## 75-在实际的项目开发中，如何避免过度设计？又如何避免设计不足？

设计模式的理论部分已经全部学习完了。现在，你可能已经蠢蠢欲动，想要赶紧实践一把，把这些理论应用到自己的项目中。不过，这里我要给你提个醒了，千万别手里拿着锤子就看什么都是钉子啊。

在我过往的项目经历中，经常遇到两种同事。

一种同事会过度设计。在开始编写代码之前，他会花很长时间做代码设计，在开发过程中应用各种设计模式，美其名曰未雨绸缪，希望代码更加灵活，为未来的扩展打好基础，实则过度设计，未来的需求并不一定会实现，实际上是增加了代码的复杂度，以后的所有开发都要在这套复杂的设计基础之上来完成。

除此之外，还有一种是设计不足。怎么简单怎么来，写出来的代码能跑就可以，顶多算是demo，看似在实践KISS、YAGNI原则，实则忽略了设计环节，代码毫无扩展性、灵活性可言，添加、修改一个很小的功能就要改动很多代码。

所以，今天我想和你聊一下，在实际的项目开发中，如何避免过度设计，以及如何避免设计不足。话不多说，让我们正式开始今天的内容吧！

### 设计的初衷是提高代码质量

创业时，我们经常会讲到一个词：初心。这词说的其实就是，你到底是为什么干这件事。不管走多远、产品经过多少迭代、转变多少次方向，“初心”一般都不会随便改。当我们在为产品该不该转型、该不该做某个功能而拿捏不定的时候，想想它符不符合我们创业的初心，有时候就自然有答案了。

实际上，应用设计模式也是如此。应用设计模式只是方法，最终的目的，也就是初心，是提高代码的质量。具体点说就是，提高代码的可读性、可扩展性、可维护性等。所有的设计都是围绕着这个初心来做的。

所以，在做代码设计的时候，你一定要先问下自己，为什么要这样设计，为什么要应用这种设计模式，这样做是否能真正地提高代码质量，能提高代码质量的哪些方面。如果自己很难讲清楚，或者给出的理由都比较牵强，没有压倒性的优势，那基本上就可以断定这是一种过度设计，是为了设计而设计。

实际上，设计原则和思想是心法，设计模式只是招式。掌握心法，以不变应万变，无招胜有招。所以，设计原则和思想比设计模式更加普适、重要。掌握了设计原则和思想，我们能更清楚地了解为什么要用某种设计模式，就能更恰到好处地应用设计模式，甚至我们还可以自己创造出来新的设计模式。

### 设计的过程是先有问题后有方案

如果我们把写出的代码看作产品，那做产品的时候，我们先要思考痛点在哪里，用户的真正需求在哪里，然后再看要开发哪些功能去满足，而不是先拍脑袋想出一个花哨的功能，再去东搬西凑硬编出一个需求来。

代码设计也是类似的。我们先要去分析代码存在的痛点，比如可读性不好、可扩展性不好等等，然后再针对性地利用设计模式去改善，而不是看到某个场景之后，觉得跟之前在某本书中看到的某个设计模式的应用场景很相似，就套用上去，也不考虑到底合不合适，最后如果有人问起了，就再找几个不痛不痒、很不具体的伪需求来搪塞，比如提高了代码的扩展性、满足了开闭原则等等。

实际上，很多没有太多开发经验的新手，往往在学完设计模式之后会非常“学生气”，拿原理当真理，不懂得具体问题具体分析，手里拿着锤子看哪都是钉子，不分青红皂白，上来就是套用某个设计模式。写完之

后，看着自己写的很复杂的代码，还沾沾自喜，甚至到处炫耀。这完全是无知地炫技，半瓶子不满大抵就是这个样子的。等你慢慢成长之后，回过头来再看自己当年的代码，我相信你应该会感到脸红的。这里我的话说得有点重，我主要还是担心你以后在项目中，过度设计被别人鄙视，所以提前给你打个预防针！

所以，我的专栏讲解中，一直是从问题讲起，一步一步给你展示为什么要用某个设计模式，而不是一开始就告诉你最终的设计。实际上，这还不是最重要的，最重要的是我想培养你分析问题、解决问题的能力。这样，看到某段代码之后，你就能够自己分析得头头是道，说出它好的地方、不好的地方，为什么好、为什么不好，不好的如何改善，可以应用哪种设计模式，应用了之后有哪些副作用要控制等等。

相反，如果你只是掌握了理论知识，即便你把23种设计模式的原理和实现背得滚瓜烂熟，不具备具体问题具体分析的能力，在面对真实项目的千变万化的代码的时候，很容易就会滥用设计模式，过度设计。

## 设计的应用场景是复杂代码

很多设计模式相关的书籍都会举一些简单的例子，这些例子仅仅具有教学意义，只是为了讲解设计模式的原理和实现，力求在有限篇幅内给你讲明白。而很多人就会误以为这些简单的例子就是这些设计模式的典型应用场景，常常照葫芦画瓢地应用到自己的项目中，用复杂的设计模式去解决简单的问题，还振振有词地说某某经典书中就是这么写的。在我看来，这是很多初学者因为缺乏经验，在学完设计模式之后，在项目中过度设计的首要原因。

前面我们讲到，设计模式要干的事情就是解耦，也就是利用更好的代码结构将一大坨代码拆分成职责更单一的小类，让其满足高内聚低耦合等特性。创建型模式是将创建和使用代码解耦，结构型模式是将不同的功能代码解耦，行为型模式是将不同的行为代码解耦。而解耦的主要目的是应对代码的复杂性。设计模式就是为了解决复杂代码问题而产生的。

因此，对于复杂代码，比如项目代码量多、开发周期长、参与开发的人员多，我们前期要多花点时间在设计上，越是复杂代码，花在设计上的时间就要越多。

不仅如此，每次提交的代码，都要保证代码质量，都要经过足够的思考和精心的设计，这样才能避免烂代码效应（每次提交的代码质量都不是太好，最终积累起来整个项目的质量就变得很差）。

相反，如果你参与的只是一个简单的项目，代码量不多，开发人员也不多，那简单的问题用简单的解决方案就好，不要引入过于复杂的设计模式，将简单问题复杂化。

## 持续重构能有效避免过度设计

我们知道，应用设计模式会提高代码的可扩展性，但同时也会带来代码可读性的降低，复杂度的升高。一旦我们引入某个复杂的设计，之后即便在很长一段时间都没有扩展的需求，我们也不可能将这个复杂的设计删除，整个团队都要一直背负着这个复杂的设计前行。

为了避免错误的需求预判导致的过度设计，我非常推崇持续重构的开发方法。持续重构不仅仅是保证代码质量的重要手段，也是避免过度设计的有效方法。在真正有痛点的时候，我们再去考虑用设计模式来解决，而不是一开始就为不一定实现的未来需求而应用设计模式。

当对要不要应用某种设计模式感到模棱两可的时候，你可以思考一下，如果暂时不用这种设计模式，随着代码的演进，当某一天不得不去使用它的时候，重构的代码是否很大。如果不是，那能不用就不用，怎么简单就怎么来。说句实话，对于10万行以内的代码，团队成员稳定，对代码涉及的业务比较熟悉的情况下，即便

将所有的代码都推倒重写，也不会花太多时间，因此也不必为代码的扩展性太过担忧。

## 避免设计不足的3个必要条件

前面大部分讲到的都是如何避免过度设计，我们再稍微讲讲如何避免设计不足。

首先，你要有一定理论知识的储备。

比如你要熟练掌握各种设计原则、思想、编码规范、设计模式。理论知识是解决问题的工具，是前人智慧的结晶。没有理论知识，就相当于游戏中没有厉害的装备，虽然可以靠身手徒手打怪，但肯定会影响你最高水平的发挥。

其次，你还要有一定的刻意训练。

很多同学很苦恼，说理论知识都学过，但是很容易忘记，遇到问题也想不到对应的知识点。实际上，这就是缺乏理论结合实践的刻意训练。我们回想一下上学的时候，我们是如何学习的。老师讲解完某个知识点之后，往往会配合讲解几道例题，然后再让你做上个几十道题去强化这个知识点。这样当你再遇到类似的问题的时候，就能不由自主地联想到相应的知识点。而工作之后，我们自己看书学知识，别说拿几个场景来实践了，大部分都是走马观花地看看，没有经过刻意的训练，知识积累不了，能力也锻炼不了，等于白学。

最后，你一定要有代码质量意识、设计意识。

在写代码之前，要多想想未来会有哪些扩展的需求，哪部分是会变的，哪部分是不变的，这样写会不会导致之后添加新的功能比较困难，代码的可读性好不好等代码质量问题。有了这样的意识，实际上，你就离写出高质量的代码不远了。

## 不要脱离具体的场景去谈设计

设计是一个非常主观的事情，不夸张地讲，可以称之为一门“艺术”。那相应地，好坏就很难评判了。如果真的要评判，我们要放到具体的场景中。脱离具体的场景去谈论设计是否合理，都是空谈。这就像我们经常说的，脱离业务谈架构都是“耍流氓”。

比如，一个手游项目是否能被市场接受，往往非常不确定。很多手游项目开发出来之后，市场反馈很差，立马就放弃了。除此之外，尽快上市占领市场也是一款手游致胜的关键。所以，对于手游项目的开发来说，往往前期不会花太多的时间在代码设计、代码质量上。

相反，如果你开发的是MMORPG大型端游，一般都要投资上亿资金，几百号人开发好几年，推倒重来的成本很大。这个时候，代码质量就非常关键了。前期就要多花点时间在设计上，否则，代码质量太差，bug太多，后期无法维护，也会导致很多用户弃而选择同类型的其他家的游戏。

再比如，如果我们开发的是偏底层的、框架类的、通用的代码，那代码质量就比较重要，因为一旦出现问题或者代码改动，影响面就比较大。相反，如果我们开发的是业务系统或者不需要长期维护的项目，那稍微放低点代码质量的要求，也是没问题的，而且，自己的代码跟其他项目没有太多耦合，即便出了问题，影响也不大。

## 课堂讨论

如何避免过度设计？如何避免设计不足？关于这个话题，你还有哪些心得体会、经验教训可以在留言区说一说，分享给大家。

如果有收获，欢迎你把这篇文章分享给你的朋友。

## 精选留言：

- Jackey 2020-04-24 00:41:19
  滥用设计模式真的是不如不用 [5赞]

- 黄林晴 2020-04-24 08:31:18
  打卡
  持续重构真的很重要
  我现在看自己去年的代码都看不下去
  相信明年的我看今年重构后的代码也看不下去 [2赞]

- 小晏子 2020-04-24 08:53:54
  避免过度设计和设计不足的经验基本上文章覆盖的很全了，如果能做到文中提到的点基本上就可以避免。另外个人感觉可以把设计方案放到组内讨论，看看别人的反应，一般过渡地太复杂的设计都会有人反对。[1赞]

- leezer 2020-04-24 07:06:27
  主要核心还是知道为啥要用设计模式 [1赞]

- Jxin 2020-04-24 11:03:50
  1.回答这两个问题。在该有的知识都具备的情况下。其实是在问，你的初心是什么。

  2.如果你的初心是“保证项目持续迭代的效率”，或者说保证项目架构的持续优异。那么快速的先落地需求+持续重构会是主色调。这种模式下，落地需求不会有太多的过度设计，设计不足也能在持续重构中被摆正。但是，这非个人的事情，需要团队甚至整个公司的共同支持与认可。

  3.如果你的初心是“写出高质量代码”。那么过度设计在所难免。可是，这问题很大吗？实际工作中，不会有人有时间一直去揣摩你的代码。而真要阅读你的代码，一般也是能看懂的。所以对团队的影响其实还是比较有限的，但对自己的认知和成长是有好处的。这么玩并不为过，重要的是保持谦逊，因为这个时候写的代码更多是实践知识点，缺少平衡架构合理和需求场景时的抉择，硬拿经典用例或伪需求来强调自己牛逼就有点令人生厌了。

- cugphoenix 2020-04-24 10:19:11
  争哥的课真的是满满干货，精髓就在于这对于设计思想的讲解，非常接地气。

- 三木子 2020-04-24 10:03:05
  学习设计模式如喝茶。一杯好茶需要时间浸泡，学习设计模式也要工作经验浸泡。茶水味道需要细品。不带味觉去喝，犹如喝白开水。设计模式不带思考去品，犹如喝了一碗没有补到身体补品！

- zs阿帅 2020-04-24 09:33:43
  现在感觉就是学完了所有招式，蠢蠢欲动，写个代码就想能用什么设计模式

- Heaven 2020-04-24 09:32:54
  在实际开发中,我认为 设计合适>过度设计>不进行设计,往往不进行设计的工程师,要么是没有了解过对应

的知识,要么是对知识的不熟悉掌握,我个人也经常犯过度设计的毛病,但是我个人的观点是,如果学完了东西,尽可能的去尝试使用它,因为不使用它,那么可能一直就没有机会去使用,或者真正合适用的场景,也想不起来,不要畏惧自己的过度设计,因为没有过度设计的错误的铺垫,可能就永远没法设计出恰到好处的代码,没有人能一蹴而就,学习这条道路上都是踩着坑过来的

- mghio 2020-04-24 08:15:25
  设计模式还是得看具体业务场景，不能硬套

- 忆水寒 2020-04-24 07:30:43
  前提是把握需求才能做好功能分析，然后才能划分功能。最后才是基于功能进行设计开发。

- 木头 2020-04-24 07:24:13
  1、业务场景的详细把握 2、分析业务场景的时候，可以划分出支撑部分、工具部分、功能部分的大致结构 3、在以上场景的概念中，转化成代码，选择适合的设计方案及模式。