第 11 章 设计两次

设计软件非常困难，因此您对如何构造模块或系统的初步思考不太可能会产生最佳的设计。如果为每个主要设计决策考虑多个选项，最终将获得更好的结果：设计两次。

假设您正在设计用于管理图形界面文本编辑器的文件文本的类。第一步是定义这个类将要呈现给编辑器其余部分的接口。与其选择想到的第一个想法，不如考虑几种可能性。一种选择是面向行的接口，该接口支持插入、修改和删除整行文本的操作。另一个选择是基于单个字符插入和删除的接口。第三种选择是面向字符串的接口，该接口能对可能跨越行边界的任意范围的字符进行操作。您无需确定每个替代方案的每个功能；在这个时间点，勾勒出一些最重要的方法就足够了。

尝试选择彼此差异较大的方法；这样您将学到更多。即使你确定只有一种合理的方法，无论如何也要考虑第二种设计，不管你认为它有多糟糕。考虑该设计的弱点并将它们与其他设计的特性进行对比将很有启发性。

在你粗略地设计出可供选择的方案后，列出每个方案的优缺点。对较高层级软件的易用性是接口最重要的考虑因素。在上面的示例中，面向行的接口和面向字符的接口都需要在使用文本类的软件代码中做一些额外的工作。面向行的接口将需要更高层级的软件在部分行和多行操作（例如剪切和粘贴所选内容）期间拆分和合并行。面向字符的接口将需要循环操作来实现多个字符的修改。还有其他值得考虑的因素：

- 一种选择是否具有比另一种有更简单的接口？在文本示例中，所有文本接口都相对简单。
- 一种接口是否比另一种接口更通用？
- 一种接口的实现是否比另一种接口的实现更有效率？在文本示例中，面向字符的方法可能比其他方法慢得多，因为它需要为每个字符单独调用文本模块。

比较了备选设计之后，您将可以更好地确定最佳设计。最佳选择可能是这些选择之一，或者您可能会发现，您可以将多种选择的特性组合到一个新的设计中，这个新的设计比任何一个原始的设计都要更好。

有时所有的备选设计都没有特别的吸引力。发生这种情况时，看看是否可以提出其他方案。使用您在备选设计中发现的问题来驱动新的设计。如果您在设计文本类并且仅考虑面向行和面向字符的方法，则可能会注意到两个方法都比较尴尬，因为它们都需要更高层级的软件来执行额外的文本操作。那是一个危险信号：如果要有一个文本类，它应该处理所有文本操作。为了消除其他额外的文本操作，文本接口需要更紧密地匹配更高层级软件中发生的操作。这些操作并不总是对应于单个字符或单个行。这种推理方式应该会让你找到一个面向范围的文本 API，它消除了早期设计的问题。

设计两次的原则可以应用在系统的不同层级上。对于模块，您可以首先使用此方法来设计接口，如上所述。然后，您可以在设计实现时再次应用它：对于文本类，您可以考虑的实现方式包括基于行的链表、固定大小的字符块或“间隙缓冲区（Gap Buffer）”。设计实现的目标与设计接口的目标是不同的：对于实现，最重要的是简单性和性能。在系统的更高

层级上探索多种设计也很有用，例如在为用户界面选择特性或将系统分解为主要模块时。在每种情况下，如果您可以比较几种选择，则更容易确定最佳方法。

设计两次不需要花费很多额外的时间。对于较小的模块比如类，您可能只需要一两个小时去思考备选设计。与您将花费数天或数周时间来实现该类相比，这是很少的时间。一开始的设计实验很可能产生明显更好的设计，其收益将超过两次设计所花的时间。对于较大的模块，您将花费更多的时间进行初始的设计探索，但是它的实现也将花费更长的时间，并且更好的设计所带来的好处也会更高。

我已经注意到，真正聪明的人有时很难接受设计两次的原则。聪明的人会发现，在他们长大的过程中，他们对任何问题的第一个快速构想就足以取得良好的成绩，而无需考虑第二种或第三种可能性。这容易导致不良的工作习惯。但是，随着这些人变老，他们将被提升到越来越困难的环境中。最终，每个人都进入了第一个想法不再足够好的境地。如果您想获得非常好的结果，那么无论您多么聪明，都必须考虑第二种可能性，或者第三种可能性。大型软件系统的设计也是这样：没有人足够优秀到总是能够第一次就做好。

不幸的是，我经常看到聪明的人坚持要实现第一个想到的想法，这会使他们无法发挥其真正的潜力（这也使得与他们一起工作会让人沮丧）。也许他们下意识地相信“聪明的人第一次就能做到”，因此，如果他们尝试多种设计，那将意味着他们并不聪明。其实不是这样的。不是说你不聪明，而是问题真的很难解决！此外，这其中是一件好事：处理一个必须认真思考的难题比处理一个根本不需要思考的难题更有趣。

设计两次的方法不仅可以改善您的设计，而且可以提高您的设计能力。设计和比较多种方法的过程将教会你使设计变得更好或更差的因素。这将使你更容易排除不好的设计，并钻研真正伟大的设计。