解密“设计模式”

有些人问我，你说学习操作系统的最好办法是学习程序设计。那我们是不是应该学习一些“设计模式”（design patterns）。这是一个我很早就有定论，而且经过实践检验的问题，所以想在这里做一个总结。

总的来说，如果光从字面上讲，程序里确实是有一些“模式”可以发掘的。因为你总是可以借鉴以前的经验，用来构造新的程序。你可以把这种经验叫做“模式”。可是自从《设计模式》（通常叫做 GoF，“Gang of Four”，“四人帮”）这本书在 1994 年发表以来，“设计模式”这个词有了新的，扭曲的含义。它变成了一种教条，带来了公司里程序的严重复杂化以及效率低下。

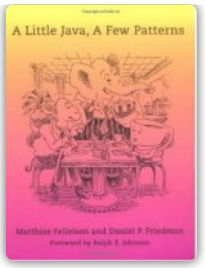
GoF 借鉴的是一个叫 Christopher Alexander 的建筑师的做法。Alexander 给一些建筑学里的“设计模式”起了名字，试图让建筑师们有一些“共同语言”。可惜的是，Alexander 后来自己都承认，他的实验失败了。因为这些固定的模式，并没能有效地传递精髓的知识，没能让新手成长为出色的建筑师。

照搬模式东拼西凑，而不能抓住事物的本质，没有“灵感”，其实是设计不出好东西的。这就像照搬“模版”把作文写得再好，也成不了作家一样。

我孤陋寡闻，当听说这本书的时候，我已经学会了函数式编程，正在 Cornell 读 PhD，专攻程序语言设计。有一天由于好奇这书为什么名气这么大，我从图书馆借了一本回来看。我很快的发现，其实这本书的作者只是给早已经存在的编程方法起了一些新的名字而已。当时我就拿起一张纸，把所有的20来个设计模式跟我常用的编程概念做了一个映射。这个映射居然是“多对一”（many-to-one）的。也就是说，多个 GoF 设计模式，居然只对应同一个我每天都用的概念。有些概念是如此的不值一提，以至于我根本不需要一个名字来描述它，更不要说多个名字！

其中极少数值得一提的“模式”，也许是 visitor 和 interpreter。很可惜的是，只有很少的人明白如何使用它们。所谓的 visitor，本质上就是函数式语言里的含有“模式匹配”（pattern matching）的递归函数。在函数式语言里，这是多么轻松的事情。可是因为 Java 没有模式匹配，所以很多需要类似功能的人就得使用 visitor pattern。为了所谓的“通用性”，他们往往把 visitor pattern 搞出多层继承关系，让你转几道弯也搞不清楚到底哪个 visitor 才是干实事的。

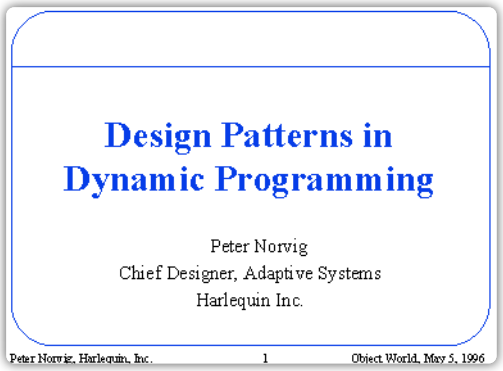
其实，函数式语言的研究者们早就知道 visitor pattern 是怎么得来的。如果你想知道如何从无到有，一步一步“发明”出 Java 的 visitor pattern，可以参考《A Little Java, A Few Patterns》（发表于 1997 年）。



而 interpreter （解释器）模式呢？看了作者们写的例子程序之后，我发现他们其实并不会写解释器，或者说他们不知道如何写出优雅的，正确的解释器。如果你想知道如何写出好的解释器，可以参考我的博文《怎样写一个解释器》。

你说我在贬低这本书的真正价值，因为 GoF 说了：“我们的贡献，就是给这些编程方式起名字。这样让广大程序员有共同的语言。” 如果这也叫贡献的话，我就可以写本书，给“空气”，“水”，“猪肉”这些东西全都起个新名字，让大家有“共同的语言”。这不是搞笑吗。

这不是我的一家之言，Peter Norvig 在 1998 年就做了一个演讲，指出在“动态语言”里面，GoF 的20几个模式，其中绝大部分都“透明”了。也就是说，你根本感觉不到它们的存在。这就像我刚才告诉你的。



在这里 Norvig 的观点是正确的，不过需要小心一个概念错误。Norvig 对“静态语言”的概念是有局限性的。有的静态语言其实也能传递函数作为参数，而且不像 Java 那样什么都得放进 class 里。这样的静态语言，其实也可以避免大部分 GoF 设计模式。而“动态语言”这个概念，在程序语言的理论里面，其实是没有明确的定义的。“动态语言”其实也能进行某些“静态类型检查”。不过在 1998 年，我还是个啥都不懂的屁孩，所以这里就不跟 Norvig 大叔计较了。

既然老人们都有历史局限性，那么为啥我还跟 GoF 找茬？本来这本书很老了，如果没有人再被它误导的话，这篇博文也就不必存在了。可是当我在 Google 实习的时候，我发现几乎每个程序员的书架上都有一本 GoF！我在 Google 实习了两次，第一次的时候代码全都是我一个人写的，所以没有使用任何 GoF 设计模式。代码直接，精巧而简单。当我第二次回到 Google，发现我的代码里已经被加入了各种 factory，visitor，…… 其实啥好事也没做，只不过让我的代码弯了几道弯，让人难以理解。

可见一本坏书，毁掉的不只是一代程序员。鉴于如此，特发此文。各位新手，希望你们敲响警钟，不要再走上这条老路，写出代码来让大家痛苦。