谈程序的“通用性”

在现实的软件工程中，我经常发现这样的一种现象。本来用很简单的代码就可以解决的问题，却因为设计者过分的关注了“通用性”，“可维护性”和“可扩展性”，被搞得绕了几道弯，让人琢磨不透。

这些人的思维方式是这样的：“将来这段代码可能会被用到更多的场合，所以我现在就考虑到扩展问题。”于是乎，他们在代码中加入了各种各样的“框架结构”，目的是为了在将来有新的需要的时候，代码能够“不加修改”就被用到新的地方。

我并不否认“通用性”的价值，实际上我的某些程序通用性非常之强。可是很多人所谓的“通用性”，其实达到的是适得其反的效果。这种现象通常被称为“过度工程” (over-engineer)。关于过度工程，有一个有趣的故事：

http://www.snopes.com/business/genius/spacepen.asp

> 传说 1960 年代美俄“太空竞赛”的时候，NASA 遇到一个严重的技术问题：宇航员需要一支可以在外太空的真空中写字的钢笔。最后 NASA 耗资150万美元研制出了这样的钢笔。可惜这种钢笔在市场上并不行销。
>
> 俄国人也遇到同样的问题。他们使用了铅笔。

这个故事虽然是假的，但是却具有伊索寓言的威力。现在再来看我们的软件行业，你也许会发现：

1. 代码需要被“重用”的场合，实际上比你想象的要少

   我发现很多人写程序的时候连“眼前特例”都没做好，就在开始“展望将来”。他们总是设想别人会重用这段代码。而实际上，由于他们的设计过于复杂，理解这设计所需的脑力开销已经高于从头开始的代价，所以大部分人其实根本不会去用他们的代码，自己重新写一个就是了。也有人到后来发现，之前写的那段代码，连自己都看不下去了，恨不得删了重来，就不要谈什么重用了。

2. 修改代码所需要的工作实际上比你想象的要少

   还有一种情况是，这些被设计来“共享”的代码，其实根本没有被用在很多的地方，所以即使你完全手动的修改它们也花不了很多时间。现在再加上 IDE 技术的发展和各种先进的 refactor 工具，批量的修改代码已经不是特别麻烦的事情。曾经需要在逻辑层面上进行的可维护性设计，现在有可能只需要在 IDE 里面点几下鼠标就轻松完成。所以在考虑设计一个框架之前，你应该同时考虑到这些因素。

3. “考虑”到了通用性，并不等于你就准确地“把握”住了通用性

   很多人考虑到了通用性，却没有准确的看到，到底是哪一个部分将来可能需要修改，所以他们的设计经常抓不住关键。当有新的需要出现的时候，才发现原来设想的可能变化的部分，其实根本没有变，而原来以为不会变的地方却变了。

   能够准确的预测将来的需要，能够从代码中抽象出真正通用的框架，是一件非常困难的事情。它不止需要有编程的能力，而且需要对真实世界里的事物有强大的观察能力。很多人设计出来的框架，其实只是照搬别人的经验，却不能适应实际的需要。在 Java 世界里的很多 design pattern，就是这些一知半解的人设计出来的。

4. 初期设计的复杂性

   如果在第一次的设计中就过早的考虑到将来，由此带来的多余的复杂性，有可能让初期的设计就出现问题。所以这种对于将来的变化的考虑，实际上帮了倒忙。本来如果专注于解决现在的问题，能够得到非常好的结果。但是由于“通用性”带来的复杂度，设计者的头脑每次都要多转几道弯，所以它无法设计出优雅的程序。

5. 理解和维护框架性代码的开销

   如果你设计了框架性的代码，每个程序员为了在这个框架下编写代码，都需要理解这种框架的构造，这带来了学习的开销。一旦发现这框架有设计问题，依赖于它的代码很有可能需要修改，这又带来了修改的开销。所以加入“通用性”之后，其实带来了更多的工作。这种开销能不能得到回报，依赖于以上的多种因素。

所以在设计程序的时候，我们最好是先把手上的问题解决好。如果发现这段代码还可以被用在很多别的地方，到时候再把框架从中抽象出来也不迟。