二

20 答疑 对于设计模式而言，场景到底有多重要?

今天是第二模块的最后一讲。在这一讲中，我们主要讲了软件的设计原理，今天，我将会针对这一模块中大家提出的普遍问题进行总结和答疑。并且，我在最后列了一个书单，这个书单里涉及到的书，可能会对你学习设计模式有一些帮助。让我们整理一下，再接着学习下一个模块的内容。

问题答疑

我们先来看一个同学提出的问题。

@山猫

如果项目初始就对Button按钮进行这么复杂的设计，那么这么项目后期的维护成本也是相当之高。

答:

我们这个模块是讲设计的，这些设计原则都是用来解决需求变更的问题的。如果你为需求变更而进行了设计，但是预期中的需求变更却从来没有发生过，那么你的设计就属于设计过度；如果已经发生了需求变更，但是你却没有用灵活的设计方法去应对，而是通过硬编码的方式在既有代码上打补丁，那么这就是设计不足。

因此，是否要使用各种设计原则和设计模式去设计一个非常灵活的程序，主要是看你的需求场景。如果你的场景就是需要灵活，就是要各种复用，应对各种变更，那么一开始就应该这样设计。如果你的场景根本不需要一个可复用的Button，那么就不需要这样设计。

所以关键还是看场景。

但是场景也会变化，一开始不需要复用，但是后来又需要复用了，那么就需要在复用的第一个场景去重构代码，而不是等将来困难局面hold不住了再重构。

同时，设计原则和设计模式只是让代码看起来复杂了，毕竟一个接口好几个实现，看起来不如 `if-else` 来得直接。但是如果习惯了这种灵活的设计，你会觉得这种设计并不复杂。对于

软件开发而言，复杂的永远是业务逻辑，而不是设计模式。设计模式是可重复的，可重复的东西即使看起来复杂，熟悉了就会觉得很简单。

看起来复杂的设计模式就是用来解决维护困难这种问题的。因此正确使用设计模式，看起来复杂了，其实维护简单了，因为关系和边界更清晰了，你不需要在一堆强耦合的代码里搅来搅去。真正维护成本高的，其实是那些所谓的简单设计，牵一发动全身，稍不注意就是各种bug。

最终，一切都要看场景，只有合适的设计，不存在好的设计。分析场景，根据场景进行相应的设计。当然，你要先知道有哪些设计原则和设计模式可以用在这样的场景，这就是我们这个专栏模块的目的。

关于依赖倒置原则，评论区有一个精彩留言，分享给你。

精选留言(11)

山猫

依赖倒置这个东西懂得人是真懂，不懂的人是一点不懂。当初为了搞懂依赖倒置原则花了相当长时间去阅读大量的文章和书籍，看了很多代码事例，后来又看了面对对象设计原则才算基本理解。

之前给公司开发培训，他们仍旧听的一脸懵逼，觉得这是个噱头。直到我跟他们说：老板就是找个写代码的人，别把自己看的那么重。你们每天写那么多bug，别怪老板说要换人。他们才理解！

作者回复: 这个解释精彩😄

2019-12-18 收起 5

留言回复则解释得更加直白：

评论 2

草原上的奔跑

老板不依赖任何员工，老板制定公司标准规范，员工遵守规范干活，嫌干的不好的，换实现(换人)

2019-12-18 评论 3

另外，我在[第13篇]留了一道思考题：

父类中有抽象方法 f，抛出异常 AException：

```
1 public abstract void f() throws AException;
```

复制代码

子类 override 父类这个方法后，想要将抛出的异常改为 BException，那么 BException 应该是 AException 的父类还是子类？

很多人都回答正确了，但也有一些回答错误的。我这里说明一下。

正确答案为， `BException` 是 `AException` 的子类。因为只有异常是子类，使用父类的地方catch异常的时候，才能catch到子类异常，也就是才满足里氏替换原则，能用子类替换父类。

最近几年，分布式架构、大数据、区块链、物联网、AI技术广泛流行。当我们说起软件开发的时候，提到的常常是这些宏大的技术架构。但是再宏大的技术也要落实到代码上，再厉害的技术终究要解决我们的业务问题。如果不能写出清晰、简单的代码，软件之间的耦合关系梳理不清楚，即使用了一些很炫酷的技术，软件开发可能还是会陷入混乱之中。

这些年，我也曾在一些知名的企业做过各种分布式系统、大数据平台开发，这些系统本身的架构也许有很大创新之处，但是真正使这些系统成功的，依然是低层那些干净、清晰的代码。这些年，我也见过一些知名的架构师、布道师，有些人也曾引领技术潮流，成为风口浪尖上的技术红人，但是真正能够一直走下去，走得远的，不是那些能给自己安了各种厉害头衔的人，而是那些能踏踏实实写出漂亮代码的人。

这个专栏的第二模块就是想传递这些信息，我们为什么要写好的代码，而不仅仅写能用的代码；以及什么叫好的代码，如何写出好的代码。

书籍推荐

人类编程的历史超过半个多世纪了，关于什么叫好的代码，如何写出好的代码也有很多研究，有许多的经典案例和著作。专栏中的内容主要都是来自这些经典的作品。

专栏[第9篇文章]，如何使用UML建模的内容主要来自《UML精粹》这本书。

其实UML本身非常简单，简单到我都觉得不值得专门阅读一本书去学习UML。UML真正需要学习的，是如何灵活使用UML去完成软件设计，如何用UML表述出自己的设计意图，以

及在什么样的场景下用什么样的模型图去表述自己的设计意图。

马丁·福勒这本书也是偏重UML的实践应用，而不是讲UML语法本身如何。我的专栏文章内容则更多来自自己的一些最佳实践：如何用UML完成设计文档。应该说，我在十几年前，得以最早抓住机会跳出开发CRUD代码，去做一些大型系统的架构和框架开发工作，正是因为我用UML比较清晰地表述了系统当时的状况和设计目标，打动了项目的领导，放手让我一个资历尚浅的新人去做系统的架构设计，也因此而改变了自己的职业发展路径。我也期望UML能帮助你找到自己的职业跳跃之路。

专栏11~15篇文章，主要讲述软件设计的基本原则，这些内容主要来自《敏捷软件开发: 原则、模式与实践》。

作者罗伯特·C ·马丁，更知名的昵称是Bob大叔。这本书的名字叫《敏捷软件开发》，但是全书主要讲的是设计原则与设计模式。作者认为，我们能够进行敏捷开发，能够快速响应需求变更，不在于什么敏捷开发过程和敏捷项目管理，而在于敏捷的软件设计。如果代码一团糟，各种耦合，各种腐化，任你用什么项目管理手段都无济于事。

但如果你的代码灵活、强壮、易于维护和变更，可以轻松应对各种需求变更，那么敏捷的项目过程管理才能带来真正敏捷的效果。

应该说，我第一次读这本书的时候，它给我的震撼相当大。人们在软件开发中遇到困难，本能地想寻找一种轻松又强大的解决办法，什么管理方法啦，外部咨询啦，购买商业解决方案啦。但是，软件终究还是存在于工程师编写的一行行代码里，如果不把这些代码搞清楚，搞好，再好的外部支持只怕也帮不上什么忙。

[第19篇]内容也是来自罗伯特·C·马丁的一本比较新的书，叫《架构整洁之道》。

这本书算是Bob大叔架构思想的一本书，讲述关于架构的过往与现在，关于架构的各种思想原理。

第20篇的内容又是来自马丁·福勒的另一本经典著作《企业应用架构模式》。

这本书是讲述企业架构模式的集大成者，我们日常使用的各种开发技术，各种解决方案，都可以在这本书里找到来源。很多业界广泛使用的技术产品，Spring，MyBatis这些，只不过是这本书里很多架构模式的一种，而同类的模式还有很多，这些模式有的被淘汰，有的在进化。

看这本书里的各种架构模式，然后再想想这些模式背后的技术在这些年中的起起落落，感觉很是沧桑。

这就是第二模块中遗留的一些问题，无论是架构，还是软件开发，终归要落到人身上，落到

人编写的一行行代码身上。我希望这个模块可以对你写代码有一些好的启发与提示。

上一页 下一页