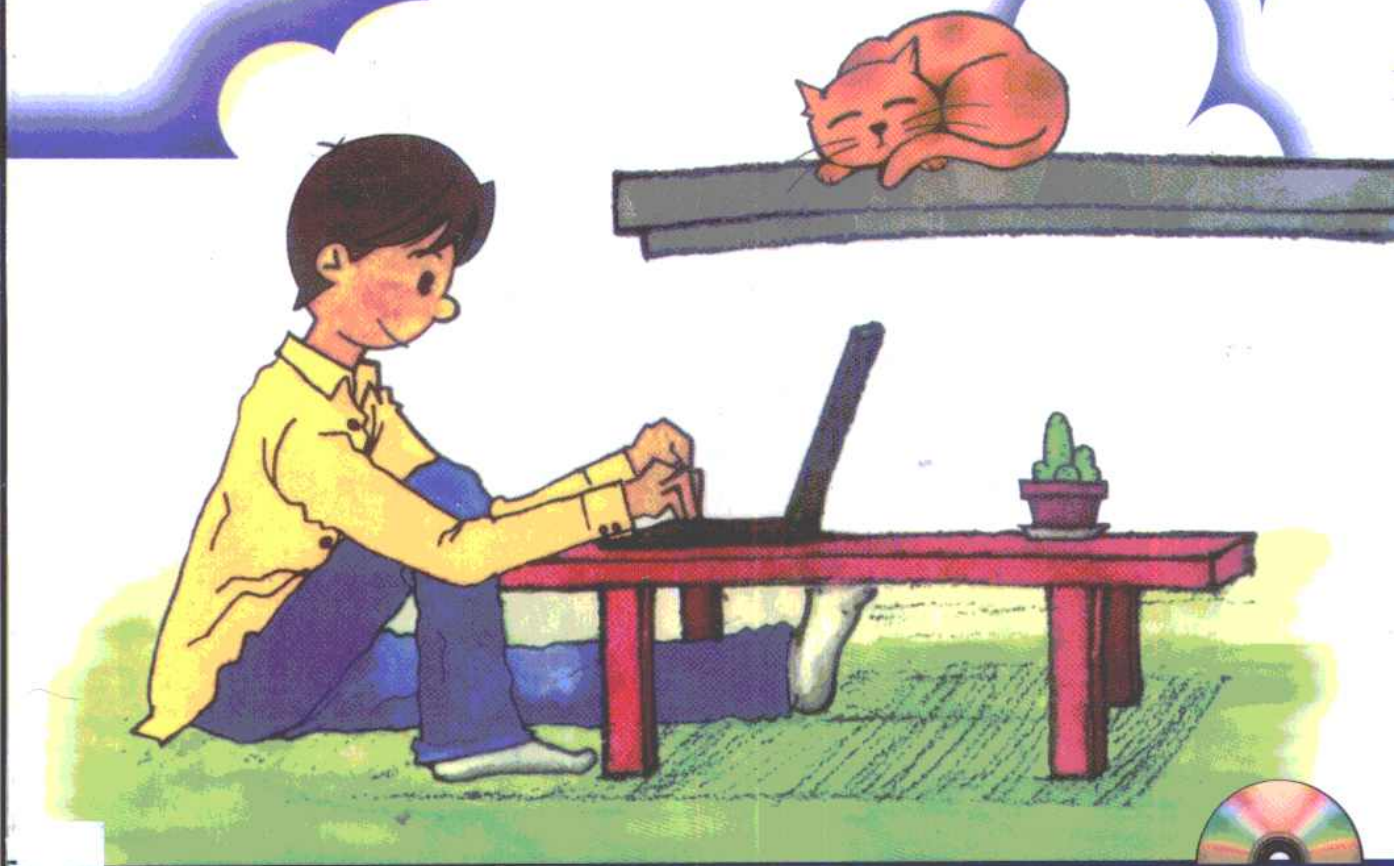
GOTOP

北京科海培训中心

陈宽达 著

Delphi 深度历险



CD-ROM

科学出版社

GOTOP 北京科海培训中心

Delphi 深度历险

陈宽达 编

科学出版社
2001

著作权合同登记号:01-2001-2528

内 容 简 介

这是一本很具特色的书籍,作者通过自身的编程经验告诉你在拥有了基本的程序设计能力后,如何进行自身发掘问题、解决问题,并在程序设计中寻找乐趣的最高层。

全书分为基本概念,操作系统,桌面秘笈,编写游戏,软件开发5大篇。主要内容包括:RAD编程工具的优劣,VCL基本概念,使用控制面板,定时器,桌面世界,背景主题工具实战,编写屏幕保护——自己的计划表,编写足球赛游戏,坦克大决战游戏,Fancy软件编写手则。

本书针对于具有Delphi编程基础并对Windows SDK有基本认识的使用者。

版 权 声 明

图书在版编目(CIP)数据

Delphi深度历险/陈宽达编. —北京:科学出版社,2001.6

ISBN 7-03-009448-4

Ⅰ.D… Ⅱ.陈… Ⅲ.Delphi语言-程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字(2001)第040514号

科 学 出 版 社 出版

北京东黄城根北街16号

邮政编码:100717

北京门头沟胶印厂印刷

科学出版社发行 各地新华书店经销

*

2001年7月第 一 版 开本:787×1092 1/16

2001年7月第一次印刷 印张:27.5

印数:1—5000 字数:674 880

定价:47.00元(含光盘)

第 1 章　RAD 的是与非

容易让初学者困惑的是 RAD 的弊端，
但 RAD 的优点是不容忽视的。
它只是使门槛降低点，让程序设计更为简单轻松。

对于刚接触 Windows 程序设计的新手而言，要在这么多的开发工具中，选择一个最适合自己的开发工具不是一件容易的事。不只是新手，就连许多老将也容易执着于同一套开发工具，宁可使用钟爱的工具埋头苦干，而不去理会更方便好用的解决方案，即使可以省下 3 倍的工作时间。

的确，进入新的程序设计领域前，开发工具的选择是最重要的一项课题。选择正确，在你设计程序时就能得心应手；万一不幸选到难学难用或者根本不适合项目性质的开发工具，那么，带给你的可能是在每一个漫漫长夜里，面对计算机独自摸索尝试的命运。

这种选择是十分无奈且困难的。如同年纪轻、阅历浅、视野小、知识窄的年轻人非得在刚放下历史课本、伟人传记的青年时代做出一生路途的抉择，做好人生的规划；刚接触一行行冷涩生硬的程序代码的入门者，也势必在概念模糊、谣言四起、真伪难分的情况下，面临选择程序语言及开发工具的难题。

1.1　选择编程语言

对于“我是 Windows 程序设计的初学者，该选择什么开发工具好呢？”这类经常见到也会永远存在的问题，我曾在网络上见过这样的回答：“Visual C++用户最多，所以推荐给你”、“VB 好学，所以选它一定没错”、“C++Builder 是 RAD 工具，简单易学，我学 3 天就会写了”等等十分没有概念，说是不负责任也不为过的言论。

不要听别人说，我们还是要根据我们的实际用途作出选择。就像人类战士通常双手握着巨剑，侏儒通常肩着战斧，而魔法师无可选择地手执魔杖一样，选择一把称心的武器还是得视个人的情况及需求来决定。

1.1.1　狂热分子的信仰

现在有许多计算机玩家常不由自主地对自己所钟情的操作系统、开发工具、应用软件甚至游戏及软件公司产生不明事理的狂热。在此情况下，很容易去找一个貌似敌对的“对

手”来用，以坚定自己的信仰、壮大自己的声势，例如 PC vs MAC、FreeBSD/Linux vs Microsoft Windows、Visual C++ vs C++Builder、Delphi vs VB 等等。

在现在的众多的开发工具面前，我只想提醒那些狂热分子，拥 X 反 Y 并没有什么不对，但是：

> 一个人若是为有了宗教信仰而骄傲、自满，甚至因此鄙薄无信仰的人，或动辄排斥与他信仰稍稍不同的人，便表示他自己还没有找到信仰，所以，他也在他自己鄙薄和排斥之列。
>
> <<疑神>> 杨牧

TANet 一位重量级的人物也曾语重心长地说：“科技的出现应该是要为人来服务的，你尽可以拥抱 X 痛骂 Y，不过切记，知己知彼百战百胜”。这是看到网络上一堆连保护模式、文件系统、系统软件、排程问题、虚拟内存都不知何物的网友痛骂 Windows 操作系统，连 API、DLL、对象导向、application framework 都摸不清脉络的新手却大声疾呼推行或反对某某程序语言或开发工具的怪现象所发的无奈之语吧。我也有同感。

1.1.2 学习目的

就我自身的经验而言，程序设计初学者的学习目的大致可分为以下几点：

不为什么，就是想学

有趣的是，这类初学者的比例还不小。也许是羡慕程序高手对于计算机系统的了若指掌，也许是希望也能拥有创作软件的能力，也许就是单纯地觉得编写程序代码是件神秘有趣的事，目的说不上来，没有很强烈的学习欲望。

这种类型的初学者由于动机不足，通常只能维持 3 分钟热度，《XX 语言入门手册》看了一整年还停留在第 2 章。若你属于这个类型，我的建议是：“跟着感觉走”。程序设计只是编写程序语言，与计算机聊天，请它为你做事的沟通方式罢了。对于一个不玩电子游戏机，也从来不想到日本去游玩或留学，工作上也没有日文需求的人来说，日文的听说读写能力不具有太大意义。那么，如果你没有事情想直接跟计算机沟通，学习程序语言又有何益？

如果没有具体的要求，最好现在就订出一个计划来。有了目标的存在，才能确认正确的学习方向，千万不要无所求式的盲目学习，那只是白费力气罢了，看电视综艺节目都还比这值得。

项目需求

由于项目的需求，各行各业有无程序设计基础的人们都得进入 Windows 程序设计领域，这是十分无奈但最常见的情况。

也许是专业分工的认识还不足。若要面对外国客户时，我们不会要求某个秘书开始 3 个月前去学该国语言，而会想办法请一位翻译人员帮忙；但是对于程序设计，常常会要求自己的员工，无论是什么背景，送到培训中心学个一两个星期，回来后就期望他们能够将一年半载的软件项目顺利做好。

当然，口语的学习及程序语言的学习在时间上还是有差别的——口语比程序语言灵活

得多了。不过，口语学到一半的人们说起话来结结巴巴，半天也凑不出字来，我们看得十分清楚；程序语言学到一半的人们，写起程序来丢三落四，往往连逻辑观念都还没搞清楚，就得从零开始编写几千行规模的程序，他们的困窘，外人不容易看得见。

我心目中的理想状态是，有着足够数量足够专业素养的计算机程序设计人员，供各行各业大小公司短期或长期聘用，与兼具计算机素养及 domain knowledge 的人员沟通合作，解决各领域的计算机软件需求。

让“人能尽其才”是最重要的，我已看过许多物理、光电、材料、数学领域的同学及朋友，为了实验室或计划的需求，得暂时抛下自己的专业，去学习程序设计来控制 RS232 连接端口，编写适配器驱动程序等等，一来短期学习所编写的软件品质让人担忧，二来浪费太多能够增强专业能力的时间。

如果因为项目的需求而进入程序设计领域，一般来说，程序语言及开发工具的选择上不会有太大的弹性，通常必须配合上司的好恶或团队原本的方案来斟酌考虑。

追求理想

许多人当初都是为了一个伟大崇高的理想才进入程序设计领域的。而就我所听闻，游戏程序编写似乎经常扮演这个“伟大崇高理想”的角色。

其实，笔者就是活生生的一个例子。高中时代，迷上创世纪系列游戏的我，成天幻想着编写一套很棒的RPG游戏，而这个梦想就成为驱使我自学C语言及汇编语言的强烈动机。一脚踏入程序设计领域后，无法自拔的我很快地全身沦陷，接着就是你们现在所见的这个状况了。

既然提到我的兴趣，顺便来谈谈游戏程序设计初学者的选择吧。

游戏程序设计是程序设计各领域中，狂热及理想成分比例最高的一个群体，有许多各种性质的程序设计师，当初都是因为热衷游戏设计，而就这样掉入程序设计的漩涡里。学习动机是毫无疑问地：“想要具备编写游戏程序的能力”。不过，按游戏种类的不同，日后学习的方向与重心也差得很远，例如 2D RPG，重点在于画面设计及故事剧情，外加 DirectDraw 提供的快速滚动条功能；3D 动画游戏，Direct3D 或 OpenGL 是跑不掉的，对于图形学的理论，算法及应用方面，也最好花些时间去努力研究学习；再如多人联机棋类游戏，DirectDraw、Direct3D、DirectInput 我想都用不上，好好研究提供联机功能的 DirectPlay 或发展一套稳固的实时数据传输链接库才是重点。

1.1.3　目前基础

现在的基础是决定工具及语言上手速度的最重要因素。

许多人在高中计算机基础课程教的是 Basic，通常以 Quick Basic 作为开发工具，上大学后接着学习 VB，衔接得很好。大多数大专院校的计算机概论课程教的是 C 或 C++ 语言，也有些信息相关科系的计算机概论课程以 Pascal 语言作为教学内容。熟悉语言之后，他们正好可以选择 C++Builder、Visual C++ 或 Delphi 作为 Windows 环境下的程序开发工具。

除了 Basic 以外，我建议你应配合你目前的学习基础，拥有 Pascal 语言基础就选择 Delphi；爱用 C/C++ 语言就选用 C++Builder 或者 Visual C++。

1.1.4 个人爱好

Basic、C/C++、Object Pascal 这 3 大程序语言，雄霸着整个 Win32 程序设计领域。Basic 易学易用，Pascal 严谨明确，C++ 强大复杂，各有各的拥护者及其理由。

易学易用的 Basic

在 Win32 环境下，Microsoft Visual Basic 是使用 Basic 语言最著名的开发工具。

Basic 语言十分简单易懂，微软希望 VB 及 VBA 维持着简单到任何想依赖计算机进行自动化程序的计算机用户都可以轻易地上手的程度。因此虽然附加的功能不断上叠，语言本身保持着 Basic 的原有特性。

Talk

公元 1964 年，BASIC 语言（全名为 Beginners All-Purpose Instruction Code）诞生，由 Dartmouth 大学的 Thomas Kurtz 及 John Kemeny 两位教授开发，他们希望 BASIC 语言能作为学生在学习功能比较强大的程序语言（如 FORTRAN、ALGOL 等等）前的踏脚石。

Basic 易学、易用的特性是众所周知的，同时它还兼具容易转译为机器码的特性，最著名的故事就是当年 Bill Gates 在毫无测试出错机会的情况下，编写出一套程序代码加数据不到 4K 内存的 Basic 直译器。

第一个 Basic 编译器是 Quick Basic（在这之前，Basic 程序都通过直译器来执行），由 Microsoft 开发。Quick Basic 发展到 4.5 版后，Microsoft 另外发展 Quick Basic Extended，也称为 PDS Basic（Personal Development System），PDS Basic 发展至 7.1 版结束，接着就是大家所熟知的 Visual Basic 了。

必须承认的是，Basic 并不适用于中大型应用程序的开发。它的先天条件不足，例如执行速度缓慢，未提供完整的对象导向程序设计机制等等。再加上后天失调，例如硬加入部分的对象导向机制（只有封装），其他规格的修改配合（例如 COM 的 *IDispatch* 接口），使得 VB 已成为微软揠苗助长下的牺牲品。即使有微软如此强而有力的极力推荐，先天缺陷仍旧无法根除，除了易学外，实在找不出其他应该使用 VB 的理由，因此我会建议所有的初学者，若能够接受其他的语言，转移阵地为上策。

Talk

VB 支持 COM，而 COM 主张的是 interface inheritance，不是 implementation inheritance，所以也算有点继承及多型的观念。

不过也由于 COM 的特性，VB 不能够直接在程序中编写类、建立对象，必须转个弯，额外建立一个 ActiveX Control 来使用，是十分麻烦、特别的作法。

严谨明确的 Pascal

在 Win32 环境下，Borland Delphi 是使用 Pascal 语言最著名的开发工具。

Niklaus Wirth（1984, ACM Turing Award 得主）于公元 1968 年开发出 Pascal 时，他主要的目的是想发展出一套能兼顾发展及执行效率，而且具有高度结构性及组织的程序语言。另外，他也希望 Pascal 能够适合教学目的，让学生易于理解计算机程序设计的概念。

Pascal 之名是由伟大的数学家 Blaise Pascal 而来。Niklaus Wirth 曾参与 ALGOL 60 语言的发展计划，而 Pascal 就是直接从 ALGOL 60 改良衍生的程序设计语言，后来它又陆续从 ALGOL 68 及 ALGOL-W 吸收许多语言上的优点。

Talk

在 Pascal 之后，Niklaus Wirth 还发明了 Modula-2 语言，另外还有 Oberon 语言，从 Modula-2 语言改良而来。

有趣的是，Niklaus Wirth 一直想设计飞机，只是他发现他需要更好的工具，于是他才设计了一个个的程序语言，并造了自己的计算机 Lilith（Module-2 语言就是为 Lilith 而设计的）；这与 Donland Knuth 为了他的 ***The Art of Computer Programming*** 惊世巨作排版不好看，特别花 10 年的工夫发展 TEX（极著名的排版语言，尤其适合数学及科学方面的文件排版）有异曲同工之妙，两位大宗师真不愧是大宗师。

在今天看来，Pascal 仍然保留着当年 Niklaus Wirth 所规划的发展原则——严谨明确的特性。由于 Borland 对 Pascal 语言的全盘掌握，使 Pascal 语言能够顺利演化为真正的对象导向程序语言——Object Pascal。就像 FreeBSD 的 coreteam 全盘控制所有 FreeBSD 套件的更新编写一样，Pascal 控制权控制在 Borland 一小撮人手中，虽失去开放性，但保持着本身的特性。

我认为它的面向对象支持恰如其分，该支持的全都支持了，不必要的也不贪多，适当地加入。它与 C++的优劣是没有答案、见人见智的，正如同大礼服及小洋装，好不好看，适不适合，因人而异。

强大复杂的 C++

在 Win32 环境下，使用 C++ 语言的开发工具为数不少，最有名的就属 Microsoft Visual C++、Borland C++Builder 及 Symantec C++。

Talk

公元 1980 年，Bjarne Stroustrup 发明了“C with Classes”语言，因为他想为 C 语言加入如同 Simula67 语言般的事件驱动机制。直到 1983 年夏天，才由 Rick Mascitti 创造出 C++这个语言名称。

如果你学过 C 语言，那么可以这样看待 C++ 语言：

- 它是比较好的 C 语言；
- 它提供数据抽象化机制；

- 它支持面向对象程序设计，所提供的语法及字眼让你可以很轻易地将面向对象落实到C++程序中。

C++语言的功能强大，勿庸置疑，template、exception handling、RTTI、Standard Library等等功能不断地加强翻新。由于广大的用户群体期望值高，再加上C++本身定位在功能强大范围广泛的通用性语言，C++自然日益复杂。著名的杂志C++ Journal上曾有段话让我印象颇深：

> 如果你认为C++还不算太复杂，那么请你解释什么叫protected abstract virtual base pure virtual private destructor，而你又会在什么时候需要它呢？
>
> Tom Cargill C++ Journal Fall 1990

虽然是最流行的面向对象程序语言，但除非你有足够的耐心及精神来全盘掌握它，否则轻易尝试的后果可能只会得到一脸的挫折。当然了，十分的复杂也带来十分的便利及高度的成就感及乐趣，我有一位朋友，工作上使用其他语言，但将C++当作兴趣来研究。

1.2 RAD的弊端

每过一段时间，TANet网络论坛上就会兴起一阵Visual C++与C++Builder孰优孰劣的讨论。前段时间，讨论周期又到了，果然又是口诛笔伐的一场论战。乐于欣赏文章的我，捕捉到一些颇为中肯的言论：

> 发信人: Meou@m2.dj.net.tw （Dadai）
> 这两个东西都蛮好用的。但是现在我摸了几个月之后，我反而比较喜欢VC++。VC++的用户界面看来繁琐，但是真的用心花个几天把VC++的功能摸熟，用起来还蛮顺手的；再加上把VC++的Application Wizard的来龙去脉搞清楚，把Class Wizard的用法弄懂，MFC一点一点慢慢弄熟之后，它的功能还蛮强大的。再加上VC++的HTML Help比BCB好上百倍，我现在觉得VC++比较好用。
> **其实这两个工具，一个是倚天剑，一个是屠龙刀，放在不会用的人手上，那一把都不顺手。**这两个工具都需要相当好的C++基础。好的C++基础对MFC、OWL、VCL来说都是基本功夫。程序学了一阵子，觉得MFC摸熟之后，设计界面不比C++Builder来得慢，**重点在于界面之下你到底要如何解决问题。**这个问题远比Application Framework及哪一套开发工具比较好要重要多了。
>
> 发信人: dyliu@ms1.hinet.net （四眼的王虫）
> VC++的在线帮助系统的确比Borland的Delphi、BCB等好多了，Borland在这一方面一直没有什么进步。有时候我用Delphi或BCB时，还会执行VC++，目的就是要用VC++的在线帮助系统，Borland应该在这一方面大力改进才行。
> **界面方面MFC比不上VCL，简单的界面还好，复杂一点的MFC就得搞上老半天。**

发信人: oesd@email.gcn.net.tw （Dadai）
VC++／MFC 的学习曲线看起来比较长，但是值得。因为你如果搞懂的话，再配合上一些 SDK 的知识及合适的开发工具，你几乎可以在 Windows 下做到任何你想做的事（剩下的只是你怎样解决你要解决的问题）。我自己觉得 MFC 应有点相关基础，最起码你要能把 VC++ Wizard 能做的东西知道如何全部用手工打造出来，不然用 Application Wizard 建出来的东西，你一样看不懂，更不要讲如何去改它了。
其实像 BCB 这种 RAD 开发工具要学得好，我觉得比 MFC 还难，因为在那漂亮界面下的底层机制往往比 MFC 复杂得多。真的大声喊 BCB 好简单的，我只看到两种人。一种是在 Win32 SDK 里面打滚多年，几千条 Win32 API 就算没用过也都大概摸过，没摸过也知道大概会叫什么名字，该往那边找。问他一个问题，脑袋里面会自动列出一堆 Win32 API，一条条过滤该如何解决。写 Windows 程序可能打字到手指头都长肌腱炎了。BCB 对他们是种解脱，VCL 更是不成问题。
而另一种则是完完全全的初学者，拿 BCB 来学 Windows 程序设计。最多只能学到组件有提供的功能都会用，组件不会的他也不会，组件不行的他也不行，组件的限制就是他的限制。应用程序写到一半，里头要调用 Win32 API 函数，他大概就挂了，要做到现有组件做不到的功能？那你要不要花钱买 VCL 链接库？

另外，再听听 Visual C++ 知名作家侯捷在他的“怀璧其罪 RAD”一文中怎么说：

RAD 并非罪恶，而是优点。要怎么用它则是 developer 自己的问题。
侯捷对于 RAD 工具如 Visual Basic 或 Delphi 或 C++Builder 并不擅长，但我知道 Visual Basic 可以调用 Windows API，做相对低阶的动作；当然，从软件工程角度来看，VB 是比较弱，因为它不具 OO 特性。至于 Delphi，我有两位这方面的专家朋友（Wolfgang 和 Xshadow），他们可以使用 Delphi 做任何事情，没有任何你想象中 RAD“该有”的限制；C++Builder 和 Delphi 系出同门，一样没有什么限制。
以下引一段我在【汗如雨下・杂感・1998 / 11】的文章片段：
● 关于 RAD（Rapid Application Development）
《Delphi 学习笔记》作者钱达智先生，是我的好友。我们之间对于 RAD 有段讨论，或许你想听。
■ 侯：BBS/News 上经常有关于 RAD 的工具见解。我认为你很够资格说些话。不少人对 RAD 有误解。如果你能提醒大家：
1. RAD 是很好的开发工具
2. 使用 RAD 并不代表不需要底层扎实的基础，那么误解的人就会比较少一点，知道该怎么做的人就会比较多一点。
■ 钱：过去在 DelphiChat、NEWS Group 以及我的书中，这样的想法都不只一次宣扬过。
其实，就我接触过的人，不论是网络或者是学生，RAD 的使用者的确是比较急功近利，也难怪会有这样的刻板印象。

虽然说过，但还是要持续宣扬这种理念，就如大哥说的，误解的人会少一点，知道该怎么做的人就会比较多一点。
RAD真是“匹夫无罪，怀璧其罪”呀。

Visual C++及C++Builder，一个用MFC一个用VCL，一个必须配合Wizards编写大量程序代码，一个虽然可用鼠标完成大部分的接口设计，不过程序逻辑及核心还是得编写程序代码，此时RAD派不上用场。不论如何，他们提到了几个重点：

1. 一个是倚天剑，一个是屠龙刀，放在不会用的人手上，那一把都不顺手。
2. 重点在于界面之下你到底要如何解决问题。
3. RAD开发工具要学得好，不比non-RAD开发工具简单，在漂亮界面下的底层机制往往出人意料地复杂。
4. RAD及non-RAD开发工具，拥有相同的“实现愿望的能力”，功夫底子好的人要起来，样样都能实现。

所以，重点在人身上。不管是RAD或non-RAD开发工具，用得熟透了的那些人永远可以做出他们想要的东西；不管是RAD或non-RAD开发工具，学得不够透彻的人们永远也只能抱怨程序语言太难、开发工具太差，无法享受程序设计的乐趣。

换个角度来看，同样是RAD开发工具的用户，有的是全然解脱，轻巧驾御开发工具，善于适用RAD的特性来加快程序开发速度及质量；有些却只能选用一下组件，能力局限于别人制作的组件功能，因为跨不出开发工具的格局，完全被RAD的服务范围限制绑死。

Talk

也许有人持有相同的怀疑态度：Delphi到底能做些什么呢？

我必须再一次大声呼喊：“在Win32下，除了驱动程序编写[1]之外，只要是其他开发工具办得到的，Delphi就办得到！”

就我所见到的，埋怨开发工具能力不足的人，通常同时在揭露自身能力的不足。而真正见到开发工具能力不足或设计不良的那些人，不会大落落地成天埋怨责怪，他们总有办法另找出路来解决问题。

1.2.1 开发工具的差异

比较Visual C++、C++Builder及Delphi这3套开发工具，我将它们之间重要的差异列在表1-1中。

表1-1 Visual C++、C++Builder及Delphi 3套工具的比较

比较项目	Visual C++	C++Builder	Delphi
设计公司	Microsoft	Borland	
前端语言	C++	C++	Object Pascal

1 其实，如果配合WinPriver的发展套件，Delphi也可以拿来编写驱动程序。

（续表）

比较项目	Visual C++	C++Builder	Delphi
Application Framework	MFC	VCL	
界面设计方式	传统（Class Wizard 及手工设计）[2]	RAD（拖拉点按）	
程序核心	手工设计（有许多链接库及类可以调用）	手工打造（有许多组件、链接库及类可以调用）	
操作原理	调用 Windows API	调用 Windows API	

其实，不论什么程序语言，不论什么开发工具，只要在同一个操作系统内，它们的操作原理都是一样的：**调用操作系统提供的服务（通常以函数调用的方式）**。在 Win32 环境下，我们称这些系统服务为 Windows API。

1.2.2　Win32开发工具的演变

原本，Win32 程序设计师编写程序的惟一方式就是调用 Win32 API 函数，这些 Win32 API 函数由 KERNEL32.DLL、USER32.DLL 和 GDI32.DLL 三大模块及其他大大小小 DLL 所提供。这些 API 函数分为许多种类，各擅其职，只要善于使用它们，在操作系统提供的能力范围内，没有做不到的事。不过，API 函数既多且繁杂，而且如同 RISC CPU 提供的指令集，每一道函数所做的事情并不多，连一些频繁使用的例行公事，例如建立新窗口、注册窗口类、更改按钮颜色等等动作，还得花上十几行程序代码来做，麻烦透了。

需求乃创造之本，于是链接库出现了；随着面向对象的浪潮，紧接着类链接库也出现了。类链接库慢慢地发展，功能不断加强，规模越来越庞大，负责的范围也逐渐涉及应用程序的生灭及操作核心，最后终于蜕变为更高层次的 application framework，最负盛名的两套 application framework 就是 Microsoft 的 MFC 及 Borland 的 OWL。虽然有类链接库及向导、专家等工具的辅助，仍有人不知足地想要发展能够更快地开发应用程序的方法，于是就有了 Visual Basic 这类可靠鼠标完成大部分界面设计工作的 RAD 开发工具，最后才是 Delphi 及 C++Builder。

1.2.3　RAD的作用

随着时间的脚步，人们总要适应大环境的变迁及进化，RAD 的确为程序开发人员省下了不少界面开发的时间。但相对地来说，因为它大为降低程序设计的门槛，使得太多的初学者沉溺于 RAD 组件的强大及使用，不知道 application framework 及 Win32 API 的地位，无论真正解决问题的数据结构及算法，甚至连程序语法都不太熟悉的状况下就可硬凑出亮丽端庄的程序外观！我想这就是 RAD 开发工具开始受到部分人们的质疑原因。

看透传统开发工具及 RAD 开发工具，摸透 MFC 及 VCL 这两套 application framework，它们只是包装一薄一厚，用法各异罢了。

MFC 薄薄的一片，让你能够全盘掌握，相对地，学习曲线既陡峭又高峻，需有足够的

2 有些厂商开发出极像 VB 设计界面的 VC++ Add-On，虽然用的是自己的 class library，不过至少也让 Visual C++朝 RAD 工具迈进了一步。

背景知识才能充分融入 MFC，享受它的好处。VCL 的包装并不彻底，但厚厚地这一层，让人完全看不到骨子里的究竟。就界面设计来说，VCL 包装即美观又好用，不过 VCL 也有不足的地方，此刻，是 RAD 也好，不是 RAD 也好，任何工具都帮不上忙，只有看自己琢磨 Win32 API 的功夫。若没有三两下子，马脚随着 framework 的不足就立即露出来了。

让我想到几年前曾发生的“命令行优劣之争”。有些高手们及 UNIX fans 喜爱命令行，一来速度快，二来有全盘掌握的感觉；而另一派当然是倾向图形使用界面了。传统确实有传统独到之处，否则为什么在电子音响大行其道的今日，仍有玩家愿意花几十倍的价格，把玩真空管音响呢？命令行用得熟，操作速度比图形使用界面还来得快上几倍倒是真的，我觉得这是命令行需要存在的最大理由。也有人对我说，“命令行比较让人懂得计算机真正的操作原理”，我告诉他，“为什么图形使用界面就会妨碍我们去了解计算机的操作原理呢？”守旧也得要有合理的理由，否则只是恋旧，潜意识的感觉其实是怕跨入新的领域而失去一向保持的优势。

所以呢，手工设计得好，还是为了鼠标操作的方便？手工设计得快，还是为了鼠标操作的省事？我想答案是很明显的，宁可在例行公事上能省时间就多省点时间，我们还有未来等着去创造呢，怎能整日沉迷在手工设计全盘掌握的感觉中呢？容易使初学者陷入迷途是 RAD 的弊端，但 RAD 所带来的好处是不容轻易抵消的。我想，RAD 无罪，它只是使门槛降低点，让程序设计更为简单轻松。

1.3 实际操作与理论

实际操作、理论往往很难分辨，让人搞不清楚究竟什么是“浅尝即可的实际操作”、什么是“实际操作架构”，什么是“实际操作理论”，以及什么是“真正的理论基础”。糟糕的是，对于没有搀杂任何程序代码的大量文字，许多人往往以为这就是所谓的“理论”。

1.3.1 参与者的类型

常跟朋友聊起，面对开发工具及程序语言的选择，大致可将所有的参与者分为 3 大类：

新手型

对所有的开发工具程序语言甚至开发平台全然陌生，稍微听过一些开发工具的名称，经常弄错也是常有的事，例如“Virtual C++”、“Virtual Basic”、“Dephli”、“Borland C Builder”等等。这个族群所占的比例最高，往往在网络论坛上询问“该学什么程序语言？”、“我想写游戏，该用什么语言？”、“XXX 及 YYY 究竟什么比较好？”这类问题，常是引发程序语言及开发工具优劣辩论大战的导火线，虽然是懵懂无辜的。

程序设计对他们而言是未知的领域，充满好奇、期待但找不着进入的门口。

专家型

所谓专家，即是训练有素的，技术实力高人一等的专才。他们通常精通某一厂商的开发工具，独钟一派程序语言，擅于编写特定领域的程序，对于相关的函数库、application framework 及实际操作细节捉摸得十分清楚，熟悉得不得了。

但是，只喜欢使用特定工具、语言甚至实际操作细节的结果，并不代表拥有深厚的信

息科学基础及宽广的信息视野。长期紧追产业技术或局限某开发工具的结果，容易导致“**技术即是知识，实际操作即是能力，实际操作架构即是理论**”的错误观点。

缺乏足够的背景知识，故步于狭窄的领域，此类型的玩家常是网络论坛上专属某某特定语言或开发工具的超级打手。

黑客型

理论上，数据结构、算法、编译器、操作系统、计算理论等等，是必备基础；实际操作方面，他们往往至少熟悉两三种以上的开发工具及程序语言，并将火力集中在与语言无关的方法论。

对他们而言，若要开发主从式数据库项目，使用 Delphi 的数据库组件；若要编写 Web 服务器端程序，用 C 语言来编写 apache module；有跨平台的需求时，使用 JDK 或 Symantec Café、Borland JBuilder 来编写 Java 程序；项目用到 VxD、WDM 或 kernel mode driver 时，可以使用 SDK、DDK 和 Visual C++，再买套 VToolsD、Driver::Work 或 WinDriver 立即动手。不执着于任何开发工具及语言，自然不会被任何公司的规划（如 Microsoft 的 VBA 吃遍天下）或美好远景（如 Microsoft 的 DNA 架构、Inprise 的 Information Network）等解决方案所羁绊，而随波逐流了。

拥有足够的背景知识，所有的语言及软件终究只是工具，能够以超然的态度来面对、比较，此类型的玩家通常把工具当玩具，把写程序当成散文写作看待。

1.3.2　参与者的入手点

见过许多程序设计的初学者，老爱劈头就问：“是否有快速的入门方法？”，我总不觉莞尔。为什么小时候学日文，长大学物理、化学、数学、经济学时，从来不会奢求所谓的“快速入门法”，但是一旦面对程序设计的学习关卡时，就想走快捷方式了呢？这是十分有趣的现象。

我想，**态度与角度**是决定各个参与者的入手点。**态度指学习这些知识技术的态度，角度指看待程序语言及开发工具的角度。**

不论你身为哪种类型的程序设计领域参与者，请试着站在黑客型玩家的角度来观察（事实上，只要努力的方向正确，你迟早也属于他们）。对无入而不自得的黑客们而言，基础知识及能力既足，各种程序语言只是与计算机对话的各种方言，发音编写各有不同而已；而各个开发工具只是各家厂商提供的不同翻译小姐。以这样的角度来看待程序语言及开发工具，你觉得还必须拥 XXX 反 YYY，或者执着各项实际操作细节吗？

面对任何一个程序语言及开发工具时，应该明了它们只是帮助你编写程序，用来控制计算机达成目的的帮手，绝非学习的目的。很多人误会了这点，将开发工具视为学习目的，往往花了数月辛勤不倦地熟悉开发工具的每一细节以及程序语言的所有语法后，才发现程序设计原来不是将一堆符合程序语法的语言元素堆砌起来就成。别因为程序语言及开发工具的复杂或新颖就慌了手脚：**程序语言是工具，程序的架构、逻辑及设计是手段，而程序执行的结果才是我们的目的。**切莫将程序语言或开发工具学过了头，忘了程序设计真正的目的。

1.3.3 这些技术是什么

举个例子来说，RFC 是说明 Internet 上众多协议、规格及结构的文件。虽然完全都是文字，只有极少数包含数据结构的定义，看起来像是很值得仔细研究阅读的教学文件。但其实它是实际操作规格，只有实际操作者才需要严谨地阅读它，搞清楚所有的细节。虽然其中的规范及协议煞有其事，但这仍是实际操作的一部分，如同公司进行项目前所编写的程序策划。不同的是，RFC 是国际性的、免费的、标准的规格书。

往下一层来看，支持整个互联网络的 TCP/IP 协议群，如 ARP、IP、ICMP、TCP、UDP 等等，有很多介绍它们的专著，说明这些通信协议的细节及通信流程，它们是什么？实际操作规格。

编写主从式或多层式架构数据库程序时，涉及的主题可能包括 VCL、BDE、SQL、ODBC、ADO、COM、DCOM、CORBA、MTS、MIDAS 等等；编写互联网络应用程序时，可能涉及 TCP/UDP、POP3、SMTP、FTP、TELNET、BSD socket、WinSock、WOSA 等等；编写 Web 应用程序时，可能涉及 HTTP、SSL、SET、HTML、DTD、XML、CGI、NSAPI、ISAPI、ASP、Java Applet、Java Script、Java Servlet、VB Script、ActiveX、Netscape Plug-in、PHP、Perl、mSQL 等等。

以上列出的一堆技术名词，有些是程序语言、有些是程序语言的应用、有些是文件语言、有些是文件语言的语法规则、有些是实际操作架构、有些是操作系统提供的机制或服务、有些是软件、有些是软件提供的机制及服务、有些是通信协议、有些是分布式对象模型等等。不论它们是什么，请记住，**它们都是实际操作，或是因实际操作而来的规划或架构，或是由实际操作衍生的副产品。**

1.3.4 全部理论都在里面

难道我就一直贬抑这些许多人抱在怀里的各项技术及知识吗？当然不。

它们都是实际操作，可是理论都在里面。

研究 TCP 通信协议，非常好。可是你研究的是 TCP/IP 各协议的封包格式、协议规格、传递流程呢？还是试图去了解背后支持它，维护效率的 Nagle Algorithm、Slow Start Mechanism、Congestion Avoidance Algorithm、Fast Retransmit and Fast Recovery Algorithm 呢？认识这些家伙，才能理解所谓“可信赖的传输层协议”，才能将对 TCP 的了解推广到其他通信技术上。

深入操作系统的内部架构及底层操作，非常好。这可使你彻底了解操作系统的操作方式，暂时抛开纸上谈兵的操作系统教科书，详细看看在现实世界中，架构在现代硬件上的现代操作系统，究竟如何妥善地担任计算机硬件与应用软件之间的桥梁。以 Windows 95 为例，你研究的是它与 DOS 的亲密关系、INT 21h 的使用时机和 thunking 机制呢？还是研究它如何提供 LPC、IPC[3]，thread scheduling / page replacment 机制，如何处理 memory thrashing / page thrashing 现象？

---

3 LPC 为 Local Procedure Call 的缩写；IPC 为 Inter-Process Communication 的缩写。

1.3.5　你看到了哪些

列出以上这些“实际操作”、“理论”模棱两可的说明，我想表达的是“淬取”、“蒸馏”、“抽象化”这些字眼。

事实就是这样，面对相同的技术、规格或架构，有人能够很快看穿外头的包装，拆开外面的包装看本质，了解其理论基础，挟着新的收获朝更广阔的领域迈进。

另外的一些人，为其能力或效果所感动迷惑，再三留恋，将实际操作细节摸得一清二楚，却往往将焦点置于使用方式或实际操作机制，虽然下方的基础及学问才距离几公分，总是无缘相见。虽然得到了高超的技术能力，却更加狭隘了视角，短浅了目光。

面对新的技术、语言及工具，以定义上看来，它们通通都是实际操作相关的产物。但是你看到了哪些？是实际操作细节？还是理论基础？你才是决定答案的人。

第 2 章　VCL 基本概念

学 Delphi 首先就要学好 VCL，
没有不精通 VCL 的 Delphi 高手。

VCL 全名叫做 Visual Component Library，VCL 是 Delphi 及 C++Builder 所用的 application framework，VCL 提供了许多组件，可供程序设计师在开发环境中操作、使用，除了这些具体的事实外，VCL 的“实质”究竟是什么？从文件的角度来看，它究竟包含了哪些文件？而我们又如何才能见到它？

线索并不多，不过我想，既然 VCL 是 Delphi 的 application framework。那么，在 Delphi 产生的可执行文件中，一定可以找到些蛛丝马迹。

2.1　Delphi程序的组成

在 Delphi 环境中，选取菜单的【File / New Application】选项，建立一个新的项目。项目建立后，立即将项目保存起来，项目命名为 Project1，单元命名为 Unit1，惟一的 form 则直接使用默认名称 *Form1*。现在我手上就有了一个最正常的 Delphi 应用程序，它是我们接下来的观察对象。

2.1.1　可执行文件成分解析

从菜单选取【Project / Options】选项，在“Project Options”对话框中把“Linker”页的 Map file 选项调整至“Publics”。然后选择【Project / Build】选项，编译、链接完成后，可以在项目目录下找到 Map file 文件 Project1.map。Project1.map 竟然包含将近 6000 行文字！我选取其中一部分列在下面，仔细分析 Map file 所包含的信息。

区段

```
#0001   Start      Length    Name              Class
#0002   0001:00000000 0003D3F4H               .text    CODE
#0003   0002:00000000 00000CF8H               .data    DATA
#0004   0002:00000CF8 00000821H               .bss     BSS
```

这些列出的可执行文件拥有的区段（section），区段可包含程序代码、数据、资源、出错信息或其他任何信息。可执行文件加载时，整个可执行文件，包括这些区段都会直接映像到该进程的地址空间，让操作系统可在内存中直接取用区段数据及直接执行区段内程序代码，使得加载、执行 PE 格式可执行文件的操作远比以前的 NE 格式可执行文件要轻松多了。