06 | 模式与框架：它们的关系与误区？

2018-08-15 胡峰

在学习程序设计的路上，你一定会碰到“设计模式”，它或者给你启发，或者让你疑惑，并且你还会发现在不同的阶段遇到它，感受是不同的。而“开发框架”呢？似乎已是现在写程序的必备品。那么框架和模式又有何不同？它们有什么关系？在程序设计中又各自扮演什么角色呢？

设计模式

设计模式，最早源自 GoF 那本已成经典的《设计模式：可复用面向对象软件的基础》一书。该书自诞生以来，在程序设计领域已被捧为“圣经”。

软件设计模式也是参考了建筑学领域的经验，早在建筑大师克里斯托弗·亚历山大（Christopher Alexander）的著作《建筑的永恒之道》中，已给出了关于“模式”的定义：

> 每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心，通过这种方式，我们可以无数次地重用那些已有的成功的解决方案，无须再重复相同的工作。

而《设计模式》一书借鉴了建筑领域的定义和形式，原书中是这么说的：

> 本书中涉及的设计模式并不描述新的或未经证实的设计，我们只收录那些在不同系统中多次使用过的成功设计；尽管这些设计不包括新的思路，但我们用一种新的、便于理解的方式将其展现给读者。

虽然该书采用了清晰且分门别类的方式讲述各种设计模式，但我相信很多新入门的程序员在看完该书后还是会像我当年一样有困扰，无法真正理解也不知道这东西到底有啥用。

早年我刚开始学习 Java 和面向对象编程，并编写 JSP 程序。当我把一个 JSP 文件写到一万行代码时，自己终于受不了了，然后上网大量搜索到底怎样写 JSP 才是对的。之后，我就碰到了《设计模式》一书，读完了，感觉若有所悟，但再去写程序时，反而更加困扰了。

因为学“设计模式”之前，写程序是无所顾忌，属于拿剑就刺，虽无章法却还算迅捷。但学了一大堆“招式”后反而变得有点瞻前顾后，每次出剑都在考虑招式用对没，挥剑反倒滞涩不少。有人说：“设计模式，对于初窥门径的程序员，带来的麻烦简直不逊于它所解决的问题。”回顾往昔，我表示深有同感。

后来回想，那个阶段我把《设计模式》用成了一本“菜谱”配方书。现实是，没做过什么菜只是看菜谱，也只能是照猫画虎，缺少好厨师的那种能力——火候。初窥门径的程序员其实缺乏的就是这样的“火候”能力，所以在看《设计模式》时必然遭遇困惑。而这种“火候”能力则源自大量的编程设计实践，在具体的实践中抽象出模式的思维。

“设计模式”是在描述一些抽象的概念，甚至还给它们起了一些专有名字，这又增加了一道弯儿、一层抽象。初窥门径的程序员，具体的实践太少，面临抽象的模式描述时难免困惑。但实践中，经验积累到一定程度的程序员，哪怕之前就没看过《设计模式》，他们却可能已经基于经验直觉地用起了某种模式。

前面我说过我刚学习编程时看过一遍《设计模式》，看完后反而带来更多的干扰，不过后来倒也慢慢就忘了。好些年后，我又重读了一遍，竟然豁然开朗起来，因为其中一些模式我已经在过往的编程中使用过很多次，另一些模式虽未碰到，但理解起来已不见困惑。到了这个阶段，其实我已经熟练掌握了从具体到抽象之间切换的思维模式，设计模式的“招数”看来就亲切了很多。

在我看来，模式是前人解决某类问题方式的总结，是一种解决问题域的优化路径。但引入模式也是有代价的。设计模式描述了抽象的概念，也就在代码层面引入了抽象，它会导致代码量和复杂度的增加。而衡量应用设计模式付出的代价和带来的益处是否值得，这也是程序员“火候”能力另一层面的体现。

有人说，设计模式是招数；也有人说，设计模式是内功。我想用一种大家耳熟能详的武功来类比：降龙十八掌。以其中一掌“飞龙在天”为例，看其描述：

> 气走督脉，行手阳明大肠经商阳..此式跃起凌空，居高下击，以一飞冲天之式上跃，双膝微曲，提气丹田，急发掌劲取敌首、肩、胸上三路。

以上，前半句是关于内功的抽象描述，后半部分是具体招数的描述，而设计模式的描述表达就与此有异曲同工之妙。所以，设计模式是内功和招数并重、相辅相成的“武功”。

当你解决了一个前人从没有解决的问题，并把解决套路抽象成模式，你就创造了一招新的“武功”，后来的追随者也许会给它起个新名字叫：某某模式。

开发框架

不知从何时起，写程序就越来越离不开框架了。

记得我还在学校时，刚学习 Java 不久，那时 Java 的重点是 J2EE（现在叫 Java EE 了），而 J2EE 的核心是 EJB。当我终于用“JSP + EJB + WebLogic（EJB 容器）+ Oracle数据库”搭起一个 Web 系统时，感觉终于掌握了 Java 的核心。

后来不久，我去到一家公司实习，去了以后发现那里的前辈们都在谈论什么 DI（依赖注入）和 IoC（控制反转）等新概念。他们正在把老一套的 OA 系统从基于 EJB 的架构升级到一套全新的框架上，而那套框架包含了一堆我完全没听过的新名词。

然后有前辈给我推荐了一本书叫 J2EE Development Without EJB，看完后让我十分沮丧，因为我刚刚掌握的 Java 核心技术 EJB 还没机会出手就已过时了。

从那时起，我开始知道了框架（Framework）这个词，然后学习了一整套的基于开源框架的程序开发方式，知道了为什么 EJB 是重量级的，而框架是轻量级的。当时 EJB 已步入暮年，而框架的春天才刚开始来临，彼时最有名的框架正好也叫 Spring。如今框架已经枝繁叶茂，遍地开花。

现在的编程活动中，已是大量应用框架，而框架就像是给程序员定制的开发脚手架。一个框架是一个可复用的设计组件，它统一定义了高层设计和接口，使得从框架构建应用程序变得非常容易。因此，框架可以算是打开“快速开发”与“代码复用”这两扇门的钥匙。

在如今这个框架遍地开花的时代，正因为框架过于好用、易于复用，所以也可能被过度利用。

在 Java 中，框架很多时候就是由一个或一些 jar 包组成的。早在前几年（2012 年的样子）接触到一个 Web 应用系统，当时我尝试去拷贝一份工程目录时，意外发现居然有接近 500M 大小，再去看依赖的 jar 包多达 117 个，着实吓了一跳。在 500M 工程目录拷贝进度条缓慢移动中，我在想：“如今的程序开发是不是患上了框架过度依赖症？”

我想那时应该没有人能解释清楚为什么这个系统需要依赖 117 个 jar 包之多，也许只是为了完成一个功能，引入了一个开源框架，而这个框架又依赖了其他 20 个 jar 包。

有时候，框架确实帮我们解决了大部分的脏活累活，如果运气好，这些框架的质量很高或系统的调用量不大，那么它们可能也就从来没引发过什么问题，我们也就不需要了解它们是怎么去解决那些脏活、累活的。但若不巧，哪天某个框架在某些情况下出现了问题，在搞不懂框架原理的情况下，就总会有人惊慌失措。

如今，框架带来的束缚在于，同一个问题，会有很多不同框架可供选择。如何了解、评估、选择与取舍框架，成了新的束缚。

一些知名框架都是从解决一个特定领域问题的微小代码集合开始发展到提供解决方案、绑定概念、限定编程模式，并尝试不断通用化来扩大适用范围。

这样的框架自然不断变得庞大、复杂、高抽象度。

我一直不太喜欢通用型的框架，因为通用则意味着至少要适用于大于两种或以上的场景，场景越多我们的选择和取舍成本越高。另外，通用意味着抽象度更高，而现实是越高的抽象度，越不容易被理解。例如，人生活在三维世界，理解三维空间是直观的，完全没有抽象，理解四维空间稍微困难点，那五维或以上理解起来就很困难了。

框架，既是钥匙，也是枷锁，既解放了我们，也束缚着我们。

两者关系

分析了模式，解读了框架，那么框架和模式有什么关系呢？

框架和模式的共同点在于，它们都提供了一种问题的重用解决方案。其中，框架是代码复用，模式是设计复用。

软件开发是一种知识与智力的活动，知识的积累很关键。框架采用了一种结构化的方式来对特定的编程领域进行了规范化，在框架中直接就会包含很多模式的应用、模式的设计概念、领域的优化实践等，都被固化在了框架之中。框架是程序代码，而模式是关于这些程序代码的知识。

比如像 Spring 这样的综合性框架的使用与最佳实践，就隐含了大量设计模式的套路，即使是不懂设计模式的初学者，也可以按照这些固定的编程框架写出符合规范模式的程序。但写出代码完成功能是一回事，理解真正的程序设计又是另外一回事了。

小时候，看过一部漫画叫《圣斗士》。程序员就像是圣斗士，框架是“圣衣”，模式是“流星拳“，但最重要的还是自身的“小宇宙”啊。

我相信在编程学习与实践的路上，你对设计模式与开发框架也有过自己的思考。欢迎给我留言，说说你有过怎样的认识变化和体会，我们一起讨论。

极客时间

程序员进阶攻略

每个程序员都应该知道的成长法则

胡峰 京东成都研究院 技术专家