) { ... }
3 }
```

这样一来，借书的关系就可以单独维护了，借书行为发生的时候，Book 和 User 两个实体对应的数据都不需要发生变化，只需要改变这个借书关系的数据就可以了。对于充血模型来说，一样可以做类似拆分。

## 内部层次划分

软件的耦合和复杂性问题往往都可以通过分层解决，模型层内部也一样，但是我们需要把握其中的度。**层次划分过多、过细，并不利于开发人员严格遵从和保持层次的清晰，也容易导致产生过多的无用样板代码，从而降低开发效率。**下面是一种比较常见的 Model 层，它是基于贫血模型的分层方式。



在这种划分方式下，每一层都可以调用自身所属层上的其它类，也可以调用自己下方一层的类，但是不允许往上调用，即依赖关系总是“靠上面的层”依赖着“靠下面的层”。最上面三层是和业务模型实体相关的，而最下面一层是基础设施服务，和业务无关。从上到下，我把各层依次简单介绍一下：

第一层 Facade，提供粗粒度的接口，逻辑上是对 Service 功能的组合。有时候由于事务需要跨多个领域模型的实体控制，那就适合放在这里。举例来说，创建用户的时候，我们同时免费赠送一本电子书给用户，我们既要调用 UserService 去创建用户对象，也要调用 SubscriptionService 去添加一条订购（赠送）记录，而这两个属于不同 Service 的行为需要放到一处 Facade 类里面做统一事务控制。在某些较小系统的设计里面，Service 和 Facade 这两层是糅合在一起的。

第二层 Service，前面已经介绍了，通常会存放仅属于单个领域模型实体的操作。

第三层数据访问层，在某些类型的数据访问中需要，比如关系型数据库，这里存放数据库字段和模型对象之间的 ORM（Object-Relational Mapping，对象关系映射）关系。


第四层基础设施层，这一层的通用性最好，必须和业务无关。某些框架会把基础设施的工作给做了，但有时候也需要我们自己实现。比如 S3Service，存放数据到亚马逊的分布式文件系统。

## CQRS 模式

你也许听说过数据库的读写分离，其实，在模型的设计中，也有类似读写分离的机制，其中最常见的一种就叫做 CQRS（Command Query Responsibility Segregation，命令查询职责分离）。


一般我们设计的业务模型，会同时被用作读（查询模式）和写（命令模式），但是，实际上这两者是有明显区别的，在一些业务场景中，我们希望这两者被分别对待处理，那么这种情况下，CQRS 就是一个值得考虑的选项。

为什么要把命令和查询分离？我举个例子来说明吧，比如这样的贫血模型：

 复制代码

```
1 class Book {
2     private long id;
3     private String name;
4     private Date publicationDate;
5     private Date creationDate;
6     ... // 省略其它属性和 getter/setter 方法
7 }
```

那么，相应地，就有这样一个 BookService：

 复制代码

```
1 class BookService {
2     public Book add(Book book);
3     public Pagination<Book> query(Book book);
4 }
```

这个接口提供了两个方法：



一个叫做 add 方法，接收一个 book 对象，这个对象的 name 和 publicationDate 属性会被当做实际值写入数据库，但是 id 会被忽略，因为 id 是数据库自动生成的，具备唯一性，creationDate 也会被忽略，因为它也是由数据库自动生成的，表示数据条目的创建时间。写入数据库完成后，返回一个能够反映实际写入库中数据的 Book 对象，它的 id 和 creationDate 都填上了数据库生成的实际值。

你看，这个方法，实际做了两件事，一件是插入数据，即写操作；另一件是返回数据库写入的实际对象，即读操作。

另一个方法是 query 方法，用于查询，这个 Book 入参，被用来承载查询参数了。比方说，如果它的 author 值为 “Jim”，表示查询作者名字为 “Jim” 的图书，返回一个分页对象，内含分页后的结果列表。

但这个方法，其实有着明显的问题。这个问题就是，查询条件的表达，并不能用简单的业务模型很好地表达。换言之，这个模型 Book，能用来表示写入，却不适合用来表示查询。为什么呢？

比方说，你要查询出版日期从 2018 年到 2019 年之间的图书，你该怎么构造这个 Book 对象？很难办对吧，因为 Book 只能包含一个 publicationDate 参数，这种“难办”的本质原因，是模型的不匹配，即这个 Book 对象根本就不适合用来做查询调用的模型。

在清楚了问题以后，解决方法 CQRS 就自然而然产生了。**简单来说，CQRS 模式下，模型层的接口分为且只分为两种：**

**命令 (Command)，它不返回任何结果，但会改变数据的状态。**


**查询 (Query)，它返回结果，但是不会改变数据的状态。**

也就是说，它把命令和查询的模型彻底分开了。上面的例子，使用 CQRS 的方式来改写一下，会变成这样：

 复制代码

```
1 class BookService {
2     public void add(Book book);
3     public Pagination<Book> query(Query bookQuery);
4 }
```

你看，在 add 操作的时候，不再有返回值；而在 query 操作的时候，入参变成了一个 Query 对象，这是一个专门的“查询对象”，查询对象里面可以放置多个查询条件，比如：

 复制代码

```
1 Query bookQuery = new Query(Book.class);
2 query.addCriteria(Criteria.greaterThan("publicationDate", date_2018));
3 query.addCriteria(Criteria.lessThan("publicationDate", date_2019));
```

读到这里，不知道你有没有联想到这样两个知识点：

第一个知识点，在 [\[第 04 讲\]](#) 我们学习 REST 风格的时候，我们把 HTTP 的请求从两个维度进行划分，是否幂等，以及是否安全。按照这个角度来考量，CQRS 中的命令，可能是幂等的（例如对象更新），也可能不是幂等的（例如对象创建），但一定是不安全的；CQRS 中的查询，一定是幂等的，且一定是安全的。

第二个知识点，在 [\[第 07 讲\]](#) 我们学习 MVC 的一般化，其中的“第二种”典型情况时，Controller 会调用 Model 层的，执行写入操作；而 View 层会调用 Model 层，执行只读操作——看起来这不就是最适合 CQRS 的一种应用场景吗？

所以说，技术确实都是相通的。

## 总结思考

今天我们详细剖析了 MVC 架构中 Model 层的方方面面，并结合例子理解了贫血模型和充血模型的概念和特点，还介绍了一种典型的模型层的内部层次划分方法，接着介绍了 CQRS 这种将命令和查询行为解耦开的模型层设计方式。

其中，贫血模型和充血模型的理解是这一讲的重点，这里我再强调一下：

贫血模型，逻辑从模型实体中剥离出去，并被放到了无状态的 Service 层中，于是状态和逻辑就被解耦开了；

充血模型，它既包含数据，也包含逻辑，具备了更高程度的完备性和自恰性。



最后，我来提两个问题，一同检验下今天的学习成果吧：

请回想一下，在你经历过的项目中，使用过什么 MVC 框架，Model 层的代码是遵循贫血模型还是充血模型设计的？

在文中应用 CQRS 模式的时候，add 方法不再返回 Book 对象，这样一来，方法调用者就无法知道实际插入的 Book 对象的 id 是什么，就无法在下一步根据 id 去数据库查询出这个 Book 对象了。那么，这个问题该怎么解决呢？

好，今天的内容就到这里，有余力还可以了解下扩展阅读的内容。有关今天的知识点，如果你在实际的工作经历中遇到过，其实是非常适合拿来一起比较的。可以谈谈你在这方面的经验，也可以分享你的不同理解，期待你的想法！

## 扩展阅读

[AnemicDomainModel](#)，Martin Fowler 写的批评贫血模型的文章，他自己提出了贫血和充血的概念，因此我们可以到概念的源头去，看看他做出批评的理由是什么。

【基础】在模型层我们经常会和数据库打交道，SQL 是这部分的基础，如果你完全不了解 SQL，可以阅读 W3school 上的 [SQL 基础教程](#)（左侧目录中的基础教程，内容简短）。

文中提到了查询对象（Query Object），这其实是一种常见的设计模式，文中举例说明了是怎么使用的，但是，如果你想知道它是怎么实现的，可以阅读 [Query Object Pattern](#) 这篇文章，上面有很好的例子。

---

# 全栈工程师修炼指南

从全栈入门到技能实战

熊燚

Oracle 首席软件工程师



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 07 | 解耦是永恒的主题：MVC框架的发展

下一篇 09 | MVC架构解析：视图 (View) 篇

## 精选留言 (11)

写留言



Luciano李鑫 置顶

2019-09-29

- 1.大型、复杂项目用贫血模型多一些，小型、简单项目用重写模型多一些。
- 2.在命令查询的时候，针对add可以返回自增id。在update的时候返回修改记录数。

作者回复: 关于 2，你说的返回自增 id 是一种很好的方法，很常用，但是它破坏了 CQRS 对于 Command 的 fire-and-forget 的要求。

当然，也有一些其它的办法，比如将 id 在客户端生成（这个生成可以是客户端自己生成，也可以是调用服务端的某一个生成 id 的接口生成），但这却又破坏了 REST 接口中 Create 方法不指定 id 的规约。

再有一个办法，是在客户端生成一个非主键、但被索引的特殊 id，例如 GUID，这样，生成了记录之后，客户端可以使用这个 GUID 去服务端获取这条数据，当然，缺点是需要额外的一列来放

置 GUID，且要保证 GUID 的唯一性。

所以，上面介绍了三种方法，各有利弊，都比较常用。当然，也有一些其它的方法。



tt

2019-09-27

赞，这是我订阅的极客时间的课程中为数不多的重度偏向概念与逻辑的课程。

我本人很少做WEB开发，只是使用过面向对象，这节课一下子就让我领悟了面向对象、面向服务、面向过程、贫血和充血模型。

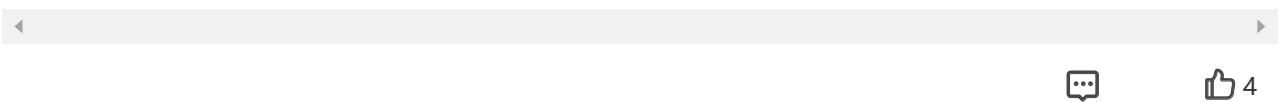
...

展开 ▾

作者回复: 感谢回复。

关于你提到的“概念与逻辑”，做个说明。

我是这样认为的，学习全栈技术比较忌讳仅仅学习单个的具体技术，之前的文章和回复中我也多次提到过，毕竟技术种类花样繁多，我们还是需要适当做一些抽象，理解一些通用和共性的东西，既包括一些概念，也包括一些套路（模式）。当然，我们需要通过许多例子来理解它们，这是没错的。



Luciano李鑫

2019-09-29

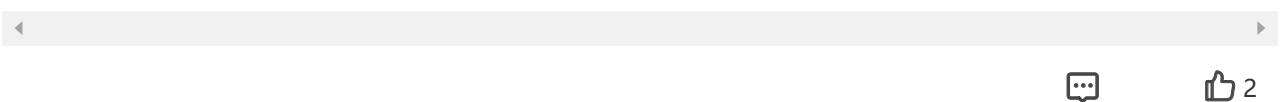
请问，为什么说在贫血模型中，service增是无状态的，而model增是有状态的，这个“状态”是怎么定义的呢？

展开 ▾

作者回复: 状态，其实就是数据。

Service 提供了一系列的过程方法，入参进，结果出，但是 Service 并未发生变化。

Model 则相反，可以创建、修改、删除，这就是状态的变化。





Kada

2019-09-28

这一课程很棒。如果有可运行的github代码示例，动手试几次，会方便理解很多。

自己实现不是不行，但肯定实现过程中遇到其它问题，就跑偏了，速度会慢，效率会低。

展开 ▾



1



桃源小盼

2019-09-27

非关系型数据库的model层，又该怎么设计呢？

展开 ▾



1



每天晒白牙

2019-09-30

MVC架构Model层设计时，有贫血模型和充血模型之分

贫血模型:把逻辑从模型实体中剥离出去，放到无状态的service层中，达到状态和逻辑的解耦(我经历过的Model层设计一般都是这种贫血模型)

...

展开 ▾



编程爱好者

2019-09-28

1.两者都有过，我个人更喜欢贫血模型，职责分明些。  
2.既然不知道，可以让服务器来告诉你。类似于tcp中的syn，ack机制，可以为这次请求增加一个消息编号，然后服务器可以通过这个消息编号告诉方法调用者。  
思考作者为什么这么设计课程，感觉很多内容是架构设计里面涉及的内容-权衡与取舍。

展开 ▾

作者回复: 关于第 2 问，消息编号的想法很好，但是，消息编号是消息编号，而写入数据库对象的 id 是对象的 id，二者并没有必然联系啊。



我叫徐小晋

2019-09-27

老师您好：

第一个问题：基本上都是使用贫血模型

第二个问题：是不是可以通过查询，然后通过id排序就可以得到最近的数据呢？

希望老师指正

展开 ▾

作者回复: 第二个问题不对，因为通过 id 排序的方式，在数据量较大的时候不可行；而且，即便能排序，你也不能保证一定能找到那个你刚插入的数据（考虑两条数据并发写入的情况）。这个问题你可以看一下其它人的答复和回复



**靠人品去赢**

2019-09-27

看了这个，我一下理解我们框架提供服务访问的模块为什么命名叫facade。



**anginiit**

2019-09-27

最近几个项目都是springMvc，使用贫血模式

展开 ▾



**許敲敲**

2019-09-27

前端框架AngularJS遵循的是MVC 模式

展开 ▾

