



下载APP



开篇词 | 这一次，我们从“丑”代码出发

2020-12-28 郑晔

代码之丑

[进入课程 >](#)



讲述：郑晔

时长 09:16 大小 8.50M



你好，我是郑晔！我又回来了！


我在“极客时间”里已经写了两个专栏，分别是《[10x 程序员工作法](#)》和《[软件设计之美](#)》，从工作原则和设计原则两个方面对软件开发的各种知识进行了探讨，帮助你搭建了一个开启程序员精进之路的框架。

不过，无论懂得多少道理，程序员依然要回归到写代码的本职工作上。所以，这次我准备和你从代码的坏味道出发，一起探讨如何写代码。



千里之堤毁于蚁穴

为什么要讲这个话题，就让我们先从一次代码评审讲起。在一次代码评审中，我注意到了这样一段代码：

 复制代码

```
1 public void approve(final long bookId) {  
2     ...  
3     book.setReviewStatus(ReviewStatus.APPROVED);  
4     ...  
5 }
```

这是在一个服务类里面写的，它的主要逻辑就是从仓库中找出一个作品，然后，将它的状态设置为审核通过，再将它存回去。前后的代码都属于常规的代码，但是，设置作品评审状态的代码引起了我的注意，于是有了下面这段对话。

我：这个地方为什么要这么写？


同事：我要将作品的审核状态设置为审核通过。

我：这个我知道，但为什么要在这里写 setter 呢？

同事：你的意思是？

我：这个审核的状态是作品的一个内部状态，为什么服务需要知道它呢？也就是说，这里通过 setter，将一个类的内部行为暴露了出来，这是一种破坏封装的做法。

同事被我说动了，于是这段代码变成了下面这个样子：

 复制代码

```
1 public void approve(final long bookId) {  
2     ...  
3     book.approve();  
4     ...  
5 }
```

之所以我注意到这段代码，完全是因为这里用到了 setter。在我看来，setter 就是一个坏味道，每次一看到 setter，我就会警觉起来。

setter 的出现，是对于封装的破坏，它把一个类内部的实现细节暴露了出来。我在《软件设计之美》中讲过，面向对象的封装，关键点是行为，而使用 setter 多半只是做了数据的聚合，缺少了行为的设计，这段代码改写后的 approve 函数，就是这里缺少的行为。

再扩展一步，setter 通常还意味着变化，而我在《软件设计之美》中讲函数式编程时也说过，一个好的设计应该尽可能追求不变性。所以，setter 也是一个提示符，告诉我们，这个地方的设计可能有问题。

你看，一个小小的 setter，背后却隐藏着这么多的问题。而所有这些问题，都会让代码在未来的日子变得更加不可维护，这就是软件团队陷入泥潭的开始。

我也一直和我团队的同学说，“写代码”有两个维度：正确性和可维护性，不要只关注正确性。能把代码写对，是每个程序员的必备技能，**但能够把代码写得更具可维护性，这是一个程序员从业余迈向职业的第一步。**

将坏味道重构为整洁代码

或许你也认同代码要有可维护性，也看了很多书，比如《[🔗 程序设计实践](#)》《[🔗 代码整洁之道](#)》等等，这些无一不是经典中的经典，甚至连怎么改代码，都有《[🔗 重构](#)》等着我们。没错，这些书我都读过，也觉得从中受益匪浅。

不过，回到真实的工作中，我发现了一个无情的事实：**程序员们大多会认同这些书上的观点，但每个人对于这些观点的理解却是千差万别的。**

比如书上说：“命名是要有意义的”，但什么样的命名才算是有意义的呢？有的人只理解到不用 xyz 命名，虽然他起出了自认为“有意义”的名字，但这些名字依然是难以理解的。事实上，大部分程序员在真实世界中面对的代码，就是这样难懂的代码。

这是因为，**很多人虽然知道正面的代码是什么样子，却不知道反面的代码是什么样子。**这些反面代码，Martin Fowler 在《重构》这本书中给起了一个好名字，代码的坏味道（Bad Smell）。

在我写代码的这 20 多年里，一直对代码的坏味道非常看重，因为它是写出好代码的起点。有对代码坏味道的嗅觉，能够识别出坏味道，接下来，你才有机会去“重构（Refactoring）”，把代码一点点打磨成一个整洁的代码（Clean Code）。Linux 内核开发者 Linus Torvalds 在行业里有个爱骂人的坏名声，原因之一就是对于坏味道的容忍。

所以，我也推荐那些想要提高自己编程水平的人读《重构》，如果时间比较少，就去读第三章“代码的坏味道”。

不过，《重构》中的“代码的坏味道”意图虽好，但却需要一个人对于整洁代码有着深厚的理解，才能识别出这些坏味道。否则，即使你知道有哪些坏味道，但真正有坏味道的代码出现在你面前时，你仍然无法认得它。

比如，你可以看看 Info、Data、Manager 是不是代码库经常使用的词汇，而它们往往是命名没有经过仔细思考的地方。在很多人眼中，这些代码是没有问题的。正因如此，才有很多坏味道的代码才堂而皇之地留在你的眼皮底下。

所以，我才想做一个讲坏味道的专栏，把最常见的坏味道直接用代码形式展现出来。在这个专栏里，我给你的都是即学即用的“坏味道”，我不仅会告诉你典型的坏味道是什么，而且也能让你在实际的编程过程中发现它们。比如前面那个例子里面的 setter，只要它一出现，你就需要立即警觉起来。

这里我也整理了一份“坏味道自查表”，把一些明显的“坏味道”信号列了出来，你可以和自己的代码做对比。



代码坏味道自查表

命名

- ☐ 命名是否具有业务含义
- ☐ 命名是否符合英语语法

函数

- ☐ 代码行是否超过（ ）行
- ☐ 参数列表是否超过（ ）个

类

- ☐ 类的字段是否超过（ ）个
- ☐ 类之间的依赖关系是否符合架构规则

语句

- ☐ 是否使用 for 循环
- ☐ 是否使用 else
- ☐ 是否有重复的 switch
- ☐ 一行代码中是否出现了连续的方法调用
- ☐ 代码中是否出现了 setter
- ☐ 变量声明之后是否有立即再赋值
- ☐ 集合声明之后是否有立即添加元素
- ☐ 返回值是否可以使用 Optional

除了为你列出来哪些代码有坏味道之外，我还会给你讲支撑这些“坏味道”之所以为“坏味道”的原因，比如说：长方法和大类之所以为坏味道，因为它们都违背了单一职责的原则。

有坏味道的代码需要经过重构才能长成新的样子，在这个专栏里，我也会提到一些重构的手法，比如，改名（Rename）、提取方法（Extract Method）等等。在今天，拜许多能力强大的 IDE 所赐，重构已经变得越来越自动化，《重构》里的很多手法已经成为了 IDE 中的一个选项。

我还想给你一个安全提示，即便 IDE 功能再强大，也不要忘了重构的重要根基：测试。即便像 Java 这样，IDE 功能已经非常强大了，依然会有一些像反射之类的场景可能会从自动化重构的鼻子底下溜走。所以，重构一段代码之前，最好能够给它写下测试，确保改动前后的代码，功能上是一致的。

如果你订阅过我的《[🔗 10x 程序员工作法](#)》和《[🔗 软件设计之美](#)》，你就会发现，三个专栏一脉相承，这些背后的道理恰恰就是我在那两个专栏中已经提到过的内容。所以，三个专栏一并服用，效果会更佳。

写在最后

最后，还是要做一个自我介绍。我叫郑晔，一个写代码超过二十年的程序员，做过与软件开发的各项工作：编代码、带团队、做咨询、写开源。正如前面所说，我已经在极客时间平台上写了两个专栏，分享我在软件开发中的各种思考。这次，我会带你进入到我的基本功里，帮你一起写好代码。

十年前，我在 InfoQ 写过一个专栏《[🔗 代码之丑](#)》，把一些真实世界的代码展示了出来，让大家看到丑陋代码是什么样子的。

不少读者都表示，那个专栏让他们受益匪浅。不过，那个系列只是我日常工作的随手之作，没有更好地整理。这个专栏就是脱胎于 InfoQ 上的《代码之丑》，我对相关内容进行了更系统地整理，保证即便看过那个《代码之丑》专栏，你依然能够在这里有所收获。

这是一条通往代码精进之路，我愿意与你一起前行，成为你在这条路上的向导。如果你想摆脱平庸的小白程序员状态，成为一个更优秀的程序员，那么，请加入我的专栏，让我们一起修炼，日益精进写代码的手艺！

13 人觉得很赞 | [提建议](#)

© 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。

下一篇 [课前热身 | 这些需求给到你，你会怎么写代码？](#)

精选留言 (19)

 写留言

**Changing**

2020-12-28

看到这个Setter，有个疑惑。在现在的项目中，service层经常出现各种setter，基本是把所有逻辑都放到service层了。之前网上查询了一些资料，把这种称之为“失血模型”。代码这样写有哪些坏处？在既有的项目中，如果要改变的话，需要从哪里做起呢？

展开 ∨

作者回复: 后面会有一讲专门讨论这个问题，简言之，就是缺了行为，暴露了细节，解决办法就是，引入行为，封装细节。



8

**邵俊达**

2020-12-28

期待已久，老师前两个专栏都学完了并且都学了不少一遍，收获良多。继续跟老师学习，打磨写代码的手艺。

作者回复: 我们一起加油！



4

**每天晒白牙**

2020-12-28

看看我的代码写的多丑

展开 ∨

作者回复: 看运气了，哈哈。



2

**escray**

2021-01-05

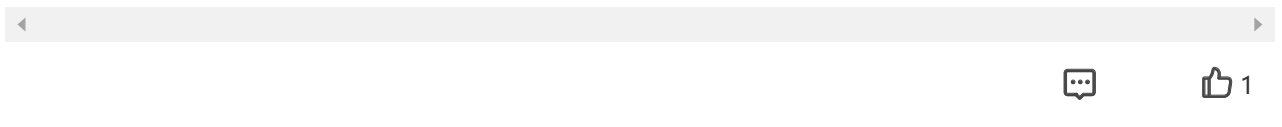
前两个专栏都还没有学完的老读者又来了。

两个专栏其实已经听了不止一遍，一直想再认真学习一遍，却没能做到。

结果现在已经脱离了编程一线，转型项目经理了，不过还是希望终有一天可以继续写代...

展开 ∨

作者回复: 多谢你对于我老专栏的总结，欢迎回来！

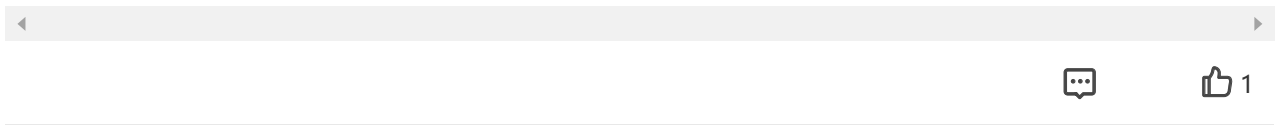


Seed2009

2020-12-29

我前同事说老师才给他们做过一周的技术指导，对您的技术很膜拜。

作者回复: 都不记得我在哪个公司留下过足迹了。😁

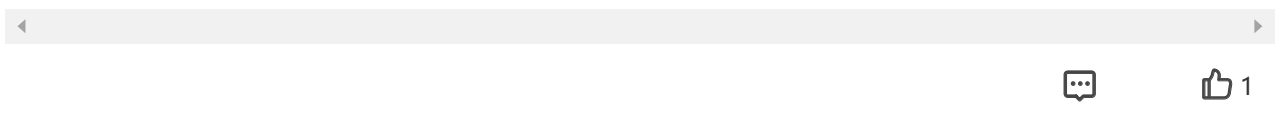


Jxin

2020-12-28

老粉前来打卡。以上书籍都拜读过。就看郑老师如何讲出花来了~

作者回复: 我也努力啊！



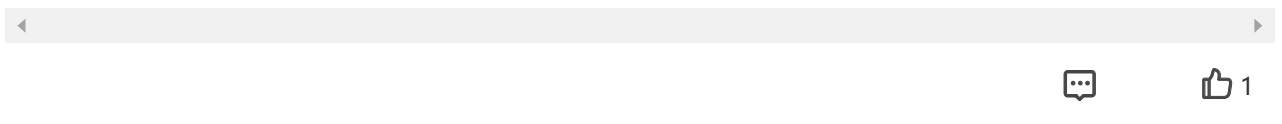
G小调

2020-12-28

```
public void approve(final long bookId) { ... book.approve(); ...}
```

book.approve() 是对setter做了封装吗

作者回复: 严格地说，是把对数据的操作封装了。

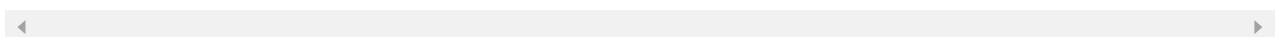


wang_acmilan

2020-12-28

来了来了，那个男人，他lei了。
这次的主题是:那味不对啊~

作者回复: 有才，你好！



**晴天了**

2020-12-28

内容都没看 先来订阅 郑老师绝对的偶像 哈哈

展开 ▾

作者回复: 我们一起努力!

**明**

2020-12-28

耶! 又多了一门“朗读并背诵全文”的课程 😊 😊

展开 ▾

作者回复: 背诵还是要求太高了, 哈哈。

**favorlm**

2021-01-22

我们现在的service层就是saveUser,serviceImpl里调用一个mybatis的mapper.save(user), 请问这样的设计形式是否是混淆了业务和技术, 请问有什么建议可以优化?

作者回复: 如果 mapper 是一个接口的话, 就还好。因为它把具体的实现隔离开了。

**CityAnimal**

2021-01-09

在最近的项目中逐渐意识到测试和重构的重要性, 但如何落地还在摸索中; 共同努力, 消除臭味

作者回复: 加油, 希望这个专栏能够帮助到你。

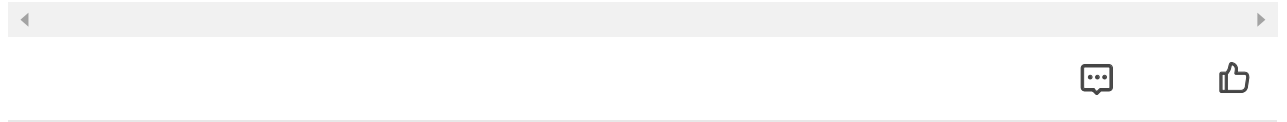


**咱是吓大的**

2021-01-02

以前听说过一个段子，说某家银行训练验钞员时只用真钞而从不用假钞，这样训练出的验钞员一碰到假钞马上就能识别出真伪，而真假交叉训练反而会弄糊涂。不知道老师怎么看这个问题？

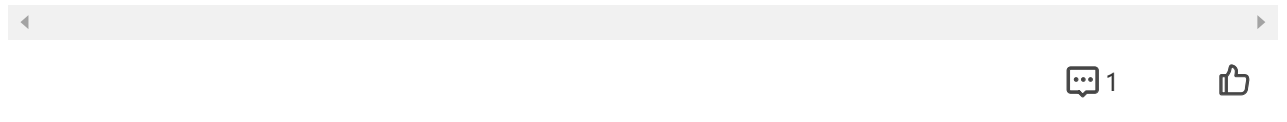
作者回复: 我不了解，不妄加评论。

**Hobo**

2020-12-31

老师，如果我需要封装一个方法能够对类的一个字段进行多种状态的更改应该怎么封装比较好？

作者回复: 为什么是一个方法而不是多个方法呢？用一个方法的结果就是一个setter的变体。

**Ankhetsin**

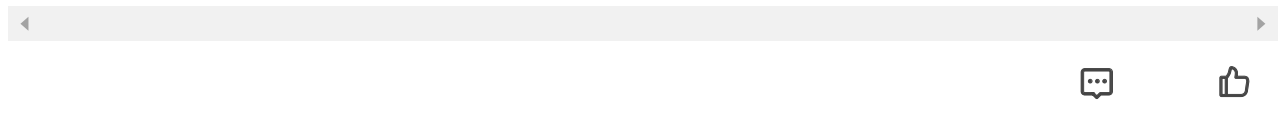
2020-12-30

一个几百行的长SQL算不算方法过长呢？一个bean有上百个字段光设置属性的值并转json不电网络库的代码肯定不止40行了。这个怎么解决？

展开 ∨

作者回复: 写出几百行的 SQL，一般都是些存储过程，而存储过程早在很多年前就已经被列到不推荐的做法里了。如果是普通的语句，除非是做某些特殊的统计，一般不建议这么做。

一个 Bean 为什么要有上百个字段呢？以我的经验看，这通常是没有想清楚就把所有的东西都塞进来了。所以，应该做的是，做职责分解，把不同的内容放到不同的接口去。

**DK**

2020-12-29

有个疑问想向老师请教，那个setter例子，如果一个状态可以改成多种，是应该为了保持可读性重复写多次差不多的代码，还是将要修改的目标状态当成参数也传进去呢

作者回复: 如果是顺着你的思路思考这个问题，那答案应该是把状态作为一个参数传递进去。

但这样思考问题，还是按照暴露细节的方式在思考，我在《软件设计之美》里讲过，封装应该是考虑行为的，究竟是什么行为导致状态的变化，这是我们应该思考的问题。

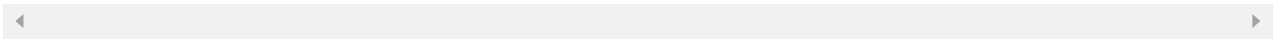


升级打怪兽

2020-12-29

培养这方面的意思，形成一套清单供自查，前期避免，还是得从不同角度思考它为啥丑，再结合清单学习，修正，补充清单，前段时间写的一个大事务模块，边改边骂自己

作者回复: 意识到自己以前写的代码不好，就是一个巨大的进步。



Geek_3b1096

2020-12-29

提升手艺~

展开 ∨



新语

2020-12-28

太棒了

展开 ∨

