万水计算机编程技术与应用系列

Borland 高级编程指南 C++ Builder

[美] Matt Telles 著
查理 李伟 郑晖 译
康博创作室 审校

中国水利水电出版社
www.waterpub.com.cn

内 容 提 要

本书深入剖析了 C++ Builder 系统的内部组成与工作机理，探讨了怎样优化程序编制工作，以提高程序开发效率的各种途径。全书共分三个主要部分，分别讲述了 C++ Builder 环境及其系统组件，完整的应用程序开发过程和示例，以及经常遇到的问题及其解决方法。

本书适用于广大程序开发人员，是程序员们提高程序开发效率，解决一些关键难点和问题的必备参考手册。

图书在版编目(CIP)数据

Borland C++ Builder 高级编程指南/（美）特列斯（Telle, M.）著；查理等译. -北京:中国水利水电出版社, 1998.8
(万水计算机编程技术与应用系列)
书名原文：High Performance Borland C++ Builder
ISBN 7-80124-758-2

I. B… II. ①特… ②查… III. C 语言-程序设计-指南 IV.TP312-62

中国版本图书馆 CIP 数据核字(98)第 15390 号

书　　名	Borland C++ Builder 高级编程指南
作　　者	（美）Telle, M. 著
译　　者	查理 李伟 郑晖
审　　校	康博创作室
出版、发行	中国水利水电出版社（北京市三里河路 6 号　100044） 网址：www. waterpub. com. cn E-mail：sale@waterpub. com. cn 电话：(010)63202266(总机)、68331835(发行部)
经　　售	全国各地新华书店
排　　版	北京万水电子信息有限公司
印　　刷	水利电力出版社印刷厂印刷
规　　格	787×1092 毫米　16 开本 25.75 印张　585 千字
版　　次	1998 年 6 月北京第一版　1998 年 6 月北京第一次印刷
印　　数	0001—5000 册
定　　价	65.00 元（含光盘）

凡购买我社图书，如有缺页、倒页、脱页者，本社发行部负责调换

前　言

程序员(特别是正在进阶的程序员)经常以这样的提问相互挑战,“你现在使用的工具顺手吗?”这个问题已经延续很久。编译器、调试器、服务器、数据库引擎以及其他所有的东西仿佛刚从石器时代向我们走来。从某种工具转而使用另外一种,不论质量还是功能,两者之间存在着惊人的差异。如果选择了错误的工具集,你的工作可能会比实际需要的更艰苦,甚至更糟。

今天,更贴切的提问可能会是“你能用好这些工具吗?”无情的竞争给予今天的开发者工具以极端深入和令人惊愕的质量,以至于熟练程序员都不可能将这样的工具推向其开发极限。程序员只有尽其所能学习有关使用工具的每个方面,在使用中锤炼技术,才能免于失败之苦。

Coriolis Group 的 High Performance 系列的设计目标是帮助你深入工具内部。这些书解释入门书籍不能涵盖的高级工具特征,并提供高任务项目,从而强制读者以专家层次考虑整个开发过程——使用隐藏于称之为技术后面的思想。

通过思考这些技术发现这一思想,并且随机地尝试直到程序正常工作为止。或者,能够获益于我们作者的经验,我们已经逾越了学习的曲线,并能在学习的道路上指点迷津。我们仔细地选择了主题、作者以及方法,确保你不会陷入泥潭,避免不需要的初级素材和不能使用的无关技术。

本书的目的是带着你和你选择的工具集去实现所追求的目标。既然已经得到了高性能的工具,能否成为高水平开发者,并保持这一荣誉就在于你了。

目　录

第一章 综 述

- 什么是 C++ Builder
- 为什么使用 C++ Builder
- 为什么编写本书
- 本书的内容是什么
- 本书适用于哪些人

1.1 什么是 C++ Builder

C++ Builder 是一种针对于 C++ Windows 程序编制的真正的快速应用程序开发工具(RAD),它正在得到整个工业界的青睐。毫无疑问,你对此也会有所耳闻,而事实上你阅读本书就足以证明你有足够的兴趣对其研究。考虑到本书的题目,你至少应该在使用 C++ Builder 工具方面有一定的经验并且希望进一步学习它。在我们涉及到 C++ Builder 系统的细节之前,让我们来从大的方面对其加以讨论:是什么使 C++ Builder 成为一个如此强大的工具?为什么 C++ Builder 对程序设计业如此重要?首先,我们将在本书的其他章节里不再使用所谓 C++ Builder 这个绕嘴的名词来称呼这一集成开发环境,我们将其简称为 CBuilder。当然,我们甚至于可以简称其为 Builder,然而,这样做将使我们陷入与 Borland 公司即将推出的 JBuilder Java 开发工具相混淆的麻烦。

CBuilder 是最早可以获得的面向 C++的真正 RAD 工具之一,而且它也是唯一提供了真正基于组件的拖放式编程的 RAD 工具。在过去几年中,这种编程风格对 Windows 程序设计所产生的影响是非常巨大的。起初,基于 MS-DOS 编辑器、C 编译器与连接器和软件开发程序包(SDK)的 Windows 编程是极易产生错误的,这使 Windows 程序编制就像噩梦一般。因此,毫不奇怪,早期的 Windows 程序充满了错误而且需要数年的时间进行开发。然而,今天编写 Windows 程序只需要几周时间而不是几年(尽管程序中仍然充满了错误,但这是另外一回事)。就像将在本书所看到的,CBuilder 不仅将帮助用户更快地开发应用程序,而且将使应用程序中的错误更少。

Windows 程序编制模型的第一步发展是 C++编程语言和 C++类库的出现,它们将成百上千行进行简单窗口显示与处理所需的代码封装在几行 C++代码中。这些 C++代码只是将以前书写的数百行 C 代码封装在 C++类中。这样做,我们不但再也不需要重写它们,而且被 C++类封装的代码中的错误也将急剧减少。毕竟,一旦代码书写完成并且经过调试,就没有必要再次重写它们。没有进行重写的代码也就不会产生新的错误。请读者记住这一点,这将在以后变得更加重要。

第二代 Windows 的发展也随之带来了第一代集成开发环境(简称 IDEs)。这些工具允

许程序员在一个单独的应用程序中完成全部编辑、编译和链接工作。在此之后不久，又出现了集成调试环境并很快被 Windows 程序设计界所接受。如果读者从来没有使用独立的开发工具进行过早期的 Windows 编程，就很难向你描述这些新工具是多么奇妙。如果你有机会接触 90 年代前的那些程序员，并且问他们关于早期 Windows 程序编制的问题，我敢肯定他们会很高兴告诉你那是多么艰难。

Windows 程序编制接下来的演化是程序框架(frameworks)发展的结果。一个程序框架就是一个应用程序骨架，它包含了应用程序的各部分代码。代码程序框架事实上与建筑框架中的大梁、铅坠和电路非常相像。回顾起来，代码程序框架实际上也许是一种倒退或者至少说是原地踏步，尽管许多人不同意这一点而且仍然作为程序框架的坚定支持者。话又说回来，还有一些人仍然热衷于争辩说 MS-DOS 仍然是一个有效的操作系统。说程序框架是一种倒退的原因是很简单的。程序框架强迫我们在一系列严格的规则下编写程序，而不是使程序的编写更加容易与灵活。

程序框架技术存在的真正的问题是其所具有的限制性。尽管程序框架通过提供普通 Windows 应用程序所具有的许多基本函数，从而加速了应用程序的开发，但它们也会给开发那些特殊的应用程序带来阻碍。如果读者曾经尝试着做某些程序框架不支持的事情(或更糟，程序框架所做与读者所想的不同)，那么就会明白笔者到底是什么意思了。一旦读者试图完成某些复杂而特殊的工作时，程序框架很快就变成了束缚双脚的小鞋了。

注意：读者如果不相信基于程序框架的应用程序开发所具有的问题，请考虑如下问题：Microsoft 事实上已经放弃更新它的主要程序框架产品 MFC 来对新的 ActiveX 和 COM(两者都是基于组件的)技术做出反应。笔者相信大多数程序框架产品(MFC、OWL 和其他产品)将会在不久的将来消失。

那么我们是不是应该放弃程序框架，回到使用那种成百上千行 C 代码的和充满无数错误的老式线性程序编制方法呢？当然不是。我们有一个比程序设计框架更好的答案：基于组件的程序设计。组件就像搭建应用程序的砖一样。用户开始于能够完成专门任务的特别组件对象，然后以适合自己需要的方式来组装它们。由于组件没有给自己强加任何结构，所以无论开发什么类型的应用程序都没有关系。事实上，组件是 ActiveX 技术的基础，而 ActiveX 又是 Internet 编程的基石。

CBuilder 的组件实现得非常好，既简单又直观。就像大多数优秀的应用程序一样，CBuilder 工作时既简洁又有很好的一致性，这大部分要归功于这一产品基于组件的特性。CBuilder 系统每一部分都有特定的职能，并将其尽可能简单而又容易地实现。组件非常接近于纯粹的 C++ 程序设计，就像读者可能见过的一样，它使整个开发的努力变得更加简单。而这一点正是 CBuilder 的风格：它是第一个基于组件的真正的 RAD 工具。

1.1.1 关于 CBuilder 进一步的内容

许多程序员热衷于接受 CBuilder 窗体的拖放技术而忽略了 CBuilder 的另一个方面。在 CBuilder 系统中并不很深的地方隐藏着一个令人难以置信地强大，并具有极度灵活性的数

据库编程环境。事实上,CBuilder 是针对 C++程序员的第一个真正的数据库编程系统。

在许多 C++开发系统中,数据库接口经常是事后添加上去的,添加的目的是想使程序员能够通过一系列有严格限制的对象来使用数据库的某些小片段。大多数 C++系统的数据库接口是由一系列紧紧环绕于底层数据库功能的包装对象所组成。对于数据库对象来说,包含需要几十个参数的方法打开并初始化一个与 ODBC 数据库的连接是很平常的。与此相比,CBuilder 将一整套数据库拖放组件内置到其系统中。CBuilder 附带一套非常完整的根本不需要任何编程的数据敏感性控件。读者完全有可能编写一个完整的数据库编辑系统来添加新记录、编辑已有的记录和删除不需要的记录,而不用编写一行 C++代码。请读者在 Visual Basic 或 Visual C++中试一试。

长远看来,CBuider 也许会更多地因为它能够实现数据库、表格和 SQL 查询而为人所知,而并非因为它能处理窗体和系统中其他可视组件。

那么,什么是 CBuilder 呢?CBuilder 就是一个提供了出众数据库功能、基于组件技术的、易于使用的并具有强大开发能力的完整灵活的 RAD 工具。当然,它还包含一个用途相当普遍,完全支持诸如模板、名字空间、异常处理和标准模板库(STL)等新技术的 ANSI 标准 C++编译器。

1.1.2 为什么编写本书

也许本书不是读者所拥有的第一本 CBuilder 书籍。当笔者编写本书时,当时的 CBuilder 书籍基本上属于两类之一。一类主要针对有经验的正在转向 CBuilder 环境的 Delphi 程序员,对 CBuilder 的一种很自然的看法是,CBuilder 事实上是使用 C++语言而不是 Object Pascal 的 Delphi(尽管笔者怀疑大多数最终使用 CBuilder 的人根本没有使用过 Delphi)。就像笔者刚才提到的,CBuilder 只是一种使用不同语言的 Delphi。为什么要改变呢?这是因为有一部分程序员使用 Delphi 是出于其具有超强组件的原因,尽管它们痛恨使用 Object Pascal。然而,我猜测读者并不是这些程序员中的一个。如果读者是他们中的一个,也用不着懊恼,本书中同样有大量适合的内容。

第二类书籍是针对那些新程序员的,他们不仅要学习一个新的开发环境,而且也要学习一种新的语言。这些人在阅读本书时不会那么得心应手,直到他们掌握了那些引导性的书籍。本书是针对富有经验(或者至少有一定经验)并需要完成一定工作的程序员的。专业程序员将会找到需出现在他们工作中的,完成所有细节问题必需的信息。我们也将考虑如何实现那些使程序取得最终成功的诀窍。通过短小但详细的代码示例,读者将学会在做需要做的事情时需要学习的知识。作为一个专业程序员,笔者了解,当看到别人程序中的优点而自己没有时间或精力来在自己的应用程序中加以实现,这是很令人沮丧的。

出于以上原因,本书实际上是以组件形式编写的。如果读者喜欢,可以通读本书,或者翻到需要的章节选择完成工作所需代码。

1.1.3 本书将讲述的内容

这里给出读者将在本书找到的主题的缩略列表:

- 使用 VCL
- 使用 ActiveX 控件
- 扩展窗体
- 编制组件
- 开发数据库应用程序
- 编制复杂的 C++ 代码

除了本书所有代码示例以外，我们还将开发一个成熟的 CBuilder wizard，通过它可以快速而简易地开发复杂的组件。最后，在本书的末尾，读者会发现一个完整的 C++ Builder FAQ 拷贝，这是由成百上千的人通过 Internet 发送电子邮件，然后搜集其中的信息整理而成。这一简单问题与回答的列表，在读者开始开发自己的复杂应用程序时应该了如指掌。

1.1.4 在阅读本书之前应该具备哪些知识

首先，读者应该了解 C++ 语言。本书不是一本关于 C++ 语言的入门书籍；有很多书籍在这方面要比笔者所能做到的更好。本书将会解释标准 ANSI C++ 与 CBuilder 中版本之间的差别。就像将要看到的，这些差别主要存在于 CBuilder 所使用的扩展中。那么读者需要对 C++ 有怎样的了解呢？其实并不需要太深。如果下面的 C++ 类定义没有吓倒你，那么在阅读本书其他章节时会很顺利。任何比这更复杂的代码将会被解释，就像本书后面将介绍的一样。

```
class TMyObject : public TObject
{
private:
    int FnDigit;
    char * FpString;
    Foo * FpFoo;
public:
    TMyObject(void);
    TMyObject( const TMyObject & aObject);
    virtual ~TMyObject(void);
    void DoSomething(void);
};
```

除了 C++ 编程技巧之外，如果读者在阅读本书之前曾经使用 C++ Builder 或 Delphi 开发过哪怕是非常简单的应用程序，也会对阅读本书有所帮助。这将使读者习惯于系统环境的行为方式和编译器与链接器工作方式。只要阅读本书和相应的代码，初学者也可以学会关于 CBuilder 编程模型、VCL 和系统内部工作方式的知识。

最后，如果读者有在 Windows 编程方面的背景，那么也没有坏处。虽然像 CBuilder 和 Delphi 这样的产品使得甚至于经理都能很方便地开发复杂的应用程序（好的，也许还没有简单到经理就能进行开发的地步，但已经很接近了），为了完成某些任务还是有必要对底层

Windows 程序设计模型有所了解。我们将用完整的一章讨论使用 Windows API 来扩展 CBuilder 程序设计模型。这样，即使那些不屑于向可视化开发环境看上一眼的擅长底层系统编程的人也会高兴。

1.1.5 本书适用于哪些人

总体上来说，本书针对那些希望学习更多的关于 CBuilder 系统内部机理与如何使系统充分发挥其作用的资深或专业程序员。初级程序员将会通过操作书中代码进行学习，但笔者希望他们能多读几遍以充分获益于每个示例。

本书由一系列独立的章节组成，每一章节与其之前的内容联系不大。这样做就使程序员能够浏览各个章节直到他们找到想要解决的问题（或者找到什么别的吸引他们注意力的东西）并且从那里开始阅读和消化本书中的代码。

尽管本书带领读者进行了几个成熟组件的创建，笔者并不希望本书成为针对 CBuilder 组件开发的手册。如果想很认真地学习组件编程，除了本书，请另外再选择一本专门针对此任务的书。

1.2 本书概述

本书有三个主要部分。第一部分，我们将通过短小示例来探究 CBuilder 环境和程序编制系统的各个组成部分。多数情形下，那些示例只说明 CBuilder 编程一或两个要点。在其他示例中，我们将探究系统更复杂的方面。在这期间，我们还将讨论有关 CBuilder 如何和为什么这样做的问题。在第一部分中我们还将讨论包括窗体绘制、图像处理、拖放式组件交互、所有者自绘控件、标准模板库、数据库和 ActiveX 控件。这样的一个部分还不坏，是吗？就像能猜出来那样，第一部分是本书中最大的一个部分。

第二部分包括一个完整的应用程序开发，从初始设计一直到使用 VCL 和许多它的组件来完全实现。这一部分将会实现一个完整的 Component Wizard，它不仅允许定义组件的基类（就像 CBuilder 附带的那个一样）而且还可以定义新的方法和属性。完成第二部分的学习后，读者应该理解关于 CBuilder 系统的组成部分。读者还应该学习到如何创建对于许多 CBuilder IDE 细节具有完全控制的扩展，包括增加新文件到一个项目、产生代码和加载文件。

第三部分包含 CBuilder FAQ（经常问及的问题）列表，其中是关于 CBuilder 的数十个问题与解答。FAQ 是按主题进行组织的（常规编程、组件、数据库、VCL）并被分解成简单的单一主题问题。在第三部分还可以找到一个从 Borland 知识库到 Internet 上专门的 C++ Builder 站点信息的列表。

总之，这是一本由程序员写给程序员的书。笔者从痛苦和可怕的经历中知道，本意是买一本旨在帮助解决问题的书，而结果却是一个改头换面的联机帮助，这种事时有发生。本书处理的问题是那些笔者在为真正的客户编写真正 C++ Builder 应用程序时曾经遇到的。笔者希望程序员在本书中能够找到那些使人发狂的问题的答案。现在还等什么，打开书开始吧。

第二章　窗体和事件处理

- 创建无标题窗体
- 在窗体上绘制
- 窗体定义内幕
- 事件处理和窗体
- 动态事件处理
- 从窗体到 MDI

窗体是在 CBuilder 系统中最基本的组件。窗体是系统中最容易看见的部分，也是用户最可能与其进行交互的部分。在大多数的 CBuilder 示例中，窗体用作包容其他组件的容器。在本章中，我们将把窗体作为一个组件来讨论。

什么是窗体呢？当然，它就是一个窗口。就像所有的窗口一样，它能够具有子控件，例如工具条、菜单和状态条。窗体，就像窗口一样，具有一些自己独特的属性，例如标题、系统菜单、最小化与最大化按钮、关闭按钮、可变大小边界等等。让我们花些时间看一下窗体以及 CBuilder 允许我们对它进行修改的部分。

图 2.1 显示了一个标准的 CBuilder 空白窗体，当开始启动 CBuilder 应用程序时用户将会看到该窗体。让我们详细地讨论这个窗体吧。

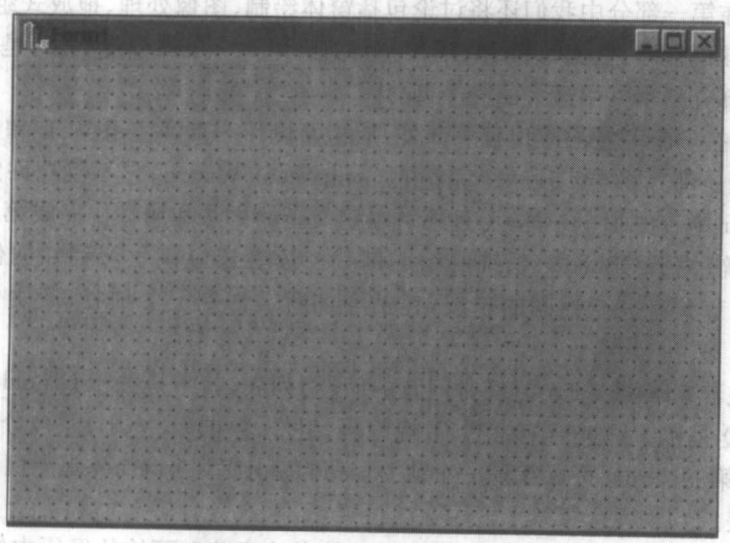

图 2.1　标准 CBuilder 空白窗体

第一个跃入读者眼帘的窗体的属性是窗体标题(Form1)，它在标题条中被明显地显示出来。这个属性被称做窗体的 Caption 属性。Caption 属性能够于设计时刻在 CBuilder 的 Ob-

ject Inspector 中被直接修改或在运行时通过代码修改。无论是在设计时，还是在运行时，改变 Caption 属性将会立即更新窗体的标题。只有一个例外，不能将 Caption 属性简单地设置成空字符串(" ")而使标题消失。

这是在 CBuilder 中的第一个专家技巧：如何来创建一个没有标题和标题条的窗体。如何能够将标题和标题条从一个 CBuilder 窗体中删除，这一点并不很明显，但是使用一点 Windows 的魔力和一些 Widows API 的知识就能够实现它。因为一个窗体事实上就是一个窗口，读者就可以在该窗体真正被创建前修改它的初始属性。如果改变窗口风格位，使其包含 WS _ POPUP 标志并删除 WS _ CAPTION 标志，就可以创建一个没有标题条的窗口。也可以创建一个窗口，它没有系统菜单、最小化或最大化按钮或者关闭按钮。出于这一原因，必须提供一种方式来关闭这样的一个窗体。

让我们来实验一下，创建一个这样的窗体。一旦完成了对 Caption 属性的学习，下一个要讨论的属性就是 Border 属性，它允许用户来控制是否窗体具有一个可变大小的边界。对于 Border 属性有多个选项，但是有两个最可能使用到的是可变边界(bsSizable)和固定的对话框风格边界(bsDialog)。用几分钟的时间来在两种边界风格间进行变换并编译结果应用程序。值得注意的是这两种边界类型在屏幕上显示不同。图 2.2 显示了一个具有可变边界的标准窗体，而图 2.3 显示了一个具有对话框风格边界的窗体。读者将会看到边界被称为“框架”。这个名称来源于窗口边界与图画框架的相似性。

第三个属性实际上是一个属性集。BorderIcon 属性是由一系列能在窗体的标题条中出现的图标或按钮组成。这个属性实际上是一个真 C++集，它包含了对于最小化按钮、最大化按钮、系统菜单和关闭按钮的选择。通过在 Object Inspector 中或使用代码把属性设置为 false，读者就可以打开或者关闭所有上面这些按钮。

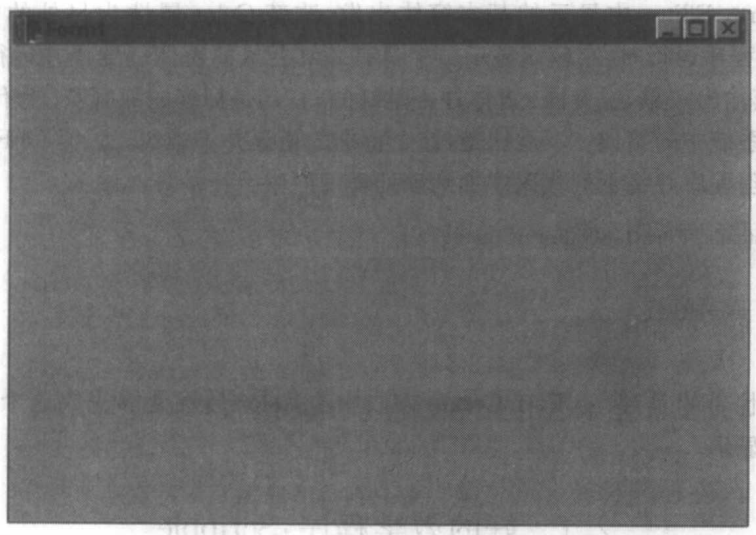

图 2.2　一个显示可变边界的窗体

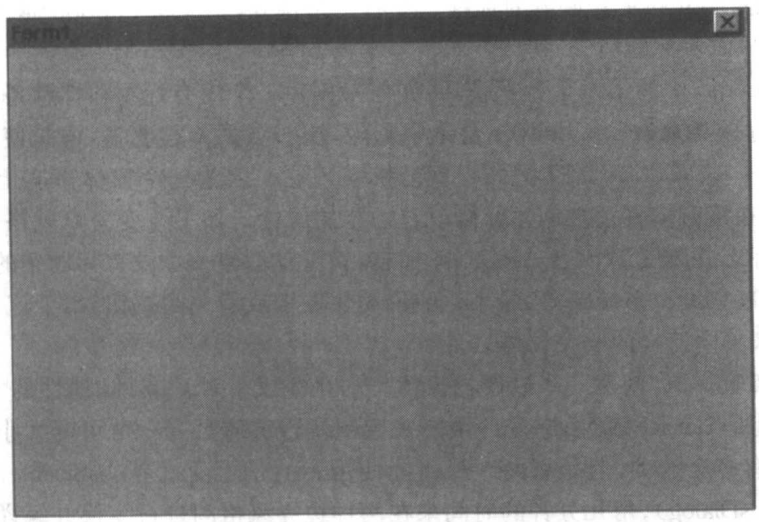

图 2.3 一个显示对话框风格或固定边界的窗体

注意：对于这些属性的更改将不会在设计时反映出来。被窗体编辑器显示出来的窗体在创建时是固定不变的。所做的改变将会在窗体第一次在应用程序中出现时得以实现。为了证明这一点，改变窗体以省略最小化和最大化按钮，请注意编辑器中显示的窗体没有变化。编译并运行该应用程序，读者将会看到事实上结果窗体并没有最小化和最大化按钮。

Color 是另外一个窗体不太明确的属性，在下面的实验中读者可以看到这一点。从在 CBuilder 的 Form Editor 中显示的基本窗体出发，改变 Color 属性为另外的什么值，比如 clRed。当应用程序运行时，它应该显示一个具有亮红背景的窗体。编译并运行新的应用程序，然后观察其输出效果，很奇怪！窗体并不是红的；它却是标准的战舰灰。为什么呢？好的，这似乎是一个系统中的错误。幸运的是，这个特别的错误并不难纠正。为 OnFormCreate 事件添加一个处理程序并将下列代码添加到该处理程序中：

```
void __fastcall TForm1::OnFormCreate()
{
    Color = clRed;
}
```

足以令人惊奇的是，改变 FormCreate 方法中的 Color 属性能够完成这个工作，然后窗体将会以红色显示。

2.1 好的教学程序：Scribble

我们已经讨论了许多设计时刻与运行时刻窗体的属性，但我们明显地在避免谈论窗体

中间的宽阔区域。窗体的整个客户区是一个独立属性，称为窗体的 Canvas 属性。Canvas 属性负责属于窗体本身的任何文本与图形的显示。它也是响应窗体 Color 属性的部分，但这是由窗体对象本身间接完成的，而不用人为干预。

本书第一个真正的例子是关于 Canvas 属性的。由于 Windows 一般要考虑其图形能力，那么第一个例子是图形的就很合适。在程序设计过去的美好日子中，当 Visual C_{++} 首次出现时，Microsoft 选择了一个叫 Scribble 的程序作为 Visual C_{++} 编程系统的教程。这个程序主要是一个使用鼠标在窗口表面自由绘图的程序，它能将鼠标在窗体表面穿过的所有点用线连接起来。

因为这个例子比较简单，Scribble 成了程序设计界内部的笑柄。许多程序员声称曾经使用过 Visual C_{++} 和 MFC(Microsoft 基本类库)而事实上他们所做的一切就是通过了 Scribble 教程。Scribble 需要几页程序来实现，而且要求程序员能够阅读书籍并输入书中的内容。这个示例没有表现出一点与 MFC 的关系，而且甚至对那些希望得到 MFC 知识去编写有用程序的程序员来说几乎没有任何用处。作为 MFC 程序设计界文雅的反叛，我们提供了 Scribble 的 CBuilder 版本。严肃的说，Scribble 对于简单的示例是很好的，因为它提供了与窗体所有组成元素之间的一些交互：属性、事件和方法。让我们在 CBuilder 中实现一个简单的 Scribble 版本。

2.1.1　Scribble：设计

Scribble 的目的非常简单。当用户按下鼠标时，程序开始绘图。当鼠标在屏幕上移动时，程序应该连接鼠标穿过的所有点来形成线段。这一程序允许用户在屏幕上“绘制”简单的(也许并不是那么简单的)图形 。图 2.4 显示一个简单的 Scribble 运行结果：一张笑脸。

在本书的介绍中，笔者曾经说过具有在 Windows 中的编程背景对于学习这里给出的程序是有用的，这正是能够帮助了解 Windows 消息系统的机会。此处的程序已经相当简单。当用户按下鼠标键并移动鼠标时，就可以连接绘制的点。对于初学者，不够清楚的是如何才能实现这一点。如果是有经验的 Windows 程序员，将会知道当用户按下鼠标左键时，Windows 会产生 WM _LBUTTONDOWN 消息给窗口。当用户移动鼠标时，Windows 将会产生 WM _MOUSEMOVE 消息。最后，当用户释放鼠标左键时，Windows 产生 WM _BUTTONUP 消息。问题是这样的：这一信息如何将自身翻译到 CBuilder 环境中？

CBuilder 通过一个称为事件处理的系统来工作。每一个在 CBuilder 环境中发送给对象的 Windows 消息将会被翻译成一个能被该对象处理的事件。如果是上面列出的消息，CBuilder 窗体对象调用三个事件处理程序：OnMouseDown、OnMouseMove 和 OnMouseUp。

注意：对于 Visual C_{++} 或 Borland C_{++} 程序员，事件处理程序的概念似乎显得有点奇怪。这两个环境都使用同一概念，但以不同的方法实现。在 Visual C_{++} 和 Borland C_{++} 中，消息映射条目定义一个消息以及用来处理这一消息的类方法。在 CBuilder(当然，还有 Delphi)中，对象的内部定义了那些被处理的消息，以及调用用户定义的处理程序。不同之处在于 CBuilder 事件处理程序，就像稍后将看到的，是动态的，

而 Visual C++和 Borland C++消息处理程序是在编译时刻定义的。两者都使用指向成员函数的指针来实现这一工作，但 CBuilder 的方法更清晰且更强大。

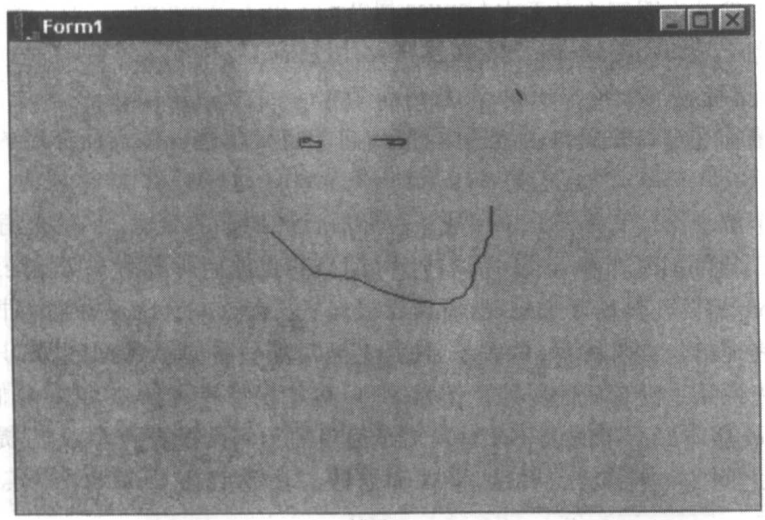

图 2.4 一个具有笑脸的 Scribble 会话

不管这些，让我们给窗体加一些处理程序来使 Scribble 应用程序完成它自身的功能。首先，给 OnMouseDown 事件增加一个处理程序。因为这是第一个例子，我们将浏览整个程序以防读者忘记。在这之后，我们将假设读者知道自己在做什么并只告知应加入哪个处理程序。

通过按 F11 键显示出 Object Inspector，如果它在窗口中不可见，或者选择 View 菜单中的 Object Inspector。在窗体上单击鼠标选择该窗体，或从 Object Inspector 顶部的下拉组合框选择它。移动到事件窗体，在 Inspector 窗体的左侧找到 OnMouseDown 事件。所有事件都以字母排序列在其中，所以应该不会有什么问题就能找到需要的事件。单击在 OnMouseDown 条目右侧的 Object Inspector 网格并输入 OnMouseDown，作为新的事件处理程序的名称。尽管让事件处理程序与其所处理的事件有相同的名字会造成混乱，但我们在本章的晚些时候会介绍如何改变它。在该窗格单元中按回车键确认输入的条目。

一旦已经输入了方法名并确认了选择，CBuilder 将在编辑窗口中创建新的处理程序。请在编辑窗口中将下列代码输入到 OnMouseDown 方法中：

```
void __fastcall TForm1::OnMouseDown(TObject *Sender, TMouseButton Button,
                                    TShiftState Shift, int X, int Y)
{
    FbMouseDown = TRUE;
    Canvas->MoveTo(X,Y);
}
```

在上述代码中显示的 FbMouseDown 标志需要加入到包含文件(Unit1.h)中并在构造函数中初始化。这个标志是用来表示鼠标是(TRUE)否(FALSE)在窗口中正处于按下状态。以下是包含文件添加的内容(加亮打印)：

```
//---------------------------------------------------------------------
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------
class TForm1: public TForm
{
__published: //IDE-managed components
             void __fastcall OnMouseDown(TObject *Sender,TMouseButton
             Button,TshiftState Shift, int x, int Y);
private: //User declarations
  Bool FbMouseDown;
public: //User declarations
        __fastcall TForm1(Tcomponent *Owner);
  };
//---------------------------------------------------------------------
extern TForm1 * Form1;
//---------------------------------------------------------------------
#endif
```

最后，以下是对类构造函数的修改(位于 Unit1.cpp 源文件中)：

```
__fastcall TForm1::TForm1(TComponext * Owner)
                  :TForm(Owner)
{
FbMouseDown=FALSE; //Initialize mouse down flag to be NOT down
}
```

在继续进行下去之前，我们来看看已经做了些什么。现在 TForm1 类有了一个用户定义的私有变量名为 FbMouseDown，它表示鼠标是否正处于按下状态。如果用户在窗口用户区范围内按下鼠标左键，OnMouseDown 方法将会被事件处理系统调用。这个方法将标志置为 TRUE，并把当前 Canvas 的绘图位置移动到鼠标所处的位置。当用户在事件处理程序中按下鼠标时，CBuilder 相当好地给出了关于鼠标处于什么位置的信息。

也许读者想了解 CBuilder 是如何知道将 OnMouseDown 事件与所编写的 OnMouse-