封装的变化之封装对象创建的变化 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/141/2021_2022__E5_B0_81_E8_A3_85_E7_9A_84_E5_c29_141223.htm 考虑一个日志记录工具。目前需要提供一个方便的日志API，使得客户可以轻松地完成日志的记录。该日志要求被记录到指定的文本文件中，记录的内容属于字符串类型，其值由客户提供。我们可以非常容易地定义一个日志对象： public class Log{ public void Write(string target, string log) { //实现内容. }} 当客户需要调用日志的功能时，可以创建日志对象，完成日志的记录： Log log = new Log().log.Write(“ error.log ” , “ log ”). 然而随着日志记录的频繁使用，有关日志的文件越来越多，日志的查询与也变得越不方便。此时，客户提出，需要改变日志的记录方式，将日志内容写入到指定的数据表中。显然，如果仍然按照前面的设计，具有较大的局限性。 现在我们回到设计之初，想象一下对于日志API的设计，需要考虑到这样的变化吗？这里存在两种设计理念，即渐进的设计和计划的设计。从本例来分析，要求设计者在设计初就考虑到日志记录方式在未来的可能变化，并不容易。再者，如果在最开始就考虑全面的设计，会产生设计上的冗余。因此，采用计划的设计固然具有一定的前瞻性，但一方面对设计者的要求过高，同时也会产生一些缺陷。那么，采用渐进的设计时，遇到需求变化时，利用重构的方法，改进现有的设计，又需要考虑未来的再一次变化吗？这是一个见仁见智的问题。对于本例而言，我们完全可以直接修改Write()方法，接受一个类型判断的参数，从而解决此问题。但这样的设计，自然要担负因为未来可

能的再一次变化，而导致代码大量修改的危险，例如，我们要求日志记录到指定的Xml文件中。 所以，变化是完全可能的。在时间和技术能力允许的情况下，我更倾向于将变化对设计带来的影响降低到最低。此时，我们需要封装变化。 在封装变化之前，我们需要弄清楚究竟是什么发生了变化？从需求看，是日志记录的方式发生了变化。从这个概念分析，可能会导致两种不同的结果。一种情形是，我们将日志记录的方式视为一种行为，确切的说，是用户的一种请求。另一种情形则从对象的角度来分析，我们将各种方式的日志看作不同的对象，它们调用接口相同的行为，区别仅在于创建的是不同的对象。前者需要我们封装“ 用户请求的变化 ”，而后者需要我们封装“ 日志对象创建的变化 ”。 封装“ 用户请求的变化 ”，在这里就是封装日志记录可能的变化。也就是说，我们需要把日志记录行为抽象为一个单独的接口，然后才分别定义不同的实现。如图一所示： 图一：封装日志记录行为的变化