



## 36 | 能力维度一：如何提升结构化设计的能力？

2022-05-17 郭东白

《郭东白的架构课》

课程介绍 >



讲述：郭东白

时长 19:22 大小 17.75M



你好，我是郭东白。

上节课我们定义了架构师这个角色，也了解了架构师的五个成长阶段，分别是程序员、兼职架构师、跨域架构师、总架构师和 CTO。以及与这五个阶段分别对应的核心能力，即：结构化设计能力、解决横向问题的能力、解决跨域冲突的能力、正确技术决策和创造生存优势的能力。

从这节课开始，我们就来探讨一下从程序员到 CTO 的能力跃迁问题。在分析的过程中，我们会着重分析两个角色间的能力差异、面临的工作挑战，从现有能力到下一个能力所以跨越的障碍，以及对应的跨越方法，最终帮助你规划自己的职业发展。

我们先来研究一下从程序员成长为架构师所必需的核心能力，也就是结构化设计的能力。



下载APP



我跟很多页面的开发工程师、CTO 交流过，我们的共同观察是：一个缺乏结构化设计能力的程序员，永远都成不了好的架构师。

我先给出软件架构能力的定义，从中可以看出结构化设计能力的关键作用。软件架构能力指的是为相对复杂的场景定义并引导实施结构化软件方案的能力，其中**结构化**，代表这个软件在其设计范围内的设计理念、代码结构和实现方式上是同质的。

我用了“同质 ( Homogenous ) ” 一词来形容结构化，指的是在设计范围内处处一致。与结构化 ( Structured ) 相反的是缺乏一致性，也就是说结构是混乱的 ( Chaotic ) 。

在这个定义中，我并没有锁定适用的人群范围，适用范围可大可小。也就是说，一个刚入行的程序员，即使没有做过任何架构规划，也完全可以培养自己的软件架构能力。

举个例子。他可以在自己的决策范围内，也就是自己写的代码里面，**寻找并给出结构化的解决方案**。这种结构化的实现往往不是产品需求的一部分，也没有人考核，更不会为此发放直接的奖金激励。但是，你会发现身边一些程序员对代码有一种几乎偏执的洁癖。在我看来，这就是对于结构性 ( Structuredness ) 的追求。

这种追求，在一个程序员的职业发展中是具有连贯作用的。从程序员到 CTO，刚开始可能是对代码结构性的追求。随着职业的发展，这种结构性会扩展到更大的范围和更广的维度中，甚至延伸到公司软件研发的各个方面。最终，代码洁癖就会演变成对于**软件系统甚至研发组织结构性的执着**，而这种执着也会固化成**对软件结构性的识别、改进和设计能力**，贯穿整个职业生涯。

我观察很多资深的软件工程师和管理者。他们的职业成长过程，其实就是把结构性的能力从一个领域迁移到另一个领域的过程。一开始是在这个领域写代码，然后就到了数据模型，接着是功能模块、团队定位和组织结构，最终就到了商业模块。

可以说，对于一个以架构师为职业追求方向的程序员，结构化设计能力是基础能力。那么，该如何提升程序员的结构化设计能力呢？



## 如何提升结构化设计能力？



在互联网时代，代码整洁主要来自编程规范的掌握和运用、设计抽象和常见设计模式的采用。对于编程规范，各大软件公司一般都有要求，网上可以检索到。这里就不再赘述。

我简单讲一下代码抽象，尤其是**面向对象编程（OOP）的设计思想**。OOP 其实是结构化思维的一个重要范式，要求我们认真思考什么是类，什么是对象，什么是对象所具备的属性，什么是对象可能发生的行为，这个行为的作用对象是谁，等等。通过这种对模型本质的思考来定义软件结构，就是提升软件结构性设计能力的有效方法。

介绍 OOP 的国内外书籍有很多，我相信每个人都看过几本，不少人甚至能对 OOP 的原理倒背如流。但是我发现大家在写代码时，却很少**习惯性地**思考 OOP 方面的问题。我认为这种频繁的深度思考习惯，恰恰是架构师成长的必需。

举个关于扑克牌游戏的简单例子。如果去 GitHub 里翻看一下这类游戏的代码，会发现很多程序员都是把每张扑克牌设计成一个普通对象。如果再认真思考一下这种设计，会发现在普通的扑克牌游戏场景中其实是不太准确的。

一般来说，当一张扑克牌出现在电子游戏这个场景下的时候，除非游戏允许出老千，每张牌都可以有猫腻，否则每张扑克牌都不能是一个对象，而是全局状态统一的单例（Singleton）。对比之下，扑克游戏里面的每一手牌是一个普通对象，每一个玩家也是一个普通对象。

对象和 Singleton 是两种完全不同的设计，它们的运行成本和代码可读性也完全不同。其实任何一本 OOP 书籍和入门的设计模式书籍，都有类似的单例更优的场景。我相信如果稍稍思考一下，也都会做出正确的选择。但多数程序员缺乏这个思考的习惯。我认为正是这个习惯，**将一般程序员和职业软件工程师区分开来了**。

不过这个思考还没有完成。要想达到优秀的软件工程师的层次，还要再深度思考一层。如果把一张牌设计成单例之后，会不会带来某些局限呢？所以这个时候就要问问自己，在**一些场景中**需要把这张牌设计成一个对象呢？







样。比如有的牌在今天晚上就会给自己带来好运气。

本来一个赌徒手里拿了一张黑桃 3 和红桃 8，应该主动弃牌。但是你在红桃 8 上加了一行小字注释：

刚才真武太郎赢牌的那一手葫芦，就是这张红桃 8 加另外一对儿 8 和一个小对儿，那次他赢了 3000 个金币。而前面那个帮你赢了 510 个金币的 All-in 的同花里，也有这张红桃 8！

一旦有了这个注释，赌场每天交易额就会增长 30%。在这种商业模式下，虽然你还在设计扑克牌游戏，但每一张牌都有自己的生命周期。因为每张牌会因为经历过某个重大事件，而引起了各自生命周期状态的变化，所以就不再是单例了。至少每场赌局中的每一张牌，都是一个独立的对象。

作为一个优秀的软件工程师，这时候就要作出判断了。我的公司处在哪个阶段？需要这么具有前瞻性的设计吗？如果现在不需要，但是未来需要，我应该采取什么样的设计才能让未来的过渡最容易呢？

如果日常就在做这种深度思考，那么恭喜，你已经养成一个架构师应该具备的思考习惯了。日积月累，这种思想实验就会成为你的核心竞争力。想想看，别人写代码就是依照产品需求文档，做个从自然语言到计算机程序的转义。而你出手就不一样了，**是一个基于对问题本质、商业价值和软件结构性思考的长远设计！**

另一个对代码整洁性帮助很大的就是**设计模式**。设计模式的主要价值包括三个方面。

一是**沉淀经验**，用一句话来简单描述就是：如果遇到这种场景，就应该采用这种设计模式。可以说，这是个系统化且高效借鉴他人成功经验的过程。

二是**传递设计原理**，也就是对代码背后思考的一种标准化注释。代码所覆盖的领域可能全新的，比如是元宇宙中的一个新物种，甚至在现实世界中都不一定有对应的存在。但如果用 `_Factory_` 或者 `_Visitor_` 这个词，别人马上就会知道这个设计背后的含义，比注释更深远也更准确。可以说，你**通过规范化的设计语言表达了自己的思想**。





的座椅一样，让另一个在你的代码高山里艰难攀行的路人，突然间找到了一个熟悉的小憩之地。

设计模式与 OOP 思维范式不同。设计模式强调趋同性，用广泛传播的知识来标准化代码的实现，从而降低实现手段和命名的多样性，也就是提升代码的一致性。当然也有人非常反对设计模式，认为设计模式多数时间都被用烂了。很多人为了引用设计模式而得出了错误的设计，这种现象的确存在。但这不是设计模式的问题，而是**使用者应用不当的问题**。

在设计模式的基础之上，一旦开始模块设计，就需要对世界做建模。这时候还需要学习问题域建模，也就是通过领域模型帮助我们与所有需求方在某个问题上达成共识。这个过程，模块二中已经有了详细解释，这里就不重复了。

确定了问题之后，接下来就可以设计服务 API 了。也就是如何以简洁明了的方式包装要解决的问题，让使用者能够轻松且**长期使用**你的服务，而不需要耗费大量的时间去理解、学习、集成、测试、维护和升级系统。

## 结构化设计过程中的具体关注点

那么在日常工作中，想做好结构化设计，应该具体关注什么呢？我认为主要是以下三点。

第一，**设计理念**。也就是说，整体设计需要与公司或部门的理念保持一致。

比如整个公司都采用分层架构，那么你的设计也要尽量采用分层架构。整个公司都采用微服务的设计理念，那么你的设计也要采用微服务的设计理念。否则写出来的代码既难维护又难理解，也难以被别人复用。

第二，**API 的结构性**。也就是说，软件模块的对外界面要有比较明显的**语义结构**。

暴露给其他调用方的 API，要有条理、表达准确，且易于理解。否则，API 在被使用的环境中，会让调用方的代码变得晦涩难懂、结构混乱。



具体而言，API 的结构性体现在三个方面。



就具有宏观的结构性。

比如说，一个实体的 API 要反映出这个实体状态和行为。因为这些状态和行为会随着实体的生命周期而发生变化，所以 API 就要围绕实体生命周期来组织。而围绕一个流程定义的 API，则要根据流程的前后顺序来组织 API。甚至在 API 命名上的语言结构性，也会大幅提升 API 的可读性和可维护性。

其次是**功能组织上的结构性**。举个例子，一个 API 可能为多个用户角色服务，每个用户角色又有多个场景，每个场景还可能有多功能。那么这些功能就应该组织成三层的结构：角色、场景和功能。不同的用户角色和不同的使用场景，都应该有不同的设计粒度。

比如订单服务可能为用户、商家、运营和审计这四个角色服务，那么面向用户的订单查询与面向商家的订单服务的粒度和内容就完全不同。

最后是**数据模型的结构性**。API 会暴露数据给调用方，那么暴露出来的数据就要有清晰的结构，遵守一定的规范。

假设你在使用 JSON 传递数据，那么 JSON 就要做到可读且合理。所谓可读，就是只看 JSON，很快就能明白这些数据的含义，及其内部对象之间的关系。所谓合理，就是包含在 JSON 中的数据结构符合最小必要的原则。

做到这三点很不容易。哪怕是所有程序员都熟悉的 GitHub，它们的 API 和数据模型中也能挑出很多毛病来。

第三，**模块内部的结构性**，也就是程序的结构性。这种结构性其实是我们前面提到的程序员设计能力的具体体现。

举个例子，如果你现在使用的是 Java，那么对 package hierarchy、package、Interface、Class 和 function 实现的设计，就体现了你的设计实力。其中：



Package Hierarchy 和 Package 的设计，体现了你对领域的理解和对领域的封装。

Interface 的设计，体现了你对一个领域对外服务的理解。




你可能说，我是个职场新人，怎么一上来就能想出很完美的设计呢？事实上，我们日常工作涉及的很多领域都有现成的国际标准，**不需要再去发明创造**。除了我们前面提到的设计模式之外，使用常见的 SprintBoot 等服务框架，依赖主流而不是小众的技术，遵守 W3C 发布的最佳实践等，都是提升代码结构性的好办法。

不过这里有一个常见的问题：如果队友水平不高，是不是要降低我的标准来与他们对齐？我的观点如下：

1. 在模块内部的结构性上，完全不需要这样做。因为这是你的私域，更好的结构性不仅可以帮助自己，还能帮助未来接替自己维护模块的人。
2. 在 API 的结构性上，可以尽量追求 API 的结构性，前提是能与整个研发团队现有的设计规范相冲突。哪怕你认为现有规范不合理，也应该建议调整规范，从而引入更先进、更可读和更容易维护的标准，而不是直接越过规范。尤其在 API 的命名、JSON schema 的定义上，如果与团队的主流选择相冲突，那么你的设计会渗透到其他人的代码里，影响他人代码的一致性和可读性。
3. 在整体设计的理念上，你没有权利做任何的发明创造，否则将会增加公司架构的混乱程度。这个时候，更好的选择是想办法去影响决策者。

## 小结

我们这节课简单聊了程序员的结构化设计能力。说句实在的，我刚做程序员时，这项能力很一般。一方面是自己当时的能力不够，另一方面是自己的主动性意愿也不强。不过后来有了转变，所以现在就把我的转变经过分享出来，希望对你有帮助。

我第一份全职工作是在甲骨文，刚开始时主要写算法代码，不需要和其他人合作太多。我后来发起了一个内部创业项目，做得比较成功，成了 Oracle 数据库的一个主要功能。由于商业上比较成功，所以有很多人参与了进来。这时候我才发现自己的设计有很多瑕疵，时也收到了很多来自内外部的批评。

对于这些批评，我起初很抵触。直到最初的代码被重构两三次之后，才终于意识到有些设计失误其实完全可以避免。于是我开始重新阅读消化软件架构方面的书籍，开启习惯性的



下载APP



是的，在我看来，习惯性地对自己所设计的软件结构性进行反思，这个过程就像是照镜子。首先要看到设计中的那些污垢，然后再去清除。

能做到这一点，结构化设计的能力自然就会提升。

## 思考题

三个课后作业，可以选择其中一道作答。跟之前一样，不是做得多，而是做得深。

1. 这节课我们简单提到了“美”，并且隐含地把“架构之美”与“结构性”画上了等号。你认可这种说法吗？如果不认同，你的观点是什么？
2. API 的延续性，是 API 结构性中非常头疼的问题。为了保证现有用户的习惯，新的系统必须要提供与过去兼容的接口和设计。你可以举几个例子吗？你碰到过类似的场景吗？是怎么平衡的呢？
3. 在你的程序员职业生涯中，追求结构化设计的最大阻力是什么？是怎么克服的？有什么经验可以分享一下吗？

欢迎把你的思考和想法分享在留言区，我会和你交流。也欢迎把课程分享给你的朋友或同时，我们一起进步。我也请你关注我的抖音号郭东白。下节课再见！

分享给需要的人，Ta购买本课程，你将得 20 元

生成海报并分享

👍 赞 2

💡 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



上一篇 35 | 模块导读：回过头来看，你觉得架构师到底是做什么的？

下一篇 37 | 能力维度二：如何提升解决横向问题的能力？





下载APP



## 精选留言 (4)

写留言



罗均

2022-05-18

东白老师的课程，总是可以给学生全新的技术视角，为学生们打开了更大的思考空间。

作业第三题，回顾写代码的历程，最大的障碍还是自身的认知局限。例如，有些场景command pattern与strategy pattern的选择，或者mediator pattern或subscription pattern的选择，在implementation过程中，因为自己的水平有限，没办法融汇贯通达到最优效果。...

展开

作者回复：谢谢！



1



术子米德

2022-05-23



\* 📁：结构化设计，如何提升？

\* 😊：设计，只要不给别人看，自己总是无比满意，尤其是刚完成的时候，得意得不行。过几天来看，这都是啥，怎么这么奇怪。这时候，就会问自己，怎么没点结构感，都不知道怎么看起。如果拿给别人看，大概率对方很雾水一头，即使解释再三，还不如自...

展开

作者回复：“设计，只要不给别人看，自己总是无比满意，”太精辟了！



2022-05-22

第三个问题：在当前的公司干了五个年头了，当时加入的时候，整个后端就我一个人，整体后端架构的设计与客户端的交互都是我基本上是我在主导。说起来比较尴尬，我其实水平真的很一般，在做这些架构之前，也就三年的工作经验。追求结构化的过程其实是痛苦的，最开始最大的疑惑是怀疑自己是不是对的。但是随着需求的不断变化，你会真的多虑了，因为没有正确的设计。所有的设计都是 trade-off。在心理上解决了这个问...

展开



**spark**  
2022-05-17

郭老师，take away~~~结构性是一个结果，追求结构性是需要思考、路径，追求结构的思考需要系统性~~~  
a.编程维度，面向对象、面向过程，设计原则，设计模式，代码规范(Google Java Format)，重构~~~  
b.思考维度，架构思维和结构的因果关系，架构思维是因，结构是果？？？计算机系统...  
展开 ∨

作者回复: 谢谢总结！

