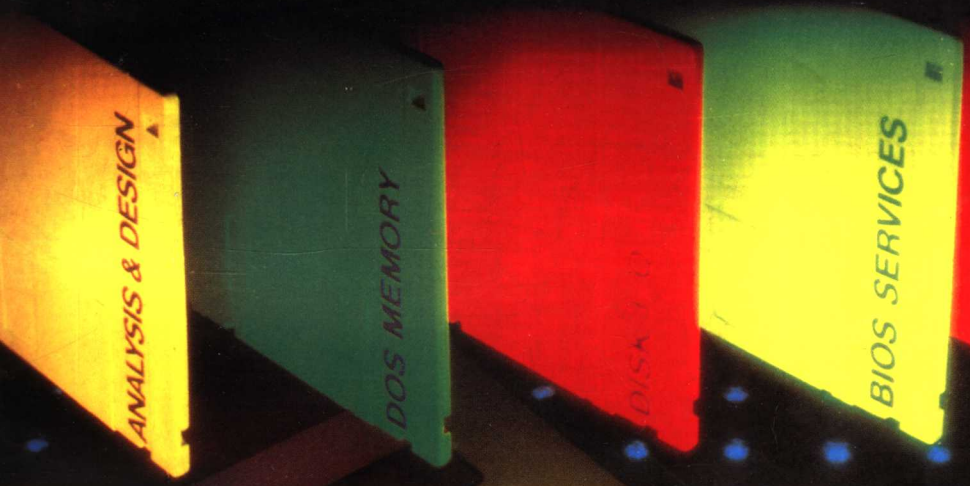
DOS 6 Developer's Guide

Jim Kyle



Reveals dozens of undocumented functions

Advanced level programming

Includes tips and tricks for exploiting DOS 6

开发者指南

SAMS PUBLISHING

科学出版社
龍門書局

希望

Disk contains complete source code, PLUS third party programs and utilities

DOS 6 开发者指南

〔美国〕Jim Kyle 著

刘燕鸣 徐建新 译

熊可宜 校

(京)新登字 092 号

内 容 简 介

本书介绍了在 DOS 6 下进行软件开发的一些技术基础，并提供了一些工具资料。本书的特色是重点讲述开发工具和技术，而不是简单地将 MS-DOS 和 BIOS 中的功能罗列出来。因此，本书对在 DOS 6 下进行软件开发的人员大有益处。

欲购本书者，请直接与北京海淀 8721 信箱书刊部联系，邮政编码：100080，电话：2562329。

版 权 声 明

DOS 6 开发者指南

〔美国〕Jim Kyle 著

刘燕鸣 徐建新 译

熊可宜 校

责任编辑 魏 玲

科学出版社
龙门书局 出版

北京东黄城根北街 16 号

邮政编码：100717

双青印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1995 年 2 月第一版 开本：787×1092 1/16

1995 年 2 月第一次印刷 印张：33

印数：1—5000 字数：770640

ISBN7-03-004748-6/TP·447

定价： 59.00 **元**

前　言

本书针对那些对在 MS-DOS 6 下开发软件感兴趣的人，不管他以前的经验如何。但是，如果读者对第三章所提到的工具一点也不了解的话，最好在使用本书前熟悉一下这些工具。因为如果要详细讲述这些工具中的任何一个，都会写成另一本书。

本书所用的“开发者”(developer)一词与“程序员”(programmer)及“分析员”(analyst)的含义有所区别。开发者的任务是把设想变成一个完整的程序或实用系统。这样，开发者首先必须使用所有的分析、设计工具来理解设想的要点，并把它变成编程思想，以便最后写出程序或实用系统。正如你所要看到的那样，开发者的注意力一直围绕着这种看法的两个要点，两者相辅相成，缺一不可。

软件开发和音乐工作在很多方面有相似之处。音乐工作和软件开发一样，最成功的从事者总是在其创作行为中充分表现出技巧和创新。如果没有坚实的技术基础，再天才的想法也很难令人接受。

本书的目的就是向你展示如何建立软件开发的技术基础，并向你提供许多工具资料，当你开发在 DOS 6 下运行的实用程序时都会用到。本书的重点是在工具和技术上，而不是简单地把所有的 MS-DOS 和 BIOS 功能列出来。我希望本书的这一特色能使它与市面上其他的书区别开来。

虽然封面和版权页上的著者只有我的名字，但这本书实际上是我 25 年来收集别人好的设想的汇总，要想把这些人的名字都写在上面是不可能的。学习就应这样：多方地收集好的设想，并保留最好的。然而这些年来我和一些人共同探讨了许多技术问题，他们向我提了许多好的建议，揭示了许多深刻的道理，因此我想在此向他们单独表示感谢。他们中有 PCED 和其他很多好的实用工具的作者：Chris Dunford，他是 IBMNET CompuServe 研究机构行政官员；Dave Angel，世界上很少有人能超过他对 MS-DOS 内核的了解；以及《计算机语言》杂志的 CompuServe 研究机构全体成员，我自 1985 年以来一直在那里担任首席官员。

在此我要特别感谢 Microsoft 公司的 Ardy Thomas 和 Eric straub，因为他们帮助我及时获得了 MRCI 的许多新特征的资料；感谢 Greg Croy 在要出版这类书时想到了我，并对 Sams 出版公司的全体编辑出版人员的工作表示感谢。

虽然这本书的出版得到了所有这些人的大力帮助，但由于我的水平有限，书中肯定存在一些错误和缺漏，我将对其负全部责任。如果发现任何错误，请务必让我知道，以便可以在今后的版本中改正。同样，如果发现某个地方需要增补资料才更完整，请写信给我，这对本书以后的版本会有帮助。我的地址是 CompuServe UserID 76703,762，经 Cmail 或经 Internet76703,762@compuserve.com。

Jim Kyle
Oklahoma 城
93 年 5 月

目　录

第一部分　为成功作计划

第二部分　你的工具箱

第三部分 技 术

第四部分　参考文献

第一部分　为成功作计划

任何软件开发的起点都是：明确了解要开发的是什么，并大致了解情况。

这部分中的两章："开始之前"和"分析和设计：一个无限的循环"目的在于向读者介绍进行有用的分析以及用有组织的方式进行目标设计所需的技术。这两点在许多程序设计书中都强调得很少，但事实上，如果没有计划，任何事实都不可能成功。

第一章"开始之前"主要讲述任何工程建立一个坚实基础所需进行的初步设计。第二章"分析和设计：一个无限的循环"则集中讲述了任何一个工程开始之前所要做的事。

第一章 开始之前

"我该从哪开始啊，请你指教"。
白兔问仙境中的红桃K*。
"从开始的地方开始"，老K回答。
"往前走到终点，然后停下"。

Lewis Carroll

Lewis Carroll在符号逻辑科学领域中做出过许多贡献。符号逻辑是计算机工业的最直接的先驱，不过，我认为还有比这更有关系的东西存在。然而，许多(即使不是大多数)软件工程似乎没有真正的起点，他们大多"一落地就跑"，急于胡乱上路，缺乏控制。

不管你开发软件是做着好玩还是为赚钱，本书将帮你以职业的方式来开发。首先一个原则就是让每一个工程都有一个真正的起点。本章及下一章集中地讲述如何做工程计划以及如何开发你所要完成的软件。

1.1 知道你从哪里开始

如果你问一些职业软件开发者："首先应做的最重要的事是什么？"你也许会得到不同的答案。然而，如果你让他们列出最重要的5件事，而不只问他们哪个是最重要的，那么大多数情况下，"知道你从哪开始"会出现在每个人的答案中。

原因很简单，拉丁哲学家Seneca给了一个较好的表述"我们的计划总是不能执行，是因为这些计划没有目标。如果一个人不知道他准备去的海港，那么永远不会顺风"。

Lewis Carroll与Cheshire更加言简意赅，当Lewis承认她不在乎去哪时，Cheshire说："这样无论你走哪条路都无所谓"。没有目标，就没有什么来指导你改进，这些努力就会像没有舵的船一样漂泊，最终将或是搁浅或是迷失方向。

决定从哪开始听起来很简单，但大多数初级开发者甚至许多有经验的职业开发者都忽视了这一点。他们盲目地向着未知前进，依靠直觉去带给他们成功。花一点时间来画一个蓝本以便找出路来会可靠得多。然后你就可以估计出你要走多远才能到达，也就可以估计出在路上你可能碰上的麻烦。

1.1.1 定下战略目标

当你开始做任何工程时，无论它多小，都必须对你为什么去做有一个清楚的认识：决定用何等努力来完成。然而我们中的许多人抱着这种态度"让我们去做些外围工作，看看会得到些什么"。我也曾屈服于这种诱惑。

* K——king，在英语中称为国王。上述句子选自《艾丽斯漫游奇境》。

“做些工作”本身并没错，并且它是了解自己知识和工具的局限性的好办法。当你在这方面“做些工作”时，你实际上是在向着这些目标前进。“学点新东西”、“找乐子”，“在委托报告来之前打发时间”，事实上在这些例子中你的活动都有一个目标。知道你该做什么会对自己有帮助。

如果你的目的是“学点东西”，则应避免用你已知的方法来做这些事。这种方法应用于时间短任务紧的工程才合逻辑。用老方法不会刺激新方法的产生，也就对实现目标没什么好处。

类似的，如果目标定在月底之前完成一个工作帐目系统，那么你就不该把时间浪费在试验新方法上。为实现这种任务目标就应首先选用经过试验为正确的方法，尤其是当你的杂货店正等着这个系统完成时更应这样。

你该如何确定一个工程的目标？第一步是显而易见的，但却经常被忽略。你应该问你自己或是把任务交给你的人“为什么要做这个工程？”

这个问题回答时一定要诚恳。“因为我想有一个新程序来卖”。虽说这是一个极好的有价值的战略目标，但却没有道出新程序要做什么或它是什么样。它所告诉你的只是能把程序卖出去就行，不管这程序到底是干什么的。这就意味着下一步你的行动方向就是找出市场上什么类型的程序走俏。

那个简单的回答已把你当前的任务从开发程序变为市场调查以确定应开发程序的类型。这已超出本书的讨论范围。无论如何在你做完“家庭作业”并在目标上做出决定后，你就得到了一个新工程和一个战略目标，在以后的开发过程中你还会回到这一点。

为一个工程建立战略目标是分析的第一步，它包含了第二章“分析和设计：一个无限的循环”中的许多内容。分析，就它本身而言，在软件开发中和设计实现软件一样是一个耗时的部分，而它因急于着手而被忽略。

许多优秀的书中已经提到这种分析技术，而且正在被广泛接受。本书中对这一点并没有在很深的深度上加以考查。

你应该马上关心的是，即使最小的任务也应有一个目标，否则你就不能决定什么时候能完成它。这个目标随着任务的进展也会不断改变，而且可能会分裂出子任务和子目标，如果没有它，你就不是在“开发”一样东西，只是在怀着得到点什么的希望中滚动车轮。

1.1.2　确定工作量

在你为自己的工程确定了一个战略目标后，下一步就是确定工作量：你应该确认这个工程可能大到什么程度，在本阶段你不必要求完全精确，但你至少应知道它只是一个 2 人·时的任务或是需要半打程序设计员在以后的三年中忙个不停。

虽然上述例子有些极端，但仍不能这样假定：显然用两天来计划一个两小时的任务是没什么好处的，而且如果你是一个人工作，你接手一个需要多个人协同工作任务的机会是很少的。当一个可能的用户问你能否接手一个特定的任务时，你应该在回答之前确定是否在你的能力范围之内，而且在每种情况下“能力”包括工作量。

要想成为一个职业软件开发者或想保持这种水平，不断估计你所遇到的每个任务的大小是很必要的。与作为一个职业音乐家一样，日常练习能使演出更流畅。当你形成估算任务的习惯后你就能较准确地给出一个快速的估计，这样使你更具有竞争力。

要想形成估算任务所要时间的习惯该怎么做呢？这和钢琴家练出用手指快速弹奏琴弦的过程一样：不断地痛苦地注意令人厌烦的细节。

起点就是形成一种好习惯，即在开始着手干之前估算每个任务所需的时间，在工作中不断记下笔记以及当你完成工作后，你就能确切知道它到底花了你多少时间，并可以与你起初的估计相比较。如果是为别人工作，你也许早就因工资单的原故而做过这些了。但是，在绝大多数情况下进入工资部门的"时间记录"与实际上花在一个工程上的时间差得太远。不仅如此，在你"存时间"的帐号上一般不只一个任务而是多个任务。

因为上述原因，你也许会发现保留自己的时间记录是很必要的，哪怕是给自己干。目的是为了搞清楚一个典型的任务要花费你多少时间。

开始你可能会感到迷惑甚至失望，因为你会发现大多数任务所费时间是你起初估计的4～20倍。而且，这是个普遍现象。在软件行业中，对时间的估计普遍是100%的错误。MS-DOS 6.0计划于1992年夏末发表。而DOS 5.0却比公布的最后"校定"计划要长6个月。计算机工业中不能准确估计工作实际将耗费的时间的现象很普遍，所以如果你的估计与实际只相差50%的话，你已经远远好于平均水平。

1.1.3 确定子任务和子目标

除了那些最简单的任务外，对任何任务你都会发现确定工作的战略目标后仍遗留下许多关于如何实现该目标的问题。这是因为，除了那些最简单的任务外，每个任务都是由许多子任务组成，每个子任务都有自身所需达到的目标，而那些子任务也是由更下一层任务组成。

如果这些你听起来觉得好像任务目标的确认是不可能的，于是工作也就不可能开始，那么你正在接近问题的实质。几千年前希腊哲家芝诺发现想要达到某个目的地的运动包括通过一系列连续点的过程，并提出了一个至今仍有许多人加以引用的诡辨以证明运动在逻辑上是不可能的。

简单地说，芝诺诡辩的要点是因为人必须首先通过路径的一半，然后再通过余下的一半才能到达终点，该过程可以继续下去，而余下的路程总是无限可分的。不断对某个值减半是不可能减为零的，因为"某个东西"的一半仍是"某个东西"而不是"什么也没有"，不管它是多么微小。

芝诺的矛盾仅说明了他是以一个错误观点来看待问题，否则我们就永不能到达任何别的地方。同样的，确定子任务并不要求你花上无限多的时间以确定完成任务所需的每一步。

你要做的只是记下在确定真正的战略目标时所想到的一些思想火花，也就是子任务的雏形。

确认目标能使你在实现了该目标时知道可以结束任务。确定子任务在你通往最终目标的路途上将会是很有用的提示。收集那些在你研究情况时进入你脑子里的思想火花能使确认子任务更容易些。

把这些思想记录到纸上能使它显得比实际上更正式，这一点很重要。这样，当你确定好战略目标后，就会对收集那些在开发过程中用得着的资料有个好主意，你的记录可以简单到只在索引卡上或"快贴(post it)"上草草几笔。我经常把灵感写到那些从程序清单上撕下来的碎纸片上。这些程序清单在我桌上多得很。

在确定了战略目标，并对工程所需时间有了一个最初的估计，对子任务有了主意后，你对任务的去向就一清二楚了。然后就可以进行你初始设计的第二个重要步骤了，并且会明白为什么你在该"旅途"上。

1.2 弄清楚你干的原因

不管你手头的项目是什么，为使你能成功，必须对任何能影响你对当前任务看法的东西加以注意。也就是说你必须注意自己的动机，项目对你的时间安排的影响，以及可能产生影响的其他因素。必须记住这不是对那些因素加以判断，也没有最好最坏之分，你只是需要注意那些重要的事实，以便在前进路上做出诚实的决定。

三个重要问题能帮助你明白为什么开发这个项目。首先，问你自己工程的动机和你自己参加的动机，如果这二者不同的话。确认这个后，问你自己该项目的首先要做的工作是什么，它将怎样和你与工程无关的活动互相影响。最后，问你自己，如果该项目不是短期能完成，那么将来的变化会怎样影响该项目。

1.2.1 动机是什么

知道该项目的动机对一个项目来说很有必要，不论该项目是你自己做着玩的还是对你的雇主的将来很重要。事实上，并不是说没有一个诚实的强有力的动机，项目就不可能成功。

许多情况下，项目的动机和你自己加入的原因会相同，当你自己唱独角戏或开发自己的程序时，这两者总是一样的，唯一可能不同的是当你为别人工作，分得一个你自己没有最初参与和建议的任务时有点差别。

为解决两种动机不同的情况，下述讨论都假设他们不同。如果你的情况是两者相同，那么，只需把不同混在一起即可。

总动机一般是由个人或主要投资团体来设定，无论它是雇主或顾客，它也许很有可能表述如“做点有效益的事”或想到“我们准备在这个工程中把编程艺术提高一段”。

然而，你也许会发现根本不可能从你的上司或顾客那儿得到动机的明白阐述。许多情况下，你不得不问一些富有外交技巧的问题以揭示项目存在的真正原因。

其他情况下，原因是很容易发现的。如果你的顾客的资料存储系统失效，而备份盘又不可读了，那么该项目的动机就很明显：对备份是解码，并恢复信息；当明显的解释已经足够时不要花时间去寻找隐含的原因。

第二个应该明确的动机是你自己参与的目的。这也许很简单而乏味如“为了保住职位或充满感情的”，“为挽救我上星期的工作”。

不管理由是什么，目的是对你干的原因和你要干的事有充分的清楚认识。这样你就不会因错误假定而被引入歧途。你不必把它们写下来或告诉他人，它们只是你自己作为一个开发者的自我指导。

强调这些带有反省性质的观点如“动机”原因是：为了做出最好工作你往往需要求助于主要动机。如果你发现工程的动力和你自己的个人动机之间严重冲突，那么最好一开始你就不要介入。

1.2.2 确定优先做的事

动机确定后，下一步当然是确定先前确定的子任务的解决顺序。比如说，在确定输出设备前就与程序的输出任务的细节打交道，是毫无意义的。然而那些不慎重的开发者总是掉到此陷

阱中，他们急于书写代码。在设定工程内部优先程序之前，你必须确定是否有与工程无关的麻烦存在，如果有，他们对工程将是个危险。

非工程麻烦

如果你是独立编程而被人雇用，并且从对你所分得的任务有一定的控制权的话，你的初始工程计划最好能含有处理非工程本身的麻烦的部分。他们对你能否有较为职业的表现有很重要的影响，如果你诚实的话就应在工程开始之前就做好这件事。当然，即使你对任务没有控制权，它同样重要，虽说你不大可能做得像前面所说的那样多。

在任何情况下，你必须有你自己对冲突情形的解决办法，并按你认为对的方式加以解决。你首先必须认识到它们的存在，以便写入你的工程计划中。为使你的计划能成功，每个可能的非工程麻烦都必须写上来，如果可能，在定时间表前加以解决。

工作中优先做的事

第一个优先做的就是当你定一个目标并解决了潜在冲突后，你需坐下来想一会，而不是急着去写代码。

每个人都会有这种冲动。尽管我有 25 年来在大中型机、小型机和微型机系统上工作的经验，但我仍经常忍不住那么做，尤其是当我做一个“快捷而令人讨厌”的程序时。然而每次我都不得不回过头来重新组织我的思想，以找出一条可行之路。

现在你可以确定你的进程了。你已经有了一个战略目标并且不会有什么非工作麻烦存在了，然而，别的因素仍需加以考虑，如：

- 该程序的最终用户想让它有什么功能？
- 最紧张的消耗是什么？
- 该程序将运行于何种类型设备之上？
- 完成工程需要多长时间？

在你能开始设计之前，你必须回答这些问题。第二章将对这个问题进行深入讨论，目前，你必须找出工程最基本的要优先考虑的事件。

不要把精力全放在如何解决问题上而应该弄明白最终用户想让这程序干什么。

你进行完第一轮需求分析之后，就可以写出解决方法草案。然而，在开始写代码之前，你最好收集用户关于分析的精确性和设计的完整性的意见。

你也许会花同你写代码一样多的时间来分析，但是，该阶段你发现的每个错误的时间决不会多于你用于写不得不扔掉的代码的时间，计划比把计划代码化任务量更小。

第四个要优先加以考虑之事实际上是把设计代码化。既使你在设计阶段把时间结算得很好，你也会发现你在写代码时不得不重新做一些设计；当然由于你的计划，这种事会少得多。

最后，你必须确信程序能符合所有的要求，如果是，你的初版宣告完成，然而如果最初设计和最终测试有很大的需求不同，那么程序就失败了。因为这种事经常发生，所以你不得不在计划发表之前给出足够的时间进行测试修改工作。

建立任务图

在你确定工程的初始优先问题后，就可以建立你的第一张任务图并给每一个标上估计的

日期。这和地图在旅行中的用途是一样的：确定你自己的方位以及你想去哪里。

任务图的最大好处就是使你可以确定子任务的时间。比如，你可以把“分析”部分的子任务都标到任务图上并标上暂时性的起止时间。

标上时间的任务被管理者称为“路标”，他们认为工程管理是一门科学而不是艺术，认为如果你把任务分解并按时间组合，以便一点一点地解决，这样你就会做得更好。你必须把分数点分得足够多，这样在两点之间你就不会误入岐途，而完成时间也会分秒不差。如果每一段都圆满解决，那么整个工程也一样顺利完成。

把计划留给后来者

在你建立任务图并把它分解成段后你就可以把你所经过的记录下来作为永久记录。你可以在你开发的系统中建一个工程计划文件，在工程进行中当你需要参考时就可随时取用。

如果你有带块操作功能的字处理器，那么可把它用于起草计划，该字处理器最好有块操作功能，它将是你早期程序开发的最有价值的工具。

块操作功能之所以有用是因为在你计划过程中，随着分析的成熟，并设计出实用程序，你经常会把大量的东西从一个地方移到另一个地方。

块操作使这种改组变得简单，别的技术都很麻烦，反而会把你原来集中在考虑词汇去向问题的精力分散了。

1.2.3 将来会怎样

在考虑实现细节如支撑平台或使用工具之前，你应考虑一下该工程的未来。该程序是否有升级的可能？该程序是打算胡乱编写以应一时之需，还是打算作为你以后工作的基础？

版本升级也就是把手头的程序作为将来在同一类型系统下运行基础的方法。

即使最初的计划不要求这一点，但你必须认识到这一点。正如许多自动仪器一样，许多程序都会不断翻新，哪怕只是换一下仪器的位置。对讲究效益的开发者来说，它是不断赚钱的基础。

虽然在许多人眼里版本升级只是一个市场工具，但实际上它却很有意义。例如，当用户提出的要求已太晚而不能直接编成程序，则可通过升级的办法来加进新功能。

类似的，由于时间或预算问题不能在第一版中完全解决，可以先把主要部分解决并在升级版本中克服该缺陷。

而且由于技术进步使得系统功能改进日新月异，你可能会发现，当新功能可能出现时，为要和它们保持一致就不得不经常升级你的程序。

在你把升级可能排除之前考虑一下这些因素。即使不需升级，而按可以升级的方法来组织程序，在你的决定改变时也不会损害什么。

如果确信没有必要升级，而你一定要升级，你一定要加倍小心地开发程序，因为这就像在踩钢丝而下边没有安全网一样。当你的用户或老板坚持说该程序仅是应一时之需只用一次，无需太在意，千万别相信。这种程序将是用得最久的。

1.3 弄明白你带的是什么

选择好一个平台，充分认识到经常变动的不可避免性，然后选择你可以打败未写的程序这

个怪物的武器，这样你和着手写程序之间只剩三件事。下一部分将详细讨论。

1.3.1 确定支持平台

多平台编程近年来是个热点，大概意思是说开发者最好能开发在常见平台如 MS-DOS、Windows、OS/2，也可能是 UNIX 上都能同等运行的程序。不幸的是，“同等”经常意味着程序所有版本都同样低劣。

虽说本书只讲了在 MS-DOS 下开发软件，而且是 MS-DOS 6.0。但并不意味着你的设计只适用该系统。在你对用户的要求作了足够的分析并想出了解决问题办法后，即使易寻变的用户想把它用于另一个系统也不必重新来做。

当在多平台下运行程序成为必要时如果你想使你的程序大多不变。你要做的事有：(a)在你计划的最初就考虑到转移到别的系统(如 OS/2、UNIX Macintosh)的可能性；(b)把在转移过程中可能不兼容的部分限制到程序的极小一部分，这样，转移到别的系统下时只需修改该小部分。

对于预见不同系统兼容的可能性，你必须确认问题中每个可能依赖正在使用的系统的部分。比如说，你的用户要求在程序中使用交互式拖动鼠标，这样就会使程序在硬件不支持定点仪的系统下失效。

类似的，要把一个依赖在 MS-DOS 环境下非常普遍的像素式图像显示器，如很平常的微机显示卡 VGA 的程序移到一个“哑终端”，没有图形显示器来作为用户界面，是不可能的。

不同系统的兼容性并不总牵涉到整个系统，有时只是硬件的一部分需要改变。刚开始 MS-DOS 所用的彩显只有 16 种颜色，此后的显示器可显示 256 色，最近的有 32,000 色，甚至一百六十万种颜色。

在这些情况下，把依赖硬件的部分“隔离”到最小可能的范围，这样当你认为需要支持不同的设备时，影响会小些。然而，为实现该目的，你必须对问题的隐患明查秋毫。

要不漏掉任何重要细节，这一点从一开始就要牢记在心。对每台设备的每一个可能影响兼容性的方面都要加以检验。当在别的系统下运行程序的时刻来到时(如果你的程序成功的话，总会有这一天)你会很高兴你做了这一切。

1.3.2 检验程序的可变性

有人说过：“不变的只有变化本身”。在软件开发界我认为这句话是对的。每个开发者都必须把设定最高级别的版本作为首要工作要求，然而到目前为止我还没能发现一个，它能准确到能把我从几乎每天都有的令人不快的惊奇中解脱出来。

于是我开始承认变化的不可避免性，并把我的注意力集中到去寻找充分解决的办法，同时还要把它对我精心准备的计划的影响减到最小。

我发现的最有效的方法就是永不把自己置于死地。每次做决定我总是留下最大可能的自由度以备应付将来的变化。

我把所有特征都置于可增加的方式，以备将来增加新特性。类似的，我让每个任务都含有“非以上任何项”的缺省选择，以便将来扩充设想到的方向，即使现在它毫无用处。而且我尽力使我设计的每个核块都是可重复使用的，这使我很容易从已存在的程序中拿一块下来加到手头的程序中去。

到目前为止，我仍不知道如何预见未来会带给我的变化，但我在所有的材料中都加了预防措施，这样一旦变化来临，应变也容易了。我极力向你推荐此法。报酬比投入丰厚得多，这是因为用这种方法做出的程序在做少量修改后的可用率很高。

实践证明，在你把程序交给用户后，用户会发明你认为不可思议的用法。如果你的设计鼓励这种作法，那么一定会获得成功。如果你用一种"讨厌"麻烦的眼光来设计你的程序，那它只会是一副产品。

1.3.3　确定工具

第三章——"语言、编译程序以及汇编程序"就工具和选用何种工具进行了讨论。在讨论之后你应该清楚一个事实。你选用的工具是由用户要求所决定的。

你会用的工具越多，灵活性就越大。你所接触的任务种类越多，你要求熟悉的工具越多。一个只致力于数据库的构造和维护的开发者可能一生除了使用多种数据库管理系统外就不会使用任何工具，但一个处理多种任务的开发者熟悉足够多的工具是值得的。

"Turing 理论"所证明的计算机先驱者艾伦·图灵在计算机发明前所发明的逻辑等式指出：任何适合一定要求的系统能做另一个适应该要求的系统可能做的任何事。今天的要求是如此之少，任何一台计算机都能适应。

也就是说你能用你掌上型机做 Cray 超级机所能做的一切。也许大型机在几毫秒中所做的事由小机器来做要花上几年，但理论上没有问题。

对语言工具来说该理论同样适用。我们的数据专业人员如果需要也可以用已知数据库工具开发一个能运行的操作系统，但是可能需要好几年，并且完成后速度一定很慢。但以上的"可能"并不意味着"实用"。

如果我的讨论你只能记住一点，那必须是：如果你想成为一个好的开发者就不要掉到一个已有的陷阱里：当你只有一只铁锤时，把一切都看成钉子。即使是一个单一的任务，只有一种工具永远不会是最好的。因此，一个开发者需要各种各样的工具。

早期"高山人"依靠一把斧子和一把刀度过寒冷的冬天。今天灵巧的人只需一些宿营和铺设工具就能应付复杂的现代生活。例如一个熟练的木匠会有一套专用工具，每一个都只能做一件事，但都接近完美。

在所有的例子中，每个人都有为应付手头工作的工具，你编程序可按以下模式：

- 如果你只需可以在其他领域荒地上生存下来的编程工具，你可以仿效"高山人"只带可以达到你的目的最少的工具即可。
- 如果你渴望成为一个"过路"开发者，你只需在编程级别中差不多高就成了，那么你只要一小套工具。
- 然而如果你奢想成为一个编程高手，鉴赏家一级的高手，那么你必须不停地收集工具，对自己为熟悉每样工具而大量花费的时间也不加报怨。要想达到这个目标，一生都太短暂，然而追求本身不就值得追求吗？

第二章主要讲述工程中问题的分析技术，并就分析结果提出可行的解决办法的技术。