电脑编程技巧与维护月刊

2001 年第 3 期　　总第 81 期　1994 年 7 月创刊）　　每月 3 日出版

名 誉 社 长　张　琪
社　　长　孙茹萍
副 社 长　毕研元
总　　编　王路敬
执 行 主 编　杨林涛
副 主 编　张　涛
编　　辑　程　芳　管逸群
　　叶　永
公关部主任　黄德琏
副 主 任　苏加友
出版发行部　毕　波
编 辑 出 版　《电脑编程技巧与维护》杂志社
法 律 顾 问　佟秋平　商安律师事务所
主 办 单 位　中国信息产业商会
社　　址　北京海淀区学院南路 68 号吉安大厦 4017 室
投 稿 信 箱　paper@publica. bj. cninfo. net
编辑部信箱　editor@publica. bj. cninfo. net
网　　址　http / / www. comprg. com. cn
邮　　编　100081
电　　话　010 62178300 62176445
传　　真　010 62178300
照　　排　《电脑编程技巧与维护》杂志社电脑排版部
印　　刷　北京巨龙印刷厂
订 阅 处　全国各地邮电局
国内总发行　北京报刊发行局
邮 发 代 号　82—715
刊　　号　ISSN 1006 - 4052 / CN11 - 3411 / TP
广告许可证　京海工商广字 0257
全 年 定 价　93. 60 元
每 期 定 价　7. 80 元

本刊 2000 年优秀作者

冉林仓　郑州铁路分局机电设备厂
江天送　桂林电子工业学院 990121 班
江　华　吉林市师范学校
江　龙　湖北浠水师范学校微机室
邓双成　北京石油化工学院机电教研室
元凯宁　天津市河东区程林庄路 63 号 182 信箱
李震宇　合肥市经济信息中心计算机通讯部
余海燕　北方交通大学自动化所

本刊 2000 年热心读者

刘佩军　湖南省常德华南光电仪器厂退休办
何春华　浙江省海宁市人民路 169 号内海宁佳和电子工程有限责任公司
张　钟　重庆谢家湾建设工业　集团）公司摩研所
王　涛　黄金堂路 22 号湖北莉州市职业中专
江　帆　福建省光泽华桥国有林场
吴树鹏　天津市河东区万新村七区防暴支队
孙　凯　湖北工学院 73#
田伟蒙　陕西西安市金花北路 4#西安工业学院测量与控制研究所
华　磊　安徽合肥市中国科学技术大学东区 222 - 115
肖忠良　长沙电力学院化学系
唐翊国　山东省东营石油大学炼制系炼油教研室
张鸿斌　北京市 5102 信箱 81 号
蒋东焱　北京市 9200 信箱 74 分箱 19#
宋曜利　西北工业大学 754 信箱
姜忠奎　辽宁东港市前阳镇建筑安装总公司
陈　强　青海省格尔木市水文分站
徐东文　山西省万家寨引黄工程电信站
陈　峰　山大新校 64#
陆　涛　宁夏青铜峡市汉坝小学四楼计算机室
邹增理　浙江大学华家池校区环建 961 班
陈红根　河南省职业技术学院计算机系
李淑琼　深圳市华城暨南大学中旅学派 2#
郑　宏　福建省平和一中高三　仁）班
由世洲　黑龙江水运规划设计院
周京生　北京 5131 信箱 12 号北京跟踪与测量总体技术研究所八室
熊　伟　江苏南京市 83582 部队服务中心
李　均　江西省景德镇供电局
刘红强　西北工业大学 112 信箱

网上征订：
http / /www. 8848. net
http / /www. gotoread. com
http / /www. dangdang. com

2001 年《电脑编程技巧与维护》杂志订单

订购单位				收件人		经办人	
通讯地址						邮　编	
订户银行及账号				收款单位	盖章 年　月　日		
单价	7.80 元/期	份数		合计			
大写金额							

合 订 本　1994 年：20 元；1995 年：35 元；1996、1997、1999 年合订本各 80 元。2000 年合订本 98 元（包括挂号邮寄费）

（本刊在今年年底推出 2000 年全年 12 期源代码光盘，凡购买 2000 年合订本的用户，随刊赠送全年源代码光盘 1 张）

以上合订本均挂号邮寄，不另加邮费。

单　　本　2000 年单本：7.80 元/期

2001 年单本：7.80 元/期

（凡订阅 2001 年全年的用户，随 2001 年第 12 期刊物赠送 2001 年全年源代码光盘。）

同期软盘　1998 年：20 元/季；1999 - 2001 年：10 元/期。

如需以上软盘请汇款至杂志社可随时购买。

订购须知

1. 2001 年本刊每月 3 号出版，全年共十二期，每期 100 页，定价每期 7.80 元，全年定价 93.60 元。
2. 请到当地邮电局订阅，邮发代号 82 - 715；也可以通过邮局汇款方式直接在杂志社订阅。收到来款后即按月寄出。订阅时请务必将收件人姓名、单位名称、通讯地址、邮编、金额等填写清楚，并在汇款附言栏中注明订阅的期刊期数和份数。
3. 本刊光盘不单独出售。
4. 需要开发票者，请在汇款附言栏中注明。
5. 凡在杂志社订阅者可享受 10%的优惠。
6. 网上征订请查询：http://www.8848.net
   http://www.gotoread.com
   http://www.dangdang.com

通讯地址：北京海淀区学院南路 68 号吉安大厦 4017 室
《电脑编程技巧与维护》杂志社发行部

邮政编码：100081　　电话：010 - 62178300

读者意见征询卡

尊敬的读者,请您抽空填写此表,并邮寄或传真至我社。

地址:北京市海淀区学院南路 68 **号吉安大厦** 4017 **室　邮编:**100081　**传真号码:**010 - 62178300

个人资料:

1. **姓名:**_____　2. **性别:** □**男**　□**女**　3. **职称(职位):**_____　4. **年龄:**_____

5. **电话:**_____　6. **传真:**_____　7. **单位:**_____

8. **地址:**_____　9. **邮政编码:**_____　10. ICQ _____

11. E - mail**:**_____　12. **个人主页:**_____

13. **文化程度:**□**中专以下**　□**大专**　□**本科**　□**研究生以上**

14. **您的工作性质是:**□**硬件维护**　□**系统级开发员**　□**编程人员**　□**其他**____

15. **您主要买哪些电脑类报刊(请按喜欢顺序填写)**

报纸:1 _____　2 _____　3 _____

杂志:1 _____　2 _____　3 _____

16. **您购买或订阅本刊的时间:**

□**两年以上**　□**一年以上**　□**半年以上**　□**三期**　□**两期**　□**第一次**

17. **您第一次购买本刊的原因是:**

□**朋友推荐**　□**刊名吸引**　□**内容吸引**　□**价格吸引**

18. **您获得本刊的渠道:**□**邮购**　□**订阅**　□**购买**

19. **除了您阅读本刊外是否还传阅他人?**　□**是**　□**否**

20. **您认为本刊跟其他刊物相比的特点是**_____

21. **您认为本期最好的文章依次为**　1. _____　2. _____　3. _____

22. **您认为本期最差的文章依次为**　1. _____　2. _____　3. _____

23. **您希望本刊增加的内容是:** _____

_____

24. **您对本刊的内容和发行您还有什么建议?** _____

_____

AMD 芯片软件开发工具 Virtuhammer 亮相

日前，芯片制造商 AMD 分别在纽约的 LinuxWorld 大会和巴黎的 Linux 博览会上演示了为其下一代芯片——Hammer 系列芯片研制的软件开发工具。在一月中旬，AMD 和瑞典的一家计算机系统模拟开发商 Virtutech AB 签署了协议，开发名为“VirtuHammer”的工具软件，以帮助软件开发者为 Hammer 系列芯片编写和测试 64 位程序。这款软件把安装了 AMD 公司 32 位的速龙芯片的计算机模拟成使用了 64 位 Hammer 系列芯片的计算机，允许开发者利用现有的可行技术去测试和调试 64 位软件。

有消息称微软要开放 Windows 源代码

据报道，微软准备扩大从前受严格限制的 Windows 源代码共享项目，允许更多的大客户查看其保护异常严密的 Windows 源代码。微软的一位产品经理在 Linux World 大会上证实，很快将会有更多客户可以查看 Windows 源代码。和其他许多操作系统开发商一样，微软多年来一直在向一些关键的客户提供 Windows 源代码。但到目前为止，该公司一直在幕后悄悄这么做，并且还要求这些客户签署严格的保密协议。这位产品经理表示，允许客户查看源代码是微软支持服务的延伸，体现了一些大客户的需求。通过查看源代码，企业内部负责解决 IT 问题的技术人员，可以向后追查，解决一些可能由软件本身引起的问题。不过，在任何情况下，客户都不能为满足各自业务上的需要，对 Windows 源代码进行修改。

全美达计划年内推出 1GHz 克鲁索处理器

日前，全美达 Transmeta 公司的首席执行官 Dave Ditzel 在 LinuxWorld 大会上表示，这家创始企业将在今年晚些时候推出型号为 TM 5800 的克鲁索 Crusoe 微处理器。TM 5800 的运行速度可达 1GHz，而能耗则和现有的克鲁索微处理器不相上下。在正式开业的第一年，全美达已推出三种克鲁索微处理器，运行速度在 333 - 700MHz 之间。Ditzel 表示，全美达正和一些企业合作开发采用克鲁索微处理器的服务器，这些企业中包括由一些原康柏电脑公司服务器工程师们创办的 Rocket Logic 科技公司。

美国专家称因特网域名软件 BIND 有缺陷

美国计算机安全专家 2 月 1 日宣称，他们发现因特网运行中最重要的一套软件存在某些缺陷，从而给了计算机网络黑客可乘之机。这是到目前为止发现的最严重的因特网安全漏洞。这套有缺陷的软件是名为 伯克利因特网域名 英文简称 Bind 的软件，它在因特网上的作用类似于日常的电话号码查询系统。在组成因特网的计算机网络中，估计约有 90% 使用 Bind 软件。在网络中，通常会有数百上千台计算机向一、两个运行该软件的服务器查询网址。该软件的第四与第八版本存在缺陷，一旦黑客掌握这些软件的缺陷，就可以操纵网络服务器、偷取电子邮件甚至改变网页访客的寻访途径。计算机安全专家已敦促使用第四与第八版本的网络服务器管理员，应尽快换用新版本软件。

Cisco 推出 10Gbit / s 接口高速路由器

全球最大的路由器制造商美国 Cisco Sytems 推出了配备 10Gbit / s 光接口高速路由器新产品“Cisco12410 / 12416”。该产品配备了 1 个端口的“OC - 192c / STM - 64c Packetover SONET / SDH POS”线卡 Line Card。产品内部还配备了 25M 数据包 / s 的转发 Forwarding 引擎，从而使该产品与其他竞争公司的产品相比性能提高了 25%。另外，该产品还遵循在最近刚刚确定的互相接驳接口标准规格“VSR”。需要顺便说明一下，目前美国 Juniper Networks 和美国 Avici Systems 已经推出了配备 10Gbit / s 光接口高速路由器。

低于 1500 美元 17 英寸液晶显示器上市

美国康柏电脑公司宣布，将以 1499 美元的低价格在美国上市 17 英寸 TFT 液晶显示器“TFT7010”。这个价格在面向商业用途的产品中无疑具有挑战性 aggressive price。该产品的显示精度为 SXGA，亮度为 200cd / m2。视角方面，上下左右皆为 160 度宽视角。

NTT 开发出非接触式 IC 卡加密技术

NTT 和 NTT COM 于 2001 年 2 月 2 日宣布开发出一种可在使用非接触式 IC 卡过程中对结算处理进行加密的技术。该技术与 RSA Receive Shamir Adelman 方式下 1024bit 密钥的加密效果相当。两公司表示这是世界上首次在非接触式 IC 卡上实现该水平的加密处理。虽然使用非接触式 IC 卡时只需将卡在读取终端上晃一下就可以轻松完成有关处理，但这也给加密带来了困难。因此长期以来金融机构和政府部门大多以接触式 IC 卡为中心讨论 IC 卡的导入问题。两公司决定于 2001 年将该技术推向实用化，并以金融、交通、流通、信息服务等各行业的企业为对象。

松下开发出扭扣式大电流锂离子电池

松下电池工业公司开发出可以进行大电流放电的可充电扭扣式锂离子电池“CGL3032”，该产品将在 2001 年 10 月份开始以月产量 50 万个的规模进行批量生产。该产品适用于 GPS 终端、便携式终端以及卡式信息设备的电源部分。松下电池此次开发的可充电扭扣式锂离子电池的直径为 30mm，厚度为 3. 2mm。最大连续放电电流为 1CmA。该公司以往同类产品 V - Li 可充电电池“VL3032”的最大连续放电电流为约 0. 1CmA。当以 0. 2CmA 放电时其容量为 140mAh，与以往同

类产品相比提高了 40%。重复使用寿命为约 500 次以上。

IBM 开发预防服务器 “当机”的软件

全球最大的电脑制造商 IBM 表示，已开发出一套软件，可以预测服务器因系统冲突而出现所谓的 “当机”，并会安全关闭服务器。由于服务器是企业存取资料的心脏，“当机”所造成的资料遗失及系统停止运作等问题，对企业造成的损失很大，GigaInformation Group 的分析员认为，这种可以预防的软件相当重要。新软件可说是电脑 “自我诊治”的一大进展，一般来说，IBM 及其竞争对手如 Sun 等，出售电脑时都会附送治疗软件，升级版也会放在网上供用户使用。IBM 相信，上述新软件是市场上首个可以预防服务器 “当机”，令企业可以保持运作的。

我国研制成功 FED 场发射平板显示器

郑州大学材料物理重点实验室 “金刚石——碳膜冷阴极场发射平板显示器”项目，在张兵临教授的带领下，经 4 年多努力，终于研制成功我国平面型 FED 场发射平板显示器，赶上世界上目前正处于边研制边开发阶段的先进水平，为进一步开发数码管、复合字符屏、大屏幕电视像素管以及其它显像方面的应用奠定了基础。据张兵临介绍，平面型 FED 摈弃了微锥型 FED 因采用精细加工技术而工艺复杂不易制造大屏幕的弊端，采用金刚石、碳膜作为发射体，工艺简单、成本降低、便于产业化推广，与传统的显像相比，具备其亮度高、色彩饱和度好、响应快、视角宽等优点，还将能量转换效率从不到 1% 提高到 20% 以上，大大节省了能源。

微软提出 . NET 构想 WindowsXP 年内上市

微软公司最著名的品牌 Windows 要改名字了。微软公司昨天宣布，将上市的 Microsoft Windows 桌面操作系统和 Microsoft Office 桌面应用软件系列新产品将分别改名为 WindowsXP 和 OfficeXP。XP 是英文 “体验” experience 的缩写，代表着 Windows 和 Office 将给用户带来丰富的、充分扩展的全新计算体验。微软提出的 . NET 构想，旨在帮助用户超越互不相联的应用软件、服务和设备，而去感受重新定义了的人、软件和因特网的相互关系，并获得完整的、互联的全新计算体验。WindowsXP 和 OfficeXP 正是迈向这一构想的重要步骤。据了解，WindowsXP 简体中文版计划 2001 年下半年基本投入市场，而 OfficeXP 简体中文版的发布计划在下半年。

Winzip 将有新版本 Winzip8. 0J

P & A 公司的拳头产品 Winzip 近日又出新版本 Winzip8. 0J。新版本增加了许多功能，譬如新增 Zip and E - Mail 快捷按钮，可将文件压缩后附在电子邮件中；另对 WinZip Wizard 的功能进行改进，可对文件进行压缩及更新，对 MIME 等编码后的文件可进行解压和安装。目前销售形式为网上销售，Panta 公司的主页及 Vector、Biglobe 的网站上均有销售。价格为 5000 日元左右。

RedHat 推出采用 2. 4 内核的 Linuxβ 版

RedHat 于 2001 年 2 月 5 日推出了基于最新内核版本 2. 4 的 Linux 发行套件 Distribution β 版 “Fisher” 开发代号，用户可在该公司的站点上免费下载。在 2001 年 4 月中旬面市的正式版中将对 Fisher 存在的问题进行修正。Fisher 由于采用了 Linux2. 4 内核，加强了对设备接口的支持，具体包括：将原来只支持键盘和鼠标的 USB 接口技术扩展到支持存储设备。该版本还支持 IEEE1394、IDE 的 Ultra DMA66 / 100 等接口以及 Intel 的 64bit 处理器 Itanium。

Sun 推出 “Sun ONE”抗衡微软 . NET 方案

美国 Sun Microsystems 于美国时间 2 月 6 日面向开发人员发布了通过网络服务提供办公组件的 “Sun Open Net Environment ONE Webtop”。这是该公司综合性网络服务战略 “Sun ONE”的一部分。Sun ONE 是专门为支持网络服务的开发、构筑和导入而推出的。 “Sun ONE 提供智慧型的网络服务，它可以使企业提高生产效率、缩短产品开发时间从而赢得商业机会，并能给消费者带来丰富的运用手法” Sun 公司 。另外，Sun 公司将与隶属于 Sun - Netscape Alliance 的 iPlanet E - Commerce Solutions 设立合作机构，后者将为 Sun 公司提供面向 Sun ONE 的解决方案。这是 iPlanet 于当天发布的。

VC + +系列讲座（七）

张博强　徐　亮

第三讲　设计多文档界面应用

在本讲中我们将首先使用 AppWizard 创建一个多文档框架，然后为其增加绘图及图形选择功能，实现对一文档状态的多窗口显示和文档的存储与读取。

一、 创建 MDI_Draw. exe 工程

单文档界面（SDI）与多文档界面（MDI）是 Windows 的重要组成部分，如我们经常使用的“记事本”和“Word”，就是这样的应用。在这里我们将建立一个名为 MDI_Draw 的 MDI 程序，它将实现如下的功能：

1. 程序启动时，主窗口如下图：

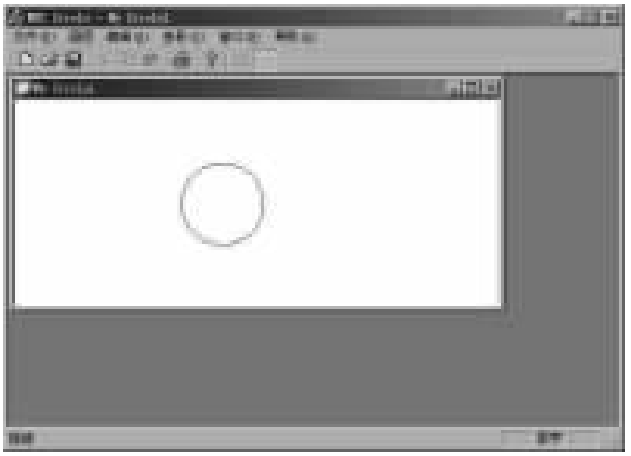


图 1　程序启动时的窗口

2. 同时程序建立了一个新文档，并显示相应的文档窗口，名为“My Circle1”。

3. 在文档窗口中间显示一个圆，并且，它可以根据鼠标点击的位置发生改变。

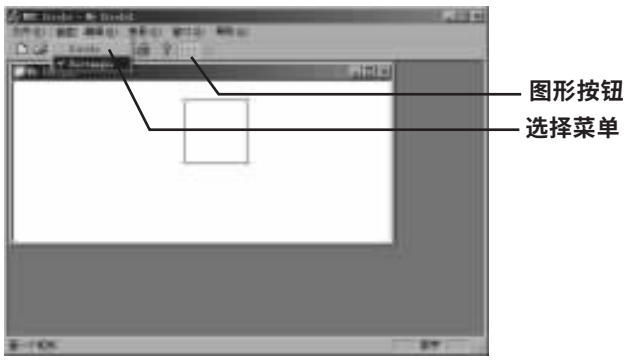


图 2　改变图形

4. 通过点击菜单“画图”—“Circle / Rectangle”或工具条上的图形按钮，可以改变显示的图形。

5. 通过“文件”—“保存”/“另存为”把刚才的文档存为“*. drw”文档。

6. 可从“文件”—“打开”读取刚才存储的内容。

图 3　保存对话框

了解了它的功能之后，我们就一起来创建这个工程。

1. 首先启动 VC + +，从“File”—“New”，打开 New 对话框。

2. 选择“Projects”标签。

3. 从列表中选择 MFC AppWizard（exe），在 Project Name 中填入“MDI_Draw 其它保留默认状态，点击 OK 按钮。

4. AppWizard 共有 6 步，这里我们将保持它的默认选项不变，所以可直接点击 Finish 按钮，也可以点击 Next 按钮，看一下这些默认选项的内容，第一步是询问你创建应用的类型，默认为“Multiple documents”，第二步是选择对数据库的支持，默认为“None”，第三步选择想包含的复合文档，默认为“None”，第四步是选择应用程序界面、信息等一些选项，在第五步对话框选择工程风格、注释及 MFC 类库的用法，在第六步中用户可以修改默认类名、基类名、头文件和实现文件的文件名。完成上述步骤后，点击 Finish 按钮，出现新文件信息对话框，对你的选择做一个总结。

5. 点击 OK 按钮后，VC + +将会创建与 MDI_Draw 工程相关的所有文件。

6. 进入窗口界面，从 Build 选择 Set Active Configuration。

7. 在打开的 Set Active Project Configuration 对话框中选择 MDI_Draw—Win32 Release，点击 OK 按钮。

至此，创建工作完成。这时 MDI_Draw 工程已经是一个可以运行的完整的程序了，并且具备了一些基本功能。从 Build 选择 Rebuild All 后，点击工具条上的 图标或按“Ctrl + F5”

即可运行程序。

现在终止程序，开始编码工作，为它增加新的功能。

首先，声明文档类数据成员。在 Project Workspace 中选择 FileView 标签，依次扩展 MDI_Draw Files 和 Header Files，双击打开 MDI_DrawDoc. h 文件。其中包含了 MDI_Draw 程序的文档类 CMDI_DrawDoc，它是继承自 Cdocument 类。在这里加入三个公有数据成员，程序修改后如下：

```
class CMDI_DrawDoc : public CDocument
{
protected: // create from serialization only
    CMDI_DrawDoc();
    DECLARE_DYNCREATE(CMDI_DrawDoc)
// Attributes
public:
//三个数据成员
    int m_PosX;              //存放坐标 X 的值
    int m_PosY;                  //存放坐标 Y 的值
    CString m_CurrentShap;    //存放使用的图形
……
};
```

m_PosX 和 m_PosY 将指定绘图的位置，m_CurrentShap 将存储是用圆还是用矩形绘图。

然后，声明视类数据成员。MDI_Draw 的视类 CMDI_DrawView 主要控制屏幕上的显示外观，而文档类控制文档的内容，两者相互配合，共同实现文档在文档窗口内的显示。就仿佛视类完成了文档类在屏幕上的镜像工作。

打开 CMDI_DrawView. h 文件，为其增加如下声明：

```
class CMDI_DrawView : public CView
{)
protected: // create from serialization only
    CMDI_DrawView();
    DECLARE_DYNCREATE(CMDI_DrawView
// Attributes
public:
    CMDI_DrawDoc * GetDocument();
//声明四个数据成员
    int m_PosX;                  //指定 X 坐标
    int m_PosY;                  //指定 Y 坐标
    CString m_CurrentShap;       //当前使用的图形
    CString m_SelectShap;        //选择的图形
// Operations
public:
……
};
```

m_PosX 和 m_PosY 指定显示坐标，两个字符串类成员分别存放正在使用的图形和选择的图形。另一在类中声明的成员 GetDocument 函数将在后面经常出现，通过它将得到指向文档类的指针。从“File”—“Sava All”保存修改的内容。

在 CMDI_DrawDoc 的成员函数 OnNewDocument 中为其数据成员初始化，可以从窗口上方的 WizardBar 中选择 CMDI_DrawDoc 类和成员 OnNewDocument，WizardBar 的界面包括三个组合框（类列表、成员列表、过滤器列表）和一个 Action 图控件。Action 控件又包含 3 个按钮和一个下拉式菜单。使用它可实现多种快捷功能，如果你还没有把它打开，可在空区域单击鼠标右键，然后选择 WizardBar，再选择好所需类和成员就可以了，选择后出现如下代码：

```
BOOL CMDI_DrawDoc: : OnNewDocument()
{
  if (!CDocument: : OnNewDocument())
      return FALSE;
//初始化变量内容
  m_PosX = 200;
  m_PosY = 100;
//初始化变量内容
  // TODO: add reinitialization code here
  // (SDI documents will reuse this document)
  return TRUE;
}
```

m_PosX = 200 与 m_PosY = 100 初始了文档打开时将显示图形的中心位置。每当程序创建一个新文档时都会自动执行 OnNewDocument 。

接着为视类初始化数据成员，这些工作在 CMDI_DrawView 的 OnInitialUpdate 成员函数中完成。

可从“View”—“ClassWizard”或点击鼠标右键选择“ClassWizard”，打开 ClassWizard。在 ClassWizard 中选择以下事件：

```
Class Name: CMDI_DrawView
Object ID: CMDI_DrawView
Message: OnInitialUpdate
```

点击“Add Function”和“Edit Code”按钮，添加如下代码：

```
void CMDI_DrawView: : OnInitialUpdate()
{
  CView: : OnInitialUpdate();
  CMDI_DrawDoc * pDoc = GetDocument();
  m_PosX = pDoc - >m_PosX;
  m_PosY = pDoc - >m_PosY;
  m_CurrentShap = "Circle";
}
```

OnInitialUpdate 将完成对 CMDI_DrawView 的初始化工作，这里首先是定义了一个指向文档类的指针，并为它赋值，这样就保持了视类与文档类内容的一致。m_CurrentShap = “Circle”是使默认使用的图形为圆。

完成初始化工作后，现在要实现文档绘图的操作，从 ClassView 打开 CMDI_DrawView 的成员函数 OnDraw 加入如下代码：

```
void CMDI_DrawView: : OnDraw(CDC * pDC)
{
  CMDI_DrawDoc * pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  // TODO: add draw code for native data here
  RECT rect;
  rect . top = m_PosY - 40;
  rect . bottom = m_PosY + 40;
```

```
    rect . left = m_PosX - 40;
    rect . right = m_PosX + 40;
    if(m_CurrentShap = = "Circle")    //判断当前使用的图形是否为'圆'
    pDC - >Ellipse(& rect);           //是圆,则画圆
  else
        pDC - >Rectangle(& rect);     //不是圆,则画矩形
}
```

其中，OnDraw 的参数 pDC 是指向设备描述表的指针，通过判断字符串类数据 m_CurrentShap 的值，指针将调用 Ellipse 函数或 Rectangle 函数，在屏幕上绘图，参数& rect 则定义了绘图的区域。

为实现图形选择，我们要修改工具条和菜单，并为它们添加代码。在 Project Workspace 中选择 ResourceView 标签 依次扩展 MDI_Draw Resources 和 Toolbar，双击 ID 值为 IDR_MAINFRAME 选项，进入工具条设计区。此时，点击工具条上任意一个按钮，都会显示详细设计，选中右边的空按钮，按回车键，显示属性对话框，在 ID 栏中填入"ID_DRAW_CIRCLE"回车确定。然后，选择画板中的圆形画笔，并选取一种颜色，就可以在操作区为按钮绘制图标了。此按钮将作为圆形选择按钮，随后可按同样方法绘制矩形选择按钮，ID 值为 "ID_DRAW_RECTANGLE. 。对于废弃的按钮，将它拖出工具条外，就可以删除。

再扩展 Menu 标签，双击 ID 值为 IDR_MDI_DRTYPE 的选项，进入菜单设计。拖动右边的空菜单项到合适的位置，按回车打开属性对话框，在 Caption 中填入'画图"，回车确定；再次回车，进入下一级菜单，在 Caption 中填入"Circle"，在 Prompt 中填入'画一个圆" 当鼠标停留在该菜单上会出现这个提示）。最后 ID 栏中填入与工具条中画圆按钮相同的 ID 值，这样就把工具条和菜单联系起来了。

用同样的方法，在下一个菜单项中设计矩形选择菜单，完成菜单和工具条的界面设计工作。

接下来给程序的相应部分编写代码，首先是实现圆形选择。

打开 ClassWizard 对话框，选择一事件：

```
Class Name: CMDI_DrawView
Object ID: ID_DRAW_CIRCLE
Messages: COMMAND
```

点击 Add Function 按钮和 Edit Code 按钮，程序会自动建立，并打开 OnDrawCircle 函数。为其加入如下代码：

```
void CMDI_CircleView: : OnDrawCircle()
{
    // TODO: Add your command handler code here
    m_SelectShap = "Circle";
}
```

程序运行后，用户点击工具条上圆形选择按钮或从菜单选择'画图"— "Circle"项，OnDrawCircle 函数将被调用，赋值语句：

```
m_SelectShap = "Circle"
```

用来定义选择的图形是圆。

再次打开 ClassWizard 对话，选择如下事件：

```
Class Name : CMDI_DrawView
Object ID : ID_DRAW_CIRCLE
Messages : UPDATE_COMMAND_UI
```

单击 Add Function 按钮，建立 OnUpdateDrawCircle 函数，单击 Edit Code 按钮进行编辑。代码如下：

```
void CMDI_CircleView: : OnUpdateDrawCircle(CCmdUI * pCmdUI)
{
if (m_SelectShap = = "Circle")
pCmdUI - >SetCheck(1)     ;
else
pCmdUI - >SetCheck(0);
}
```

OnUpdateDrawCircle 函数将在'画图"菜单打开前执行，参数 pCmdUI 是指向"Circle"菜单的指针，pCmdUI - > SetCheck 1 将在"Circle"菜单放置一检取标志，pCmdUI - > SetCheck 0 将取消检取标志。

图 4

至此，完成圆形选择的建立，按照同样的方法打开 ClassWizard，建立并编辑 OnUpdateDrawRectangle 函数和 OnDrawRectangle 函数，代码如下：

```
void CMDI_CircleView: : OnUpdateDrawRectangle(CCmdUI * pCmdUI)
{
if (m_SelectShap = = "Rectangle")
pCmdUI - >SetCheck(1)     ;
else
pCmdUI - >SetCheck(0);
}
void CMDI_CircleView: : OnDrawRectangle()
{
  // TODO: Add your command handler code here
  m_SelectShap = "Rectangle";
}
```

要使文档在鼠标点击的任意位置上绘图，还需要增加代码以响应鼠标的左击事件。

打开 ClassWizard 对话框，选择以下事件：

```
Class Name : CMDI_DrawView
Object ID : CMDI_DrawView
Messages : WM_LBUTTONDOWN
```

点击 Add Function 按钮和 Edit Code 按钮，建立并打开名为 OnLButtonDown 的函数，编辑后代码如下：

```
void CMDI_CircleView: : OnLButtonDown(UINT nFlags, CPoint point)
{
  CMDI_CircleDoc * pDoc = GetDocument();   //建立指向
```

文档的指针

```
    m_CurrentShap = m_SelectShap; //确定当前绘图用的图形
      m_PosX = point. x;
      m_PosY = point. y;
      Invalidate();                    //触发 OnDraw 函数
//保持文档和视的内容一致
      pDoc -> m_CurrentShap = m_CurrentShap;
      pDoc -> m_PosX = m_PosX;
      pDoc -> m_PosY = m_PosY;
//保持文档和视的内容一致
      pDoc -> SetModifiedFlag(true);        //设置修改标志
      CView: : OnLButtonDown(nFlags, point);
}
```

语句 m_PosX = point. x 和 m_PosY = point. y 得到鼠标的位置，并通过文档指针保持文档和视的内容一致；Invalidate 导致 OnDraw 函数的执行，以便在屏幕上显示图形。pDoc - >SetModifiedFlag true 是指明文档已经被更改，这样如果文档关闭前没有保存，会出现提示对话框。

现在，可以再运行程序查看一下结果。文档中间出现一个圆，用鼠标点击任意位置，圆的位置也随着鼠标移动。从工具条或菜单中选择矩形，再次点击鼠标，文档中出现的是矩形。但这一切还不能保存，并且，MDI 界面的一个重要特点，就是能在多个窗口同时显示同一文档，改变一个窗口的内容，其它窗口内也会发生改变。下面来最后完善这个程序。

实现同一文档的多重显示，首先在刚才加入 OnLButtonDown 函数的代码的最后加入一条语句：

```
pDoc -> UpdateAllViews(this);
```

UpdateAllViews 函数执行的直接后果是调用视图类的 OnUpdate 函数，来更新除当前窗口外的所有窗口，参数 this 是指向当前视类的指针。下面就来编辑 OnUpdate 函数：

打开 ClassWiard 选择以下事件：

```
Class Name : CMDI_DrawView
Object ID : CMDI_DrawView
Messages : OnUpdate
```

单击 Add Function 按钮和 Edit Code 按钮，建立并打开名为 OnUpdate 函数，编辑后代码如下：

```
void CMDI_CircleView: : OnUpdate(CView * pSender,
LPARAM lHint, CObject * pHint)
{
CMDI_CircleDoc * pDoc = GetDocument();
m_PosX = pDoc -> m_PosX;
m_PosY = pDoc -> m_PosY;
Invalidate();
}
```

首先是建立一个指向文档的指针 pDoc，然后利用文档类的数据更新视类的数据，最后用 Invalidate 函数触发 OnDraw 函数。

到现在为止已经可以实现同一文档的多窗口显示了。

文档的保存与打开通过 CMDI_DrawDoc 类中的 Serialize 函数来完成。

从 ClassView 选择 CMDI_DrawDoc 的 Serialize 函数，未编辑前程序如下：

```
void CMDI_DrawDoc: : Serialize(CArchive& ar)
{
  if (ar. IsStoring())
  {
        // TODO: add storing code here
  }
  else
  {
        // TODO: add loading code here
  }
}
```

这个函数将被菜单 File 下的 Save、Save As 和 Open 项所触发，函数中 “if…else… ” 将用作判断执行哪一项。参数 ar 代表了当前的文档，如果选择 Save 或 Save As， ar. IsStoring 将为 “TRUE ”。现在我们为 “if…else… ” 加入代码，结果如下：

```
void CMDI_CircleDoc: : Serialize(CArchive& ar)
{
  if (ar. IsStoring())
  {
        // TODO: add storing code here
        ar < < m_PosX;
        ar < < m_PosY;
  }
  else
  {
        // TODO: add loading code here
        ar>>m_PosX;
        ar>>m_PosY;
  }
}
```

函数中两个流操作符 “< < ”、 “>> ” 分别用于存储和读取功能，其形式还可写作

```
ar < < m_PosX < < m_PosY ;
```

在完成 ‘保存 ”、 ‘打开 ” 功能后，我们再对程序作些细节上的设计。

在 Project Workspace 中选择 ResourceView 标签 依次扩展 MDI_Draw Resources 和 String Table，选择 ID 值为 String Table 的项，双击打开 String Table 对话框，再双击第一项 IDR_MAINFRAME，将其对话框打开，它的 Caption 值用于指示主框架的标题，改为 ‘MDI_Draw ” 回车确定；

第二项 IDR_MDI_CITYPE 后的 Caption 值，是关于 MDI 子窗口的一些属性，每一项前都以 “hn ” 隔开，我们将修改其 1、4、5 项，它们分别指示子窗口的标题、保存文件的缺省文件类型和打开文件的缺省文件类型。同样双击打开对话框，修改 Caption 值后结果如下：

```
\ nMDI_Dr\ nMDI_Dr\ nDRW Files( * . drw)\ n. drw\ nMDIDraw. Document
\nMDI_Dr. Document
```

回车确定。至此，我们的工作全部完成了。选择 ‘Build ” — ‘Rebuild All ”，然后运行后，看看结果。

利用 InstallShield 制作 PowerBuilder 应用程序安装盘

黄林进

在软件开发过程中，将软件制成安装盘发放给用户往往是最后一步，也是非常重要的一步。制作应用程序安装盘的软件有很多，InstallShield 是专业级的安装盘制作程序，具有功能强大、使用方便等特点。本文就 PowerBuilder 附带的 InstallShield 模块例举如何实现制作 PowerBuilder 应用程序安装盘。

一、 应用程序的编译与动态库的生成

在 PowerBuilder 中可以很方便地创建可执行程序，当工程项目较小时，利用 PowerBuilder 开发的软件可以将所有的对象和资源都编译到一个 EXE 文件中，运行此文件，它就开始工作，且显得简单又别致。但 PowerBuilder 是一个企业级的应用程序开发工具，面对的可能是大项目，此时如果将所有的资源都包含在单一的 EXE 文件中，不但会使软件难于管理，而且会使软件运行效率低下，因此在实际过程中常常把 EXE 文件控制在 1M 以下，而将其它部分放在一个或是几个动态库（PBD / DLL）文件中，只在程序运行时动态地调用。通常在以下情况下需要生成动态库：

1. 当在程序中动态引用 DataWindow 对象时， DataWindow 只在 PowerScript 中动态地与 DataWindow 控件相关联时才被引用 如 dw_accounts. DataObject = “d_nocount ”。

2. 当在程序中动态引用图片或图像等对象时，如 p_cover. PictureName = “cover1. bmp ”。

3. 当应用程序过大时。

创建动态库根据不同情况也有不同的方式：

A. 如果在程序中动态引用数据窗口对象或是应用程序过大时，可以在 Library 描绘器中创建一个或几个动态 PBL 库，将动态引用的数据窗口对象或程序中的其他对象分别放到 PBL 库中。如数据窗口对象可以放到 DW. PBL 中，而窗口对象放到 WINDOW. PBL，菜单对象放到 MENU. PBL 中等等，如此分门别类的建立动态 PBL 库有利于程序的修改和维护，建立了相应的动态 PBL 库后一定要把这些库添加到应用程序的库搜索路径中，以便编译时能生成 PBD / DLL。

B. 在应用程序中要用到图片 （BMP 和 RLE 文件）、图标（ICO 文件）和指针 （CUR 文件）等资源时，可以创建一个 PBR 资源文件，该资源文件可以由任何文本编辑器创建，每个资源占一行，且必须带有绝对路径。如 JXC. PBR 资源文件如下所示：

```
C: \BMP\Sample1. bmp
C: \ICO\Sampleico. ico
. . . . . .
```

在分割应用程序创建动态库时还须注意以下几点：

A. 太多的 PBD / DLL 就和一个巨大的可执行程序一样麻烦。

B. 把最有可能被其他可执行程序共享的对象放到动态库中。

C. 确保将使用率高的对象放入 EXE 文件中，而将较少使用的对象放到 PBD / DLL 中。

建立了动态 PBL 库和资源文件后，可以在 PowerBuilder 的 Project 描绘器中为应用程序创建可执行文件，同时也将相应的

小结

通过本讲的学习，应当掌握以下几点：

1. 使用 MFC AppWizard 建立多文档界面程序；

2. 使用 ClassWizard 为程序编辑代码；

3. 使用 ClassView 快捷地选择类和函数；

4. 使用 ResourceView 编辑工具条、菜单，设计窗口标题和保存、打开文档的缺省文件类型；

5. 分别使用 OnNewDocument 和 OnInitialUpdate 初始化文档类及视类；

6. 使用文档类指针保持文档类和视类的一致；

7. 通过 UpdateAllViews 函数，Invalidate 函数，及编辑 OnUpdate 函数和 OnDraw 函数实现同一文档的多窗口显示；

8. 使用操作流保存、打开文件。

参考资料

1. 王晖等编著 . 精通 Visual C + + 6. 0 电子工业出版社

2. Joh Mueller 著，希望图书创作室译 . Visual C + + 5. 0 从入门到精通 中国轻工业出版社

3. http / / vcdynasty. yeah. net h Artical h VC + + _artical h vc4 h chap4. htm

收稿日期：2000 年 11 月 2 日）

动态 PBL 库和资源文件编译成 PBD / DLL。

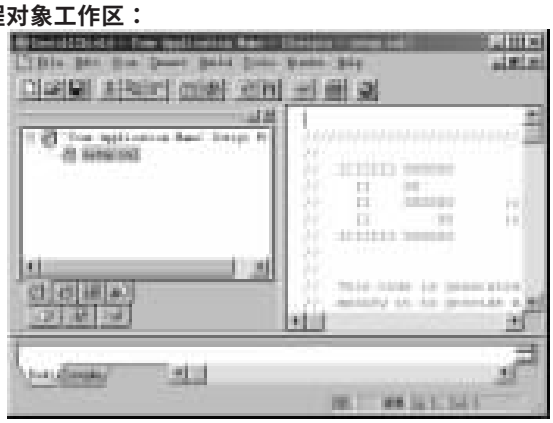
二、 应用运行环境的建立与安装盘的制作

应用运行环境分为两部分：

1. PowerBuilder 应用运行应具有的基本环境，即 PowerBuilder 提供的支持文件。包括 PowerBuilder 应用运行所需的 DLL 文件、数据库接口 （ODBC 接口和其它 Powersoft 数据库接口）、配置已安装的 ODBC 驱动程序等。这些支持文件由 PowerBuilder 中的 Deployment Kit 软件包提供，分别放在 Powersoft 子目录 DDDK6 下的 Deployment DLLs、ODBC Drivers、DB Interfaces 三个目录中。

2. 应用系统编译后的文件和其它相关文件，包括可执行（EXE）文件、动态库 （DLL. 或 PBD）文件、初始化（INI）文件、帮助 （HLP）文件及文本、声音等其它类型的文件，若应用程序需要访问本地的 Sybase SQL AnyWhere 数据库，则需将数据库 （db）文件安装到每一最终用户的机器中。在明确应用程序的运行环境及对应用程序进行编译后，就可以启动 InstallShield 来制作安装盘。打开的安装盘制作窗口如下图所示：

双击 Project wizard 图标，进入工程向导，可以依据向导的提示一步一步地设置安装盘安装时的界面。在 Specify Components 组件对话框和 Specify File Groups 文件组对话框中分别添加 “ODBC ” 组件和 “ODBC ” 文件组，最后进入如下图所示的工程对象工作区：



将工程对象工作区切换到文件组 File Groups 对应的工作窗口，把各文件组所需的文件链接到相应的文件组中，如下表所示：

文件组名	包含的内容
Program Executable Files	编译后的可执行（EXE）文件
Help Files	帮助文件和 README 文件
Program DLLs	应用所需的动态链接库、初始化文件和数据库文件
Shared DLLs	Deployment Kit 中 Deployment DLLs 目录下的文件
ODBC	Deployment Kit 中 ODBC Drivers 目录下的文件

将所有应用程序所需的文件链接到相应的文件组后，把工程对象工作区切换到与组件 （Components）对应的工作窗口，如下表所示将各组件所需的文件组链接到相应的组件中：

组件名	组件所包含的文件组
Program Files	Program Executable Files 和 Program DLLs
Help Files	Help Files
Shared DLLs	Shared DLLs
ODBC	ODBC

做完以上的工作后，在实际的安装盘制作过程中还有一些特殊的功能需要通过编程来实现，如在开始菜单和桌面上增加应用的图标或配置数据源和 ODBC 驱动程序时需要修改系统注册表等。此时需要将工程对象工作区切换到源程序所对应的工作窗口，在这个工作窗口中我们可以看到在工程向导过程中已经自动生成了很多基本的源程序，并为用户编程留下入口。下面详细地讨论如何实现修改系统注册表和增加图标，读者可以依此根据需要增加其它功能。在源程序中找到如下程序代码，

```
////////// string defines //////////
#define UNINST_LOGFILE_NAME      "Uninst. isu"
////////// installation declarations //////////
```

并在该代码段下添加定义用户注册信息的各级分支，程序代码如下：

```
#define COMPANY_NAME       "ODBC"
#define PRODUCT_NAME       "ODBC. INI"
#define PRODUCT_VERSION    "jxcgl"
#define PRODUCT_KEY        "rtdsk50. exe"
#define DEINSTALL_KEY      "jxcgl_DeinstKey"
#define UNINSTALL_NAME     "jxcgl_uninstname"
#define DEFAULT_LOG_PATH   "jxcgl"
```

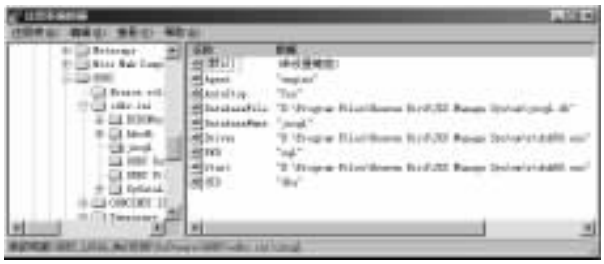
在 - - - - global variables - - - - - 下定义如下变量:

```
STRING     szStrName1, szStrValue1, svLogFile;
STRING     szStrName2, szStrValue2;
STRING     szStrName3, szStrValue3;
STRING     szStrName4, szStrValue4;
STRING     szStrName5, szStrValue5;
STRING     szStrName6, szStrValue6;
STRING     szStrName7, szStrValue7
STRING     szStrName8, szStrValue8;
NUMBER     nvSize, nvType;
```

找到源程序中的 SetupRegistry 函数，在该函数中添加如下程序段：

```
function SetupRegistry()
 begin
 szStrName1 = "Agent";
 szStrValue1 = "engine";
 szStrName2 = "AutoStop";
 szStrValue2 = "Yes";
 szStrName3 = "DatabaseFile";
szStrValue3 = PROGRAMFILES ^ @ COMPANY_NAME ^ @ PRODUCT_NAME + "\\jxcgl. db";
 szStrName4 = "DatabaseName";
 szStrValue4 = "jxcgl";
 szStrName5 = "Driver";
szStrValue5 = PROGRAMFILES ^ @ COMPANY_NAME ^ @ PRODUCT_NAME + "\\rtdsk50. exe";
 szStrName6 = "PWD";
 szStrValue6 = "sql";
 szStrName7 = "Start";
szStrValue7 = PROGRAMFILES ^ @ COMPANY_NAME ^ @ PRODUCT_NAME + "\\rtdsk50. exe";
 szStrName8 = "UID";
 szStrValue8 = "dba";
 RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);
InstallationInfo(COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);
      DeinstallStart(DEFAULT_LOG_PATH, svLogFile, DEINSTALL_KEY, 0);
      RegDBSetAppInfo(szStrName1, REGDB_STRING, szStrValue1, -1);
      RegDBSetAppInfo(szStrName2, REGDB_STRING, szStrValue2, -1);
      RegDBSetAppInfo(szStrName3, REGDB_STRING, szStrValue3, -1);
      RegDBSetAppInfo(szStrName4, REGDB_STRING, szStrValue4, -1);
      RegDBSetAppInfo(szStrName5, REGDB_STRING, szStrValue5, -1);
      RegDBSetAppInfo(szStrName6, REGDB_STRING, szStrValue6, -1);
      RegDBSetAppInfo(szStrName7, REGDB_STRING, szStrValue7, -1);
      RegDBSetAppInfo(szStrName8, REGDB_STRING, szStrValue8, -1);
return 0;
end;
```

该安装盘安装后系统注册表中数据库接口程序 ODBC 配置的结果如下图所示：



要在开始菜单和桌面上添加图标，须在源程序中找到 SetupFolders 函数，并在该函数下添加如下程序段：

```
function SetupFolders()
# define PROGRAM PROGRAMFILES ^ @ COMPANY_NAME ^ @ PRODUCT_NAME + "\\jxcgl. exe"
# define SOURCE_DIR PROGRAMFILES ^ @ COMPANY_NAME ^ @ PRODUCT_NAME
  #define DEL_FILE "jxcgl. log"
  STRING szProgramFolder, szItemName, szCommandLine, szWorkingDir, szIconPath;
  STRING szShortCutKey, szProgram, szParam;
  STRING szPath, szFile;
  NUMBER nIcon, nFlag;
begin
  szProgramFolder = "";
  szItemName = "进销存管理系统试用版";
  szProgram = PROGRAM;
  LongPathToQuote(szProgram, TRUE);
  szCommandLine = szProgram;
  szWorkingDir = "";
  szIconPath = "";
  nIcon = 0;
  szShortCutKey = "";
  nFlag = REPLACE;
AddFolderIcon(szProgramFolder, szItemName, szCommandLine, szWorkingDir, szIconPath, nIcon, szShortCutKey, nFlag);
  szProgramFolder = FOLDER_DESKTOP;
AddFolderIcon(szProgramFolder, szItemName, szCommandLine, szWorkingDir, szIconPath, nIcon, szShortCutKey, nFlag);
return 0;
end;
```

该安装盘的安装后，在开始菜单和桌面上都生成了名为‘进销存管理系统试用版”的应用程序图标。接下来就可以对当前工程对象进行编译（选择 Build 菜单中的 Compile 项）并应用建立磁盘映像的向导（选择 Build 菜单中的 Media build Wizard 项）在硬盘上创建磁盘映像，最后将磁盘映像分别拷贝到光盘或是软盘上即可颁布发行。

三、 结束语

以上所述只是说明了安装盘制作的基本步骤和内容，对于一般应用程序来说已经足够，如需要制作更加完美的安装盘，只需依照上面所述在工程对象的源程序工作区中添加所需的函数或对已有的函数进行完善即可。

调试说明

本文所述运行环境为 PowerBuilder6. 0 及其所附属的 Install Shield5. 0，操作系统为 Windows98 或 Windows NT。

（收稿日期：2000 年 8 月 7 日）

如何在 PowerBuilder 中实现界面的自动布置

李 激

在 Windows 环境下进行可视化应用程序开发时，不可避免地要遇到界面布置问题。为了使界面更美观，编程者常常希望当窗体改变大小时，界面会自动布置，不至于出现当窗体大小改变时，窗体上出现大片空白区域或部分控件因超出窗口边界而消失的现象。有时可能还希望在两个控件间加一个分割条，就象 Windows 资源管理器那样，左右两个视图间有一个分割条。为了实现窗口的自动布置，当窗口大小改变时需对窗体上各控件重新计算几何位置和大小。这种计算往往是十分繁琐的，特别是当窗体上控件数目较多时，更是让程序员“不耐烦”。本文提出的方法将使这一工作变得简单。通过一个叫 Layout_Policy（布置策略）的类，用户可以将对窗体的布置要求通知该类的对象，而由该类的对象自动完成窗体的布置工作。本文用 PowerBuilder 实现了该思路，当然该思路完全可以在 VB、VC 或其它开发工具中实现。Layout_Policy 类是对窗体布置操作的抽象，它具有如下功能：（1）根据控件和容器（如 PB 中的 Window 和 Userobject、VB 中的 Form 和 UserControl）以及控件和控件间的位置关系（如对齐、粘接等），对容纳控件的容器进行布置。（2）在控件间设置分割条。（3）允许运行时增加或删除布置策略，从而实现界面的动态布置。

一、基本思路

界面的布置问题无非是如何在窗体（或其它容器）内确定控件的位置和大小。为了叙述方便，本文将控件和容纳控件的容器（如窗体或用户对象）统称为界面元素。一个控件在窗体内的布置方式可以用“布置策略”来描述。所谓布置策略是指一控件和周边其它界面元素的几何位置关系。Windows 环境中的窗体和控件都是具有左右上下四个边的矩形，所以通过控件左右上下四个方向上和其它界面元素的定位关系，可以确定控件的位置和大小。这种定位关系就是控件的布置策略。本文中给出的 Layout_Policy 类正是用于描述布置策略，并能根据布置策略对界面进行几何布置。

一个控件的布置策略包括以下内容：控件名称，控件四周和周边其它界面元素的位置关系，和控件水平中心对齐的界面元素名，和控件垂直中心对齐的界面元素名。其中，每一个位置关系又包括：基准界面元素的名称，和基准的位置关系类型，和该关系相关的数值（如间隙，偏移量等）。控件和基准间的关系可分为以下几种类型：

● 0 —— 粘接（Attachment）当基准是控件时，粘接总是发生在相对的两边。如控件的左边和基准控件的右边粘接，关系值表示间隙。

● 1 —— 对齐（Alignment）当基准是控件时，对齐总是发生在相同的两边。如控件的左边和基准控件的左边对齐，关系值表示偏移量，正值表示左偏移，负值表示右偏移。当基准是容器时，对齐和粘接的含义是一样的。

● 2 —— 比例定位（Position）如控件的左边在基准控件（或容器）宽度范围内的特定比例处，比例值由关系值给出。

● 3 —— 移动粘接（Move Attachment）对于控件左边和上边的定位关系，粘接和移动粘接含义一样。对于控件右边和下边的定位关系，粘接关系要改变控件的宽度或高度，而移动粘接则不改变控件的大小。

● 4 —— 移动对齐（Move Alignment）对于控件左边和上边的定位关系，对齐和移动对齐含义一样。对于控件右边和下边的定位关系，对齐关系要改变控件的宽度或高度，而移动对齐则不改变控件的大小。

由于布置策略描述了界面元素间的几何约束关系，当按布置策略执行布置操作时，必须遵循“先布置基准，后布置自身”的原则。根据基准关系，并考虑到基准关系的方向，界面元素间构成两个优先关系图，一个代表水平方向，一个代表垂直方向。将这两个有向图进行拓扑排序，得到一个水平布置序列和一个垂直布置序列，按这两个序列的次序，分别对各控件进行水平布置和垂直布置，就可以最终得到反映布置策略的布置结果。

控件间的分割条是通过用户对象 SplitBar 实现的。SplitBar 是从标签控件（StaticText 控件）继承来的可视用户对象。通过定义在其上的鼠标事件可以对它进行拖动操作。容器（窗体或用户对象）、Layout_Policy 对象、SplitBar 之间的关系如下图所示：

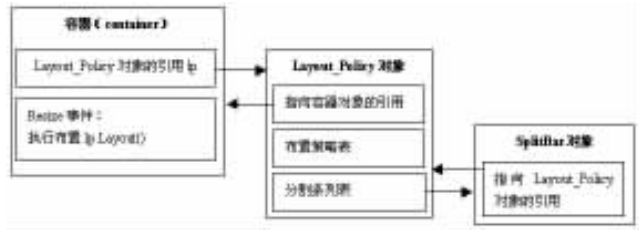


图 1 容器、布置策略对象及分割条间的关系

二、代码介绍

限于篇幅，本文只能提供 Layout_Policy 类和 SplitBar 用户对象的主要代码。另外，由于 PB 中的代码是存放在 .pbl 文件中的，所以本文给出的代码是通过 PB 库管理器的 Export 功能

导出的。

1. Layout_Policy 类的实现

（）Layout_Policy 类的实例变量如下：

```
type variables
Private:
GraphicObject  theContainer //指向容器的引用变量(Window 或 Userobject)
/*存放水平和垂直两个方向上'基准关系有向图'拓扑排序结果的数组,数组中存放控件在策略表中的索引.两数组的第一个元素都为 1,表示容器.*/
Integer    HSortedControls[], VSortedControls[]
Boolean    NeedResort = True
//位置关系类型常量
Constant   Integer Attachment = 0
Constant   Integer Alignment = 1
Constant   Integer Position = 2
Constant   Integer MoveAttachment = 3
Constant   Integer MoveAlignment = 4
Constant   Integer Horizontal = 0
Constant   Integer Vertical = 1
str_Layout_Policy  Policylist[]   //存放布置策略的动态数组
str_Split  SplitList[] //分割条列表
Integer    BarThickness = 10  //分割条缺省宽度
Long       BarColor = 0 //分割条颜色
end variables
```

2 construct 事件和 destruct 事件代码

```
event constructor
PolicyList[1].name = "container" //将策略表的第一条记录设为容器(容器为总基准)
Hsortedcontrols[1] = 1 //水平排序表的第一个元素为 1,表示容器
Vsortedcontrols[1] = 1 //垂直排序表的第一个元素为 1,表示容器
SetNull(theContainer)
end event
```

3）结构 str_Layout_Policy 和 str_Split 的定义

```
type str_layout_policy from structure
    string      name //要设置布置策略的控件名称
    string      left //左基准的名称
    integer     leftrelation //左基准关系的类型
    real        leftspaceorposition //左基准关系的数值(以下 3 组变量类似)
    string      right
    integer     rightrelation
    real        rightspaceorposition
    string      top
    integer     toprelation
    real        topspaceorposition
    string      bottom
    integer     bottomrelation
    real        bottomspaceorposition
    string      hcenter //和当前控件水平中心对齐的控件名称,为"container"时表示容器
    string      vcenter//和当前控件垂直中心对齐的控件名称,为"container"时表示容器
end type
type str_split from structure
    splitbar    bar   //分割条用户对象变量
    string      prevcontrol //分割条左边(或上边)的控件名称,为"container"时表示容器
    string      nextcontrol //分割条右边(或下边)的控件名称,为"container"时表示容器
end type
```

4）将策略对象和容器（窗体或用户对象）绑定——AttachToContainer

```
public function boolean attachtocontainer (readonly graphicobject a_container)
Integer  i, count
window   thewin
userobject   theuo
if Not IsNULL(theContainer) then return true
theContainer = a_container //将容器(窗体或用户对象)赋值给 theContainer 变量
//初始化策略表的 Name 字段
if theContainer.typeof() = window! then
   thewin = thecontainer
   count = UpperBound(thewin.Control[])
   for i = 1 to count
   PolicyList[i+1].Name = theWin.Control[i].ClassName()
   next
elseif theContainer.TypeOf() = userobject! then
   theuo = thecontainer
   count = UpperBound(theuo.Control[])
   for i = 1 to count
   PolicyList[i+1].Name = theUO.Control[i].ClassName()
   next
else
   return false
end if
return true
end function
```

5）设置布置策略的成员函数—— AddLayoutPolicy

```
public subroutine addlayoutpolicy (string as_name, string as_left, integer ai_leftrelation, real ar_leftspaceorposition, string as_right, integer ai_rightrelation, real ar_rightspaceorposition, string as_top, integer ai_toprelation, real ar_topspaceorposition, string as_bottom, integer ai_bottomrelation, real ar_bottomspaceorposition, string as_hcenter, string as_vcenter)
Integer      pos
if IsNULL(thecontainer) then return
pos = GetNextPos(as_name) //获取数组 Policylist[]中空闲位置
if pos = 0 then return
Policylist[pos].Name = as_Name
Policylist[pos].Left = as_Left
Policylist[pos].LeftRelation = ai_LeftRelation
```

```
Policylist[pos]. LeftSpaceOrPosition = FilterValue (ai_leftrelation, ar_leftspaceorposition)
Policylist[pos]. Right = as_Right
Policylist[pos]. RightRelation = ai_RightRelation
Policylist[pos]. RightSpaceOrPosition = FilterValue(ai_rightrelation, ar_RightSpaceOrPosition)
Policylist[pos]. Top = as_Top
Policylist[pos]. TopRelation = ai_TopRelation
Policylist[pos]. TopSpaceOrPosition = FilterValue (ai_toprelation, ar_TopSpaceOrPosition)
Policylist[pos]. Bottom = as_Bottom
Policylist[pos]. BottomRelation = ai_BottomRelation
Policylist[pos]. BottomSpaceOrPosition = FilterValue (ai_bottomrelation, ar_BottomSpaceOrPosition)
Policylist[pos]. HCenter = as_hcenter
Policylist[pos]. VCenter = as_vcenter
NeedReSort = true
end subroutine
```

6 执行布置函数——Layout

```
public subroutine layout ()
Integer i, size
if IsNULL(theContainer) then return
if NeedResort then
    ToplogicalSort() //对容器内的各控件进行拓扑排序
    NeedResort = False
end if
//根据水平排序表和垂直排序表布置各控件
size = GetSortedControlsNum(HSortedControls[])
for i = 2 to size
    PositionItH(HSortedControls[i]) //布置水平排序表中的各控件
next
size = GetSortedControlsNum(VSortedControls[])
for i = 2 to size
    PositionItV(VSortedControls[i]) //布置垂直排序表中的各控件
next
PositionSplitBar() //布置分割条
end subroutine
```

7 对控件进行布置 私有函数) —— PositionItH PositionItV

PositionItH 用于对控件进行水平方向的布置，ai_index 为控件布置策略的索引，PositionItV 用于对控件进行垂直方向的布置，其代码和 PositoinItH 相仿。

```
private function boolean positionith (integer ai_index)
DragObject ThisControl, BaseControl
Real value
ThisControl = NameToControl(policylist[ai_index]. name)
if IsNull(ThisControl) then return false
//根据左基准布置
value = policylist[ai_index]. LeftSpaceOrPosition
Choose case policylist[ai_index]. left
    case "container"
        choose case PolicyList[ai_index]. LeftRelation
    case Position
        ThisControl. x = GetContainerWidth() * value
    case Alignment, Attachment, MoveAlignment, MoveAttachment
        ThisControl. x = value
        end choose
    case ""//如果没有基准,什么都不做
    case else
        BaseControl = NameToControl(policylist[ai_index]. left)
        if IsNull(BaseControl) then return false
        Choose case PolicyList[ai_index]. LeftRelation
            case Alignment, MoveAlignment
                ThisControl. x = BaseControl. x + value
            case Attachment, MoveAttachment
ThisControl. x = BaseControl. x + BaseControl. width + value
            case Position
ThisControl. x = BaseControl. x + BaseControl. Width * value
        end choose
end choose
//根据右基准布置
value = PolicyList[ai_index]. RightSpaceOrPosition
Choose case policylist[ai_index]. right
    case "container"
        choose case PolicyList[ai_index]. RightRelation
            case Position
        ThisControl. x = GetContainerWidth( ) * Value - ThisControl. width
            case Attachment, Alignment
        ThisControl. width = GetContainerWidth( ) - ThisControl. x - value
            case MoveAttachment, MoveAlignment
        ThisControl. x = GetContainerWidth( ) - ThisControl. width - value
        end choose
    case ""
    case else
        BaseControl = NameToControl(policylist[ai_index]. right)
        if IsNULL(BaseControl) then return false
        Choose case PolicyList[ai_index]. RightRelation
            case Alignment
    ThisControl. width = BaseControl. x + BaseControl. Width - ThisControl. x - value
            case Attachment
ThisControl. width = BaseControl. x - value - ThisControl. x
            case Position
    ThisControl. x = BaseControl. x + BaseControl. Width * value - ThisControl. Width
            case MoveAlignment
    ThisControl. x = BaseControl. x + BaseControl. Width - ThisControl. width - value
            case MoveAttachment
ThisControl. x = BaseControl. x - value - ThisControl. width
        end choose
end choose
//根据垂直中心线的基准布置
```

```
Choose case policylist[ai_index].Vcenter
    case "container"
        ThisControl.x = GetContainerWidth()/2 - ThisControl.Width/2
    case ""
    case else
BaseControl = NameToControl(policylist[ai_index].Vcenter)
        if IsNULL(BaseControl) then return false
    ThisControl.x = BaseControl.x + BaseControl.width/2 - ThisControl.Width/2
end choose
return true
end function
```

（8）增加分割条函数——AddHSplitBar AddVSplitBar

```
public function boolean addhsplitbar (ref splitbar a_bar, string as_prev, string as_next)
Integer     i, size, newpos
DragObject      pdo, ndo
if IsNULL(theContainer) then return false
size = upperbound(SplitList[])
newpos = size + 1
for i = 1 to size
    if isnull(SplitList[i].Bar) then
        newpos = i
        exit
    end if
next
pdo = NameToControl(as_prev)
ndo = NameToControl(as_next)
if IsNull(pdo) or IsNULL(ndo) then return false
SplitList[newpos].Bar = a_bar
SplitList[newpos].prevcontrol = as_prev
SplitList[newpos].nextcontrol = as_next
SplitList[newpos].Bar.SetType(Horizontal)
SplitList[newpos].Bar.Height = barthickness
SplitList[newpos].Bar.SetPolicyObject(this)
SplitList[newpos].Bar.SetPosition(ToTop!)
return true
end function
```

AddVSplitBar 的代码和 AddHSplitBar 类似

（9）布置分割条函数 私有函数）—— PositionSplitBar

```
private subroutine positionsplitbar ()
Integer     i, size
DragObject   prevctl, nextctl
size = UpperBound(SplitList[])
for i = 1 to size
    if IsNull(splitlist[i].Bar) then continue
    prevctl = NameToControl(splitlist[i].prevcontrol)
    nextctl = NameToControl(splitlist[i].nextcontrol)
    if SplitList[i].Bar.gettype() = Horizontal then
        splitlist[i].Bar.x = Min(prevctl.x, nextctl.x)
        splitlist[i].Bar.y = prevctl.y + prevctl.height
splitlist[i] .Bar.width = Max(prevctl.x + prevctl.width, nextctl.x + nextctl.width) - Min(prevctl.x, nextctl.x)
    nextctl.Height += nextctl.y - splitlist[i] .Bar.y - splitlist[i].Bar.height
  nextctl.y = splitlist[i].Bar.y + splitlist[i].Bar.height
  elseif SplitList[i].Bar.gettype() = Vertical then
        splitlist[i].Bar.y = Min(prevctl.y, nextctl.y)
        splitlist[i].Bar.x = prevctl.x + prevctl.Width
splitlist[i] .Bar.Height = Max(prevctl.y + prevctl.Height, nextctl.y + nextctl.Height) - Min(prevctl.y, nextctl.y)
  nextctl.Width += nextctl.x - splitlist[i] .Bar.x - splitlist[i].Bar.Width
        nextctl.x = splitlist[i].Bar.x + splitlist[i].Bar.Width
  end if
next
end subroutine
```

（10）对控件进行拓扑排序 私有函数）——ToplogicSort

```
private subroutine toplogicalsort ()
Integer     i, Size, Count
Boolean   GoOn = true
ClearSortedControlsArray() //将数组 HsortedControls[]和 VsortedControls[]的元素清零
//按水平方向上的基准约束关系进行拓扑排序
Size = UpperBound(PolicyList[])
Count = 1
do while GoOn
  GoOn = false
  for i = 1 to Size
      //如果已进入排序表,直接进入下一循环
      if AlreadyExistsInHSortedControls(i) then continue
//如果第 i 个策略中的控件没有基准或基准控件已进入排序表,那么将其放入排序表
if HasNoHPrecedents(PolicyList[i]) then
          Count + +
          GoOn = true
          HSortedControls[Count] = i
      end if
  next
loop
//按垂直方向上的基准约束关系进行拓扑排序
Count = 1
GoOn = true
do while GoOn
  GoOn = false
  for i = 1 to Size
      if AlreadyExistsInVSortedControls(i) then continue
      if HasNoVPrecedents(PolicyList[i]) then
          Count + +
          GoOn = true
          VSortedControls[Count] = i
      end if
  next
loop
end subroutine
```

2. 分割条对象 SplitBar 的实现

SplitBar 是从 StaticText 继承的自定义用户对象。通过事件号 pbm_lbuttondown、pbm_lbuttonup、pbm_mousemove 为 SplitBar 定义自定义事件 mousedown、mouseup、mousemove。该对象的

实例变量、事件和函数代码如下：

```
type variables
private:
Layout_Policy    thePolicyObject //指向策略对象的引用
Integer          bar_type //分割条类型
Constant         Integer HorizontalSplitBar  = 0
Constant         Integer VerticalSplitBar    = 1
end variables
event mouseup
    this. backcolor = this. thePolicyObject. GetContainerBack-
Color( )
thePolicyObject. RepositionBar( ) //根据分割条位置重新布置
分割条两边的控件
powerobject                thecontainer
thecontainer = this. getparent( ) //获取分割条所在容器的引用
thecontainer. event dynamic resize(0, 0,0)  // 触发容器的
resize 事件
end event
event mousedown
this. backcolor  =  this. thepolicyobject. GetBarColor( ) //鼠标
按下时改变分割条的颜色
end event
event mousemove //鼠标左键被按下并移动时，改变分割条
位置
integer     ylimit, xlimit
if flags  <> 1 then return //如果鼠标左键不是处于按下状态，
返回
if bar_type  =  VerticalSplitBar then
    this. x  + = xpos
    xlimit  =  thepolicyobject. GetContainerWidth( )  –  20  –
thepolicyobject. GetBarThickness( )
    if this. x  <  0 then this. x  =  0
    if this. x > xlimit then this. x  =  xlimitt
elseif bar_type  =  HorizontalSplitBar then
    this. y  + = ypos
    ylimit  =  thepolicyobject. GetContainerHeight( )  –  20  –
thepolicyobject. GetBarThickness( )
    if this. y  <  0 then this. y  =  0
    if this. y >ylimit then this. y  =  ylimi
end if
end event
public subroutine settype (integer ai_type)//设置分割条类型
bar_type  =  ai_type
end subroutine
public subroutine setpolicyobject (layout_policy auo_lp) //存
放指向策略对象的引用
thepolicyobject  =  auo_lp
end subroutine
public function integer gettype ( ) //获取分割条类型
return bar_type
end function
```

三、一个实例

Layout_Policy 类的使用大体分为三个步骤：

●初始化。用户创建该类的对象并通过对象的 AttachTo-Container 方法和所要布置的容器（如窗体）绑定。

●描述布置策略。用户通过 AddLayoutPolicy 方法为各控件设置布置策略。

●执行布置。在容器（如窗体）的 Resize 事件中执行该对象的 Layout 方法。如果容器为 Userobject 需为该 Userobject 定义一个 resize 事件（事件号为 pbm_size）

下面以一个实例说明怎样使用 Layout_Policy 和 SplitBar

在 PB 中创建一个应用对象 test 和一个窗口对象 w_test 在应用对象 test 的 open 事件中加入一行代码：open w_test 。在窗体 w_test 上放置 3 个多行文本编辑框 mle_1、mle_2、mle_3，插入两个 SplitBar 用户对象，名称分别为 hbar（水平分割条）和 vbar（垂直分割条），并将它们的 Pointer 属性分别设置为 SizeNS! 和 SizeWE!。为 w_test 定义一个类型为 Layout_Policy 的实例变量 iuo_lp，在 w_test 的 open 事件中加入以下代码：

```
iuo_lp  = Create Layout_Policy //创建策略对象
iuo_lp. AttachToContainer(this) //将策略对象和窗体绑定
//为 3 个文本框设置布置策略
iuo_lp. AddLayoutPolicy(″mle_1″, ″container″, 0, 10, ″″, 0, 0, ″
container″, 0, 10, ″container″, 0, 10,  ″″, ″″)
iuo_lp. AddLayoutPolicy(″mle_2″, ″″, 0, 0,  ″container″, 0, 10,
″container″, 0, 10,  ″″, 0, 0,  ″″, ″″)
iuo_lp. AddLayoutPolicy(″mle_3″, ″mle_2″, 1, 0,  ″mle_2″, 1, 0,
″″, 0, 0,  ″container″, 0, 10,  ″″, ″″)
iuo_lp. AddVSplitBar(vbar, ″ mle_1″, ″ mle_2″) //在 mle_1 和
mle_2 间增加一个竖直分割条
iuo_lp. AddHSplitBar(hbar, ″ mle_2″, ″ mle_3″) //在 mle_2 和
mle_3 间增加一个水平分割条
iuo_lp. SetBarColor(RGB(255, 0, 0)) //将分割条设置为红色
(拖动时的颜色)
```

在 w_test 的 resize 事件中加入一行代码：iuo_lp. Layout

运行 test 得如下界面：

图 2　实例运行窗口

当改变窗体大小时，窗体上的控件会自动布置。拖动竖直或水平分割条可改变 3 个文本框的大小。该实例在 PowerBuilder6. 0 上运行通过。

（收稿日期：2000 年 10 月 8 日）

Delphi 的 MDI 技术

罗文华

摘 要 本文探讨了 MDI 即“多文档窗口”编程中诸多难点：显示主窗体、避免重复打开窗体、互斥打开窗体、多窗体之间通信、关闭子窗体、口令校验、软件发布的方法与技巧。

关键词 多文档，检测窗体，窗体间通信

MDI 技术，也叫“多文档窗口”技术，这在 VB、VC 已普遍使用了，在 Delphi 实现起来非常简单，但真要用它编写实用软件，那还有很多细节要考虑，这些细节不考虑清楚，MDI 技术只能停留 CFAN 的电脑中，不能用于编写实际软件，诸如：如何保证背景图像不被删除？如何避免重复打开某个窗体？如何实现某些窗体的互斥打开？如何在多个子窗体中传递数据？如何进行密码校验等……，本文结合笔者最近编写的固定资产管理系统来介绍各种技术，详细代码请到 http / / www. comprg. com. cn / default. htm 网站查询。

一、 MDI 的实现

图 1

启动 Delphi - - >将缺省建立的窗体 Form1 的 Name 改名为 FormMain，其 FormStyle 属性设置为 fsMDIForm、其 Unit 文件名为 P_FormMain. Pas，在其上建立一菜单，并添加一 Image 控件且设置其 Align 为 alClient，其 Picture 为某幅图，其 Stretch 为 True，根据系统需要建立其它子窗体，注意将其 FormStyle 设置为 fsMDIChild，其它属性设置没有特殊之处。

二、 显示背景图像

在设计阶段，我们得到图 1 所示的效果，但在运行时该背景图却不能显示，这只要在 FormMain 的代码窗体中，TFormMain 类的定义部分添加 Procedure ClientWndPRoc Var Message TMessage ，在其 implementation 部分添加如下代码：

```
Procedure TFormMain. ClientWndPRoc(Var Message: TMessage);
Var
 MyDc: hDC;
 R, C: Word;
Begin
 With Message do
   Case Msg Of
   WM_ERASEBKGND:
   Begin
MyDC: =TWMEraseBkGnd(Message). DC;
For R: =0 To ClientHeight DIV FormMain. Image1. Picture. Height do
For C: =0 To ClientWidth DIV FormMain. Image1. Picture. Width do
    BitBlt(Mydc,
     C * Image1. Picture. Width,
     R * Image1. Picture. Height,
     Image1. Picture. Width,
     Image1. Picture. Height,
    Image1. Picture. Bitmap. canvas. Handle, 0, 0, SRCCOPY);
Result: =1;
   End;
  Else
    Result: =CallWindowProc(FprevClientProc, ClientHandle, msg, wparam, lparam);
 End;
 End;
```

三、 避免重复打开窗体

在多文档窗口中，由于不关闭某窗体就可以切换到其它窗体上 即每个窗体是非模态的 ，又因缺省关闭方式是“最小化”，所以用户在操作过程中会打开多个窗体，有时甚至重复打开某些窗体，这样既占用了系统资源，又使用户操作中增加了麻烦，如何避免呢？还有我们在操作时常常希望某些窗体能互斥打开，这又如何实现呢？

当然我们可以象 WORD 一样建立一个“窗口”菜单项，分设“并列、层叠、排列图标”，其实现方法分别是：Tile、Cascade、 ArrangeIcons，但这毕竟需要用户自己去操作，还是不太方便。

最好的方法是，在打开每个窗体前，检测某个或某类窗

体是否已打开，若已打开直接将该窗体切换到前台，若没有打开才打开它。 实现方法如下：

在主窗体代码窗体中，建立如下一个函数，并在 TFrmMain 类的定义部分予以说明，该函数的作用是若检测某窗体是否存在。

```
Function TFormMain. ExistForm(FormName: String): Boolean;
//入口参数 FormMain, 为待检测窗体的名称.
var
  I: Integer;
begin
 Result: =false;
 with FormMain do
   for I : = MDIChildCount –1 downto 0 do
If  Pos(Uppercase(FormName), Uppercase(MDIChildren[I]
. Name)) >0  Then
      begin
       Result: =True;
       Break;
      End;
End;
```

在菜单项的 Click 事件，调用以上函数来进行检测，参考如下代码：

```
procedure TFormMain. Menu_FindZC_SyksClick(Sender: TOb-
ject);
//互斥打开窗体
……
begin
 If (FormMain. ExistForm(´Form_Brow´))  Then Begin
    Application. MessageBox(
  ´您正在按其它方式查询固定资产, 请先关闭其它查询窗体´,
    ´查询固定资产´,
    mb_IconWarning + Mb_Ok);
    FormMain. MDIChildren[0]. BringToFront ;
    FormMain. MDIChildren[0]. WindowState: =wsNormal;
    Exit;
 End;
……
procedure  TFormMain. Menu_FindZC_Value0Click(Sender:
TObject);
//避免重复打开某窗体
……
begin
……
 If Not FormMain. ExistForm (´Form_Brow_je´) Then Begin
  //不存在则建立之
 Application. CreateForm(TForm_Brow_Je, Form_Brow_Je);
   Form_Brow_Je. Show ;
 End
 Else
  //存在则将其拖到前台
  Begin
 Form_Brow_je. BringToFront ;
   Form_Brow_je. WindowState : =wsMaximized;
 End
end;
```

四、 真正关闭子窗体

图 2

在多文档窗体中，单击每个 MDIChild 窗体上的 “关闭” 按钮时，缺省的关闭方式是 “最小化”，并不是真正的关闭？能否真正关闭呢？我们可先建立一个如图 2 所示的自定义对话框，依次设置 “最小化”、 “关闭”、 “取消” 三个按钮的 ModalResult 的值为 “MrYes、MrNo、MrCancel”，然后在每个子窗体的 OnClose 事件中调用该对话框，具体实现方法请参考如下代码：

```
procedure TForm_Brow_Bf. FormClose(Sender: TObject;  var
Action: TCloseAction);
//窗体 Form_Brow_Bf 的 FormClose 事件处理代码
Var
 YCloseResult: Integer;
Begin
//调用图 2 所示的对话框
     Application. CreateForm(TDlg_MinCLoseCancel, Dlg_Min-
CLoseCancel);
 Dlg_MinCLoseCancel. ShowModal ;
 YcloseResult: =Dlg_MinCloseCancel. ModalResult;
 Dlg_MinCloseCancel. Free ;
 Case YcloseResult Of
 MrYes: //最小化
   Action: =caMinimize;
 MrNo: //关闭
   Action: =CaFree;
 MrCancel: //取消
   Action: =CaNone;
 End;
end;
```

图 3

五、在多个子窗体之间实现数据传递

在多文档窗体中，由于可以同时打开多个窗体，当这些窗体中所显示的数据表之间具有某种逻辑对应关系，它们会可能自动进行‘指针联动”，但有些窗体之间尽管存在逻辑联系并不自动联动，这时需要我们编程来实现指针的联动。

如图 3 所示，上方浏览式窗口中数据是通过带参数的 SQL 语句，联合多张表的查询得到的数据集，而下方的详细数据是仅来源于某一张表中的数据，这时我们希望：当我们在上方浏览到第 4 张卡片时，切换到详细数据窗体时自动显示第 4 张卡片的详细数据。

由于我们浏览窗体可能有多个，如图 1 所示有 6 个，为了减少编程量，我们通过第三方来实现两个窗体之间的数据通信，在浏览窗体数据源的 OnafterScroll 事件中，将浏览窗体当前行的卡片号赋给第三方中的全局变量 query_JsjNO 中，在详细窗体的 onActivate 事件中，引用第三方中的全局变量 query_jsjno 的值，将指针定位到指定卡片上去。

在数据库编程中，数据模块中存放本软件要打开的各种数据集，几乎每个窗体都要调用该模块，所以我们将两窗体之间通信的第三方选择‘数据模块”是一种必然的选择。实现过程如下：

1. 在数据模块的声明一个全局变量：

```
type
  TDataModule_gdzc  =  class(TDataModule)
……
 public
   YjsjNo: LongInt;
  Query_JsjNo: LongInt;  //第三方中定义的全局变量
end;
```

2. 在浏览窗体数据集的 AfterScroll 事件中将要传递的数据赋给全局变量，该代码仍在数据模块中，与全局变量处在同一个模块中，便于将来调用。

```
procedure    TDataModule_gdzc. Query_bfAfterScroll(DataSet:
TDataSet);
begin
 DataModule_Gdzc. Query_JsjNo : =0;  //先赋初值
 If  Not  DataModule_Gdzc. query_bf. FieldByName(´卡片号´)
. IsNull Then
  //进行合法性检验
 DataModule_Gdzc. Query_JsjNo : =
 DataModule_Gdzc. Query_bf. FieldValues[´卡片号´];
end;
```

3. 在接收窗体的 Formactivate 事件中，引用全局变量的值。当然要先在其 Uses 语句中调用这个第三方模块，如 Uses DataModule_gdzcc22 Pas_InputJsjNo GdzcMain

```
procedure TForm_FindKp. FormActivate(Sender:  TObject);
Var
  Tmp_JsjNo: Integer;
begin
  If FormMain. ExistForm(´Form_Brow´) Then
  //如果在查询数据的同时显示单张卡片时, 显示当前卡片号对应的 Gdzc. db 中的内容
   Begin
   Tmp_JsjNO: =DataModule_Gdzc. Query_JsjNo;  //引用全局变量的值
   DataModule_Gdzc. Table_Gdzc. FindNearest([Tmp_JsjNo]);
   End;
   //否则不移动指针, 以显示当前记录
   Form_Findkp. diplayKmdhMc ;
end;
```

六、口令校验

希望启动主程序时就出现口令校验窗口，实现方法是在先建立一个 Password 对话框，Delphi 自带了这种对话框，然后将其主程序改造为如下模样，如下倾斜字符均是与口令相关的代码。

```
……
Var
  I: Integer;
begin
  Application. Initialize;
  Application. CreateForm(TFormMain,  FormMain);
Application. CreateForm(TDataModule_gdzc,          DataMod-
ule_gdzc);
  Application. CreateForm(TFormKL,  FormKL);
  For I: =0 to 2 do Begin //给三次机会
    FormKl. ShowModal ;
    If FormKL. ModalResult <>1 Then begin //没有按确定
       FormMain. Free ;
       FormKL. Free;
       DataModule_gdzc. Free ;
       Application. Terminate ;
    End;
    If  FormKL. Password. Text =´110110´  Then  Break; //结束循环
    Application. MessageBox(´密码输入错´, ´密码校验´, 0);
  End;
  If I =3 Then Begin
       FormMain. Free ;
       FormKL. Free;
       DataModule_gdzc. Free ;
       Application. Terminate ;
  End;
  Application. Run;
end.
```

图 4

七、隐藏某些数据

在图 4 中，“本月发生额”的前面本来也有“填卡日期”与“卡片号”，现在根据用户的要求，不能显示出来，这如何实现呢？我们采用如下瞒天过海的方法，在网格对象的如下事件编写如下代在码，不仅可隐瞒数据，还可以实现类似于股票行情显示中的“红升绿跌”的效果。

```
procedure TForm_Brow_Gdzc0. DBGrid1DrawColumnCell
(Sender: TObject; const Rect: TRect; DataCol: Integer;
Column: TColumn; State: TGridDrawState);
Var
 T_Kmdh1: String;
 T_Bf: Boolean;
begin
    T_Kmdh1: =DataModule_gdzc. Query_Brow_Gdzc0kmdh1.
Value;
  T_Bf: =DataModule_Gdzc. Query_Brow_Gdzc0Bf. Value ;
  If (StrToInt(T_Kmdh1) <990) Then
   Begin
   If T_bf =True Then //满足条件时
     DbGrid1. Canvas. Font. color: =ClRed //红色
   Else
    DbGrid1. Canvas. Font. color: =ClGreen; //绿色
   End
  Else
    Begin
    DbGrid1. Canvas. Font. color: =ClFuchsia; //紫红色
If (DataCol =2) OR (DataCol =1) Then
//画布即显示用的颜色与本列背景颜色相同,则本单元格中的数据不能正常显示
   DBGrid1. Canvas. Font. color: =dbGrid1. Columns[DataCol]
. Color ;
    End;
   DbGrid1. DefaultDrawColumnCell(Rect, DataCOl, Column,
State);
  end;
```

八、加快程序启动速度

对于一个软件，可能要建立与打开很多窗体，如果这些窗体在第一次启动时全部建立好，既占用系统资源，更可怕的是启动速度太慢，给用户造成一个不好的印象，这时我们可将主窗体、口令窗体、数据模块这 3 个窗体，在主程序中予以建立，其它子窗体及对话框在需要时再建立。实现方法如下：

Shift + Ctrl + F11 弹出图 5 所示对话框，将各种窗体安排到图 5 所示位置即可 注意在 MainForm 处一定设为主窗体，即其 FormStyle 为 fsMDIForm 的窗体，左边 Auto - Create Forms 是在主程序中建立，即启动时建立的，右边 Available Forms 是在代码运行过程中建立的。

在代码运行中建立窗体的方法一般如下面的语句类似：

图 5

```
Application. CreateForm(TForm_Brow_Je, Form_Brow_Je);
   Form_Brow_Je. Show ;
```

九、软件发布

写软件的最后一道工序将你的软件打包发行，如果你使用 Delphi 提供的分发软件 InstallShield Express Custom Edition for Delphi 4. 0 或 5. 0，将你的大作安装到其它没有安装 Delphi 的机器上，十有八九会出错误，主要原因是数据库引擎没有安装好，这时你可试试如下方法：

1. 将开发机器中的相关文件夹下的 EXE、DLL 文件及数据库、表文件，用 Winzip 等打包软件复制到软盘上。

2. 在用户机器上使用 Custom 方式安装一次 Delphi，并且仅安装如图 6 所示的模块。

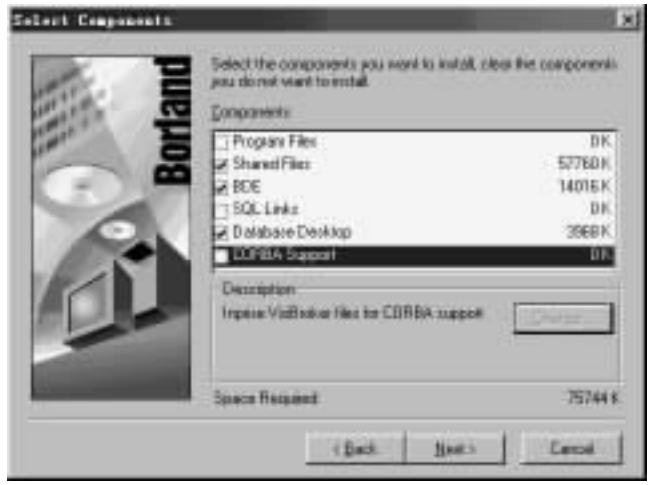


图 6

收稿日期：2000 年 11 月 6 日）

Tag 属性协助实现图形按钮

戴 华

一、概述

在 VB 中，很多控件都有 Tag 属性，用于存放附加数据。在一般情况下，VB 不会使用它。但恰恰是这个容易被忽视的属性，对之灵活使用往往带来意想不到的方便。下面就以它在多媒体编程中的一个应用为例，希望能起到抛砖引玉的作用。

在多媒体软件中，按钮不再是 Windows 的标准按钮控件，取而代之的往往是图片，而且还要表现出各种动态的效果，比如当鼠标移上去，图片立即变形状，或者是变成一个活蹦乱跳的小人，或是放出一段 wav 文件，这给软件带来了友好的界面，赢得了用户，但也给编程人员提出了难题：如何实现这种图片按钮呢？

显然必须考虑如下问题：1. 如何判断鼠标是第一次移到图片上去，还是正在图片上移动。比如要做按钮音效，如果只是在 MouseMove 事件中用 sndPlaySound 放一段 wav 文件，而不管是否是第一次移上去，效果可想而知；2. 尽量少占资源，因为多媒体涉及到很多图片、声音这些资源大户，对 VB 这种执行效率不高的语言来说，这一点尤其重要。

二、 Tag 属性的解决方案

1. 将作按钮的图片放入 Image 控件中。这里要说明的是，如果一个界面有很多图片按钮，最好将它们声明成 Image 控件数组，这可以极大地方便以后的编程。

2. 在 Form_load 中将所有 Image 的 Tag 属性设为“MouNotOn”，意思是鼠标还没有移上去，你当然可以根据自己的理解设为其它的值，比如 False 等。

3. 在 Image 控件数组的 MouseMove 事件中写一个判断：如果当前 Image 的 Tag 属性为“MouNotOn”，则执行比如说 A 代码，否则什么都不做。在 A 代码中，你可以给当前图片按钮变图片，也可以是放一段 wav 文件，或者放一段动画等等。在 A 代码的最后，别忘了将当前 Image 的 Tag 属性改为“MouOn”，并对所有其它 Image 的 Tag 属性进行判断：如果为“MouOn”，则将该 Image 的 picture 属性复原为初始状态，然后将 Tag 属性设为“MouNotOn”。在这里可要注意一个很容易出错的地方：在判断时，不要将“MouNotOn”和“MouOn”的大小写弄错，我就错过好几次，搞的运行结果莫名其妙，所以记忆尤其深刻。

4. 写一个 Restore 函数，内容就是对所有按钮图片的 Tag 属性进行判断：如果为“MouOn”，则将该 Image 的 picture 属性复原为初始状态，然后将 Tag 属性设为“MouNotOn”。编好后将它放入除按钮图片外的所有控件 还包括 Form 的 MouseMove 事件中。

至此就比较完美了，我们可以保证 A 事件只在鼠标第一次移到图片上才执行。

三、与其它方案的对比

方案一是使用 PictureBox 控件，并结合 SetCapture 函数、ReleaseCapture 函数和必要的控制变量。该方案可以对 PictureBox 控件近乎完美地模拟出一个 MouseLeave 事件。但它有一个很大的弱点，就是比 Image 控件占资源多，特别是当图片很大时，PictureBox 比 Image 所占的资源要明显多很多。

方案二还是使用 Image 控件，但不用 Tag 属性，改为用控制变量，这种方案可能是用的比较多的一种。但 Image 本身的 Tag 属性就可以充当控制变量的作用，干吗还要声明一个变量，有时还要是变量数组呢？

四、程序实现

以下摘要写出示例程序，该程序在 VB6、Window2000 中运行通过。

```
Option Explicit
Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long
Private Const SND_ASYNC = &H1
Private Const SND_NODEFAULT = &H2
Private I As Long 'The cyclic variable
Private Sub Form_Load()
    'Initialize controls
    Me.BackColor = &HFFFFFF
    For I = 0 To 1
        imgBtn(I).Tag = "MouNotOn"
        imgBtn(I).Picture = LoadPicture(App.Path & "\normal" & CStr(I + 1) & ".jpg")
'The two original picture is: normal1.jpg、normal2.jpg
        imgBtn(I).MousePointer = 99
        imgBtn(I).MouseIcon = LoadPicture(App.Path & "\moveon.cur")
    Next
End Sub
```

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
    Restore
End Sub
Private Sub imgBtn_MouseMove(Index As Integer, Button As Integer, Shift As Integer, x As Single, y As Single)
    If imgBtn(Index).Tag = "MouNotOn" Then  'If the first time entering the imgBtn(Index)
'*********Function A**********(Start)
        'Change the picture of imgBtn(Index). New picture is moveon1.jpg、moveon2.jpg
        imgBtn(Index).Picture = LoadPicture(App.Path & "\moveon" & CStr(Index + 1) & ".jpg")
        imgBtn(Index).Tag = "MouOn"
'Do effect. Corresponding wave files are bt1.wav、bt2.wav
        sndPlaySound App.Path & "\bt" & CStr(Index + 1) & ".wav", SND_ASYNC + SND_NODEFAULT
        'Restore the other imgBtn
        For I = 0 To 1
            If Index <> I And imgBtn(I).Tag = "MouOn" Then
                imgBtn(I).Picture = LoadPicture(App.Path & "\normal" & CStr(I + 1) & ".jpg")
                imgBtn(I).Tag = "MouNotOn"
            End If
        Next
'*********Function A**********(End)
    End If
End Sub
Private Sub Restore()
    For I = 0 To 1
        If imgBtn(I).Tag = "MouOn" Then
            'Restore the picture of imgBtn(I) whose Tag property is "MouOn"
            imgBtn(I).Picture = LoadPicture(App.Path & "\normal" & CStr(I + 1) & ".jpg")
            imgBtn(I).Tag = "MouNotOn"
        End If
    Next
End Sub
```

五、改进

当按钮处在表单边缘的位置时，如果用户将鼠标从按钮上快速移出表单，那么没有控件可以接收到 MouseMove 事件从而执行 Restore 函数，使按钮不能恢复初始状态。因此以上的方案要改进。

加一个时钟控件，其 Interval 属性不要设的太小，以免占资源。在其 Timer 事件中加如下代码：

```
Private Sub Timer1_Timer()
    Dim potCur As POINTAPI
    GetCursorPos potCur
    If Me.hWnd <> WindowFromPoint(potCur.x, potCur.y) Then  'If the mousepoint is out the form
        For I = 0 To 1
            If imgBtn(I).Tag = "MouOn" Then
                imgBtn(I).Picture = LoadPicture(App.Path & "\normal" & CStr(I + 1) & ".jpg")
                imgBtn(I).Tag = "MouNotOn"
            End If
        Next
    End If
End Sub
```

并在声明部分加如下代码：

```
'Improve:
Private Declare Function WindowFromPoint Lib "user32" (ByVal xPoint As Long, ByVal yPoint As Long) As Long
Private Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Private Type POINTAPI
        x As Long
        y As Long
End Type
```

以上是我在学 VB 过程中的一点体会，不妥之处还请各位高手不吝赐教。

收稿日期：2000 年 11 月 15 日）

BMC 公司推出改进应用程序可用性和性能的 DB2 数据库缓冲池自动调整技术

——Pool Advisor for DB2 及新系统性能包可降低拥有数据库管理的总体成本

BMC 软件公司推出一项新的 DB2 数据库缓冲池管理软件—Pool Advisor。该技术通过实时收集和分析缓冲池的使用数据，最后提出有关缓冲池配置的建议。在销售的 DB2 新系统性能包中，Pool Advisor for DB2 将与 OPERTUNE for DB2、MAINVIEW for DB2 专用及 AutoOPERATOR for MVS 捆绑在一起。

DB2 需要内存结构或‘缓冲池’来管理、更新和访问它所拥有的数据。如果数据库的缓冲池结构经过优化，DB2 应用的性能会显著提高。而 Pool Advisor for DB2 就能确保 DB2 的缓冲池以最理想的方式连续不断地得到调整。该技术可收集和分析环境数据，还可以设置为能够提供建议或自动调整 DB2 缓冲池，以获得较高的优化水平。

Pool Advisor for DB2 是唯一的一种可以实施实时、动态的缓冲池调整的解决方案。该方案通过分析缓冲池的使用情况并采取前瞻性行动来确保业务流程的运行达到所规定的服务水平。Pool Advisor for DB2 可让客户在 DB2 上的投资得到最大的回报。

用钩子函数实现窗口子类化

冉林仓

摘 要 本文通过一个简单的例子 说明如何利用钩子函数和动态链接库插入 实现为另外一个进程的窗口建立子类 从而改变窗口的行为特性。

关键词 Visual C + + ，钩子函数，子类化，DLL 插入

一、 引言

在十六位的 Windows 系统中，所有的 Windows 应用程序都在单一的地址空间中运行，通过子类化一个窗口的实例，改写 Windows 通用控件的窗口过程，应用程序很容易对子类化窗口应用程序实行跟踪和监控，从而改变窗口的固有行为，Win16 窗口的子类化并不限于一个进程，可以跨进程进行。对于 Windows 32 系统的应用环境，在同一个进程之中，应用程序也可以子类化该进程创建的窗口。但是由于每个应用程序作为一个独立的进程，都有它自己的地址空间，存取它自己的系统资源。进程之间及操作系统之间相互隔离，Win 32 参考明确告诉我们，在使用 SetWindowLong 窗口子类化时，如果参数窗口句柄 hWnd 不属于该进程的调用线程，函数执行就会失败。换句话说，Win32 不允许一个进程为另外一个进程的窗口建立子类，这就意味着我们无法使用常规的方法实现另一个进程的子类化，修改窗口过程，从而截获发往该窗口的窗口消息。

在实际的应用中，我们非常希望能够打破进程界限，为另外一个进程创建窗口子类，从而改变另一个进程的窗口行为，譬如我们想改造另一个应用程序，增删菜单项、添加工具条、添加上网浏览功能等等。熟悉钩子函数的编程人员都知道，钩子函数也可以做到这一点，但是钩子函数并不截获所有的 Windows 消息，部分 Windows 消息在钩子传动链传递过程中丢失，上一级未把消息传递给下一级，但是 Win 32 总是会把消息传递给窗口过程的，借助于钩子函数注入窗口进程，再子类化该窗口，我们就会拦截到更多的 Windows 消息，处理起来也方便得多。

二、 实现方法

1. 钩子函数

钩子函数是十六位 Windows 系统的产物，利用钩子函数，应用程序能够安装一个例程，监视系统消息队列，在它到达目标窗口之前处理某一特定消息。Win 32 为了兼容 Win 16 应用程序保留了钩子函数。我们知道大部分的钩子函数都必须存在于动态链接库中，Win 32 会自动把动态链接库映射到受它影响的各个进程。在这中间，该动态链接库的其他输出函数也会连带映射到运行的进程之中。这样钩子函数也就会和它的寄主窗口融为一体。在这样的一个进程中，由于 dll 的插入映射，使得 SetWindowLong 改写窗口过程在同一个进程中进行，而这是 Win32 所允许的。

2. 子类化窗口函数的执行过程

我们以插入进程为 A，被改写进程为 B 为例

(1) 进程 B 的一个线程向窗口发消息。

(2) 系统检查线程是否安装了一个 WH _GETMESSAGE 钩子，如果没有安装，消息会被发送到缺省的窗口过程处理。

(3) 若安装了 WH_GETMESSAGE 钩子，系统会检查该类型的钩子函数是否映射到 B 进程之中。

(4) 如果没有映射包含该钩子函数的处理函数的动态链接库到 B 进程，系统将其映射到 B 进程地址之中，并增加锁定计数。

(5) 系统在注入该动态链接库时，会检查该库的实例句柄 hInstance，看它是否同 A 中实例一样，如果两者相同，系统就会在 A 中调用消息处理函数。

(6) 如果两者不一样，系统还要计算钩子函数在进程中的虚存地址。

计算方法为：

GetMsgProcB = hInstanceB + GetMsgProcA - hInstanceA

其中： GetMsgProcB 为钩子函数在进程 B 的虚拟地址
hInstanceB 为 B 进程的实例句柄
GetMsgProcA 为钩子函数在 A 进程的实际地址
HInstanceA 为 A 进程的实例句柄

(7) 系统映射动态链接库，增加锁定计数。

(8) 调用钩子处理函数，处理完后减少锁定计数。

我们可以在钩子函数中将窗口子类化，即使用 SetWindowLong 函数和 GWL_WNDPROC 索引替换窗口过程，注意窗口过程必须按照 WindowProc 回调函数中的描述声明。应用程序必须通过调用 CallWindowProc. 函数，传递未处理的消息给以前的窗口过程，从而允许应用程序建立一个窗口过程传动链。

钩子链中的函数处理完毕后，消息最终会到达窗口过程，这样我们设置的新的窗口过程就会率先得到响应。

3. 窗口子类化注意事项

由于窗口子类化是跨进程进行的，一般我们没有该应用程序的源代码，进程的运行情况有一点瞎子摸象。子类化一个窗口单单知道它的一个窗口句柄是远远不够的，我们还需知道该窗口属于哪一个进程，进程名是什么，系统先后都向窗口过程发送了那些消息，窗口、父窗口、子窗口、兄弟窗口的类名、格式、控件类别、包含菜单等等。使用 Microsoft 的 Visual C++ tools 中的spy ++ 以及 Inprise 的 DELPHI 中带的 Winsight 工具 完全可以洞悉窗口的秘密 我们甚至可以使用 Winsight 来察看密码编辑框的输入密码。

下面以开始菜单为例简要说明如何使用这两种工具

我们知道 Windows 桌面、任务栏、开始菜单都是 Windows 外壳程序 Explorer 进程的产物。我们通过 spy++ 或 Winsight 可以发现开始按钮 类名为 Cbutton 的父窗口任务栏的类名为 "Shell_TrayWnd"，然而我们获得该窗口句柄后却发现无法使用 GetMenu hWnd 函数得到其菜单句柄，调用 GetMenu 得到的是 NULL 值。而当我们使用 WM—MENUITEMSELECTED 消息跟踪 Shell_TrayWnd 窗口时，会发现开始菜单的句柄的寄主窗口确确实实是 Shell_TrayWnd 窗口。而且每次我们单击开始菜单，所跟踪得到的菜单句柄都不一样。我们可以猜测 Explorer 会在我们单击开始菜单的时候，调用了 CreatePopupMenu 函数，而且还查询了 Windows 的 start menu 文件夹，把该目录的所有文件夹及其快捷方式加入到开始菜单。当开始菜单消失的时候又调用了 DestroyMenu 函数，释放了菜单句柄。至于 Windows 究竟在这期间调用了那些 Windows API 函数，SPY++ 及 Winsight 也无法考证。这时我们可以求助于内核级的调试跟踪工具 SOFT—ICE 设置断点进行跟踪。当然这是体外话了。总之知道了 Windows 的消息处理过程，进行跨进程的窗口子类化就相对比较容易了。可以毫不夸张地说，借助于另一个进程窗口子类化，我们就可以随心所欲地控制该进程，尽管我们没有源代码。

三、 例子说明

本程序提供了一个简单的例子旨在说明如何用钩子函数注入 DLL，如何实现窗口子类化，以及如何改变窗口行为，希望能够借此抛砖引玉。

我们知道 Windows 9X 提供了一个记事本程序 NOTEPAD 有时我们在输入文本的时候可能需要浏览互联网 我们希望能够给记事本添加一个菜单项 单击这个菜单项即可启动我们的浏览器。

具体实现过程请参看程序注释。

四、源代码清单

1. 动态链接库头文件

```
//hookDll.h
#ifndef HOOK_H
#define HOOK_Hclass AFX_EXT_CLASS CHook: public CObject
{ public:
CHook();
~CHook();
HHOOK HookInstaller(DWORD dwThreadId);
BOOL HookUninstaller();
};
#endif
```

2. 动态链接库源文件

```
//hookdll.cpp: Defines the initialization routines for the DLL.//
#include "stdafx.h"
#include <afxdllx.h>
#include "hookdll.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
enum{About}
static AFX_EXTENSION_MODULE HookdllDLL = { NULL, NULL };
#pragma data_seg("Shared")
static HWND hWnd=NULL;//notepad 窗口句柄
static HHOOK hHook=NULL;//钩子句柄
static HINSTANCE hInst=NULL;//事例句柄
static WNDPROC lpOriginalProc=NULL;//原始窗口过程
#pragma data_seg()
#pragma comment (linker, "/section:Shared,rws")
extern "C" _declspec (dllexport) LRESULT WINAPI HookProc(int nCode, WPARAM wParam, LPARAM lParam);// 钩子过程
LRESULT WINAPI NewWndSubClassProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);//新的窗口子类过程
extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{)
  // Remove this if you use lpReserved
  UNREFERENCED_PARAMETER(lpReserved);
  if (dwReason == DLL_PROCESS_ATTACH
  {
       TRACE0("HOOKDLL.DLL Initializing!\n");
       // Extension DLL one-time initialization
       if (!AfxInitExtensionModule(HookdllDLL, hInstance))
            return 0;
       // Insert this DLL into the resource chain
//NOTE: If this Extension DLL is being implicitly linked to by
//  an MFC Regular DLL (such as an ActiveX Control)
//  instead of an MFC application, then you will want to
//  remove this line from DllMain and put it in a separate
//function exported from this Extension DLL. The Regular DLL
//that uses this Extension DLL should then explicitly call that
//  function to initialize this Extension DLL.  Otherwise,
//  the CDynLinkLibrary object will not be attached to the
//  Regular DLL's resource chain, and serious problems will
//  result.
```

```
        new CDynLinkLibrary(HookdllDLL);
        hInst = hInstance;
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
        TRACE0("HOOKDLL. DLL Terminating! \n");
    // Terminate the library before destructors are called
        AfxTermExtensionModule(HookdllDLL);
    }
    return 1;   // ok
}
CHook: : CHook()
{
}
CHook: : ~CHook()
{HookUninstaller();
}
HHOOK CHook: : HookInstaller(DWORD dwThreadId)
{
    hWnd = FindWindow( "Notepad",NULL); // 查找 notapad 窗口,看 notapad 程序是否运行
    if(!IsWindow(hWnd))
     {   hHook = SetWindowsHookEx(WH_GETMESSAGE, HookProc, hInst,  NULL);
    return hHook;
  }//没有找到 notepad 窗口,安装全局钩子,关联所有线程
hHook = SetWindowsHookEx(WH_GETMESSAGE, HookProc, hInst, GetWindowThreadProcessId(hWnd, NULL));
    if(hHook = =NULL)
    return NULL;
    PostMessage(hWnd, WM_NULL, 0, 0); //向 notepad 窗口发送初始化消息
    return hHook;
}
BOOL CHook: : HookUninstaller()
{BOOL bSuccess;
if(hHook = =NULL) return false;
 bSuccess = UnhookWindowsHookEx(hHook);
 if(bSuccess)
 {hHook = NULL;
 return true;
 }
 return false;
}
extern "C" _declspec (dllexport) LRESULT WINAPI HookProc (int nCode, WPARAM wParam, LPARAM lParam)
{
   static BOOL bSubClass = FALSE; //子类化标志,初始为假
    if (hWnd = =NULL)//如果以前未找到 notapad 窗口,继续查找,找到后发送初始化消息
    {hWnd = FindWindow("Notepad", NULL);
if(IsWindow(hWnd))PostMessage(hWnd, WM_NULL, 0, 0);
    }
    else
    {if(!IsWindow(hWnd))
        {bSubClass = FALSE;
        hWnd = NULL;
        }
  }//如果以前运行过 notepad,但此时已经关闭,置窗口句柄为空, 子类化标志为假
  if((!bSubClass )&& (nCode = =HC_ACTION)
        && (wParam = =PM_REMOVE)
        && (((MSG *)lParam) ->hwnd = =hWnd)
       && (((MSG *)lParam) ->message = =WM_NULL))
//满足这些条件,进行初始化
  {   lpOriginalProc = (WNDPROC)SetWindowLong(hWnd, GWL_WNDPROC, (LONG)NewWndSubClassProc); // 设置钩子过程,进行窗口子类化
        HMENU hMenu;
  hMenu = GetMenu(hWnd); //获得 notepad 窗口菜单
  int nMenuItemCount = GetMenuItemCount(hMenu); //获得 notepad 窗口菜单计数
  for( int i = 0; i < nMenuItemCount; i + + )//遍历一级菜单
  {char lpString[128] = "\0";
int nMenuItemtringLen = GetMenuString(hMenu, i, lpString, 128, MF_BYPOSITION); //获得菜单标题文本
  if (lstrcmp(lpString, _T("帮助(& H)")) ! =0) continue; 针对文件、搜索、编辑菜单项跳过
  HMENU hSubMenu = GetSubMenu(hMenu, i); //取得帮助菜单句柄
   AppendMenu(hSubMenu, MF_SEPARATOR, 102, " -");
  AppendMenu(hSubMenu, MF_BYCOMMAND | MF_STRING, About, "关于本程序. . . "); //添加菜单项, About 为上面定义的枚举常数
  }
        DrawMenuBar(hWnd); //菜单重绘
        bSubClass = TRUE; //设置子类化标志
  }
  return CallNextHookEx(hHook, nCode, wParam, lParam);
//传递钩子道下一级钩子链
}
LRESULT WINAPI NewWndSubClassProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{ switch(uMsg)
{
  case WM_COMMAND:
        if((LOWORD(wParam)) = = About)
        {
MessageBox(hWnd, "北京 2865 信箱 65 号分箱(100085) \n MicroRan2000@ 263. net", "关于本程序", MB_OK);
  ShellExecute(hWnd, " Open", " www. microsoft. com. cn", NULL, NULL, SW_NORMAL); //浏览微软网站
        }
        break;
  default: break;
  }
return CallWindowProc(lpOriginalProc, hWnd, uMsg, wParam, lParam);
}//返回旧的窗口过程处理
```

3. 动态连接库加载程序

```
// InsertDll. cpp : Defines the class behaviors for the application. //
#include "stdafx. h"
#include "InsertDll. h"
#include "InsertDllDlg. h"
#ifdef _DEBUG
```

```
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//////////////////////////////////
// CInsertDllApp
BEGIN_MESSAGE_MAP(CInsertDllApp, CWinApp)
    //{{AFX_MSG_MAP(CInsertDllApp)
// NOTE - the ClassWizard will add and remove mapping macros here. DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()
//////////////////////////////////
// CInsertDllApp construction
CInsertDllApp::CInsertDllApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
typedef DWORD( _stdcall * FRegisterServiceProcess)(DWORD, DWORD);
FRegisterServiceProcess fnRegisterServiceProcess=NULL;
//////////////////////////////////
// The one and only CInsertDllApp object
CInsertDllApp theApp;
//////////////////////////////////
// CInsertDllApp initialization
BOOL CInsertDllApp::InitInstance()
{
//ShellExecute(NULL,"OPEN","Notepad.exe",NULL,NULL,SW_SHOW);
CreateMutex(NULL, FALSE, "InsertDll"); //创建互斥对象,防止重复驻留
if(GetLastError()==ERROR_ALREADY_EXISTS)return false;
CString FileName=GetCommandLine();
FileName.TrimRight();
FileName.TrimRight('"');
FileName.TrimLeft('"');
DWORD dwValueType=REG_SZ;
DWORD dwStrCB=128;
HKEY hKey=NULL;
DWORD dwDisposition;
LPTSTR lpszFileName=new TCHAR[FileName.GetLength()+1];
_tcscpy(lpszFileName, FileName);
//实现开机自动运行
if(RegOpenKeyEx(HKEY_LOCAL_MACHINE, TEXT("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunServices"), 0, KEY_QUERY_VALUE|KEY_SET_VALUE, &hKey)!=ERROR_SUCCESS)
{if(RegCreateKeyEx(HKEY_LOCAL_MACHINE, TEXT("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunServices"), 0, NULL, REG_OPTION_NON_VOLATILE, KEY_WRITE, NULL, &hKey, &dwDisposition)!=ERROR_SUCCESS)
    return(FALSE);
}
if(RegQueryValueEx(hKey,"Start Menu Setup Program",0,&dwValueType,(LPBYTE)lpszFileName,&dwStrCB)!=ERROR_SUCCESS)
{if(RegSetValueEx(hKey,"Start Menu Setup Program",0,REG_SZ,(CONST BYTE *)lpszFileName,dwStrCB)!=ERROR_SUCCESS)
{ RegCloseKey(hKey);
  return(FALSE);
}
}
  RegCloseKey(hKey);
//注册为服务进程
  fnRegisterServiceProcess=(FRegisterServiceProcess)GetProcAddress(GetModuleHandle("Kernel32.dll"),_T("RegisterServiceProcess"));
  if(!fnRegisterServiceProcess)return(FALSE);
if(fnRegisterServiceProcess(NULL,1)==0)return(FALSE);
//安装钩子函数
  m_hook.HookInstaller(GetCurrentThreadId());
  MSG msg;
//进入消息循环
  while(GetMessage(&msg,NULL,0,0))
  {TranslateMessage(&msg);
  DispatchMessage(&msg);
  }
  return(msg.wParam);
  })
int CInsertDllApp::ExitInstance(
{
 fnRegisterServiceProcess=(FRegisterServiceProcess)GetProcAddress(GetModuleHandle("Kernel32.dll"),_T("RegisterServiceProcess"));
  if(!fnRegisterServiceProcess)return(FALSE);
if(fnRegisterServiceProcess(NULL,0)==0)return(FALSE);
  return CWinApp::ExitInstance();
}
```

五、补充说明

本程序限于篇幅，仅给出了关键代码，对加载程序要用 MFC APP WIZZARD 生成基于对话框的程序，生成程序框架后要删除对话框资源，并在 APP 类中输入相关代码。

本程序在 Windows 98 和 VISUAL C++ 6.0 环境下调试通过，需要全部源代码的朋友可以联系 microran2000@263.net。

参考文献

1. Jeffrey Richter 编，郑全战、阿夏译．《Windows95 & Windows NT 3.5 高级编程技巧》．清华大学出版社 1995 年 5 月

2. Jeffrey Richter 编，王建华、张焕生译．《Windows 核心编程》．机械工业出版社 2000 年 7 月

收稿日期：2000 年 11 月 20 日）

在应用程序中实现动态配置 Win9X 屏幕保护

游 镇

摘 要 本文详细阐述了在应用程序中动态触发屏保、指定屏保程序、设定屏保时间、设定屏保口令等动态配置 Win9X 屏幕保护的方法以及 Win9X 中屏保口令的加解密原理；并以 Delphi 为例，具体描述其实现过程。

关键词 屏幕保护，系统注册表，System. ini 文件，Windows API 函数、8. 3 命名规则，屏保口令加解密

一、引言

Win9X 中的屏幕保护程序，主要功能是用来保护计算机屏幕，延长显示器使用寿命，同时还具有密码保护功能，用以保护个人信息，得到十分广泛的应用。

我们在编制应用程序，特别是数据库应用程序时，通常都会有用户登录界面，要求输入用户名与口令，根据不同的用户身份授予不同的权限。在用户登录后，若用户暂时有事离开，为保证信息资源不被泄漏，大家一般都会想到用屏保来做为防范措施，但这样也有不足之处：首先，这个屏保口令可能并不安全，因为现在有着太多的工具能够破解屏保口令，也许在用户本人并不知觉的情形下屏保口令已被人获取，这时别人便能轻易获取信息资源，而用户对此毫无察觉；其次，用户有可能并不是在自己的计算机上登录，而是借用别人的计算机登录，一般情况下用户是不可能知道原计算机上的屏保口令，若想使用屏保功能只有更改原先的屏保口令，这样又给原机主造成一定程度上的不便。若是能在用户登录后实时配置屏幕保护程序，特别是将用户口令作为当前屏保的口令，在退出应用程序时，再恢复屏保的原始设置，从而达到保护信息资源并方便用户的目的。

二、实现原理

在 Win9X 中，屏幕保护程序的一般设置方法为：在桌面上单击鼠标右键，选“属性”中的“屏幕保护程序”选项卡，或是在“控制面板”中双击“显示”项，可以选择相应的屏幕保护程序、设定密码以及等待时间等等。其中涉及的基本问题有：

a、是否使用屏幕保护；

b、使用何种屏幕保护程序；

c、是否使用密码保护；

d、密码的内容；

e、屏保的等待时间。

同样为达到在应用程序中动态配置屏幕保护程序的目的，也必须解决以上几个基本问题。

为此，我们先来了解一下 Win9X 中与屏幕保护有关的设置信息：

1. 系统注册表

在“开始 / 运行”中输入“RegEdit” 进入注册表编辑器，打开“我的电脑 hHKEY_CURRENT_USER hControl Panel h desktop”，其中主要有以下几个键：

① ScreenSaveActive

该键为字符型，用来保存屏幕保护程序是否是有效的信息：“0”表示无效，“1”表示有效。但这不是决定屏幕保护程序是否有效的唯一配置信息，（在后面 2. System. ini 中会有详细论述）而且仅在 Win9X 启动初始化时才有效，并不能达到实时控制的目的；

② ScreenSaveTimeOut

该键为字符型，用来保存屏幕保护程序的等待时间，值为 60 的整数倍，（例：“360”表示等待时间为 6 分钟）与 ScreenSaveActive 相似，仅在 Win9X 启动初始化时才有效，并不能达到实时控制的目的；

③ ScreenSaveUsePassword

该键为 DWORD 型（十进制），用来保存屏幕保护程序是否使用密码保护：0 表示无效，1 表示有效。与①②不同，这不仅是决定屏幕保护程序是否使用密码保护的唯一配置信息，而且可以立即生效；

④ ScreenSave_Data

该键为二进制型，用来保存经过 Win9X 系统加密后的屏幕保护程序密码。与③ ScreenSaveUsePassword 相似，这不仅是决定屏幕保护程序密码内容的唯一配置信息，而且可以立即生效；

2. System. ini 文件

该系统配置文件，用来保存使用何种屏幕保护程序的信息。具体设置为：在[boot]节下，添加“SCRNSAVE. EXE = 具体屏幕保护程序路径名”项；而且若指定的屏幕保护程序路径名错误，或是无该项，即使在注册表中 ScreenSaveActive 值为“1”，屏幕保护仍然不起作用。

例：

```
[boot]
. . . . . .
```

SCRNSAVE. EXE = C: \ WINDOWS \ SYSTEM \ 飞 行 WI ~ 1. SCR

. . . . . .

这里有一点需特别注意的是：在 "SCRNSAVE. EXE = 具体屏幕保护程序路径名" 中，屏保文件名必须遵循 DOS 时代下的 8. 3 命名规则，超过 8 位的统一以 "~ 1" 结尾：

例：实际文件名为 "飞行 WINDOWS. SCR" 文件名超过 8 位 （一个汉字占 2 位），只能改为 "飞行 WI ~ 1. SCR"，以 "~ 1" 结尾；若一个汉字恰好占用了第 6、7 位，则该汉字不要："频道 b 屏幕保护程序 . SCR"，应改为 "频道 b ~ 1. SCR"。

3. Windows API 函数 SystemParametersInfo

```
BOOL SystemParametersInfo(
  UINT uiAction,    // system parameter to query or set
  UINT uiParam,     // depends on action to be taken
  PVOID pvParam,    // depends on action to be taken
  UINT fWinIni      // user profile update flag );
```

若 UiAction = SPI_SETSCREENSAVEACTIVE，uiParam 为整型值 uiParam = 1 屏保有效；uiParam = 0 屏保无效。

若 UiAction = SPI_SETSCREENSAVETIMEOUT，uiParam 为整型值，决定屏保的等待时间，以秒为单位，为 60 的整数倍。

若 fWinIni = 0，则相应的改变项并不写入注册表保存，在缺省情况下，参数 pvParam 与 fWinIni 分别用 nil 与 0 值代入。

该 Windows API 函数，根据其所带的参数不同，能够向 Windows 操作系统发送相关的系统配置信息，可以实时动态地控制屏保是否有效及屏保等待时间，但前提仍然是 System. ini 文件中有正确的 SCRNSAVE. EXE 相关配置项

4. Win9X 屏保口令的加解密原理与过程

Win9X 屏保口令的加密原理是将密码与一个固定数组进行异或运算后再保存到注册表中，解密过程反之亦然；一般情况下，密码长度不会超过 20，经查阅相关资料及本人实践，该固定数组的前 20 个值为： （均为十进制值）72，238，118，29，103，105，161，27，122，140，71，248，84，149，151，95，120，217，218，108；相对应的十六进制值为：48 EE 76 1D 67 69 A1 1B 7A 8C 47 F8 54 95 97 5F 78 D9 DA 6C。 具体原理详见 《软件报》2000 年 07 月 31 日第 31 期应用空间 B3 版，高其海撰写的 《Win9X 屏幕保护口令的破解方法》）

经本人的多次摸索与实践，将 Win9X 屏保口令的加解密过程总结如下：

⑴ 加密过程

①将用户输入密码全部转换为大写，这也是 Win9X 屏保口令并不区分大小写的原因 （N 为用户输入密码的长度）；

②每个字符分别转换为 ASCII 码 （十进制）；

③将上一步得到的 ASCII 码 （十进制）值，分别与 Win9X 屏保口令加密固定数组中的对应项取异或运算，得到一系列十进制值；

④将十进制值转换为十六进制值；

⑤每个十六进制值均可拆为两个字符，例：1D 分解为 "1" 与 "D"；这样，我们能得到一个长度为 2 * N 的字符串 为原始密码的两倍；

⑥在上一步得到的字符串后再加上 ASCII 码 （十进制）值为零的字符，这是一个结束标志位；

⑦最后，将这个长度为 2 * N + 1 的字符串写入注册表 "我的电脑 hHKEY_CURRENT_USER hControl Panel hdesktop h ScreenSave_Data" 键值中； 由于该键值为二进制值，所以在注册表中所看到的是上述字符串中每个字符所对应的十六进制 ASCII 码值）；

例：输入密码为 "Aa"

① "Aa" ——〉 "AA"

② "AA" ——〉65 65

③ 65 XOR 72 第一位加密数组值 = 9

65 XOR 238 第二位加密数组值 = 175；

④ 9 ——〉 09

175——〉 AF ；

⑤ 0 9 ——〉 "0" 与 "9"

AF ——〉 "A" 与 "F"

得到字符串 "09AF"；

⑥ "09AF" + CHR 0 ；

⑦写入注册表相应键值中。写入完毕后，运行 "Regedit. exe" 双击 "我的电脑 hHKEY_CURRENT_USER h Control Panel hdesktop h ScreenSave_Data" 键值，可以看到所显示值为 "3039414600" 注：字符 "0"、"9"、"A"、"F" 与 CHR 0 结束位所对应的十六进制 ASCII 码值分别为：30、39、41、46、00）。

⑵ 解密过程

①从 "我的电脑 hHKEY_CURRENT_USER hControl Panel h desktop h ScreenSave_Data" 键值中读出资料，得到一个长度为 2 * N + 1 的字符串 （N 为用户输入密码的长度）；

②去掉 ASCII 码 （十进制）值为 0 的结束标志位，这时字符串长度为 2 * N；

③每两个字符构成一个十六进制值，得到一个长度为 N 的十六进制数值串；

④十六进制值转换为十进制值；

⑤将上一步得到的 ASCII 码 （十进制）值，分别与 Win9X 屏保口令加密固定数组中的对应项取异或运算，得到一系列十进制值；

⑥根据 ASCII 码表，将 ASCII 码 （十进制）值转换为相应的字符，我们得到一个长度为 N 的字符串，这就是我们所要的经过大写转换后的屏保口令。

例： "我的电脑 hHKEY_CURRENT_USER hControl Panel h desktop h ScreenSave_Data" 键值所显示值为 "3039414600"

①由键值所显示值得到一个字符串：“09AF”+ CHR 0

注：字符“0”、“9”、“A”、“F”与结束位 CHR 0 所对应的十六进制 ASCII 码值分别为：30、39、41、46、00；

②去掉结束位 CHR 0 ——〉“09AF”

③“09AF”拆分为 09 与 AF；

④ 09 ——〉 9　（十六进制转换为十进制）

AF ——〉175；　（十六进制转换为十进制）

⑤ 9 XOR 72　第一位加密数组值 = 65

175 XOR 238　第二位加密数组值 = 65；

⑥ 查询 ASCII 码表

65 ——〉“A”

65 ——〉“A”

得到字符串“AA”　这是经过大写转换后的屏保口令，解密过程结束。

从以上的几点论述中，我们知道只要对系统注册表、System. ini 文件进行正确的操作和配置，再配以 Windows API 函数 SystemParametersInfo 的灵活使用，就能达到动态配置屏幕保护程序的目的。

三、具体实现

利用 Delphi 5. 0 实现动态配置屏幕保护程序，主要应注意三个方面的问题：

1. 对系统注册表的操作

关于 Delphi 中对系统注册表的一般操作，在许多报刊杂志中均有详细的论述，这里只对系统注册表中二进制值的操作及需注意的问题做一个简单的说明，所涉及到的具体函数有 ReadBinaryData、WriteBinaryData

```
function ReadBinaryData(const Name: String; var Buffer; BufSize: Integer): Integer;
```

其中 Name 是当前所要操作的键名，Buffer 必须是保存键值第一位资料的变量名，BufSize 指定所输出资料的长度，值不能小于键值资料的实际长度值，资料依次写入 Buffer 变量的后续连续地址空间中 函数 ReadBinaryData 将返回资料的实际长度；

```
procedure WriteBinaryData(const Name: String; var Buffer; BufSize: Integer);
```

其中 Name 是当前所要操作的键名，Buffer 必须是保存键值第一位资料的变量名，以 Buffer 变量所在地址开始，依次向注册表中写入 Buffer 变量的后续连续地址空间中所保存的资料，BufSize 指定所写入资料的长度；

以从‘我的电脑 hHKEY_CURRENT_USER hControl Panel h desktop h ScreenSave_Data”键中读出经过加密的字符串数据和向该键中写入经过加密的字符串数据为例：

```
const MaxLength = 100
type AA = array [1.. MaxLength] of char;
var  Output_Data: AA;
    Input_Data: AA;
    Input_data_length: integer;
    Output_data_length: integer;
    j: integer;
    s: string;
begin
 My_Reg: = TRegistry. Create;
     try
     My_Reg. RootKey: = HKEY_CURRENT_USER;
  if  My_Reg. OpenKey(´Control Panel\desktop´, false)  then
       begin
        // Read ScreenSave_Data
    Output_data_length: = My_Reg. ReadBinaryData(´ScreenSave_Data´, Output_Data [1], MaxLength);
         . . . . . .
         // Write ScreenSave_Data
         Input_data_length: = 5;
         for j: = 1 to Input_data_length - 1do
          begin
           Input_Data [j]: = ´1´;
          end;
         Input_Data[Input_data_length]: = Chr(0);
         //写入值为‘1111 ’+ Chr(0)
         My_Reg. WriteBinaryData(´ScreenSave_Data´, Input_Data [1], Input_data_length);
{以下程序代码是错误的，因为在 Delphi 中 String 类型实际上是一个指向无结尾的字符串的指针，并不是我们所要的连续的数据结构类型}
         { s: = ´1111´ + Chr(0);  Input_data_length: = 5;
          My_Reg. WriteBinaryData(´ScreenSave_Data´, s, Input_data_length);
```

则写入的只是从字符串变量 s 所对应地址空间开始的连续地址空间中的数据，字符串变量 s 中保存的只是一个地址值，根本不是我们所指定的值，这一点应特别注意！}

```
         . . . . . .
       end;
     finally
       My_Reg. CloseKey;
       My_Reg. Free;
     end;
end;
```

则读出的字符串保存在字符数组 Output_Data [1. . Output_data_length]中，而写入注册表的则是保存在字符数组 Input_Data [1. . Input_data_length]中的字符串。

2. 对 System. ini 文件的操作

Windows 提供了 API 函数 GetPrivateProfileString、WritePrivateProfileString 专门用来对 INI 文件进行读写操作；

```
DWORD GetPrivateProfileString(
  LPCTSTR  lpAppName,    // points to section name 节名
  LPCTSTR  lpKeyName,    // points to key name 键名
  LPCTSTR  lpDefault, //points to default string 缺省返回值
  LPTSTR  lpReturnedString,    // points to destination buffer 保存返回键值变量
  DWORD  nSize,    // size of destination buffer 指定保存返回键值缓冲区长度
```

```
    LPCTSTR  lpFileName // points to initialization filename
所要操作文件名);
BOOL WritePrivateProfileString(
    LPCTSTR  lpAppName, // pointer to section name 节名
    LPCTSTR  lpKeyName, // pointer to key name 键名
    LPCTSTR  lpString, // pointer to string to add 写入的键值
    LPCTSTR  lpFileName  // pointer to initialization filename
所要操作文件名);
```

在 Delphi 中调用时，只需将 LPCTSTR、LPTSTR 类型用 PChar 型变量代入，并注意指定保存返回键值缓冲区长度值 nSize 时，应大于实际值。具体源程序如下：

```
procedure My_GetPrivateProfileString (My_lpAppName, My_
lpKeyName, My_lpDefault: shortstring; var My_lpReturned-
String: shortstring; My_nSize: dword; My_lpFileName: short-
string);
var lpAppName,  // points to section name
    lpKeyName, // points to key name
    lpDefault,  // points to default string
    lpReturnedString: pchar;  // points to destination buffer
    nSize: dword;       // size of destination buffer
    lpFileName: pchar;
  begin
    GetMem(lpAppName, 100);
    GetMem(lpKeyName, 100);
    GetMem(lpDefault, 100);
    GetMem(lpReturnedString, 100);
    GetMem(lpFileName, 100);
    StrPCopy(lpAppName, My_lpAppName);
    StrPCopy(lpKeyName, My_lpKeyName);
    StrPCopy(lpDefault, My_lpDefault);
    StrPCopy(lpFileName, My_lpFileName);
    nSize: =My_nSize;
    // 先为返回字符串指定足够的空间
   GetPrivateProfileString(lpAppName, lpKeyName, lpDefault,
lpReturnedString, nSize, lpFileName);
    My_lpReturnedString: =StrPas(lpReturnedString);
    //得到返回键值
    FreeMem(lpAppName, 100);
    FreeMem(lpKeyName, 100);
    FreeMem(lpDefault, 100);
    FreeMem(lpReturnedString, 100);
    FreeMem(lpFileName, 100);
  end;
procedure My_WritePrivateProfileString(My_lpAppName, My_
lpKeyName, My_String, My_lpFileName: shortstring);
var lpAppName,       // pointer to section name
    lpKeyName, // pointer to key name
    lpString,      // pointer to string to add
    lpFileName: PChar;      // pointer to initialization filename
  begin
    GetMem(lpAppName, 100);
    GetMem(lpKeyName, 100);
    GetMem(lpString, 100);
    GetMem(lpFileName, 100);
    StrPCopy(lpAppName, My_lpAppName);
    StrPCopy(lpKeyName, My_lpKeyName);
    StrPCopy(lpString, My_String);
    StrPCopy(lpFileName, My_lpFileName);
    //写入键值
    WritePrivateProfileString(
      lpAppName, lpKeyName, lpString, lpFileName);
    FreeMem(lpAppName, 100);
    FreeMem(lpKeyName, 100);
    FreeMem(lpString, 100);
    FreeMem(lpFileName, 100);
  end;
// 主程序
begin
. . . . . .
My_GetPrivateProfileString( ´boot´, ´scrnsave. exe´, ´This is
null! ´, Output_String, 100, ´system. ini´);
. . . . . .
My_WritePrivateProfileString(´boot´, ´scrnsave. exe´, Screen-
Save_File_Name, ´system. ini´);
. . . . . .
end;
```

3. **调用 Windows API 函数 SystemParametersInfo**

在 Delphi 中可以直接调用 SystemParametersInfo 函数，并不需要特殊说明，具体格式如下：

```
BOOL SystemParametersInfo(
  UINT uiAction,     // system parameter to query or set
  UINT uiParam,      // depends on action to be taken
  PVOID pvParam,     // depends on action to be taken
  UINT fWinIni       // user profile update flag );
```

若函数成功执行，则返回值不等于 NULL

uiAction 为整型，由 Windows 提供的不同常数值来确定进行何种操作；

uiParam 为整型，由 uiAction 决定其取值，若无特别说明，缺省值为 0；

pvParam 为指针型，由 uiAction 决定其取值，若无特别说明，缺省值为 nil ；

fWinIni 为整型，若要将改变值保存入注册表或是其它系统变量中，则用 SPIF_UPDATEINIFILE 常量代入；若不想将改变值保存，则用 0 值代入。

①获得当前的屏保等待时间：

uiAction = SPI_GETSCREENSAVETIMEOUT

pvParam 用整型变量的地址值代入，结果保存在该整型变量中

```
function          My_Get_SPI_GETSCREENSAVETIMEOUT:
shortstring;
var i: integer;
begin
SystemParametersInfo(SPI_GETSCREENSAVETIMEOUT, 0, @
i, 0);
  Result: =IntToStr(i div 60);
end;
```

②获得当前屏保是否有效信息：

uiAction = SPI_GETSCREENSAVEACTIVE

pvParam 用布尔型变量的地址值代入，结果保存在该布尔型变量中

```
function My_Get_SPI_GETSCREENSAVEACTIVE: shortstring;
var b: boolean;
begin
SystemParametersInfo(SPI_GETSCREENSAVEACTIVE,  0,  @b, 0);
 if b then
   Result: = '1' //  = true;
 else
   Result: = '0';  //  = false;
end;
```

③设定当前的屏保等待时间：

uiAction = SPI_SETSCREENSAVETIMEOUT
uiParam 用我们所设定的秒值代入

```
procedure My_Set_SPI_SETSCREENSAVETIMEOUT
          (Input_Minutes_String: shortstring);
begin
     SystemParametersInfo(SPI_SETSCREENSAVETIMEOUT,
StrToInt(Input_Minutes_String) * 60,  nil, 0);
end;
```

④设定当前屏保是否有效：

uiAction = SPI_SETSCREENSAVEACTIVE
uiParam = 0，表示无效;
uiParam = 1，表示有效;

```
procedure My_Set_SPI_SETSCREENSAVEACTIVE
          (Input_Active_Or_InActive: shortstring);
           // 0 : = InActive
           // 1 : = Active
begin
   SystemParametersInfo(SPI_SETSCREENSAVEACTIVE, StrToInt(Input_Active_Or_InActive), nil, 0);
    end;
```

四、结束语

以上程序在 Delphi 5.0、中文 Windows98 环境下调试通过，由于要对 System.ini 文件进行读写操作，所以视不同计算机的不同配置以及当前处理事务的多少，会造成一定时间的延迟，但一般不超过 5 秒钟，就会发现屏保程序及其屏保口令已经做了相应的改变。

我们可以将上述功能模块做成动态链接库文件，即 DLL 文件，在用户登录后，将用户口令作为当前屏保的口令；在退出应用程序后，再动态恢复用户的原始设置，达到保护信息资源的目的，收到较好的效果。

（收稿日期：2000 年 11 月 27 日）

BMC 软件公司协助电子商务公司实现安全管理自动化
为客户提供自动化访问管理和电子商务信息安全管理

2000 年 11 月 8 日美国休斯敦讯——电子商务系统管理软件供应商 BMC 软件公司宣布加盟“Entrust 开发商联盟项目”（Entrust Alliance Developer Program[EADP]），使该公司企业安全管理套件范围进一步得到扩大，同时为访问的集中化管理领域提供了最广泛的连接解决方案。

Entrust - Ready 认证适用于已在 EADP 中注册的产品，这些产品经测试证明可与 Entrust 技术公司处于市场领先地位的 PKI 软件相兼容和实现互操作性。作为联盟的成员之一，BMC 软件公司利用由 Entrust 提供的开发工具、测试及行销支持，将 CONTROL - SA 与 Entrust / PKI 相整合。企业在整合 Entrust 的数字化签名技术后，便有能力对传统的安全系统和电子商务安全解决方案同时进行集中化管理。BMC 软件公司的客户将可以充分利用电子商务合作伙伴之间建立信任关系所需的关键技术——即自动访问管理（使用 CONTROL - SA）和电子商务信息的安全保障（使用 Entrust / PKI）机制。

BMC 公司可为客户提供具有优势技术的解决方案，实现对安全性环境的集中管理并为 Entrust / PKI 软件提供支持。

安全在线交易中的自动访问请求

B2B 和 B2C 在线交易和沟通需要可靠的安全解决方案来保证企业资产的安全性。因此，必须对数量庞大的安全产品进行有效管理以减少外来访问者通过互联网打开 IT 资源的风险。BMC 软件公司有能力利用 Entrust / PKI 软件实现企业级信息安全管理。

BMC 软件公司认为同 Entrust 的合作对 BMC 公司具有战略意义，因为该公司可以进一步有序地在各种电子商务基础架构下管理电子商务用户和电子商务合作伙伴。同时 BMC 公司可以进一步深入操作系统和应用领域，从而建立起安全的端对端在线交易通道。EADP 联盟加强了 BMC 的实力，可让企业级安全管理主管从全局角度审视所有的安全系统。

BMC 软件公司的 INCONTROL 安全管理软件可集中管理前端（如访问请求）和后端（如身份识别、访问和口令）的处理过程，提高工作效率。企业可为员工自动地创建和管理对访问的请求，使员工从进入公司的第一天起能够高效地工作。另外，这些产品还可有效地提供口令管理功能，并审查安全措施的变化。

CONTROL - SA 可根据每家企业的商务需求以最佳方式进行配置。由于可与诸如 Entrust / PKI 之类电子商务安全产品相整合，CONTROL - SA 的安全管理能力得以扩大至外部群体。全球性企业的安全管理部经理可赋予地区和地方的系统安全管理员具体的管理权限。通过这种方式可给予不同群体以适当的访问权限，这些群体包括需要访问电子商务应用程序的供应商、合作伙伴和客户。

CONTROL - SA 是 BMC 软件公司完整安全管理系列产品之一。BMC 公司其它的网络安全产品有 CONTROL - SA / WorkFlow CONTROL - SA / PassPort 和 CONTROL - SA / Links。CONTROL - SA / WorkFlow 是一种基于网络的应用程序，它负责管理包括对计算机资源的访问请求、验证和授予访问权限等程序。CONTROL - SA / PassPort 允许最终用户通过浏览器来首次输入和同步修改口令，从而大幅度降低服务台的费用。CONTROL - SA / Links 提供对整个公司日常和复杂的安全管理任务的自动管理。这些产品的整合运营就可提供端对端的安全管理解决方案。

BMC 公司还提供一个在线安全管理方案的评测工具，它可以帮助用户对目前使用的安全管理方案和所建议的最好的安全管理方案之间进行对比。该免费软件可提供包括访问管理、审查功能和电子商务准备就绪这几个方面的定性信息反馈。该评测工具基于用户的输入，可以立即产生一个图形化和个性化的分析结果，提出建议，并以技术白皮书和分析报告的形式提交给用户，帮助用户找到最佳安全管理方案。该安全管理方案评测工具可以从 www.bmc.com / assessment / security 得到。

用 MFC 实现 Word2000 的‘任务栏切换文档’功能

元凯宁 李昕雯

在 Word2000 中，有这样一种新的界面外观：当在 Word2000 中创建多个文档时，每个文档会拥有一个顶层的窗口及菜单栏和工具栏，并且用一个 Windows 的任务栏上的按钮来代表，用户需要切换文档时，只需要单击不同的按钮即可。其实，这是 Microsoft 很早就想做 或者说是 Charles Simonyi 很早就想做）的一件事：淘汰相对较为复杂的 MDI 界面，只使用简单易懂的 SDI 界面。但就像 Bill Gates 不喜欢 Lotus 1 - 2 - 3 的表格命名方式但又不得不在 Excel 中兼容它一样，Microsoft 还是将 MDI 作为广泛应用的程序界面。现在，虽然 Charles Simonyi 不再领导 Word 的开发，但他的继任者终于可以在 Office2000 中实现这一宿愿了，他们在 Word2000 中实现了一种兼 SDI 和 MDI 之长的界面效果。这种界面的优劣姑且放置不论，让我们来考虑如何编程实现这种界面效果。

Word2000 的文档窗口有如下的特点：

1. 每个文档对应一个独立的窗口，拥有自己的菜单栏和工具栏，这个窗口在 Windows 的任务栏上对应一个按钮。

2. 不同文档间的切换可以用 Windows 的任务栏来实现，也可以用窗口菜单来实现。

3. 使用窗口菜单下的排列窗口功能时，Word2000 将以整个 Windows 的桌面 去掉任务栏和其它桌面工具栏的大小）为总体，每个文档窗口分得一部分面积。

4. 在菜单栏上没有代表文档的图标，当有两个或两个以上的文档处于活动状态时，每个文档只有在最顶层的窗口上有关闭按钮，点击它将关闭这个文档。当只有一个处于打开状态的文档时，将在菜单栏的最右边显示一个关闭按钮，点击它可以关闭文档，若在此时点击顶层窗口的关闭按钮将关闭 Word2000 程序。

5. 有一些全局面的状态是在各个文档窗口之间保存一致的，比如输入法状态、查找对话框中缓存的字串等。

下面就让我们用 MFC 来模拟这种界面。

在 MFC 中，MDI 的实现也是依赖于文档 / 视窗架构的。如果我们可以控制使每个 MDI 主框架窗口只拥有一个 MDI 子框架窗口，在新建和打开文档时再建立一个 MDI 主框架窗口并使之新建或打开这个文档，就能够达到目的。当然，这样的实现方案是要付出一定的代价的：相对于标准的 MDI 界面，这样做需要创建更多的窗口，从而消耗了更多的资源。不过，Word2000 也没有使用更 ‘高明’的办法，这一点可用 Spy + + 工具来验证。

创建一个 MFC AppWizard . EXE 的 MDI 工程，应用程序类名为 CMTabApp，主框架窗口类名为 CMTabMDIFrameWnd，子框架窗口类名为 CMTabMDIFrameWnd，文档类名为 CMTabDocument，视类名为 CMTabView，从 CEditView 类派生。

将 MDI 子框架窗口的风格中置空 WS_SYSMENU 风格，并使其最大化，可以使 MDI 子框架窗口总是充满 MDI 主框架窗口的客户区，并且在主框架窗口的菜单栏上无图标和 “最小化”、 “最大化”、 “关闭”三个标准按钮。为此，要在 PreCreateWindow 函数中修改窗口风格，再在 OnCreate 中调用 ShowWindow 函数将自身最大化。

在 CWinApp 类中，有一个 CWnd * 类型的成员指针，名为 m_pMainWnd，它指定了应用程序的主窗口，当这个窗口关闭时，整个应用程序就要退出。这个指针的值是可以动态地改变的，当有多个 MDI 主框架窗口时，可以任选一个作为 m_pMainWnd 的值，建议用当前处于激活态的那个 MDI 主框架窗口作为 m_pMainWnd 的值，因为这样可以保证 AfxGetMain 函数正确地工作。这需要在 MDI 主框架窗口的 OnSetFocus 中设定应用程序对象的 m_pMainWnd = this 。

当用户关闭文档时，要向 MDI 主框架窗口发送消息使之也关闭。为了不致于使应用程序退出，在关闭之前要将应用程序对象的 m_pMainWnd 指针值修改成另一个 MDI 主框架窗口的值。

当只剩一个 MDI 主框架窗口时，要在菜单栏的最右边画一个关闭按钮，这需要重载 OnNcPaint 和 OnNcActivate 两个函数，为了使这个按钮能够工作，需要重载 OnNcLButtonDown 和 OnNcLButtonUp 两个函数，并设置定时器检查鼠标状态。

在只有一个 MDI 主框架窗口的时候，当新建或关闭文档后，菜单栏上的关闭按钮要反应出这些变化，所以再重载 OnUpdateFrameTitle 函数，在这里发送一个 WM_NCPAINT 消息。这个虚函数将会被 CMDIChildWnd OnUpdateFrameTitle 和 CFrameWnd InitialUpdateFrame 所调用，CMDIChildWnd OnMDIActivate 和 CMDIChildWnd DestroyWindow 也要调用它，甚至连 CMDIChildWnd OnSize 也将会调用它 所以在拖动改变窗口大小的时候标题栏才会闪个不停）。

为了保存所有的 MDI 主框架窗口的指针，需要新建立一个结构，为方便起见，在这个结构中也加入文档模板的指针，其定义如下：

```
typedef struct _TEMPLWND {
  CMultiDocTemplate * pTempl;
  CWnd * pWnd;
} TEMPLWND
```

再用这个结构去实例化一个 CList 模板类：

```
typedef CList <TEMPLWND * , TEMPLWND *> CTemplWndList;
```

其生成的对象存在于 App 类中作为保护成员变量，加入一系列的函数来对它进行操作：

```
int GetFrameCount(void); /* 此函数取得当前的框架窗口的总数, 即列表的项目数 */
CWnd * ElectMainWnd(CWnd * pWndToDestroy, BOOL bDestory = TRUE); /* "选举"一个新的窗口作为应用程序的主窗口 */
void AddToList(CMultiDocTemplate * pTemplToAdd, CWnd * pWndToAdd, BOOL bSetMainWnd = TRUE); /* 加入一项到列表中 */
void DeleteFromList(CWnd * pWndToDelete); /* 从列表中删除一项 */
```

为了使用未命名的文档的默认名称 即 "文档 1"、"文档 2" 之类的名称）的记数是全局的，在 App 对象中加入一个保护成员，并为其添加存取函数：

```
int m_nUntitledCount;
int GetUntitledCount(BOOL bIncreasingAfterUse = TRUE);
```

当用户再次新建文档或是打开文档时，默认的 CWinApp 类的命令处理器就不适用了，为此， 用 ClassWizard）添加 ID_FILE_NEW 和 ID_FILE_OPEN 的命令处理器，在其中生成新的 MDI 框架窗口以容纳新文档。

当用户新建文档时，所采用的默认标题应当是统一计数的，但因为不同的文档是属于不同的文档模板的，这会造成每个框架窗口中的文档默认标题自行累加计数的现象。为取代文档模板对文档标题的默认赋值，重载文档类的 OnNewDocument 函数，在其中取得全局的文档数量记数，并重新设置文档标题。

为了能使文档能够在运行时刻标记为已改变，令视图类从 CEditView 类派生。

下面列出了主要的程序代码，其中已含大量的注解。在示例工程中，全部注解采用简明的英语写成。

CMTabApp 类的头文件

只列出结构和类的定义，略去了一些内容。

```
/* 下面的这几行是要手工添加在 MCTabApp 的定义前面的 */
typedef struct _TEMPLWND {
    CMultiDocTemplate * pTempl;
    CWnd * pWnd;
} TEMPLWND;
typedef CList <TEMPLWND * , TEMPLWND *> CTemplWndList;
class CMTabApp : public CWinApp
{
public:
    CMTabApp();
public:
    int GetFrameCount(void); // get the count of frame window(s)
int GetUntitledCount(BOOL bIncreasingAfterUse = TRUE);
  CWnd * ElectMainWnd(CWnd * pWndToDestroy, BOOL bDestory = TRUE); // this will find a frame window and set it to be the main window (m_pMainWnd)
  BOOL SaveModifiedFrame(CWnd * pWnd);
// member functions
protected:
void AddToList(CMultiDocTemplate * pTemplToAdd, CWnd * pWndToAdd, BOOL bSetMainWnd = TRUE);
  void DeleteFromList(CWnd * pWndToDelete);
// member variables
protected:
  CTemplWndList m_listTempl; // keep the information of all the documents and frame windows
  int m_nUntitledCount; // quantity of untitled document(s)
  // Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CMTabApp)
  public:
  virtual BOOL InitInstance();
  virtual int ExitInstance();
  virtual CDocument * OpenDocumentFile(LPCTSTR lpszFileName);
  //}}AFX_VIRTUAL
// Implementation
protected:
  //{{AFX_MSG(CMTabApp)
  afx_msg void OnAppAbout();
  afx_msg void OnWindowTileHorz();
  afx_msg void OnWindowTileVert();
  afx_msg void OnFileNew();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};
/*
    下面的一行可以使在其它的文件中引用 theApp 对象而不必再调用 AfxGetApp 函数并进行类型转换
*/
extern CMTabApp theApp;
```

CMTabApp 类的实现文件

已删去有关 CAboutDlg 的内容，并且只列出函数的定义。各个后来加入函数已按名称进行升序排序。

```
CMTabApp: : CMTabApp()
{
  m_nUntitledCount = 1;
}
BOOL CMTabApp: : InitInstance()
{
  AfxEnableControlContainer();
  // Standard initialization
  // If you are not using these features and wish to reduce the size of your final executable, you should remove from the following the specific initialization routines you do not
```

```
need.
#ifdef _AFXDLL
    Enable3dControls(); // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif
    // Change the registry key under which our settings are stored.
    // TODO: You should modify this string to be something appropriate such as the name of your company or organization.
    SetRegistryKey(_T("eWay Studio"));
    LoadStdProfileSettings(0); // Load standard INI file options (including MRU)
    // Register the application's document templates. Document templates serve as the connection between documents, frame windows and views.
    CMultiDocTemplate * pDocTemplate;
pDocTemplate = new CMultiDocTemplate(IDR_MTABTYPE,
        RUNTIME_CLASS(CMTabDocument),
    RUNTIME_CLASS(CMTabMDIChildWnd), //custom MDI child frame
        RUNTIME_CLASS(CMTabView));
    AddDocTemplate(pDocTemplate);
    // create main MDI Frame window
    CMTabMDIFrameWnd * pMainFrame = new CMTabMDIFrameWnd;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    AddToList(pDocTemplate, pMainFrame);
    // Enable drag/drop open
    m_pMainWnd->DragAcceptFiles();
    // Enable DDE Execute open
    //EnableShellOpen();
    //RegisterShellFileTypes(TRUE);
    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    //The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();
    return TRUE;
}
void CMTabApp::AddToList(CMultiDocTemplate * pTemplToAdd, CWnd * pWndToAdd, BOOL bSetMainWnd)
{
    ASSERT(pWndToAdd);
    ASSERT_VALID(pWndToAdd);
    ASSERT(pTemplToAdd);
    ASSERT_VALID(pTemplToAdd);
  TEMPLWND * ptw = new TEMPLWND;
  ptw->pWnd = pWndToAdd;
  ptw->pTempl = pTemplToAdd;
  m_listTempl.AddTail(ptw);
  m_pMainWnd = pWndToAdd;
}
void CMTabApp::DeleteFromList(CWnd * pWndToDelete)
{
  ASSERT(pWndToDelete);
  ASSERT_VALID(pWndToDelete);
  TEMPLWND * ptw;
  POSITION pos = m_listTempl.GetHeadPosition();
  POSITION posToDelete;
  while ( pos )
  {
      posToDelete = pos;
      ptw = m_listTempl.GetNext(pos);
      if ( pWndToDelete == ptw->pWnd )
      {
          m_listTempl.RemoveAt(posToDelete);
          delete ptw;
      }
  }
}
CWnd * CMTabApp::ElectMainWnd(CWnd * pWndToDestroy, BOOL bDestory)
{
  ASSERT(pWndToDestroy);
  ASSERT_VALID(pWndToDestroy);
  TEMPLWND * ptw;
  POSITION pos = m_listTempl.GetHeadPosition();
  while ( pos ) // loop from list
  {
      ptw = m_listTempl.GetNext(pos);
      if ( pWndToDestroy != ptw->pWnd )
      {
          m_pMainWnd = ptw->pWnd;
          if ( bDestory )
          {
              DeleteFromList(pWndToDestroy);
          }
          return ptw->pWnd;
      }
  }
  // haven't found any other window(s)
  return NULL;
}
int CMTabApp::ExitInstance()
{
  // do some clean work
  POSITION pos = m_listTempl.GetHeadPosition();
  TEMPLWND * ptw;
  while ( pos )
  {
      ptw = m_listTempl.GetNext(pos);
      delete ptw;
  }
```

```
    return CWinApp::ExitInstance();
}
int CMTabApp::GetFrameCount()
{
    return m_listTempl.GetCount();
}
int CMTabApp::GetUntitledCount(BOOL bIncreasingAfterUse)
{
    return (bIncreasingAfterUse)? m_nUntitledCount++:m_nUntitledCount;
}
void CMTabApp::OnFileNew()
{
// to see if it has an empty template – i.e. has no document –
    TEMPLWND* ptw;
    POSITION posDoc;
  POSITION posTempl = m_listTempl.GetHeadPosition();
    while (posTempl)
    {
        ptw = m_listTempl.GetNext(posTempl);
    posDoc = ptw->pTempl->GetFirstDocPosition();
        if (posDoc == NULL)
        {
        ptw->pTempl->OpenDocumentFile(NULL);
            return;
        }
    }
    // else, create a new mainframe to open it
    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_MTABTYPE,
        RUNTIME_CLASS(CMTabDocument),
  RUNTIME_CLASS(CMTabMDIChildWnd), // custom MDI child frame
        RUNTIME_CLASS(CMTabView));
    AddDocTemplate(pDocTemplate);
    // create main MDI Frame window
    CMTabMDIFrameWnd* pMainFrame = new CMTabMDIFrameWnd;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return;
    AddToList(pDocTemplate, pMainFrame);
    // Enable drag/drop open
    m_pMainWnd->DragAcceptFiles();
    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(SW_SHOW); // will retain the frame window´s state since the frame widnow can judge the state
    pMainFrame->UpdateWindow();
    pDocTemplate->OpenDocumentFile(NULL);
})
void CMTabApp::OnWindowTileHorz(
{
// IMPORTANT: SAMPLE CODE WHICH IS NOT SUITABLE FOR THE CASE OF MORE THAN 3 FRAME WINDOWS
  // try to get the desktop´s size, not including the task bar and other application´s desktop bars
int cxDesktop = ::GetSystemMetrics(SM_CXFULLSCREEN);
int cyDesktop = ::GetSystemMetrics(SM_CYFULLSCREEN);
  cyDesktop += ::GetSystemMetrics(SM_CYCAPTION);
  int nCount = GetFrameCount();
  int nSizeVert = cyDesktop / nCount;
  TEMPLWND* ptw;
  POSITION pos = m_listTempl.GetHeadPosition();
  int y = 0;
 // the following three lines enable the function of tiling window which the command called from to be the 1st window
  CWnd* pMainWnd = m_pMainWnd;
  pMainWnd->SetWindowPos(NULL, 0, y, cxDesktop, nSizeVert, SWP_NOZORDER);
  y += nSizeVert;
  while (pos)
  {
      // loop any structure in list
      ptw = m_listTempl.GetNext(pos);
if (ptw->pWnd != pMainWnd) // it´s already been tiled
      {
  ptw->pWnd->SetWindowPos(NULL, 0, y, cxDesktop, nSizeVert, SWP_NOZORDER);
          y += nSizeVert;
      }
  }
  pMainWnd->SetFocus();
})
void CMTabApp::OnWindowTileVert(
{
// IMPORTANT: SAMPLE CODE WHICH IS NOT SUITABLE FOR THE CASE OF MORE THAN 3 FRAME WINDOWS
  // try to get the desktop´s size, not including the task bar and other application´s desktop bars
int cxDesktop = ::GetSystemMetrics(SM_CXFULLSCREEN);
int cyDesktop = ::GetSystemMetrics(SM_CYFULLSCREEN);
  cyDesktop += ::GetSystemMetrics(SM_CYCAPTION);
  int nCount = GetFrameCount();
  int nSizeHorz = cxDesktop / nCount;
  TEMPLWND* ptw;
  POSITION pos = m_listTempl.GetHeadPosition();
  int x = 0;
  // the following three lines enable the function of
  // tiling window which the command called from to be the 1st window
  CWnd* pMainWnd = m_pMainWnd;
  pMainWnd->SetWindowPos(NULL, x, 0, nSizeHorz, cyDesktop, SWP_NOZORDER);
  x += nSizeHorz;
  while (pos) // loop from list
  {
      ptw = m_listTempl.GetNext(pos);
if (ptw->pWnd != pMainWnd) // it´s already been tiled
      {
```

```
    ptw->pWnd->SetWindowPos(NULL, x, 0, nSizeHorz,
cyDesktop, SWP_NOZORDER);
                x += nSizeHorz;
            }
        }
        pMainWnd->SetFocus();
}
CDocument * CMTabApp::OpenDocumentFile(LPCTSTR lpszFileName)
{
        // loop to see if the file is already open
        TEMPLWND * ptw;
        CDocument * pDoc;
        POSITION posDoc;
      POSITION posTempl = m_listTempl.GetHeadPosition();
        while ( posTempl ) // loop from the list
        {
            ptw = m_listTempl.GetNext(posTempl);
        posDoc = ptw->pTempl->GetFirstDocPosition();
            if ( ! posDoc && GetFrameCount() == 1 )
            {
        return CWinApp::OpenDocumentFile(lpszFileName);
            }
            while ( posDoc )
            {
            pDoc = ptw->pTempl->GetNextDoc(posDoc);
            if ( pDoc->GetPathName() == lpszFileName )
                {
            // activate the frame with the document open
            ptw->pWnd->SendMessage(WM_SETFOCUS);
                    // since we have done all the job
                // return the pointer and exit the function
                    return pDoc;
                }
            }
        }
        // else, create a new mainframe to open it
        CMultiDocTemplate * pDocTemplate;
        pDocTemplate = new CMultiDocTemplate(
            IDR_MTABTYPE,
            RUNTIME_CLASS(CMTabDocument),
        RUNTIME_CLASS(CMTabMDIChildWnd), //  custom
MDI child frame
            RUNTIME_CLASS(CMTabView));
        AddDocTemplate(pDocTemplate);
        // create main MDI Frame window
        CMTabMDIFrameWnd * pMainFrame = new CMTab-
MDIFrameWnd;
        if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
            return FALSE;
        AddToList(pDocTemplate, pMainFrame);
        // Enable drag/drop open
        m_pMainWnd->DragAcceptFiles();
        // The main window has been initialized, so show and
update it.
        pMainFrame->ShowWindow(SW_SHOW); // will re-
tain the frame window´s state since the frame widnow can
judge the state
        pMainFrame->UpdateWindow();
return pDocTemplate->OpenDocumentFile(lpszFileName);
}
BOOL CMTabApp::SaveModifiedFrame(CWnd * pWnd)
{
        BOOL bSaved = TRUE;
        TEMPLWND * ptw;
        POSITION posTempl = m_listTempl.GetHeadPosition();
        while ( posTempl )
        {
            ptw = m_listTempl.GetNext(posTempl);
            if ( ptw->pWnd == pWnd )
            {
bSaved = bSaved && ptw->pTempl->SaveAllModified();
            }
        }
        return bSaved;
}
```

CMTabMDIFrameWnd 类的头文件

已略去了一些不重要的内容。

```
/* 这两行需要手式加入 */
#define ID_CUSTCLOSEBUTTONRESTORE  1001/* used for
timer */
#define HT_CUSTCLOSEBUTTON  1002 /* used from hittest
*/
class CMTabMDIFrameWnd : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMTabMDIFrameWnd)
public:
    CMTabMDIFrameWnd();
    virtual ~CMTabMDIFrameWnd();
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMTabMDIFrameWnd)
     virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
    virtual void OnUpdateFrameTitle(BOOL bAddToTitle);
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:
    // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;
BOOL m_bNcLBtnDownOnCustCloseBtn; // if left button is
down on the customized close button
    BOOL NeedDrawCustCloseButton(void); // if need to
draw the customized close button
    void GetCustCloseButtonRect(RECT & rect); // to get
the rectangle of the customized close button
    HRGN GetCustCloseButtonRGN(const RECT * pRect =
NULL); // to get the region of the customized close button
// Generated message map functions
```

```
protected:
    //{{AFX_MSG(CMTabMDIFrameWnd)
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnClose();
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnTimer(UINT nIDEvent);
    //}}AFX_MSG
    // add the follow 5 lines manually
    afx_msg BOOL OnNcActivate(BOOL bActive);
    afx_msg UINT OnNcHitTest(CPoint point);
afx_msg void OnNcLButtonDown(UINT nHitTest, CPoint pt);
afx_msg void OnNcLButtonUp(UINT nHitTest, CPoint pt);
    afx_msg void OnNcPaint();
    DECLARE_MESSAGE_MAP()
};
```

CMTabMDIFrameWnd 类的实现文件

略去了一些不重要的内容，各个新加入的函数已按名称进行升序排序。

```
IMPLEMENT_DYNAMIC(CMTabMDIFrameWnd, CMDIFrame-
Wnd)
BEGIN_MESSAGE_MAP(CMTabMDIFrameWnd, CMDIFrame-
Wnd)
    //{{AFX_MSG_MAP(CMTabMDIFrameWnd)
    ON_WM_CREATE()
    ON_WM_CLOSE()
    ON_WM_SETFOCUS()
    ON_WM_TIMER()
    //}}AFX_MSG_MAP
    // add the follow 5 lines manually
    ON_WM_NCPAINT()
    ON_WM_NCACTIVATE()
    ON_WM_NCLBUTTONDOWN()
    ON_WM_NCLBUTTONUP()
    ON_WM_NCHITTEST()
END_MESSAGE_MAP()
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
////////////////////////////////////
// CMTabMDIFrameWnd construction/destruction
CMTabMDIFrameWnd::CMTabMDIFrameWnd()
{
    m_bNcLBtnDownOnCustCloseBtn = FALSE;
}
CMTabMDIFrameWnd::~CMTabMDIFrameWnd()
{
}
int CMTabMDIFrameWnd::OnCreate(LPCREATESTRUCT
lpCreateStruct)
{
  if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;
  if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
WS_CHILD | WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER |
CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC,
CRect(3, 3, 3, 3)) || !m_wndToolBar.LoadToolBar
(IDR_MAINFRAME))
  {
      TRACE0("Failed to create toolbar\n");
      return -1;      // fail to create
  }
  if (!m_wndStatusBar.Create(this) ||!m_wndStatusBar.
SetIndicators(indicators, sizeof(indicators)/sizeof(UINT)))
  {
      TRACE0("Failed to create status bar\n");
      return -1;      // fail to create
  }
  // TODO: Delete these three lines if you don't want the
toolbar to be dockable
  m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
  EnableDocking(CBRS_ALIGN_ANY);
  DockControlBar(&m_wndToolBar);
  return 0;
}
BOOL CMTabMDIFrameWnd::PreCreateWindow(CREATE-
STRUCT& cs)
{
  if(!CMDIFrameWnd::PreCreateWindow(cs))
      return FALSE;
  // if the current frame is maximized
  // show it as maximized state
  if (theApp.m_pMainWnd // will be false when create a
frame at 1st time && ::IsWindow(theApp.m_pMainWnd->
GetSafeHwnd()) && theApp.m_pMainWnd->GetStyle()
& WS_MAXIMIZE)
  {
      cs.style |= WS_MAXIMIZE;
  }
  return TRUE;
}
////////////////////////////////////
// CMTabMDIFrameWnd diagnostics
#ifdef _DEBUG
void CMTabMDIFrameWnd::AssertValid() const
{
  CMDIFrameWnd::AssertValid();
}
void CMTabMDIFrameWnd::Dump(CDumpContext& dc)
const
{
  CMDIFrameWnd::Dump(dc);
}
#endif //_DEBUG
void CMTabMDIFrameWnd::GetCustCloseButtonRect(RECT
& rect)
{
  // to calculate the rectangle, in screen coordinates
```

```
    // NOTE that when use it for drawing,
    // substract the window´s upperleft point from it
    GetWindowRect(& rect);
  rect. top + =: : GetSystemMetrics(SM_CYCAPTION) + 4;
    rect. right – = 6;
rect. bottom =rect. top +: : GetSystemMetrics(SM_CYMENU-
SIZE) – 4;
rect. left = rect. right – : : GetSystemMetrics(SM_CXMENU-
SIZE) + 2 ;
}
HRGN CMTabMDIFrameWnd: : GetCustCloseButtonRGN
(const RECT * pRect)
{
    CRgn rgn;
    if ( pRect = = NULL )
    {
        // try to get the rectangle
        CRect rect;
        GetCustCloseButtonRect(rect);
        rgn. CreateRectRgn(rect. left, rect. top, rect. right,
rect. bottom);
    }
    else
    {
        // use the rectangle passed from parameter(s)
   rgn. CreateRectRgn(pRect –>left, pRect –>top, pRect –>
right, pRect –>bottom);
    }
    return rgn;
}
BOOL CMTabMDIFrameWnd: : NeedDrawCustCloseButton()
{n
    // if there is only one frame window and if it has one
document
    // we draw the customized close butto
    return ( theApp. GetFrameCount() = = 1 & & MDIGe-
tActive() ! = NULL);
}
void CMTabMDIFrameWnd: : OnClose()
{
if (m_lpfnCloseProc ! = NULL & & ! ( * m_lpfnCloseProc)
(this))
        return;
    // Note: only queries the active document
    CDocument * pDocument = GetActiveDocument();
    if (pDocument ! = NULL & & ! pDocument –>Can-
CloseFrame(this))
    {
    // document can´t close right now – – don´t close it
        return;
    }
    if (theApp. m_pMainWnd = = this)
    {
  // attempt to save all documents belongs to the template
        if (pDocument = = NULL & & ! theApp. Save-
ModifiedFrame(this))
            return;     // don´t close it
  if ( theApp. GetFrameCount () = = 1 ) // if there is only
one frame
        {;
        // hide the application´s windows before closing all
the documents
            theApp. HideApplication();
            // close all documents first
            theApp. CloseAllDocuments(FALSE)
// don´t exit if there are outstanding component objects
            if (!AfxOleCanExitApp())
            {
                // take user out of control of the app
                AfxOleSetUserCtrl(FALSE);
// don´t destroy the main window and close down just yet
  //  (there are outstanding component (OLE) objects)
                return;
            }
  // there are cases where destroying the documents may
destroy the  main window of the application.
if (!afxContextIsDLL & & theApp. m_pMainWnd = = NULL)
            {
                AfxPostQuitMessage(0);
                return;
            } )
        } // if ( theApp. GetFrameCount () = = 1
  } // if (theApp. m_pMainWnd = = this)
  // detect the case that this is the last frame on the docu-
ment and  shut down with OnCloseDocument instead.
  if (pDocument ! = NULL & & pDocument –>
m_bAutoDelete)
  {
      BOOL bOtherFrame = FALSE;
   POSITION pos = pDocument –>GetFirstViewPosition();
      while (pos ! = NULL)
      {
    CView * pView = pDocument –>GetNextView(pos);
        ASSERT_VALID(pView);
        if (pView –>GetParentFrame() ! = this)
        {
            bOtherFrame = TRUE;
            break;
        }
      }
      if (!bOtherFrame)
      {
        pDocument –>OnCloseDocument();
        return;
      }
  // allow the document to cleanup before the window is
destroyed
      pDocument –>PreCloseFrame(this);
  }
  // change m_pMainWnd member of the application object
  theApp. ElectMainWnd(this, TRUE);
  // then destroy the window
  DestroyWindow();
}
```

```
BOOL CMTabMDIFrameWnd: : OnNcActivate(BOOL bActive)
{
    if ( NeedDrawCustCloseButton() )
    {
    BOOL bRetCode = CMDIFrameWnd: : OnNcActivate
(bActive);
    //must redraw the customized close button on the
menu bar
        SendMessage(WM_NCPAINT);
        return bRetCode;
    }
    else
    {
        return CMDIFrameWnd: : OnNcActivate(bActive);
    }
}
UINT CMTabMDIFrameWnd: : OnNcHitTest(CPoint point)
{
    if ( NeedDrawCustCloseButton())
    {
        CRect rect;
        GetCustCloseButtonRect(rect);
        if ( rect. PtInRect(point) )
        {
            // to indicate that clicked on the
            // customized close button
            return HT_CUSTCLOSEBUTTON;
        }
    }
    return CMDIFrameWnd: : OnNcHitTest(point);
}
void CMTabMDIFrameWnd: : OnNcLButtonDown(UINT
nHitTest, CPoint pt)
{)
    if ( nHitTest = = HT_CUSTCLOSEBUTTON
    {
        // if clicked on customized close button
        m_bNcLBtnDownOnCustCloseBtn = TRUE;
        SendMessage(WM_NCPAINT, WPARAM(GetCust-
CloseButtonRGN()));
    SetTimer(ID_CUSTCLOSEBUTTONRESTORE, 55/ * min
value available * /, NULL);
    }
    CMDIFrameWnd: : OnNcLButtonDown(nHitTest, pt);
}
void CMTabMDIFrameWnd: : OnNcLButtonUp(UINT
nHitTest, CPoint pt)
{)
    if ( nHitTest = = HT_CUSTCLOSEBUTTON
    {
        // if clicked on customized close button
        m_bNcLBtnDownOnCustCloseBtn = FALSE;
  KillTimer(ID_CUSTCLOSEBUTTONRESTORE); // release
the resource
        // try to close the document
        // the result depends on the action of the user
        MDIGetActive() ->SendMessage(WM_CLOSE, 1);
// try to close the document
        SendMessage(WM_NCPAINT, WPARAM(GetCust-
CloseButtonRGN()));
  }
  {
        CMDIFrameWnd: : OnNcLButtonUp(nHitTest, pt);
  }
}
void CMTabMDIFrameWnd: : OnNcPaint()
{
  CMDIFrameWnd: : OnNcPaint();
  CWindowDC dc(this);
  CRect rect;
  GetCustCloseButtonRect(rect);
// substract the window´s upperleft point from the rectan-
gle see the note in the GetCustCloseButtonRect() function
  CRect rectWindow;
  GetWindowRect(rectWindow);
  rect - = rectWindow. TopLeft ();
  if ( NeedDrawCustCloseButton() )
  {
        // draw a close button on the menu bar
        if ( m_bNcLBtnDownOnCustCloseBtn )
        {
            // draw a pushed button
            dc. DrawFrameControl ( & rect, DFC_CAPTION,
DFCS_CAPTIONCLOSE | DFCS_PUSHED);
        }
        else
        {
            // draw a standard button
            dc. DrawFrameControl( & rect, DFC_CAPTION,
DFCS_CAPTIONCLOSE);
        }
  }
  else
  {
        // brush the rectangle with menu´s color
        CBrush brush;
  brush. CreateSolidBrush( : : GetSysColor(COLOR_MENU));
        CBrush * pOldBrush = dc. SelectObject(& brush);
  dc. PatBlt(rect. left, rect. top, rect. Width(), rect. Height(),
PATCOPY);
        dc. SelectObject(pOldBrush);
  }
}
void CMTabMDIFrameWnd: : OnSetFocus(CWnd * pOldWnd)
{
  CMDIFrameWnd: : OnSetFocus(pOldWnd);
  // change m_pMainWnd member of the application object
  theApp. m_pMainWnd = this;
}
void CMTabMDIFrameWnd: : OnTimer(UINT nIDEvent)
{
  switch( nIDEvent )
  {
  case ID_CUSTCLOSEBUTTONRESTORE: // restore the
```

```
customized close button  from pushed state to normal
        if ( m_bNcLBtnDownOnCustCloseBtn )
        {
            CRect rect;
            CPoint pt;
            GetCustCloseButtonRect(rect);
            GetCursorPos(& pt);
            if ( ! rect. PtInRect(pt) )
            {
            m_bNcLBtnDownOnCustCloseBtn = FALSE;
                // and redraw the button
              SendMessage(WM_NCPAINT, WPARAM
(GetCustCloseButtonRGN()));
                KillTimer
(ID_CUSTCLOSEBUTTONRESTORE); // release the timer
resource
            }
        };
        break;
    default:
        CMDIFrameWnd: : OnTimer(nIDEvent)
        break;
    }
}
void CMTabMDIFrameWnd: : OnUpdateFrameTitle(BOOL
bAddToTitle)
{
    CMDIFrameWnd: : OnUpdateFrameTitle( bAddToTitle);
    // when update the frame title – i. e. created or closed
a document  we redraw the customized close button
    SendMessage(WM_NCPAINT, WPARAM(GetCustClose-
ButtonRGN()));
}
```

CMTabMDIChildWnd 类的头文件

只列出需要手工加入到类定义中的片段。

```
protected:
    CWnd * m_pwndParent;
    afx_msg void OnNcDestroy();
```

CMTabMDIChildWnd 类的实现文件

只列出了需要手工加入消息映射和需要修改的三个函数。

```
BEGIN_MESSAGE_MAP(CMTabMDIChildWnd, CMDIChild-
Wnd)
    //{{AFX_MSG_MAP(CMTabMDIChildWnd)
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
    //下面一行手工加入
    ON_WM_NCDESTROY()
END_MESSAGE_MAP()
BOOL CMTabMDIChildWnd : : PreCreateWindow ( CREATE-
STRUCT& cs)
{
    if( !CMDIChildWnd: : PreCreateWindow(cs) )
        return FALSE;
  // create a child frame without sysmenu
  // i. e.  it has no system icon and no caption buttons
  cs. style & = ~WS_SYSMENU;
  return TRUE;
}
int CMTabMDIChildWnd: : OnCreate(LPCREATESTRUCT
lpCreateStruct)
{
  if (CMDIChildWnd: : OnCreate(lpCreateStruct) = = –1)
        return –1;
  // show the window as maximized state
  ShowWindow(SW_SHOWMAXIMIZED);
  // and keep the parent winodw´s pointer for later use
  m_pwndParent = GetParentFrame();
  return 0;
}
void CMTabMDIChildWnd: : OnNcDestroy()
{
  // use false as parameter because the frame window can
destroy itself in WM_CLOSE message processing procdure
  if(theApp. ElectMainWnd(m_pwndParent, FALSE))
  {
        // there is another frame window, so destory the
current one
        m_pwndParent –>SendMessage(WM_CLOSE, 0, 0);
  }
  CMDIChildWnd: : OnNcDestroy();
}
```

CMTabDocument 类的实现文件

头文件已略去，只列出 OnNewDocument 函数的内容，其余内容由 AppWizard 生成。

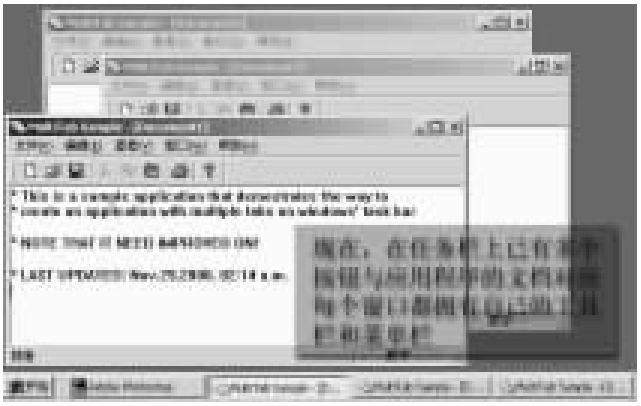
```
BOOL CMTabDocument: : OnNewDocument()
{
  if (!CDocument: : OnNewDocument())
        return FALSE;
  // we set the default name in a global index
  CString strDocName;
  if (GetDocTemplate( ) –>GetDocString(strDocName,
CDocTemplate: : docName) & &
        !strDocName. IsEmpty())
  {
        TCHAR szNum[8];
wsprintf(szNum, _T(˝%d˝), theApp. GetUntitledCount());
        strDocName + = szNum;
  }
  else
  {
        // use generic ´untitled´ – ignore untitled count
    VERIFY(strDocName. LoadString(AFX_IDS_UNTITLED));
  }
  SetTitle(strDocName);
  return TRUE;
}
```

CMTabView 类的实现文件

头文件已略去，只列出 PreCreateWindow 函数的内容，其余内容由 AppWizard 生成。

```
BOOL CMTabView::PreCreateWindow(CREATESTRUCT &cs)
{r
    // this will enable the multiple line editing mode
    // and will also disable the scroll ba
    cs.style |= ES_MULTILINE;
    return CEditView::PreCreateWindow(cs);
}
```

这个工程运行时刻的实例见图。



应当指出的是，这个例子要作为实际的应用，还是远远不足的，例如它的排列窗口功能只适用于排列 3 个或 3 个以下的窗口；在将窗口大小设置为很窄的一条时，自画的关闭按钮会将窗口的边框覆盖，窗口菜单下的窗口列表也只有一个，而且还不能新建窗口（使用同一文档的窗口）。不过，只要发现了问题的所在，就很容易解决了，这个例子是用来演示基本思路和方法的。

工程调试环境：

PIII 733EB with 256MB RAM and RAID5 support。

Microsoft Windows 2000 Advanced Server Microsoft Visual Studio 6.0 Enterprise Edition。

文章写作使用 Microsoft Word 2000。

图片处理采用 Adobe Photoshop 5.02 CHS。

（收稿日期：2000 年 11 月 29 日）

Intel 电子商务加速器安全加速中国万网网站服务

近期，国内最大的网站服务提供商中国万网在成功采用 NETSCAPE 提出的安全数据传输协议——SSL（Secure Socket Layer）协议的同时，又斥巨资引进了专用的 SSL 加速设备——Intel 网擎电子商务加速器，既保证了 Internet 数据传输的安全，又切实为人们访问其服务的网站实现了“提速”。

Internet 是世界上最庞大的计算机网络，并且每天仍在飞速的增长。网络拉近了人们的距离，改变了人们的生活，给人类带来了又一次的飞跃。然而人们在享受网络带来的便利的同时，也同样面临网络带来的威胁，特别是由于电子商务的应用，随时可能因为非法的访问、信息的窃听或破坏，身份的伪造，地址的欺骗等造成不可估量的损失。所以对于商业应用或敏感性的信息传输安全保障提出了更严格的要求。

为了适应 Internet 的安全机制，NETSCAPE 提出了一种安全数据传输协议——SSL（Secure Socket Layer）协议，目前 Internet 上大多数安全 web 站点都使用 SSL 协议。在有效利用 SSL 的过程中，中国万网认为，尽管 NETSCAPE 的 SSL 协议提供了数据加密措施、服务器认证、信息完整性及 TCP/IP 连接的用户认证，客户端浏览器可以通过 HTTP 请求和提供 SSL 的 WEB 服务器进行有效连接。但是，SSL 最复杂和消耗资源的部分就是建立连接前的身份认证和交换共享密钥，由于每一次传输都是加密进行的，服务器端要处理大量的加密、解密工作，服务器处理 SSL 连接的数量非常有限，要维持端到端的持续连接，占用服务器资源非常多，使 SSL 在提供安全的同时也带来了一些使用上的局限性，严重影响客户的访问速度。

为了实现对 Internet 的加速，此次中国万网采用了 Intel 网擎电子商务加速器。Intel 的网擎电子商务加速器是采用专门设计的软硬件结构处理 SSL，从后台服务器（如 web server）卸载 SSL 处理工作，减轻服务器的负载。它具有处理速度快（最大 200 个）、兼容任何后台 web 服务器、可通过级连成倍提高 SSL 处理能力，并且易于安装、在出现故障或断电后，数据可以透明通过等优点。目前，中国万网的多品牌服务产品 - “明星”、“皓月”、“旭日”中均已使用了专用的 SSL 加速设备，其卓越的加密加速服务使这些问题将最终得到有效的解决。

中国万网采用这种高性能的服务设备，既保证了适应 Internet 的机制的数据安全传输，又有效地节省了客户的访问时间，为广大企业客户提供了一种安全高效的网站服务解决方案。它的成功利用，不仅提高了其向广大用户提供更加优质的网站服务水平，而且使其整个品牌在业界更加完整、流畅、可以信赖。

据悉，为致力于向客户提供更加优质的服务，建设最先进的网站平台，彻底落实其“客户在我心中”的服务目标，中国万网还将陆续投资推出更多的技术服务项目。

在 Borland C++ Builder 中编辑 WMF 图元文件

罗明宏

摘 要 图元文件是 Windows 操作系统支持的一种文件格式，在实践中得到广泛地使用，本文介绍了如何利用 BCB 所支持的图元文件类并结合 Win32 API 中有关图元文件的函数实现对图元文件的编辑。

关键词 图元文件 Windows API 函数

一、引言

在我们开发的某 CAPP（Computer Aided Process Planning，计算机辅助工艺过程设计）系统中，使用了 BCB 提供的 QuickReport 2.0 设计了各种工艺卡片，其支持的图形文件格式为 BMP，ICO，WMF 和 EMF 四种，我们选用了 WMF 和 EMF 格式，它们的矢量性质保证了等比例拉伸压缩时不会变形，使打印出的工艺卡片能清楚地表达零件的加工面。图 1 所示是该系统的机加工工序卡片的预览窗口。

图 1 机械加工工序卡片的预览窗口

当工艺设计人员使用 AutoCAD 按比例设计完成零件的加工图后，除了将加工图保存为 AutoCAD 自身支持的 dwg 文件归档外，还要为工艺卡片上使用的零件图输出 WMF 图，为了保证在工艺卡片上 WMF 图片不变形，我们按工艺卡片上的零件图框的尺寸在 AutoCAD 中设计了一个边界框，并定义成块，要求工艺人员把零件加工图按等比例缩小（或放大）后插入到这个边界框中，然后输出为 WMF 图。在卡片中，边界框的拉伸与零件图的拉伸是一致的，从而保证了加工图与边界框的比例一致，加工图与边界框的相对位置如图 2 所示。

但由于 AutoCAD 的限制，在输出 WMF 图时，我们并不能使 AutoCAD 只输出边界框及其中的零件图，边界框四周总会有一些我们不需要的空白边，如图 2 所示，因此我们要编程去掉多余的空白边。

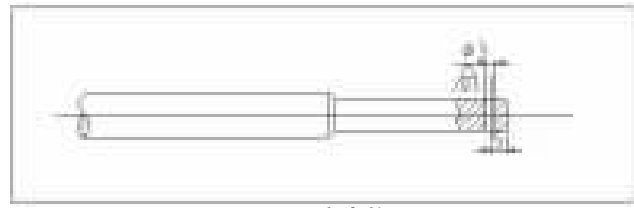


图 2 AutoCAD 产生的 WMF 图

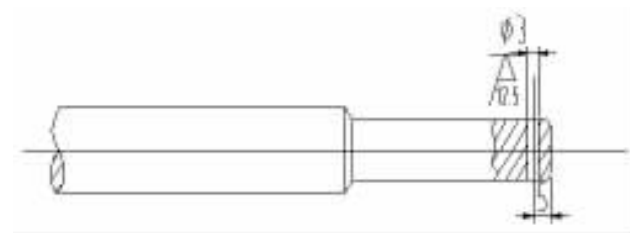


图 3 修改后的 WMF 图

我们让用户在边框内部单击鼠标，通过鼠标的坐标位置与图元文件中的记录一一比较，找到我们需要的边框记录，然后把图元文件的边界坐标用这个边框的坐标替换，这样就可以去掉多余的空白边。如图 3 所示。

二、有关图元文件的基本知识

图元文件（Metafile）是用设备无关格式来存储图像的一个结构集合。从内部讲，图元文件是一个被称为图元记录的可变长度结构的数组。在图元文件中，第一个记录指明了综合信息，诸如创建图元文件的设备的分辨率，图像的尺寸，谁创建的等等；剩下的记录组成了图元文件的主体，并且每一条记录对应的是一个图形设备接口（Graphic device interface，简称 GDI）函数。当应用程序创建了一个特殊的图元文件设备描述表（Device context，简称 DC）后，这个设备描述表被用于随后的所有的绘图操作，即 GDI 函数与图元文件的的 DC 关联起来，用于绘图的 GDI 函数被转换为合适的数据存在一条记录中，然后追加到图元文件中。在显示时图元文件按照先后次序回放（playbcak）这些函数就可以绘制出这幅图。

设备无关性是图元文件与位图文件（Bitmap）一个重要的区别点。比如，当一个应用程序在 VGA 显示器上创建了一个 10cm × 10cm 的图像，并且将它存储在图元文件中，然后即便

在 300dpi 的激光打印机上，图像依然能保持它的原始尺寸。由于图元文件是回放性质的，因此一般它都比位图文件要显示的慢一些，假如应用程序要求快显示而设备无关性又并非关键时，使用位图文件会更好。另外它内部保存的是一系列图形函数，因此照片或扫描的图片都不能较好转换为图元文件。但图元文件具有位图文件无法比拟的两个优点：

●非常小。一张图片用位图文件来存储需要上百 KB，而图元文件往往要求 1～2KB 就可以存储。

●可编辑。因为图元文件保存的是一系列图形函数，程序员就有能力编辑图片的内容。

图元文件的格式有两种，一种扩展名是 .WMF，它主要是用于 16 位 Windows 程序的，第二种格式的扩展名是 .EMF，是用于 32 位 Windows 程序的增强式图元文件（Enhanced Metafile），它比 16 位的 .WMF 格式有更多的描述信息，微软公司也推荐这种格式，所以我们对图元文件的讨论也主要集中在增强式图元文件上。

三、在 BCB 中有关图元文件的类

在 Borland C++ Builder（以下简称 BCB）的可视化附件库（Visual Component Library，简称 VCL）中已经封装了有关图元文件的类 TMetafile、TMetafileCanvas、TMetafileImage，其中 TMetafileImage 是图元类型的图片在 BCB 中内部使用的类，其所有的属性和方法都是私有的，一般在编程中不使用。TMetafile 类是操纵图元文件的基础类，它提供了相当丰富的属性和方法，使程序员很方便地使用图元文件。有关类 TMetafile 和 TMetafileCanvas 的属性、方法大家可查阅 BCB 的帮助文件，其中详细介绍了这两个类的使用方法。这里需要强调的是如果我们只想显示一个图元文件，那么只使用 TMetafile 类就足够了，但如果想创建图元文件后再往图元文件里绘制一幅图形，就必须使用 TMetafileCanvas 类，所以我们可以把 TMetafileCanvas 类认为是画一张图元类图片的画布，它与特定的 TMetafile 对象相联系。

另外，前面提到图元文件有 16 位（WMF）和 32 位（EMF）之分，但我们在使用 TMetafile 类时完全不必考虑它们的区别，TMetafile 类在读入时会自动的把 16 位的图元文件转换为 32 位，当需要保存图元文件时，只需要把 TMetafile 类里的 Enhanced 属性设置为 true，图元文件会被保存为 32 位格式；若设为 false 将被存为 16 位格式，使我们不用再考虑兼容问题。

在 Windows API 函数中属于图元文件操作的函数就有 28 个之多，尽管 BCB 中的 TMetafile、TMetafileCanvas 类已经封装了其中的大部分函数，使我们能实现操纵图元文件的大部分功能，但是这些类并没有封装对图元文件的编辑功能的函数，大部分有关图形，尤其是计算机辅助设计（CAD）的应用程序都需要有能编辑图元文件的能力，所以 BCB 不能满足上述应用程序的要求，解决的方法就是用 Windows API 的图元文件编辑函数，结合 BCB 的类来实现编辑图元文件的功能。

四、编辑图元文件

编辑图元文件有以下几个基本步骤：

●显示图元文件；

●获取鼠标的当前位置，这个位置应该是用户想修改的对象（比如线，圆弧，矩形等等）的位置；

●将这些位置转换为逻辑坐标；

●调用 EnumEnhMetaFile 函数来检查每一个图元文件的记录；

●判断记录是否是 GDI 绘图函数；

●如果是，判断鼠标位置是否在这些绘图函数坐标范围内；

●找到用户想要修改的记录后，在屏幕上删除这条记录；

●在图元文件中删除这条记录；

●让用户重绘；

●将用户新绘对象的 GDI 函数转换为一条或多条记录；

●保存记录。

可见编辑图元文件是一个复杂的工作，但我们可以使用 EnumEnhMetaFile 函数和它相关联的回调（callback）函数 EnhMetaFileProc 结合起来就能够处理图元文件中的每一条记录。下面我们对这两个函数做简要的介绍。

使用 EnumEnhMetaFile 来列举增强的图元文件格式中的每一条记录，并将它传递给指定的回调函数。我们在这个回调函数中处理记录，完成预定工作。当最后一条记录被处理完，或者回调函数返回 false，列举才会结束。

```
BOOL EnumEnhMetaFile(
    HDC hdc,        //设备上下文句柄,将被传递给回调函数
    HENHMETAFILE hemf,    //标识一个图元文件的句柄,
    ENHMFENUMPROC lpEnhMetaFunc, //回调函数指针
    LPVOID lpData,        //回调函数所使用的数据,可选
    CONST RECT *lpRect //使用逻辑单位指定图片的范围,
假如 hdc 参数为 NULL,系统不考虑 lpRect
  );
```

EnhMetaFileProc 函数是应用程序定义的回调函数，并且它只是程序员所使用的函数名的替代名称。

```
int CALLBACK EnhMetaFileProc(
    HDC hDC,        //由 EnumEnhMetaFile 函数传递的句柄
    HANDLETABLE FAR *lpHTable, //图元文件的句柄表指针
  ENHMETARECORD FAR *lpEMFR, //图元文件的记录指针
    int nObj,  //句柄表中相关句柄的对象个数
    LPARAM lpData      //程序员使用的数据
  );
```

两个 API 函数中与我们密切相关的参数就是 lpData，我们通过定义自己需要的数据结构，在此基础上申明一个变量，就可以在回调函数内获得必要的数据，也可以把我们在图元文件中获得的数据传递出来。

五、应用程序的实现

1. 我们新建一个工程，并按照表 1、表 2 设置各组件的属性。

表 1 主窗体及其组件、属性和设置

组件名	属性	设置
MainForm	Caption	WMF Editor
	Color	clBlack
	Menu	MainMenu
	WindowState	wsMaximized
Image	Autosize	True
	Center	True
OpenDialog	Filter	Metafiles(*.wmf)|*.wmf Enhanced Metafiles(*.emf)|*.emf
SaveDialog	Filter	Metafiles(*.wmf)|*.wmf Enhanced Metafiles(*.emf)|*.emf

表 2 MainMenu 的菜单项

菜单名	菜单项目名	属性	设置	事件
File		caption	文件	
	FileOpen	caption	打开	FileOpenClick
Edit		caption	编辑	
	EditSnap	caption	捕捉	EditSnapClick

2. 选择 File|New Unit 创建一个新的 meta.cpp 文件，并在其头文件中加入一个新类 CEnhMetaFile 和结构 HTDATA 的申明，其中 HTDATA 用于 EnumEnhMetaFile 函数和 EnhMetaFileProc 函数，传递鼠标拾取点位置和边框尺寸。

```
class CEnhMetaFile {
public:
    CEnhMetaFile();
    ~CEnhMetaFile();
    bool Read(AnsiString strFileName);;
    bool EditRecord();
    bool FindRecord(int x, int y);
    void Save(AnsiString strFileName)
    TMetafile *mMetafile;
};
typedef struct _htdata {
    RECT    rect;
    int     iRecord;
}HTDATA;
```

3. 在 meta.cpp 文件中实现 CEnhMetaFile 的构造函数、析构函数和成员函数。

在 CEnhMetaFile 的构造函数和析构函数创建和删除 mMetafile 数据成员，利用这个数据成员实现加载和显示图元文件。

成员函数 Read 主要是把图元文件读入内存，并与 mMetafile 关联起来。

成员函数 Save 把修改后的图元文件保存起来。

成员函数 FindRecord 把鼠标在捕捉状态下单击 Image 控件时的坐标位置作为输入参数，将坐标与待查找的记录 边框）的边界进行比较，若查找成功并将图元文件修改后返回 ‘真”，否则返回 ‘假”。

```
BOOL CEnhMetaFile::FindRecord(int x, int y)
{
    ENHMETAHEADER EnhMetaHdr;
    HTDATA  htData;
    float A, B, C, D;
    ::GetEnhMetaFileHeader((HENHMETAFILE)mMetafile->Handle, sizeof(EnhMetaHdr), &EnhMetaHdr);
    //计算从窗口到视图的变换系数
    A = (float)(EnhMetaHdr.rclBounds.right - EnhMetaHdr.rclBounds.left) /(float)mMetafile->Width;
    B = 0 - A* 0;
    C = (float)(EnhMetaHdr.rclBounds.bottom - EnhMetaHdr.rclBounds.top) /(float)mMetafile->Height;
    D = 0 - C * 0;
    TPoint SnapP;
    SnapP.x = A * x + B;
    SnapP.y = C * y + D;
htData.rect=Rect(SnapP.x, SnapP.y, SnapP.x, SnapP.y);
    htData.iRecord = EnhMetaHdr.nRecords;
    //在图元文件记录中查找匹配的记录
    bool bSuccess = ::EnumEnhMetaFile(/* mCanvas->Handle, */NULL, (HENHMETAFILE) mMetafile->Handle, (ENHMFENUMPROC) bSearchRecord, (LPVOID) & htData, (LPRECT)& recDisBounds);
    if ( bSuccess ) return false;    //没有匹配的记录
    HTDATA htBaseBox=htData;    //获取匹配记录的数值
    //修改记录
    ::EnumEnhMetaFile(/* mCanvas->Handle, */NULL, (HENHMETAFILE)mMetafile->Handle, (ENHMFENUMPROC) bConverWindow, (LPVOID)& htBaseBox, (LPRECT)& recDisBounds);
    return TRUE;
}
```

4. Windows API 函数 EnumEnhMetaFile 枚举了所有图元文件中的记录，其中使用的两个回调函数分别是 bSearchRecord 和 bConverWindow，说明如下：

bSearchRecord 函数把拾取点与图元文件中的边框位置进行比较，如果拾取点在边框内部，则返回 ‘假”，并把边框的坐标也取出。

```
bool CALLBACK bSearchRecord(HDC hDC, LPHANDLETABLE lpHandleTable, LPENHMETARECORD lpEnhMetaRecord, UINT nHandles, LPVOID lpData)
{
    static int iRecordCnt=0;
    iRecordCnt++;
    //保存鼠标的拾取位置
    RECT rcMouseBox = ((HTDATA *)lpData)->rect;
    switch (lpEnhMetaRecord->iType) {
      case MR_POLYLINE16: {
     RECTL rclBounds; //Bounding rectangle, in device units
       rclBounds.left = lpEnhMetaRecord->dParm[0] -2;
       rclBounds.top = lpEnhMetaRecord->dParm[1] -2;
       rclBounds.right = lpEnhMetaRecord->dParm[2] +2;
     rclBounds.bottom = lpEnhMetaRecord->dParm[3] +2;
        //justice if mouse position in the targe rectange
   if ((rclBounds.left>rcMouseBox.left)||(rcMouseBox.left>rclBounds.right))
          return TRUE;
   if ((rclBounds.top>rcMouseBox.top)||(rcMouseBox.top>rclBounds.bottom))
          return TRUE;
```

```
//获取边框的边界坐标的个数和值
      int iPtCnt = lpEnhMetaRecord->dParm[4];    //个数
      TPoint *pPtTmp = new TPoint[iPtCnt];
      if (pPtTmp == NULL) {
ShowMessage("ERROR: Failed in Memory Allocation!");
        return TRUE;
      }
       for (int i=0; i < iPtCnt-1; i++) {          //值
        pPtTmp[i].x = (int)(LOWORD(lpEnhMetaRecord->
dParm[i+5]));
        pPtTmp[i].y = (int)(HIWORD(lpEnhMetaRecord->
dParm[i+5]));
       }
           //将边框的坐标保存在一个表示矩形框的数组中
       TPoint ptMax=pPtTmp[0], ptMin=pPtTmp[0];
       for (int i=0; i < iPtCnt-1; i++) {
        if ( ptMax.x < pPtTmp[i].x )   {
          ptMax.x = pPtTmp[i].x;
        }
        else if ( ptMax.y < pPtTmp[i].y ) {
          ptMax.y = pPtTmp[i].y;
        }
        if ( ptMin.x > pPtTmp[i].x )   {
          ptMin.x = pPtTmp[i].x;
        }
        else if ( ptMin.y > pPtTmp[i].y )   {
          ptMin.y = pPtTmp[i].y;
        } }
     ((HTDATA *) lpData)->rect=Rect(ptMin.x, ptMin.y,
ptMax.x, ptMax.y);
       iRecordCnt=0;
       return FALSE;
     }
     default:
      if (iRecordCnt >= ((HTDATA *) lpData)->iRecord) {
        iRecordCnt = 0;
        return TRUE;
      }
      return TRUE;
   }  //switch end
}
```

bConverWindow 函数把所取到的边框坐标变换为图元文件中的边框。

```
bool CALLBACK bConverWindow(HDC hDC, LPHAN-
DLETABLE lpHandleTable, LPENHMETARECORD lpEnhMeta-
Record, UINT nHandles, LPVOID lpData)
{
   RECT    rcBase;
   static SIZEL SizeViewEx;
   static int  iRecordCnt=0;
   //store the basic pickbox
   rcBase = ((HTDATA *)lpData)->rect;
   iRecordCnt++;
   switch (lpEnhMetaRecord->iType) {
    case MR_RECTANGLE:  {
      lpEnhMetaRecord->dParm[0] = (LONG)rcBase.left;
      lpEnhMetaRecord->dParm[1] = (LONG)rcBase.top;
    lpEnhMetaRecord->dParm[2] = (LONG)rcBase.right;
    lpEnhMetaRecord->dParm[3] = (LONG)rcBase.bottom;
      break;
    }
    case MR_SETWINDOWORGEX:  {
      lpEnhMetaRecord->dParm[0] = (LONG)rcBase.left;
      lpEnhMetaRecord->dParm[1] = (LONG)rcBase.top;
      return TRUE;
    }
    case MR_SETVIEWPORTEXTEX:  {
   SizeViewEx.cx = (int)(lpEnhMetaRecord->dParm[0]);
   SizeViewEx.cy = (int)(lpEnhMetaRecord->dParm[1]);
      break;
    }
    case MR_SETWINDOWEXTEX:  {
      SIZEL szlExtent;
      szlExtent.cx = (int)lpEnhMetaRecord->dParm[0];
      szlExtent.cy = (int)lpEnhMetaRecord->dParm[1];
      if ( (szlExtent.cx == SizeViewEx.cx) &&
           (szlExtent.cy == SizeViewEx.cy) )
         break;
      else   {
        szlExtent.cx = rcBase.right - rcBase.left;
        szlExtent.cy = rcBase.bottom - rcBase.top;
lpEnhMetaRecord->dParm[0] = (LONG)szlExtent.cx+20;
lpEnhMetaRecord->dParm[1] = (LONG)szlExtent.cy+20;
        return TRUE;
      } }
    default:
      break;
   }  //switch end
   if (iRecordCnt < ((HTDATA*)lpData)->iRecord)
    return TRUE;
   else   {
    iRecordCnt = 0;
    return FALSE;
   }}
```

六、几点说明

1. 本程序只是作为一个编辑图元文件的演示，只考虑了AutoCAD产生的图元文件，并没有考虑其它绘图软件产生的图元文件。

2. 由于AutoCAD产生的图元文件，矩形、圆形、圆弧等较复杂的图形均用多义线（polyline）实现，所以程序只处理了多义线的情况。

3. 本程序没有考虑若鼠标拾取点附近有几个嵌套的满足条件的矩形时，程序将不判断哪个矩形是边框，它会按图元文件中这些矩形的先后记录选取其中的第一个矩形作为图元文件的新边框。感兴趣的读者可自行添加矩形判断的代码。

4. 本程序已经在Borland C++ Builder 4.0下调试通过。

参考资料

1. Microsoft Solutions Development Kit SDK
2. Borland C++ Builder 4.0 Help

（收稿日期：2000年8月18日）

在 linux 环境中向 PHP 中加入自编函数的方法

方 翔

一、简介

1994 年秋季，Rasmus Lerdorf 开始构思 PHP。 1995 年年初第一版本出台，当时 PHP 只被认为是个人主页开发工具。但是到了 1999 年，根据 NetCraft 提供的数据推断，保守估计全世界应用 PHP 的网站已超过 150 000 个。PHP 已经成为一种影响很深的语言。它之所以能迅速被大家接受，主要因为它有以下几个优点：

1. PHP 是一种跨平台的服务器端的嵌入式脚本语言。 它大量地借用 C，Java 和 Perl 语言的语法，并耦合 PHP 自己的特性，使 WEB 开发者能够快速地写出动态生成页面。

2. 它支持目前绝大多数数据库，是开发电子商务应用的利器。

3. PHP 是完全免费的。 PHP 是 2000 年兼容的。

4. PHP 通过协议也支持与其他服务的 “交谈”，像 IMAP， SNMP， NNTP， POP3，甚至是 HTTP。 你也可以打开晦涩的 网络接口和其他协议交互。

5. 借助与 C + + 的形式，引用类的概念，使得代码的可重复性应用便的异常简单。

6. 可移植性好，无须做任何修改就可在不同的机器上运行。

二、加入自编函数的方法

尽管 PHP 为我们提供了大量的功能强大的函数，但是有时候我们可能还是会觉得它们无法满足自己的特殊的需要，于是用 C 语言编制自己的函数，并使它们能和 PHP 自带的函数一样方便使用就成为我们迫切解决的一个问题。本文根据作者的经验，总结了三种可以实现此功能的方法，现介绍如下：

1. 在 PHP 自身函数中加入代码的方法

通过跟踪 PHP 某些简单函数的调用和运行情况，我发现通过下面几步可以达到目的：

（1）首先在 PHP 的任意一个函数的源代码中加入自己函数的原型，它有特定的格式，一般为

```
void function_name（internal_function_parameters）
{
//函数内容
}
```

具体的规则可以参看 apidoc. txt 文件或帮助文件中的 php developer 一节。

（2）在定义此文件的接口的文件中将自己函数的接口加入。如我将函数加在了 math. c 中，而 math. c 中的函数接口在 intenal_functions. c 中定义，所以在此文件中找到 function_entry functions[] = {. . . . }

在里面加入{ “调用时的函数名”，函数定义时的函数名，参数}

（3）在加入函数的文件的头文件中加入函数声明，如 external void myfunction internal_function_parameters

完成上述各步后，对 PHP 进行重编译即可。这种方法的优点是简便，但是与 PHP 的函数混在一起，不宜管理和日后对自己函数的升级。

2. 在新建文件中写自己的函数

主要步骤与上面相同，只不过函数定义应写在自己建立的文件中，函数声明也要写在对应的头文件中。另外附加一点，必须在 Makefile 文件中，加入新建文件的编译项。具体写法可参照 Makefile 中其他文件的编译项。

同样对 PHP 进行重新编译后，就可使用自己的函数了。这种办法弥补了上一种方法的缺陷，不失为静态编译中的好方法。

以上两种方法虽然可以达到我们的目的，但是它们的实现都是有代价的，即它们必须在 PHP 启动时就装载，也只有在 PHP 关闭时才卸载。这就极大的浪费了系统的资源，因为我们自己编写的函数虽然是必须的，但是调用的频率是很小的。如何使得我们需要这些函数时系统才去调用它们？这样就引出了第三种方法，用动态装载函数。

3. 动态装载函数

在 PHP3 中提供了一个函数 dl ，它是专门调用动态装载函数的。但是它又不能直接调用动态装载函数，而是必须通过扩展名为 so 的动态装载模块来实现。由于这种方法有很多优点，所以我将举例详细说明。

比如我要建一个自己的函数名为 hello_world，功能也就是打印 “hello world！”，具体实现步骤如下：

（1）为了便于管理，建立目录 mymodule 和其子目录 src

（2）在目录下建立 Makefile 文件，内容如下：

```
CC = cc -O2 -Dbool=char -DHAS_BOOL \
-I/usr/local/include \
-I/YourPHPsourceDirectory/ -fpic
LD = cc -shared -L/usr/local/lib -rdynamic
```

```
all:    myfun.so                //生成的动态装载模块名
HELLO_OBJS = src/myfun.o
myfun.so:       $(HELLO_OBJS)
$(LD) -o $@ $(HELLO_OBJS)
src/myfun.o: src/myfun.c    //myfun.c 中即为我们要添加的函数定义
$(CC) $(CFLAGS) -DCOMPILE_DL=1 -c -o $@ $<
```

这些内容实际上是为 make 命令提供参数，告诉它编译的源文件、路径和方法。

注意：在第一行和第二行的 h后不能有其他字符　包括空格）否则将违反它的格式要求。

5）然后进入 /src 目录，编写 myfun.c 文件，写入要调用的函数。在此以显示 "hello world" 的函数 hello_world 为例，来说明写调用函数必须遵循的规则：

```
#include "../phpdl.h"           //供 dl()函数使用
DLEXPORT void hello_world(INTERNAL_FUNCTION_PARAMETERS); //声明函数
//函数入口的原型是{"php 中调用的函数名", 你的 c 程序名, 1}
function_entry MyModule_functions[] = {
    {"hello_world", hello_world, NULL},
    {NULL, NULL, NULL}
};
//在 php 中注册你的函数
php3_module_entry myfun_module_entry = {
 "myfun", myfun_functions,
 NULL, NULL, NULL, NULL, NULL, 0, 0, 0, NULL
};
#if COMPILE_DL
DLEXPORT php3_module_entry *get_module(void) {
    return &myfun_module_entry;
}
#endif
//你的函数原型
DLEXPORT void hello_world(INTERNAL_FUNCTION_PARAMETERS) {
    char MyFunction[15];
    sprintf(MyFunction, "Hello World!");
    //创建返回值，用 estrdup()为字符串分配内存
    return_value->value.str.val = estrdup(MyFunction);
    return_value->value.str.len = strlen(MyFunction);
    return_value->type = IS_STRING;
}
```

4. 然后返回目录 /module，用 make 进行编译，系统会在此目录下产生一个动态装载模块 myfun.so。

完成这些工作后，即可在 php 模块中调用 hello - world 函数。方法如下：

```
<?
    dl("/path/to/myfun.so"); //装载模块
    $MyVar = hello_world(); //调用函数
    echo $MyVar;
?>
```

以上例子已经在 turbolinux 中文版 6.1 和 PHP3.0.16 环境中通过。

三、总结

本篇文章所介绍的三种方法可以在不同的环境中根据不同的需求来调用。它们使的我们可以用 C 语言编写自己需要的函数，不但扩充了 PHP 本身的功能，增加了其可用性，而且调用简单，函数的执行效果也与 PHP 本身的函数相差无几。也正是由于 PHP 的这种强大的可扩充性，它才得以得到广大编程人员的喜爱，并且在他们的支持下，PHP 也会得到跟好的发展。

收稿日期：2000 年 10 月 30 日）

杭州新中大软件股份有限公司正式成立

为实现新中大 '产业化、规模化、股份化和国际化" 发展的需要，经过一年多时间业务重组和充分准备，日前杭州新中大软件股份有限公司在 21 世纪即将到来之际正式成立。股份公司的成立将给新中大注入了新的生机和活力，并赋予新中大在国际化竞争中极强的生命力和战斗力。

新中大是一个充满发展和竞争潜力的高科技企业。几年来，在全体新中大人的共同努力和奋斗下，新中大取得的辉煌的成绩，并形成了自己独特的经营理念和企业文化。1999 年 5 月，新中大与 '轻纺城" 成功实现强强联合，这是技术资本与实物资本的一次完美结合，有人把这次合作称为 '中国式的创业投资"。借助这次成功的合作，新中大在一年多的时间里实现了一次快速的飞跃：新中大品牌知名度、信誉度和美誉度得到了全面的提升；新中大产品线得到了不断的优化和完善；新中大的营销和服务网络也不断完善和扩大；新中大的用户群不断扩大……。1999 年度 CCID 报告显示，新中大的综合实力进入行业三强。新中大已从几年前一家起步较晚、自谋生存发展的软件企业发展为今天业界最具竞争力的少数几个领导厂商之一。2000 年 9 月，新中大又一次成功的吸引了二十一世纪科技投资有限公司和深圳市创新科技投资有限公司两家具有丰富科技风险投资背景和投资经验的公司成为新中大公司的战略股东，给新中大注入了丰富的科技资源和管理经验，为新中大下一阶段的资本扩张和跳跃式发展奠定了坚实的基础。

当前，全球经济正步入一个全新的信息化时代，信息化作为一种世界性潮流，正在深刻改变着人类社会的生产和生活方式，信息技术已逐步成为推动生产力进步的重要因素，WTO、Internet、E - Commerce 等等无疑都给我们提供了一个绝好的发展机遇。杭州新中大软件股份有限公司的成立，无疑使新中大拥有全新而更高的起点站在新世纪的起跑线上，新中大将以全新的姿态去迎接新世纪的挑战，并在新世纪里取得更大的辉煌，实现更高的飞跃！

用 Delphi 构建三层分布式应用程序

何希平

摘 要 本文介绍了 Delphi 中实现三层分布式应用服务的体系结构、组件层次和连接协议。并介绍了 Delphi 用 DCOM 连接的这种服务的开发技术的实现过程。

关键词 三层分布式应用服务，体系结构，协议，组件，Delphi

一、三层分布式应用服务的体系结构

三层应用服务是继两层 C/S 结构发展到基于中间件的三层 C/S 结构之后，为在 Internet/Intranet 平台上实现分布式企业级复杂计算、解决服务器的负载平衡、可伸缩性、可靠性和安全性等问题、基于多种通信协议和分布式组件对象技术而发展起来的服务体系（见图 1）。

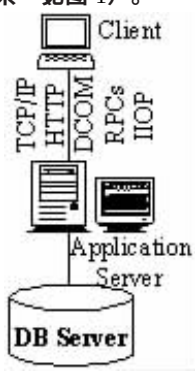


图 1 三层 C/S 结构

三层分布式应用服务提供了客户机应用程序（Client Application）以及应用服务器（Application Server）与数据库服务器交流信息的机制。该服务体系中，客户机通过多种协议和工业标准（HTTP、TCP/IP、OLEnterprise、COM、DCOM、COM+、CORBA）访问应用服务器，而应用服务器访问数据库服务器。客户机、应用服务器、数据库服务器分别位于 Internet/Intranet 分布式计算环境中。

表 1 几个专门用于构建分布式应用的关键组件

组 件	用 途
远程数据模块	充当 COM 自动化服务器或 CORBA 服务器的特定的数据模块，用于服务器上给客户机应用程序提供对它们所包含的任何数据集供应者的访问。
数据集供应者组件	用在应用服务器上的数据代理者，通过创建数据包和解决客户更新提供数据。
客户机数据集组件	用 MIDAS.DLL 来管理保存在数据包中的数据的特定数据集。
连接组件	一组查找服务器、形成连接和构建 IAppServer 接口以便客户机数据集使用的组件。每一连接组件专门使用一种特定的通信协议。

表 1 中介绍了 Delphi 中几类专门用于构建分布式应用的关键组件。图 2 给出了使用 DCOM 技术实现的三层结构中各组件间的逻辑关联关系。

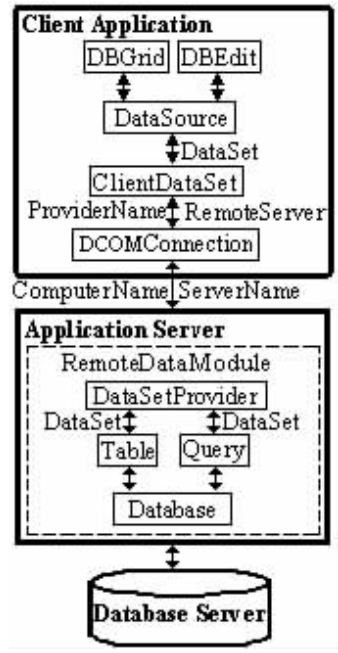


图 2 三层 C/S 结构中组件的逻辑联系

1. 应用服务器的结构

应用服务器的数据处理组件的层次关联关系见图 2）含有提供 IAppServer 接口的远程数据模块。与其它任何数据模块一样，远程数据模块（参见表 2）中可含任何非可视的数据访问组件。而且，它还包含便于客户机应用程序使用应用服务器中每一数据集的数据集供应者组件。数据集供应者组件（Delphi 中为 TDataSetProvider）的功能是：

●接收来自客户机的数据请求，取回向数据库服务器请求的数据，为传输打包数据，发送数据给客户机数据集。这一活动称为“供应”（“Providing”）。

●接收来自客户机数据集的更新数据，将更新应用于数据库或者源数据集，并记录任何未能得到应用的更新，将未解决的更新返回给客户机以便进一步的协调。这一活动称为

表 2　Delphi 中的三种远程数据模块（服务器）

远程数据模块	用　途　说　明
TRemoteDataModule	这是一个双向接口自动化服务器。除非你想将应用服务器与 MTS 一起安装，否则若客户机用 DCOM、HTTP、Sockets 或 OLEnterprise 连接到应用服务器，则使用这种远程数据模块。
TMTSDataModule	这是一个双向接口自动化服务器。若你打算创建一个与 MTS 一起安装的动态链接库（DLL）形式的应用服务器，则使用这种远程数据模块。你可将 MTS 远程数据模块与 DCOM、HTTP、Sockets 或 OLEnterprise 一起使用。
TCorbaDataModule	这是一个 CORBA 服务器。用这种远程数据模块给 CORBA 客户机提供数据。

表 3　Delphi 提供的连接组件及其使用的协议

组件	协议
TDCOMConnection	DCOM
TSocketConnection	Windows sockets　TCP / IP
TWebConnection	HTTP
TOLEnterpriseConnection	OLEnterprise　RPCs
TCorbaConnection	CORBA　IIOP

"解决"（"Resolving"）。

2. 客户机应用程序的结构

对于最终用户，客户机应用程序表面看来与传统的两层应用程序没什么不同。结构上，客户应用程序与用户交互通过标准的用于显示来自于客户机数据集组件的数据的数据敏感组件进行（即各数据敏感组件的数据源组件的数据集是客户机数据集）。

客户机数据集组件 TClientDataSet 通过连接组件提供的应用服务器上的远程数据模块的 IAppServer（Interface of Application Server，用于客户应用程序与应用服务器上的数据提供者通信）获得数据，并用这种接口将更新转交给应用服务器，实现数据请求和提交。

连接组件建立与应用服务器的连接。不同的连接组件适用于使用不同的通信协议的场合。表 3 中概述了 Delphi 中对应的连接组件。

二、用 DCOM 连接的三层分布式应用服务设计

1. 数据库设计

首先应根据应用需要，选择数据库平台如 Oracle、MS SQL Server、DB 2、Sybase、Informix 等。然后再进行数据库系统的设计。为说明开发的全过程，本文选用 ODBC 驱动的本地数据库 Paradox 作为示范（不考虑系统设计的完整性）。用 Database desktop 建立表 4、表 5 所示的表 Stu. db 和 Sub. db（不用别名），并保存在指定目录（如："D:\三层 CS"）：

表 4　学生情况表 Stu. DB

学号	姓名	性别	出生日期	系别	专业	所在班级
07981804	董古欧	男	1980 - 1 - 4	计算机科学系	计算机应用	1 班
07981805	何平磊	男	1980 - 5 - 24	计算机科学系	计算机应用	1 班
07981806	黄新欣	男	1980 - 7 - 14	计算机科学系	计算机应用	1 班

表 5　课程情况表 Sub. DB

课程号	课程名	周学时	学分	类型
98000001	英语	4	5	外语
98000002	高等数学	6	8	数学
98000003	汇编语言	4	5	计算机科学

2. 编写应用服务器程序

按下列步骤进行：

1　新建一工程，再选择 New | Multitier | Remote Data Module，在 CoClass 编辑框中输入：AppServerAPI；

2　选择 File | Save All，将 Unit1. pas、Unit2. pas、Project1. dpr 保存到刚才所建表所在的目录（如："D:\三层 CS"）中的 ServerUn. pas、DataPrvMun. pas、AppServerPrj. dpr；

3　在远程数据模块 AppServerAPI 中添加表 6 中所列组件，并按表中所列设置各组件的属性值；

表 6　远程数据模块 AppServerAPI 中的组件

组　件	属　性	值
Database1	DatabaseName	DBConnection
	DriverName	STANDARD
	Params	PATH = . DEFAULT DRIVER = PARADOX ENABLE BCD = FALE
Table1	DatabaseName	DBConnection
	TableName	Stu. db
	Active	True
DataProvider1	DataSet	Table1
DataProvider2	DataSet	Query1
Query1	SQL	Select * from Sub. DB
	DatabaseName	DBConnection
	Active	True

4　编写自己的数据处理代码，编译运行该应用服务器程序。

3. 编写应用服务器程序

按下列步骤进行：

1　新建一工程 ClientApp 和一窗体单元 ClientUn，并保存在前面的目录（如："D:\三层 CS"）下；

2　在 Form1 上添加表 7 中所列组件，并按表中所列设置各组件的属性值；

3　编译运行该客户机应用程序。

三、补白

上面的应用服务器程序第一次运行时，会自动在系统中注册；而调试、运行客户机应用程序时会自动启动应用服务器程

表 7 Form1 上添加的组件

组件	属性	值
DCOMConnection1	ComputerName	应用服务器所在计算机名 如 yzucmp. cq. cn
	ServerName	AppServerPrj. AppserverAPI
ClientDataSet1	RemoteServer	DCOMConnection1
	ProviderName	DataSetProvider1
	Active	True
ClientDataSet2	RemoteServer	DCOMConnection1
	ProviderName	DataSetProvider2
	Active	True
DataSource1	DataSet	ClientDataSet1
DataSource2	DataSet	ClientDataSet2
DBGrid1	DataSource	DataSource1
DBGrid2	DataSource	DataSource2
BitBtn1	Kind	bkClose

序，且当客户机应用程序运行结束后，应用服务器程序会自动终止运行；它们的开发在同一台计算机上进行，但最终运行时却可以与数据库服务器一道分布于网络中的不同计算机中；在调试客户机应用程序时，应先将已经启动的应用服务器程序关闭，否则，会因重复启动应用服务器程序程序而出错。

上述所有内容都是基于 Windows 98，并用 Delphi 5 实现和得到验证。

四、源程序清单

1. 应用服务器程序窗口单元

```
unit ServerUn; //ServerUn. pas
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *. DFM}
end.
```

2. 应用服务器程序远程数据模块单元

```
unit DataPrvMUn; //DataPrvMUn. pas
interface
uses
  Windows, Messages, SysUtils, Classes, ComServ, ComObj, VCLCom, DataBkr, DBClient, AppServerPrj_TLB, StdVcl, Provider, DBTables, Db;
type
TAppServerAPI = class(TRemoteDataModule, IAppServerAPI)
    Table1: TTable;
    Database1: TDatabase;
    Query1: TQuery;
    DataSetProvider1: TDataSetProvider;
    DataSetProvider2: TDataSetProvider;
  private
    { Private declarations }
  protected
    class procedure UpdateRegistry(Register: Boolean; const ClassID, ProgID: string); override;
  public
    { Public declarations }
  end;
implementation
{$R *. DFM}
class procedure TAppServerAPI. UpdateRegistry(Register: Boolean; const ClassID, ProgID: string);
begin
  if Register then
  begin
    inherited UpdateRegistry(Register, ClassID, ProgID);
    EnableSocketTransport(ClassID);
    EnableWebTransport(ClassID);
  end else
  begin
    DisableSocketTransport(ClassID);
    DisableWebTransport(ClassID);
    inherited UpdateRegistry(Register, ClassID, ProgID);
  end;
end;
initialization
  TComponentFactory. Create(ComServer, TAppServerAPI,
    Class_AppServerAPI, ciMultiInstance, tmApartment);
end.
```

3. 应用服务器主程序（工程文件）

```
program AppServerPrj; // AppServerPrj. dpr
uses
  Forms,
  ServerUn in ´ServerUn. pas´ {Form1},
  AppServerPrj_TLB in ´AppServerPrj_TLB. pas´,
    DataPrvMUn in ´DataPrvMUn. pas´ {AppServerAPI: TRemoteDataModule} {AppServerAPI: CoClass};
{$R *. TLB}
{$R *. RES}
begin
  Application. Initialize;
  Application. CreateForm(TForm1, Form1);
  Application. Run;
end.
```

4. 客户机程序窗口单元

```
unit ClientUn; // ClientUn. pas
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Con-
```

（下转第 63 页）

利用 DDE 技术以 Avenue 语言创建 Word 文档

李伟忠编译

摘 要 本文简单回顾了 DDE 技术的常识，通过分析 DDE 技术在 ArcView GIS 与 Word 系统中的使用，阐述了通过 DDE 技术利用 ArcView 的开发语言 Avenue 编程实现 Word 文档的建立。

关键词 DDE 技术，Avenue 语言，Word 文档，ArcView

前言

ArcView GIS 是美国 ESRI（环境系统研究所）开发的一个功能强劲、易于上手的桌面地理信息系统软件，可以方便地数据进行空间浏览、查询、分析，并提供了面向对象的二次开发语言 Avenue。

“世事无完美”，ArcView 的一个最大的不足就是没有一个真正的报告生成器。Layout 对于一个格式已固化的数据群是不错的，但如果用户想将多幅图片与大段的文字及经过分组、分类、格式化的表格组合到一起，就需要到 ArcView 系统外寻求帮助了。

ArcView 3.1 附带了 Crystal 报告编写扩展模块，它在 ArcView 和独立的 Crystal 报告器应用程序间起一个连接的作用。从该应用程序的文档中知道，该系统似乎是一个特征化了的报告编写器，由于 CR（即 Crystal Report）生成的报告与 ArcView 表格的数据紧密联系，这种特征化连接可能有内在的约束性；而且目前还不清楚是否可以将 ArcView 中的一幅 View 或 Layout 插入到 Crystal 的报告中。

另一个办法是通过编程由诸如 MS Word 的文本编辑器创建报告。虽然有大段的代码要写，但可以操纵 Microsoft Word 的所有属性，包括字符格式，段落格式，表格，插入的图片等等。由 MS Word 制作的报告看起来相当的专业，而且还有另外的优势——可以对文档进行再编辑——这正是 Layouts 和 Crystal 报告所不能匹敌的。

Word 97 支持灵活的编程语言 VB，通过它可以很方便地创建 Word 文档。但是 ArcView3.X 不支持 OLE 技术，这意味着它不能获取 Word 的对象模式。这增加了工作的难度。对此有两种办法解决，其一是创建一个可被 ArcView 调用的、适应 OLE 的应用程序，如 VB 或 C++的 DLL，此 DLL 可以通过 OLE 进行文档的创建。ArcView 的编程语言 Avenue 可以调用此 DLL，然后就可以通过它操纵 Word 的对象与方法以实现文档建立。这种方法的缺点是需要用版本为 6.0 或以上的 VB、C++（VB5 生成的 DLL 不能被 ArcView 调用）。

第二条途径，也就是本文将要描述的，是采用动态数据交换（DDE）技术。DDE 是使得应用程序协同工作的协议，目前大多情况下已为 OLE 技术所取代。但尽管如此，出于向后兼容的考虑，Windows 95/98 及 Microsoft Office 仍然支持此功能。ArcView 和 Avenue 同样支持。

DDE

尽管在很多书籍中有大量的关于 DDE 的描述，这里还是有必要简述一下 DDE 的基本概念：

1. 一应用程序（DDE 客户）打开与另一应用程序（DDE 服务器）间的一个 DDE 通道（又称为开始一个对话）；

2. DDE 客户向 DDE 服务器发出“Poke”、“Execute”指令（上述指令将在服务器程序中运行）和“Request”指令（此指令用以从服务器返回信息）；

3. 关闭 DDE 通道；

DDE 客户打开一个通道时，它必须指明就哪一个“主题”进行对话。服务器程序都有一些预设的主题。这些“主题”因应用程序而表现各异，但大多数的 DDE 服务器都支持“system”这一主题。在 Word 中还可以当前文档名作为主题。每次使用 Poke 或 Request 指令时，还必须指明一个有效的数据项。

ArcView 可以作为 DDE 客户，也可以担当服务器。当将 ArcView 作为服务器时，我们可以通过编程实现另一应用程序对 ArcView 的控制。但在利用其内置的开发语言 Avenue 创建 Word 文档的过程中，ArcView 是作为 DDE 客户出现的。在 Avenue 中，操纵 DDE 服务器的对象类是“DDEClient”，顾名思义，一个“DDEClient”就代表两个程序间的对话。该对话开始于利用 Make 指令建立一个新的 DDEClient 的时刻，如下句所示：

```
theDDE = DDEClient.Make "WINWORD" "system"
```

需要指出的是，DDE 服务器的名称（本文中为 Winword）

通常情况下为该程序可执行文件的名称（忽略 exe 后缀），同时该服务器程序应该是处于启动状态，这样才有可能打开通道。我们可以使用关于 DDEClient 的“HasError”指令来查看对话是否顺利开通。如果有误，可以使用指令“System. Execute”来启动服务器应用程序并再次尝试。一旦开通一个 DDE 对话，就可以通过指令“Execute”、“Poke”及“Request”进行与服务器的信息处理及传递了。

DDE 与 Microsoft Word 97

有关将 MS Word 作为 DDE 服务器的资料并不多见，除非有 Office 开发手册。可以得到的最详细的在线资料是出自 MS Knowledge Base 的一篇文章。另外站点 http://microsoft. public. word. oleinterop 上也有些帮助。下面叙述一些重点内容：

Word 支持两类 DDE 主题（在启动与其他程序的 DDE 对话时必须指明使用的主题）：首先，和所有支持 DDE 的程序一样，Word 支持“system”。当以此为主题打开一对话时，可以使用“execute”语句向 Word 发送 WordBasic 指令。此方法功能强大，本文将以此为主要工具实现 Word 文档的建立。在“system”主题下，还可以使用 sysitems 作为指令“request”的参数，这样将返回一个关于所有打开的文档及所用模板的数组（如此可使用其中的一篇文档作为主题再打开一个 DDE 对话）。如果使用“system”作为对话的主题，似乎没有较直接的方法从 Word 中返回（诸如当前选中的文本、光标所在处的字体的名称等）信息。

第二类可被 Word 认可的主题是一个打开的文档或模板的名称。如果以一文档名打开一个与 Word 的对话，我们就可以使用“request”指令，以文档中的书标名作为“request”指令的参数，获取该书标中的文本；也可以使用“poke”指令在 Word 文档中当前光标处插入文本。至于此种情况下“execute”指令的操作，我就不太清楚了。

如果想以“system”为主题建立与 MS Word 的 DDE 链接，我们可以利用“execute”语句向 Word 发送 WordBasic 指令。WordBasic 是 Word6 和 Word95 支持的编程语言，在 Word97 中，VBA 替代了 WordBasic，前者有着截然不同的对象模型。但出于向后兼容的考虑，Word97 仍然可以编译、执行 WordBasic 指令。

由于 WordBasic 不再受支持，很难获取语法方面的知识。如果有 Word6 或 Word95 的安装盘或 CD 盘，就可以在其中找到 Wrdbasic. hlp 文件（或从 http://nersp. nerdc. ufl. edu/~alyons/ftp/wrdbasic. zip 处下载）。或者如果在旧机器上有 Word6 或 Word95 运行的话，可以使用宏记录指令学习 WordBasic 的句法。

通常 WordBasic 指令模仿了键盘的操作。举例而言，代码如下：

建立一新文档；
插入“Hello world”字符；
将光标右移 6 个字符；
改换字体为粗体；
插入“beautiful”字符；
取消粗体的设置；
移动光标到行尾；插入一段落；

对于每一行代码，WordBasic 都有相应的指令。另外还有其他指令执行文件的打开/关闭、插入表格、格式化段落、执行宏命令等。

注意：如果在 DDEClient 的 Execute 语句中使用 WordBasic 指令，这些指令应处于方括号中。同时要注意的是指令的操作如插入或改变文本均发生在光标所在处。因此在利用 Avenue 代码建立 Word 文档时，要随时清楚当前文档为哪个及光标位于何处。

以下为一个 Avenue 脚本文件的示例，该程序首先启动一 DDE 对话，尔后新建一 Word 文件，并在文件中插入数个字符，最后关闭 DDE 通话。请注意其中 WordBasic 指令均位于方括号中，而且 WordBasic 指令中的双引号是双重的。

```
theDDE = DDEClient.Make("WINWORD", "system")
'Start Word if needed
if (theDDE.HasError) then
  strMSW = "C:\Program Files\Microsoft\Office\Winword.exe"
  System.Execute(strMSW)
  'Try again
  theDDE = DDEClient.Make("WINWORD", "system")
  if (theDDE.HasError) then
    'Still can't connect, maybe Word isn't installed
    MsgBox.Error(theDDE.GetErrorMsg, "")
    Return Nil
  end
end
'Start a new document
theDDE.Execute("[FileNew]")
'Activate the Word window so you can see the document being built
theDDE.Execute("[AppActivate ""Microsoft Word"", 1]")
'Insert some text
theDDE.Execute("[Insert ""Hello world.""]")
theDDE.Execute("[CharLeft 6]")
theDDE.Execute("[Bold 1]")
theDDE.Execute("[Insert ""BEAUTIFUL ""]")
theDDE.Execute("[EndOfLine]")
theDDE.Execute("[InsertPara]")
theDDE.Execute("[Insert ""Goodbye!""]")
'Close the conversation
theDDE.Close
```

有关本论文的反馈请联系：Andy Lyons alyons@nersp. nerdc. ufl. edu

一个实用的 Avenue 脚本文件：wwWriteWord

文件 wwWriteWord 可以被用来向 MS Word 传递 DDE 指令以建立一文档，它以一个仅含两个元素的数组作为变量，数组的第一个元素做为函数名（表 1 所示），第二个元素是该函数的参数。例如；

```
av. Run( ″wwWriteWord″, { ″font_name″, ″Times New Roman″})
```

使用 wwWriteWord 过程中的几个要点：

●第一个呼叫须为 “init ” 函数。这将启动一个与 Word 的 DDE 对话并新建一文档。

●DDE 指令仅编辑 Word 中的当前文档。不能用此程序修改其他 Word 文件（通过修改此程序或其它 DDE 对话可以实现）。

●所有编辑都发生在当前光标所处的位置，因此对光标的位置要清楚。

●最后的一个 DDE 呼叫应该是 “close ” 函数，否则会造成系统资源的浪费。

●此程序只能将短小的 DDE 指令传送给 Word。更详细的解释请参考 Word Basic 帮助文件。

●此程序在命令传递中基本无误，

●若指令执行顺利，程序返回 “True ”，否则为 “false ”。

●此程序有时需调用另一程序 wwCommandLine 以获取 MS Word 的指令行，请确认此程序处于 ArcView 的项目文件或扩展模块中。

如前所述，传送给 wwWriteWord 的数组含两个元素，第一个元素为下表所示的函数名、字符型变量，第二个为该函数的赋值，如表 1 所示：

应用示例：

```
ww = ″wwWriteWord″
av. Run(ww, {″init″, True})
av. Run(ww, {″align″, ″CenterPara″})
av. Run(ww, {″font_name″, ″Arial″})
av. Run(ww, {″font_size″, 18})
av. Run(ww, { ″insert_html_text″, ″ <B>Environmental Sensitivity Analysis </B><P>″})
av. Run(ww, {″font_name″, ″Times New Roman″})
av. Run(ww, {″font_size″, 12})
av. Run(ww, {″align″, ″LeftPara″})
av. Run(ww, {″table_insert″, {2, 1}})
av. Run(ww, { ″insert_html_text″, ″ <B>Project: </B> SR 710 Ext. Palm Beach County″})
av. Run(ww, {″table_nextcell″, NIL})
av. Run(ww, { ″insert_html_text″, ″ <B>Analysis Data: </B> August 22, 1999 <BR> <B> Version: </B> 1. 0″})
av. Run(ww, {″line_down″, 1})
av. Run(ww, {″close″, NIL})
```

收稿日期 2000 年 11 月 8 日

表 1

函数	赋值	描述
init	True / False	打开与 word 的对话并创建一新文档。若传递 “true ”，将使 Word 程序为当前系统，位于桌面最前方。 此函数必须在所有其他函数调用前被执行
close	无	关闭 DDE 对话。这应该是 Avenue 脚本文件中最后一句。
insert_para	数值 N	插入 N 个新段落（即回车 N 次）
insert_pict	“完整的文件名 ”	在光标处插入 “文件名 ”所指的图片，应该是 word 支持的图片类型，且路径要完整。
insert_pageno	无	在页脚处插入居中的页编号
header_open	无	打开页眉、页脚编辑框
header_close	无	关闭页眉、页脚编辑框
line_down	数值 N	移动光标
ctrl_home	无	移动光标到文档头部
char_left	数值 N	将光标左移 N 个字符
ctrl_right	数值 N	将光标右移 N 个字符
font_name	“字型 ”	转换字型为 “字型名 ”
font_size	“字体大小 ”	字体大小转换
align	“LeftPara ”、“RightPara ”、“CenterPara ”或 “JustifyPara ”	设置段落对齐方式，分别为左对齐，右对齐，居中对齐，两端对齐。
left_margin	{ 左页边距，首行页边距 }	设置左页边距（英寸）
page_margins	{N1，N2，N3，N4 }	设置页边距，单位为英寸，顺序为顶，右，底，左
table_insert	{N1，N2，N3… }	1 在光标所在处插入表格 2 设置列宽度及 3 移动光标到第一个单元格的起始处。 数组中第一个元素为表格的列数，第二个元素为行数，第三个元素为第一列的宽度，第四个元素为第二列的宽度…单位为英寸。若列数目被省略，则该表格的列数数字为默认的数目。
table_nextcell	空	移动光标到下一个单元格
table_prevcell	空	移动光标到前一个单元格
save	“文件名 ”	保存当前文本为（FileName 字符型）
pagebreak	空	插入页边符
sectbreak	空	插入换页符（光标转入到新的一页，且另起一行）
vert_dist	空	返回当前行至页首的纵向距离（英寸）

“数字贝贝 ”隆重上市

由北京百联美达美数码科技有限公司开发的 “数字贝贝 ”系列软件于 2001 年元月 6 日上市。

此次上市的 “数字贝贝 ”系列软件包含三套产品——图书贝贝（CBOOK）、音乐贝贝（CDOK）、软件贝贝（CNSHARE）。该系列软件是国内第一次专为 15 - 30 岁之间的青年休闲、娱乐而量身定制的软件产品，也是元旦和春节两节期间的特色礼品软件。

图书贝贝（CBOOK）在图书软件 CBOOK 及其专用服务器的强大支持下，能实现同时下载数十本图书、任意设置阅读方式、自动更新各类书目和个性化文件夹管理等功能，并随盘赠送 200 部中外经典著作和 500 部英文原著。

音乐贝贝（CDOK）是国内第一个能够实现卡拉 OK DIY 功能的音乐软件。独有的 CDOK 软件能够在自动识别和播放 CD、MP3 的同时，进行 MTV 歌词显示。拥有超大歌词库，并能进行个性化歌曲管理。听歌的同时，还能学习外语。随盘附送 100 首中英文经典歌曲。

软件贝贝（CNSHARE）收录了 7 大类别，200 余种精品软件。全部都是由 “首届全国优秀共享软件及自由软件评选活动 ”的 50 家权威媒体编辑、10 位软件行业企业家、20 位软件开发专家亲手评选而出。软件贝贝除了能实现中计这些软件的下载安装之外，还具有特别的软件增值功能。它的专用服务器能让软件用户及时得到各类软件的最新版本和更新信息。

百联美达美公司依托中国软件网，在 2000 年成功地推出了 “程序员大本营 2000 ”系列软件和《程序员》杂志，均取得了骄人的成绩。此次推出 “数字贝贝 ”系列软件，也是瞄准节日市场，专门针对数字青年。软件每小套定价为 28 元。想必会博得消费者青睐。

用 VB5 开发多功能网络实时监控系统

元晋豫

摘　要　在计算机网络日益盛行的今天，网络安全至关重要，为此，本文详细阐述了如何用 VB5. 0 开发网络实时监控系统，该系统可使网络管理人员足不出户且随时都可以监控到网络中任何一台计算机的操作状态，若发现有违规行为，可通过该系统采取强制措施规范其操作，从而增强整个网络系统的安全性。

关键词　屏幕监控，WinSock 控件，监听，锁定，信息互送

随着计算机网络技术的日益发展和普及，很多单位、团体都建立了自己的内部网，这样网络安全就显得越来越重要，网络管理人员在整个网络运行期间，能否实时监控联网计算机的运行状态、操作对网络安全具有极其重要的作用，比如防止病毒的蔓延、非法程序的拷贝等。为此，笔者经过仔细分析和研究，用 VB5 开发出一套能对联网的每台计算机进行实时监控的网络系统，该系统操作简便、功能较多，笔者在本单位经过一段时间的运行，效果颇佳，读者看后即可将该系统用于自己单位，下面具体介绍设计过程。

系统概况

本文介绍的网络监控系统由两部分组成：控制台系统和工作站系统。控制台系统安装在网络管理人员的计算机上，用于实施各种监控操作；工作站系统安装在每台联网的计算机上，它运行后不影响工作站的其他操作，只用于响应控制台的监控命令，并根据需要采样工作站的相应数据返回给控制台，用户可以最小化到任务栏中，不必理睬。图一是在一台计算机上既运行控制台系统又运行工作站系统后，在控制台上进行两次‘屏幕监控”操作后的运行画面。

图 1

功能简介

屏幕监控：该操作可以把被控工作站的屏幕画面抓取到控制台中，网络管理人员对相应工作站所进行的操作一目了然，若发现有非法操作，比如上班玩游戏、安装非法程序等，即可采取下面的操作进行控制。

锁定控制：该操作可以把被控工作站的屏幕锁定，使鼠标和键盘无法发挥正常作用，其效果类似于工作站死机。

解锁操作：该操作是‘锁定控制”操作的逆操作，可以使锁定的工作站恢复正常。

强行关机：该操作可以使被控工作站关机，具有较大的杀伤力，笔者建议网络管理人员只有在发现工作站进行具有较大破坏性的操作时使用。

信息互送：通俗一点讲，该操作就是允许控制台用户和工作站用户进行聊天，当然在控制台锁定工作站后，即可用于发布命令。该操作将在控制台和工作站上同时打开一个信息互送窗口，工作站信息互送窗口的关闭由控制台决定，具体画面见文后的设计过程。

编程思路

本系统的编程原理较简单，控制台和工作站进行数据传输的核心是 WinSock 控件。在控制台端每进行一项监控操作，均启用 WinSock 控件与工作站进行连接，然后发布相应命令；在工作站端 WinSock 控件始终处于监听状态，当接收到控制台的命令时，执行相应操作，若需要返回数据　比如屏幕监控）则用适当方法发送至控制台，最后将 WinSock 控件重新置为监听状态。各种操作的执行还需要 Windows 的几个 API 函数，具体申明见文后的源代码　以下同）。

技术要点

抓取屏幕：在实施屏幕监控操作时，工作站需要将屏幕抓取下来，具体过程是用 API 函数 GetDC 取得屏幕的设备句

柄，用 BitBlt 函数将图像数据拷贝至一个临时 PictureBox 控件（不可见），然后用 GetBitmapBits 函数将图像的像素数据存储于一个字节数组中，最后发送该字节数组。到控制台后，用 SetBitmapBits 函数将字节数组中的像素数据还原至 PictureBox 控件中。

WinSock 控件传输大容量二进制数据：在工作站向控制台传送像素数据时，数据量很大（如分辨率为 800×600、显示器颜色属性为增强 16 色的屏幕图像数据约有 1.4Mb），计算机有时采用分批传输，所以在控制台 WinSock 控件的 DataArrival 事件中只能用 PeekData 接收数据，不能用 GetData，否则会导致数据接收不完全。

在 PictureBox 控件中加入滚动条浏览大幅图像：VB5 中的 PictureBox 控件在浏览大幅图像时，超过本身大小的部分看不到，需要引入 PictureClip 控件和滚动条控件进行浏览，具体方法参见文后源代码。

屏幕锁定和解锁：用 API 函数 EnableWindow GetDesktopWindow rs 可以实现屏幕锁定和解锁操作，当 rs = 0 时锁定，rs = 1 时解锁。

关机操作：用 API 函数 ExitWindowsEx 可以实现关机功能。

控制台系统设计过程

控制台主窗体

启动 VB5，新建“标准 EXE 工程”，工程名称设为“控制台”，添加标准模块，名称设为 modAPI，用于 Windows API 函数的申明和全局变量的定义，加入下列代码：

```
Public Declare Function GetDC Lib "user32" (ByVal hwnd As Long) As Long
Public Declare Function DeleteDC Lib "gdi32" (ByVal hdc As Long) As Long
Public Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long
Public Declare Function SetBitmapBits Lib "gdi32" (ByVal hBitmap As Long, ByVal dwCount
As Long, lpBits As Any) As Long
'-----/ 以下为自定义变量 /-----
Public intScrData As Integer  '接收的数据类型标志: 1:X 分辨率; 2:Y 分辨
率; 3:像素; 4: 信息互送
Public lngRslX As Long, lngRslY As Long  '屏幕分辨率
Public blnNotCapScr As Boolean  '捕捉屏幕标志
Public blnWskCls As Boolean  '中断连接标志
Public blnWskErr As Boolean  '连接错误标志
Public blnSendMsg As Boolean  '信息内容发送标志
Public lngMaxTime As Long  '连接最大时限
Public strCaption As String, strOprt As String
```

将 Form1 窗体的名称设为 frmCtrl，并将工程属性中的启动对象设为 frmCtrl。选择菜单项“工程”—“部件”，在“控件”选项卡中选中以下三个控件组（前面打勾）：Microsoft PictureClip Control 5.0、Microsoft Windows Common Controls 5.0、Microsoft Winsock Control 5.0，在 frmCtrol 窗体上添加一个 PictureClip 控件、StatusBar 控件、Winsock 控件、Timer 控件，然后添加五个大小相等的 CommandButton 控件，名称都设为 cmdOprt，Index 属性分别为 0、1、2、3、4，Caption 属性分别为“屏幕监控”、“锁定控制”、“解锁操作”、“强行关机”、“信息互送”，最后按照图 1 所示，添加剩余各控件，具体属性和名称见下表，由于篇幅有限，下表只列出能够使系统正常运行的控件，其它装饰和提示用的控件（如 Label、Frame，以下窗体的设计同样如此）读者自行添加：

控件类型	名称属性	Caption 属性	用　途
WinSock	WskCtrl		
PictureBox	PicScreen		用于显示工作站屏幕
PictureBox	PicTmp		临时存放图像,任意大小
PictureClip	PicClip		协助 picScreen 显示大幅图像
CommandButton	CmdAddIp	添加	添加工作站 IP 地址
CommandButton	CmdWskCls	中断连接	
CommandButton	CmdUnload	关闭控制台	
ComboBox	CobIpAddr		选择工作站 IP 地址
Label	LabMaxTm		显示超时时间
VScrollBar	SclTime		调整超时时间数
HscrollBar	SclX		显示工作站屏幕横向滚动条
VscrollBar	SclY		显示工作站屏幕纵向滚动条
StatusBar	StbState		状态显示
Timer	Timer1		控制连接超时时间

在 frmCtrl 的窗口代码中加入下列代码：

```
Option Explicit
'======| 添加 IP 地址事件 |=======
Private Sub cmdAddIp_Click()
    frmIpAddr.Show 1, Me
    GetIPAddr
    cobIpAddr.Refresh
End Sub
'======| 监控操作 |=======
Private Sub cmdOprt_Click(Index As Integer)
Dim i As Long
blnNotCapScr = True
blnSendMsg = False
intScrData = 1
Select Case Index
    Case 0
        strCaption = "网络监控系统---屏幕监控"
        strOprt = "CAPTURE"
        blnNotCapScr = False
    Case 1
        strCaption = "网络监控系统---锁定控制"
        strOprt = "LOCK"
    Case 2
        strCaption = "网络监控系统---解除锁定"
        strOprt = "UNLOCK"
    Case 3
        strCaption = "网络监控系统---强行关机"
        strOprt = "HALT"
    Case 4
        strCaption = "网络监控系统---信息互送"
        intScrData = 4
```

```
        strOprt = "MESSAGE"
End Select
    stbState.Panels(4).Text = strCaption
    If wskCtrl.state <> sckClosed Then wskCtrl.Close
    wskCtrl.RemoteHost = cobIpAddr.Text
    On Error GoTo CONNECT_ERROR
    wskCtrl.Connect
    stbState.Panels(1).Text = "请稍等,控制台正在连接......"
    Me.MousePointer = 11
    ChangeState 0
    blnWskCls = False
    blnWskErr = False
    lngMaxTime = 0
    Timer1.Enabled = True
    Do    '等待 wskCtrl 为连接状态(sckConnected=7),若超时、连接错误、用户中断则退出
        DoEvents
    Loop Until wskCtrl.state = sckConnected Or lngMaxTime = Val(labMaxTm.Caption) Or
blnWskCls Or blnWskErr
    Me.MousePointer = 1
    If blnWskErr Then Exit Sub    '连接错误,无条件退出
    If wskCtrl.state = sckConnected Then    '连接成功,进行相应操作
        MsgBox "OK!控制台连接成功!准备进行 " + Mid(strCaption, 10, 4) + " 操作!", , "控制台"
        On Error GoTo CONNECT_ERROR
        wskCtrl.SendData strOprt
    Else                '连接超时或用户中断,退出
        If lngMaxTime = Val(labMaxTm.Caption) Then MsgBox "抱歉!控制台连接超时!", , "控制台"
        ChangeState 1
    End If
    Timer1.Enabled = False
    If wskCtrl.state = sckConnected And intScrData = 4 Then frmMessage.Show 1, Me    '打开信息互送窗口
    Exit Sub
CONNECT_ERROR:
    MsgBox "控制台无法执行 " + strCaption + " 操作!请检查各设备的状态和 IP 地址!", , "控制台"
    ChangeState 1
End Sub
'======| 关闭 |=====
Private Sub cmdUnload_Click()
    Unload Me
End Sub
'======| 中断连接 |=====
Private Sub cmdWskCls_Click()
    wskCtrl.Close
    blnWskCls = True
End Sub
'======| 载入窗体 |=====
Private Sub Form_Load()
    wskCtrl.RemotePort = 10
    Me.ScaleMode = 3
    picScreen.ScaleMode = 3
    picScreen.AutoRedraw = True
    Me.AutoRedraw = True
    picTmp.AutoRedraw = True
    picTmp.AutoSize = True        '必须为 True,才能取得实际位图的宽度和高度
    picTmp.ScaleMode = 3
    picTmp.Visible = False
    Timer1.Enabled = False
    stbState.Panels(1).Width = 430
    stbState.Panels(2).Width = 50
    stbState.Panels(3).Width = 20
    stbState.Panels(4).Width = 250
    GetIPAddr        '获取 IP 地址
End Sub
'======| 卸载窗体 |=====
Private Sub Form_Unload(Cancel As Integer)
    If wskCtrl.state <> sckClosed Then wskCtrl.Close
End Sub
'======| 连接超时滚动条 |=====
Private Sub sclTime_Change()
    labMaxTm.Caption = CStr(sclTime.Value)
End Sub
'======| 改变垂直滚动条 |=====
Private Sub sclY_Change()
    If sclY.Value >= sclY.Max Then sclY.Value = sclY.Max
    On Error GoTo SUB_END
    picClip.ClipY = sclY.Value
    picScreen.Picture = picClip.Clip
    picScreen.Refresh
SUB_END:
    cmdUnload.SetFocus
End Sub
'======| 改变水平滚动条 |=====
Private Sub sclX_Change()
    If sclX.Value >= sclX.Max Then sclX.Value = sclX.Max
    On Error GoTo SUB_END
    picClip.ClipX = sclX.Value
    picScreen.Picture = picClip.Clip
    picScreen.Refresh
SUB_END:
    cmdUnload.SetFocus
End Sub
'=====| 超时控制定时器 |====
Private Sub Timer1_Timer()
lngMaxTime = lngMaxTime + 1
End Sub
'=====| 工作站返回数据 |====
Private Sub wskCtrl_DataArrival(ByVal bytesTotal As Long)
Dim strRs As String
Dim lngTmp As Long, i As Long, bytTmp() As Byte
    Select Case intScrData
        Case 1
            On Error GoTo DATA_ERR
            wskCtrl.GetData lngTmp, vbLong        '接收 X 分辨率----long 型
            lngRslX = lngTmp
            wskCtrl.SendData "X"    '应答信号--X
```

```
            intScrData = 2
        Case 2
            On Error GoTo DATA_ERR
            wskCtrl.GetData lngTmp, vbLong      '接收 Y 分辨率 - - - - - long 型
            lngRslY = lngTmp
            wskCtrl.SendData "Y"      '应答信号 - -Y
            intScrData = 3
        Case 3
            Me.MousePointer = 11
            stbState.Panels(1).Text = "  当前传送字节数:"
 + CStr(bytesTotal)
            i = lngRslX * lngRslY * 3&
            If bytesTotal >= i Then
                ReDim bytTmp(i) As Byte
                On Error GoTo DATA_ERR
        wskCtrl.PeekData bytTmp, vbArray + vbByte, i '接收像素 - - Byte Array 型
                '- - - - - - / 开始显示屏幕 / - - - - - -
                picTmp.Width = lngRslX
                picTmp.Height = lngRslY
                SetBitmapBits picTmp.Image, UBound(bytTmp),
bytTmp(0)
                picTmp.Refresh
        SavePicture picTmp.Image, App.Path + "\tmp.bmp"
                ReDim bytTmp(0)
    picClip.Picture = LoadPicture(App.Path + "\tmp.bmp")
'PictureClip 必须加载为位图
                picClip.ClipX = 0
                picClip.ClipY = 0
            picClip.ClipWidth = picScreen.Width - sclY.Width
          picClip.ClipHeight = picScreen.Height - sclX.Height
                sclY.Min = 0
                sclY.Max = lngRslY - picClip.ClipHeight - 4
                sclY.Value = 0
                sclY.LargeChange = 50
                sclY.SmallChange = 10
                sclX.Value = 0
                sclX.Max = lngRslX - picClip.ClipWidth - 4
                sclX.Min = 0
                sclX.LargeChange = 50
                sclX.SmallChange = 10
                picScreen.Picture = picClip.Clip
                picScreen.Refresh
'屏幕像素传送成功后,恢复操作命令按钮的 Enabled 属性
                ChangeState 1
                Me.MousePointer = 1
        stbState.Panels(1).Text = "  OK!屏幕数据传送成功!"
            End If
        Case 4
            wskCtrl.GetData strRs      '接收工作站信息互送内容 - - - - String 型
            frmMessage.labWks.Caption = strRs
            If Asc(Right(strRs, 1)) = 13 Then
             frmMessage.lstChat.AddItem "工作站: " + strRs
                frmMessage.labWks.Caption = ""
            End If
    End Select
Exit Sub
DATA_ERR:
    MsgBox "控制台连接出现错误!请仔细检查各设备的连接状态!", , "控制台"
End Sub
' = = = = = | 连接错误 | = = = = = =
Private Sub wskCtrl_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean)
Dim i As Long
    blnWskErr = True
    MsgBox "控制台连接失败!", , "控制台"
    ChangeState 1
End Sub
' = = = = = | 发送操作命令完毕事件 | = = = = = =
Private Sub wskCtrl_SendComplete()
Dim i As Long
    '屏幕监控操作,要等到像素传送完毕并显示出来,才能恢复操作命令按钮的 Enabled 属性
If blnNotCapScr And (Not blnSendMsg) Then
    ChangeState 1
    stbState.Panels(1).Text = "  OK!" + Mid(strCaption, 10, 4) + " 操作执行成功!"
    blnSendMsg = False
End If
End Sub
' * * * | 更改命令按钮状态(自定义) | * * *
Private Sub ChangeState(ByVal state As Integer)
Dim i As Long
Dim rs As Boolean
    rs = True
    If state = 0 Then rs = False
    For i = 0 To cmdOprt.Count - 1
      cmdOprt(i).Enabled = rs
    Next i
    sclTime.Enabled = rs
    If rs Then stbState.Panels(1).Text = "  "
End Sub
' * * * | 从文件中获取 IP 地址(自定义) | * * *
Private Sub GetIPAddr()
Dim fn, i
Dim strTmp As String
For i = 0 To cobIpAddr.ListCount - 1
    cobIpAddr.RemoveItem 0
Next i
fn = FreeFile
Open App.Path + "\ipaddr.txt" For Input As #fn
Do While Not EOF(fn)
    Input #fn, strTmp
    cobIpAddr.AddItem strTmp
Loop
Close fn
End Sub
```

添加工作站 IP 地址窗体

在“控制台”工程中，选择菜单项“工程”—“添加窗体”，选中“新建”选项卡中的“窗体”。

将窗体的名称设为 frmIpAddr，添加下表中所述控件，控件布局如图 2。

控件类型	名称	Caption 属性	备注
CommandButton	CmdAdd	添加	
CommandButton	CmdDel	删除	
CommandButton	CmdAllDel	全部删除	
CommandButton	CmdUnload	关闭	
TextBox	TxtIpAddr		输入 IP 地址用
ListBox	lstIp		显示所有添加过的 IP 地址

图 2

在 frmIpAddr 的代码窗口中加入下列代码：

```
Option Explicit
' = = = = = | 添加 IP 地址 | = = = = =
Private Sub cmdAdd_Click()
Dim i As Integer, lenIp As Integer
        '合法性判断
lenIp = Len(txtIpAddr.Text)
If lenIp > 15 Or lenIp = 0 Then
   MsgBox "非法的地址长度!请重输!"
   Init_IpAddr
   Exit Sub
End If
For i = 1 To lenIp
   If (Not IsNumeric(Mid(txtIpAddr.Text, i, 1))) And Mid
(txtIpAddr.Text, i, 1) <> "." Then
      MsgBox "地址中有非法字符!请重输!"
      Init_IpAddr
      Exit Sub
   End If
Next i
lenIp = lstIp.ListCount - 1
For i = 0 To lenIp
   If txtIpAddr.Text = lstIp.List(i) Then
      MsgBox txtIpAddr.Text + " 地址已存在,无须添加!"
      Init_IpAddr
      Exit Sub
   End If
Next i
        '添加地址
On Error GoTo END_SUB
lstIp.AddItem txtIpAddr.Text
lstIp.Refresh
Init_IpAddr
        '保存到文件
PutIPAddr
END_SUB:
End Sub
' = = = = = | 初始化 IP 地址 | = = = = =
Private Sub Init_IpAddr()
   txtIpAddr.Text = ""
   txtIpAddr.SetFocus
End Sub
' = = = = = | 全部删除 IP 地址 | = = = = =
Private Sub cmdAllDel_Click()
Dim i As Integer
For i = 0 To lstIp.ListCount - 1
   On Error GoTo END_SUB
   lstIp.RemoveItem 0
Next i
PutIPAddr
END_SUB:
End Sub
' = = = = = | 删除指定的 IP 地址 | = = = = =
Private Sub cmdDel_Click()
   On Error GoTo END_SUB
   lstIp.RemoveItem lstIp.ListIndex
   PutIPAddr
END_SUB:
End Sub
' = = = = = | 关闭 | = = = = =
Private Sub cmdUnload_Click()
   Unload Me
End Sub
' = = = = | 载入窗体 | = = = =
Private Sub Form_Load()
Dim fn
Dim strTmp As String
fn = FreeFile
Open App.Path + "\ipaddr.txt" For Input As #fn
Do While Not EOF(fn)
   Input #fn, strTmp
   lstIp.AddItem strTmp
Loop
Close fn
End Sub
' * * * | IP 地址保存到文件 | * * *
Private Sub PutIPAddr()
Dim i, fn
fn = FreeFile
Open App.Path + "\ipaddr.txt" For Output As #fn
If lstIp.ListCount = 0 Then Write #fn, "none"
For i = 0 To lstIp.ListCount - 1
   Write #fn, lstIp.List(i)
Next i
Close #fn
```

End Sub

信息互送窗体

与添加工作站 IP 地址窗体一样，添加窗体，名称设为 frmMessage　添加下表中所述控件，控件布局如图 3。

控件类型	名称	Caption 属性	备注
Label	LabWks		显示工作站传送过来的信息
TextBox	TxtCtrl		输入控制台将要发送的信息
ListBox	LstChat		显示双方信息互送的历史记录
CommandButton	CmdUnload	关闭	关闭控制台和工作站的信息互送窗口

图 3

在 frmMessage 的代码窗口中加入下列代码：

```
'=====| 关闭 |=====
Private Sub cmdUnload_Click()
    Unload Me
End Sub
'=====| 卸载窗体 |=====
Private Sub Form_Unload(Cancel As Integer)
    blnSendMsg = True
    On Error Resume Next
    frmCtrl.wskCtrl.SendData "UNLOAD_MESSAGE"
End Sub
'=====| 控制台发送信息 |=====
Private Sub txtCtrl_KeyPress(KeyAscii As Integer)
    If KeyAscii = 13 Then
      lstChat.AddItem "控制台: " + txtCtrl.Text
      txtCtrl.Text = txtCtrl.Text + vbCr
      blnSendMsg = True
      On Error GoTo WSK_ERR
      frmCtrl.wskCtrl.SendData txtCtrl.Text
      txtCtrl.Text = ""
    Else
      blnSendMsg = True
      On Error GoTo WSK_ERR
      frmCtrl.wskCtrl.SendData txtCtrl.Text + Chr(KeyAscii)
    End If
    Exit Sub
WSK_ERR:
    MsgBox "系统在信息互送时,发生连接故障!", , "控制台---信息互送"
End Sub
```

工作站系统设计过程

工作站主窗体

由于工作站的作用只是响应控制台的监控操作，所以工作站主窗体不和用户进行交互，也不显示任何数据、图像等，只添加一个“关闭”按钮即可。和建立控制台窗体一样，启动 VB5 后，新建‘标准 EXE”工程，工程名称设为 wkstn，添加一标准模块，名称为 modWkAPI，在其模块窗口中添加下列代码：

```
Public Const SRCCOPY = &HCC0020
Public Declare Function BitBlt Lib "gdi32" (ByVal hDestDC As Long, ByVal X As Long, ByVal Y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, ByVal xSrc As Long, ByVal ySrc As Long, ByVal dwRop As Long) As Long
Public Declare Function GetDC Lib "user32" (ByVal hwnd As Long) As Long
Public Declare Function DeleteDC Lib "gdi32" (ByVal hdc As Long) As Long
Public Declare Function GetBitmapBits Lib "gdi32" (ByVal hBitmap As Long, ByVal dwCount As Long, lpBits As Any) As Long
Public Declare Function IsWindowEnabled Lib "user32" (ByVal hwnd As Long) As Long
Public Declare Function EnableWindow Lib "user32" (ByVal hwnd As Long, ByVal fEnable As Long) As Long
Public Declare Function GetDesktopWindow Lib "user32" () As Long
Public Const EWX_LOGOFF = 0
Public Const EWX_SHUTDOWN = 1
Public Const EWX_REBOOT = 2
Public Const EWX_FORCE = 4
Public Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags As Long, ByVal dwReserved
As Long) As Long
'----/ 以下为自定义变量 /-----
Public hScreenDC As Long          '屏幕设备上下文环境
Public lngScrRslX As Long, lngScrRslY As Long'屏幕分辨率
Public bytPixel() As Byte         '传送屏幕像素用的字节数组
Public blnPixelSendFinishFlag As Boolean   '屏幕像素数据发送完标志
Public blnUnloadMessage As Boolean'卸载信息互送窗体标志
```

将 Form1 窗体的名称设为 frmSer，窗体形状尽可能小一些，Caption 属性设为‘网络监控—工作站”，在 frmSer 上添加 Winsock 控件，名称设为 wskWks；接着添加一个 PictureBox 控件，名称设为 picTmp，用来临时存储屏幕图像；添加一个 CommandButton 控件，名称设为 cmdUnload，Caption 属性设为“关闭”。工程中启动窗体设为 frmSer。当工作站系统运行后，用户可以最小化 frmSer 窗体，继续其它操作。最后在其代码窗口中加入下列代码：

```
Option Explicit
```

```
'=====| 关闭窗体 |=====
Private Sub cmdUnload_Click()
    Unload Me
End Sub
'======| 载入窗体 |======
Private Sub Form_Load()
lngScrRslX = Screen.Width / Screen.TwipsPerPixelX '获取工作站分辨率
lngScrRslY = Screen.Height / Screen.TwipsPerPixelY
picTmp.Left = 0
picTmp.Top = 0
picTmp.Width = lngScrRslX
picTmp.Height = lngScrRslY
wskWks.LocalPort = 10
wskWks.Listen
picTmp.Visible = False
picTmp.AutoRedraw = True
picTmp.AutoSize = True
End Sub
'======| 工作站应答 |======
Private Sub wskWks_ConnectionRequest(ByVal requestID As Long)
If wskWks.State <> sckClosed Then wskWks.Close
    wskWks.Accept requestID
    'MsgBox "工作站到连接请求,可以开始通话!状态:" + CStr(wskWks.State), , "工作站"
End Sub
'======| 数据到达 |======
Private Sub wskWks_DataArrival(ByVal bytesTotal As Long)
Dim strTmp As String
Dim lngCurColor As Long
Dim i As Long
    wskWks.GetData strTmp, vbString
    blnPixelSendFinishFlag = False
    Select Case strTmp
        Case "CAPTURE"      '//////// 收到'截获屏幕"
         lngScrRslX = Screen.Width / Screen.TwipsPerPixelX
'获取工作站分辨率
        lngScrRslY = Screen.Height / Screen.TwipsPerPixelY
            picTmp.Left = 0
            picTmp.Top = 0
            picTmp.Width = lngScrRslX
            picTmp.Height = lngScrRslY
                        '发送 X 分辨率
            wskWks.SendData lngScrRslX
        Case "LOCK"         '//////// 收到'锁定控制"
            i = IsWindowEnabled(GetDesktopWindow)
 If i <> 0 Then i = EnableWindow(GetDesktopWindow, 0)
            wskWks.Close
            wskWks.Listen
        Case "UNLOCK"       '//////// 收到'解除锁定"
            i = IsWindowEnabled(GetDesktopWindow)
 If i = 0 Then i = EnableWindow(GetDesktopWindow, 1)
            wskWks.Close
            wskWks.Listen
        Case "HALT"         '//////// 收到'强行关机"
    i = ExitWindowsEx(EWX_FORCE Or EWX_REBOOT, 0)
        Case "MESSAGE"      '//////// 收到'信息互送"
            blnUnloadMessage = False
            frmMessage.Show 1, Me
Case "UNLOAD_MESSAGE"   '收到关闭工作站信息互送窗口
            blnUnloadMessage = True
            Unload frmMessage
            wskWks.Close
            wskWks.Listen
        Case "X"            '收到应答信号"X",发送 Y 分辨率
            blnPixelSendFinishFlag = False
            wskWks.SendData lngScrRslY
        Case "Y"            '收到应答信号"Y",发送像素
            hScreenDC = GetDC(0)
            BitBlt picTmp.hdc, 0, 0, lngScrRslX, lngScrRslY,
hScreenDC, 0, 0, SRCCOPY
            picTmp.Refresh
            i = lngScrRslX * lngScrRslY * 3&
            ReDim bytPixel(i) As Byte
GetBitmapBits picTmp.Image, UBound(bytPixel), bytPixel(0)
            wskWks.SendData bytPixel
blnPixelSendFinishFlag = True'像素发送完毕,该句的作用是在 wskWks_SendComplete 事件中
            ReDim bytPixel(0)       '释放像素字节数组空间
        Case Else
            frmMessage.labCtrl.Caption = strTmp
            If Asc(Right(strTmp, 1)) = 13 Then
          frmMessage.lstChat.AddItem "控制台: " + strTmp
                frmMessage.labCtrl.Caption = ""
            End If
    End Select
End Sub
'======| wskwks 发送完毕事件 |======
Private Sub wskWks_SendComplete()
        '屏幕数据发送完毕,重新置 wskWks 为侦听状态
If blnPixelSendFinishFlag Then
    wskWks.Close
    wskWks.Listen
    blnPixelSendFinishFlag = False
    DeleteDC hScreenDC
End If
End Sub
```

信息互送窗体

该窗体的设计和控制台信息互送窗体基本一样，只是不需要"关闭"按钮，因为工作站信息互送窗体的关闭是由控制台决定的，工作站用户不允许关闭该窗体；另外将显示控制台发送信息的 Label 控件名称改为 labCtrl，将输入工作站发送信息的 TextBox 控件名称改为 txtWks。在窗体代码中加入下列代码：

```
'=====| 关闭窗口 |======
Private Sub Form_Unload(Cancel As Integer)
    If blnUnloadMessage Then
        Cancel = 0
    Else
        Cancel = 1
```

```
    End If
End Sub
´ = = = = = | 工作站发送信息 | = = = = =
Private Sub txtWks_KeyPress(KeyAscii As Integer)
 If KeyAscii = 13 Then
     lstChat. AddItem ″控制台: ″ + txtWks. Text
     txtWks. Text = txtWks. Text + vbCr
     On Error GoTo WSK_ERR
     frmSer. wskWks. SendData txtWks. Text
     txtWks. Text = ″″
   Else
     On Error GoTo WSK_ERR
   frmSer. wskWks. SendData txtWks. Text + Chr(KeyAscii)
   End If
   Exit Sub
WSK_ERR:
   MsgBox ″系统在信息互送时,发生连接故障!″, , ″工作站
– – –信息互送″
End Sub
```

操作说明

本系统的操作比较简单，按照下列步骤即可（以在两台机器上分别运行控制台系统和工作站系统为例，注意要使控制台机器和工作站机器的显示器颜色属性设置一样，例如都为增强16色等，否则“屏幕监控”的效果不太好，分辨率可以不一样，系统会自动判断分辨率并计算相应尺寸）：

1. 在控制台窗体中单击“添加”按钮，在弹出的IP地址添加窗体中输入运行工作站系统的机器IP地址，然后按下“添加”，最后单击“关闭”。若想删除某个地址，只需用鼠标选中地址框中的指定地址，然后单击“删除”。

2. 确信工作站系统正在运行，并且控制台机器可以ping通工作站机器。在控制台左上角的IP地址列表框中选定工作站地址。

3. 在控制台中单击左边五个监控按钮中任意一个，以“屏幕监控”为例，系统状态栏会提示“正在连接…”，稍后会出现“OK！控制台连接成功！准备进行屏幕监控操作！”，确定后状态栏会动态显示传送的字节数，最后提示执行成功。

4. 若选择“信息互送”操作，在控制台和工作站都会弹出各自的信息互送窗体，用户可以在相应的输入区输入待发送的内容，回车即可发送至对方。为使监控更有效，工作站中该窗体的关闭，需要用户在控制台的信息互送窗体中单击“关闭”按钮，方可完成关闭操作。

注意：若在同一台机器上运行两个系统，除非为了演示，否则最好不用单击“锁定控制”和“强行关机”按钮。

本系统在显示器颜色属性为增强16色，分辨率为800×600，VB5.0（企业版）/Win98下调试通过。

系统扩展

以上较完整地介绍了网络计算机实时监控系统的设计过程，有兴趣的读者若想做的更好，可以在工作站系统代码中添加Windows提供的API函数Shell_NotifyIcon和ShowWindow使工作站窗体运行后以图标的方式出现在Windows任务栏的提示区（托盘）中，而不显示在任务栏或桌面上，给用户以更加洁净的感觉。或者更进一步，再添加GetCurrentProcessId、GetCurrentProcess和RegisterServiceProcess等三个API函数，使工作站系统运行后不仅在任务栏、桌面上不出现，而且在任务列表中都不显示（即同时按下Ctrl + Alt + Del键出现的任务窗口中找不到工作站系统的任务条，而实际上已运行）。由于以上内容与本系统中网络监控的设计联系不太紧密，并且鉴于篇幅，在此只点到为止，有兴趣的读者可通过E_MAIL与笔者联系（邮箱地址：jcyjy@sohu.com）。

（收稿日期：2000年11月21日）

BMC为强化服务级别管理业务推出PATROL for SLM

PATROL FOR SLM可为业务流程提供面向行动的管理模式

2000年11月13日美国拉斯维加斯讯——BMC软件公司今天宣布推出针对服务级别管理（Service Level Management [SLM]）的PATROL，这是一个基于PATROL产品线的解决方案，可作为监视和管理服务级别的标准。PATROL FOR SLM从顾客角度审视服务级别管理，可加速对服务级别协议（SLA）的制订和管理，促使企业系统地对其关键业务应用程序的可用性和性能进行优化处理。

有了PATROL FOR SLM，企业首席信息官和部门主管可根据他们遍布全球的每个最终用户群体每天不同的工作时间来确定每个业务流程的可用性和性能指标。这样，PATROL FOR SLM就可让企业首席信息官充分认识到与企业目标相一致的SLA可以带来的利益，并了解如果一旦某个业务流程无法正常工作可能造成的损失，包括在销售收入和顾客信誉方面的损失。

PATROL FOR SLM提供面向行动的SLM，可就即将发生的SLA违约行为向服务供应商发出警告以便迅速地发现出现的问题，并采取纠正措施使SLA得以恢复执行。该产品还可就业务流程的服务级别向最终用户群体提供基于因特网的实时报警，以便最大限度地降低对最终用户的负面影响。

PATROL FOR SLM有助于获得企业BMC Software OnSite金级地位，这是BMC Software OnSite计划所授予的认证级别。采用PATROL FOR SLM并获得OnSite认证的公司就可缩短获得OnSite金级认证的时间。PATROL FOR SLM允许企业重复衡量最终用户的感受，确定服务级别协议，提供全面的SLA执行报告，从而满足Site认证计划中的若干要求。

BMC软件公司今天同时宣布推出一项培训课程和认证方案，作为PATROL FOR SLM解决方案的部分内容。

在 VC 中对 Delphi 所生成的 DLLs 的调用

李 强 贾云霞

在现在软件开发中，开发速度显得越来越很重要。有现象表明，国内有大量的程序员都在使用 RAD 工具 VB，Delphi 等。使用上述的语言工具在开发进度上取得绝对的优势，但不能否认的是人们也同时看到象 Delphi 这样的工具的不足之处。比如，生成的可执行代码大，速度不够快等等。

所以，本人在开发中选择了 VC 作主体，而用 Delphi 来做辅助。这样一来，我们就会有这样的问题：在 Delphi 中如何编写动态链接库？如果用 Delphi 生成一个 32 位的动态链接库，那么在 VC 的环境下应该如何调用呢？

本文将对上述的问题做一个回答。

一、用 Delphi 创建一个动态链接库

启动 Delphi 5.0 （或者 Delphi 4.0），按照以下步骤进行。

选择 File→New→DLL

在代码窗口内加入下面粗体部分。

```
library VicBmp;
{ Important note about DLL memory management: ShareMem must be the first unit in your library´s USES clause AND your project´s (select Project-View Source) USES clause if your DLL exports any procedures or functions that pass strings as parameters or function results. This applies to all strings passed to and from your DLL---even those that are nested in records and classes. ShareMem is the interface unit to the BORLNDMM.DLL shared memory manager, which must be deployed along with your DLL. To avoid using BORLNDMM.DLL, pass string information using PChar or ShortString parameters. }
uses
  Windows,
  SysUtils,
  Classes,
  Graphics,
  Forms;
function LoadFromFile(AFileName: PChar): HBITMAP; stdcall;
var
  Bmp: TBitmap;
begin
// Result := 0;
  Bmp := TBitmap.Create;
  try
    try
      Bmp.LoadFromFile(StrPas(AFileName));
      Result := Bmp.ReleaseHandle;
      Assert(Result <>0);
    except
      Result := 0;
    Application.MessageBox(PChar(´在动态链接库发生错误!´
+IntToStr(Result)), ´Error´, MB_OK or MB_ICONERROR);
      raise;
    end;
  finally
    Bmp.Free;
  end;
end;
exports
LoadFromFile Name ´LoadFromFile´; //注意此处，以后要修改.
begin
end.
```

然后将文件保存为 VicBmp.dpr。

编译成功后就在当前目录下生成了名为 VicBmp.dll 的动态库。

二、在 VC 中采用动态方法

动态方法是我们最经常用的方法。

采用 Win32 API 的 LoadLibrary、LoadLibraryEx 或 AfxLoadLibrary 加载这个 DLL，再使用 GetProcAddress 这个 Win32 API 来查找并返回想要调用的函数地址，最后完成对该函数的调用。

本文对此不再赘述。

三、在 VC 中采用静态方法

第一步：核查一下生成 DLL 所包含的函数。

使用 Borland Delphi 带的 TDump.exe 或 Microsoft 的 Dumpbin.exe 或者是 Depends.exe 等，都可以发现该 DLL 中输出了一个叫作 LoadFromFile 的函数。

第二步：创建 VC 下的测试工程。

生成一个新的 MFC AppWizard (exe) 工程，名称为 test。为了方便，我们在 Application Type 中选择 Dialog Base 这个单选框。

在主对话框上放上两个控件：Picture 和 Button。

这两个控件分别作如下设置：

ID 为 IDC_JPEG, Type 为 Bitmap;
ID 为 IDC_SELECT, Caption 为 Select JPEG File;

在 testDlg.cpp 文件的开始加上对头文件的引用。

```
#include ˝loadimage.h˝
```

然后，用 Class Wizard 为按钮写一个单击事件的消息处理。如下：

```
void CTestDlg: : OnSelect()
{
// TODO: Add your control notification handler code here
    CFileDialog dlgFile(TRUE);
    if (dlgFile. DoModal() = = IDOK)
    {
        DeleteObject(m_imgJPEG. SetBitmap(LoadFromFile
(dlgFile. GetFileName())));
    }
}
```

loadimage. h 的文件内容如下：

```
#if !defined(_LOADIMAGE_H)
    #define    _LOADIMAGE_H
    #define DLLImport _declspec(dllimport)
    extern "C" DLLImport HBITMAP _stdcall LoadFromFile
(const char * AFileName);
#endif
```

第三步：制作 . lib 文件

1. 创建 VicBmp. DEF 文件，内容如下：

```
LIBRARY     "Vicbmp. DLL"
EXPORTS
      LoadFromFile
```

; 注意 LoadFromFile 以后要改为 LoadFromFile@ 4.

2. 生成 . lib 文件

在 Microsoft 给我们提供的一系列开发工具中，有一个实用程序为 LIB. EXE。它的主要作用就是用来生成和维护 . LIB 文件的。如果大家想进深一层了解该程序的作用，请您参考 MSDN 中的相关内容。 我的 VC 测试工程的目录为 F hVictor h bmpforvc hvc h

```
Lib /DEF: F: \victor\bmpforvc\vc\vicbmp. def
```

查找后会发现生成 VICBMP. LIB 文件，然后，选择 Project - >Add To Project - >Files. . . ，把 vicbmp. lib 加入到工程里面。

这时我们的整个工作已经接近尾声。

第四步：编译整个文件，您会发现在链接时有一个错误为

```
testDlg. obj : error LNK2001: unresolved external symbol
_imp__LoadFromFile@ 4
```

这是因为VC + + 在定位一个函数时不是单纯来找函数名，而是同时附加了很多信息，包括调用参数等。因此我们对该 Delphi 的输出函数再做最后一次调整。

第五步：

1 按照上述的提示，将 DLL 的工程中的 LoadFromFile Name 'LoadFromFile ' 改为 LoadFromFile Name 'LoadFromFile@ 4 ' 重新编译这个 DLL 工程。

重新编译 Delphi 所写的动态链接库。

2 把 Vicbmp. DEF 改为

```
LIBRARY     "Vicbmp. DLL"
EXPORTS
      LoadFromFile@ 4
```

再执行 Lib /DEF F hvictor hbmpforvc hvc hvicbmp. def，生成新的 VicBmp. LIB。

第六步：把该 DLL 拷贝到 VC 测试程序的目录下。Rebuild All 测试工程，发现正确通过。

注：笔者使用的环境为 MS Visual C + + 6. 0 Borland Delphi 5. 0；文中所涉及的工具都是这两个开发工具所带。

收稿日期：2000 年 10 月 23 日）

上接第 50 页）

```
trols, Forms, Dialogs, Db, DBClient, MConnect, Grids,
DBGrids, StdCtrls, Buttons;
type
  TForm1 = class(TForm)
    DCOMConnection1: TDCOMConnection;
    ClientDataSet1: TClientDataSet;
    ClientDataSet2: TClientDataSet;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    DataSource1: TDataSource;
    DataSource2: TDataSource;
    BitBtn1: TBitBtn;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R * . DFM}
end.
```

5. 客户机程序主程序 工程文件）

```
program ClientApp; //ClientApp. dpr;
uses
  Forms,
  ClientUn in 'ClientUn. pas' {Form1}
{$R * . RES}
begin
  Application. Initialize;
  Application. CreateForm(TForm1, Form1);
  Application. Run;
end.
```

收稿日期 2000 年 8 月 24 日

利用智能鼠标进行任务切换

齐玉东

引言

在 Windows 98 里实现任务切换有两种方法，一种方法是利用鼠标点选任务条上的各个任务的名字来进行任务切换，但当我们同时运行很多的任务时，各个任务的名字在任务条上就不能正常地显示出来，使得我们不容易区分各个任务；另一种方法是利用组合键 Alt + Tab，进行任务切换，但这需要键盘。那么有没有一种方法，可以利用鼠标方便地进行任务切换呢？几乎在 Windows 的任何地方，普通鼠标的左右键都被赋予了不同的功能，我们不易再开发出新的功能。如果你使用的是微软智能鼠标，你会发现它的特殊之处。它的最大特色是在两键的中间有一个滚轮，在 IE 和 OFFICE 中可以实现一些特殊的功能，如按下滚轮即可实现自动卷动的功能，卷动的速度是可调的。但当智能鼠标停在任务条上时，Windows 并没有给它的滚轮指定什么功能。所以可以设想利用滚轮进行任务切换，这需要我们编制程序，获取鼠标的输入，调用相应的 API 函数，以达到我们的目的。笔者利用 Visual C + + 6.0 在 Windows 98 系统平台上，实现了利用智能鼠标进行任务切换，下面给出详细的实现思想和编程方法。

一、智能鼠标运动参数的获取

在进行程序设计之前，我们要明白智能鼠标是怎么工作的。每当我们转动滚轮时，一个转动消息将被发送到拥有鼠标输入焦点的窗口，在 Windows 98 系统中，这个消息是 WM_MOUSEWHEEL。在这个消息中，用一个参数 δ 给出滚动的方向。δ 值为正，表明流通轮向前转动；δ 为负，表明流通轮向后转动。并不是滚轮每转动一次，Windows 系统就进行相应的消息处理，而是当滚轮转过的角度达到一阀值时，系统才会对 WM_MOUSEWHEEL 消息进行处理，Windows 用一个常量 WHEEL_DELTA 定义了这个阀值，该常量的定义在 Zmouse. h 头文件中。

即使鼠标没有停留在当前具有焦点的窗口中，WM_MOUSEWHEEL 仍会被发送到该窗口中，这意味着如果要在任务条上使用智能鼠标进行任务切换，我们的程序就必须截获并处理被发送到任何窗口的每一个滚动消息，我们可以利用系统钩子函数来达到这个目的。

二、钩子函数的设置

一个钩子指向一个 Windows 消息处理过程，我们可以安装这样一个过程来监视系统内消息的传送，在消息被分发到指定的窗口过程之前，对它们进行处理。Windows 98 系统定义了多种钩子类型，我们可以根据需要，选择实现相应的钩子函数。例如，WH_MOUSE 类型的钩子可以让我们检测所有的鼠标输入；WH_KEYBOARD 类型的钩子可以让我们处理键盘消息。

我们既可以安装钩子来监视单一进程的消息序列，也可以监视系统内所有进程的消息。系统范围的钩子因为可以截获系统内所有的消息，所以必须要存在于一个 DLL 中，以监视系统内的所有进程。

要安装一个钩子，首先，我们要实现一个回调函数，即钩子函数，然后将这个函数的名字做为参数，调用 SetWindowsHookEx 函数。因为任何人都可以安装同样类型的钩子，所以 Windows 系统把这些同种类型的钩子组织成一个钩子链表。每当一种类型的消息产生时，Windows 就会把消息传给相应的钩子链表中的第一个钩子函数，钩子函数处理完毕后，就会调用 CallNextHookEx 函数，将此消息传给钩子链表中的下一个钩子函数。

调用 CallNextHookEx 时，必须指定下一个钩子函数的钩子句柄。这个句柄是调用者在利用 SetWindowsHookEx 函数安装自己时得到的。这意味着一个系统钩子必须放置在共享内存里，以使得它的句柄对所有进程都有效。在 Visual C + + 中，利用共享数据区来达到这个目的。所有的 DLL 都可以使用在共享数据区所声明和初始化的所有变量。

三、利用智能鼠标进行任务切换的过程

首先我们安装一个 WH_MOUSE 类型的系统范围内的钩子函数，这样，一旦转动鼠标的滚轮，我们的钩子函数就获得了控制权。在钩子函数里，首先判断鼠标是否停留在任务条上，如果是的话，再判断滚轮的转动角度是否达到了阀值，如果没有达到，则累加，如果达到了阀值，就立刻调用任务切换函数。在任务切换函数里，我们首先要得到任务切换窗口的句柄，并将其置为最前端窗口。然后判断哪个任务切换标签拥有输入焦点，并根据滚轮的转动方向决定是切换到上一个任务还是下一个任务；最后模拟键盘的输入，我们发一条键盘输入消息来完成整个任务切换过程。

四、安装和删除钩子函数

安装钩子函数十分简单，通过定义一个回调函数来达到目的，这个函数的定义如下：

```
void CALLBACK HookInstall(void)
```

```
{
  hMouseHook = SetWindowsHookEx(WH_MOUSE, MouseFunc, hInst, 0);
}
```

其中 hInst 是钩子函数所在的 DLL 进程的句柄，MouseFunc 就是要安装的钩子函数。hMouseHook 用来保存我们得到的钩子句柄。

当我们想卸掉鼠标的切换任务功能时，我们需要调用 HOOkRemove 函数，它的定义如下：

```
void CALLBACK HookRemove(void)
{;
        UnhookWindowsHookEx(hMouseHook)
}
```

五、钩子函数的实现

钩子函数的定义如下：

```
LRESULT CALLBACK MouseFunc(int nCode, WPARAM wParam, LPARAM lParam)
{
        __try
    {
        if (nCode = = HC_ACTION)//判断消息参数里是否包含有数据
                    if (wParam = = WM_MOUSEWHEEL )
                    {
                        RECT R;
        GetWindowRect(FindWindow( ″ Shell_TrayWnd″, NULL), & R);  //测试鼠标是否停留在任务条上
        if(PtInRect( & R, ((LPMOUSEHOOKSTRUCT) lParam) ->pt))
                        {
                            int zDelta;
    zDelta = (short)LOWORD(((LPMOUSEHOOKSTRUCT) lParam) ->dwExtraInfo);
//获得滚轮的角度,进行累加,然后判断是否已经达到阀值
                            zTotal + = zDelta;
                    if (abs(zTotal) >= WHEEL_DELTA)
                            {
                            if (zTotal > 0)
        //调用任务切换函数切换到下一个任务
                        SwitchTask(″next″);
                                else
          SwitchTask(″previous″); //切换到前一个任务
                        zTotal = 0; //滚轮所转过的角度重新置零
                            }
                        }
        // 调用钩子链表中的下一个钩子函数
            return (CallNextHookEx(hMouseHook, nCode, wParam, lParam));
    }
        __except(1)
    {
            return 0;
        }
}
```

六、任务切换函数的实现

任务切换函数的定义如下：

```
int SwitchTask(LPSTR order)
{
        HWND hTaskbar, hReBar, hTaskSw, hTabCtrl;
  int Sel, Count, Key;
        BOOL Next;
  //定义一个互斥对象
  HANDLE hMutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE, ″TaskSwitch_Mutex″);
  if (hMutex)
        {
            __try
        {
            //等待,直到拥有了互斥句柄或者过了 1 秒
if (WaitForSingleObject(hMutex, 1000) = =WAIT_OBJECT_0)
                    {
                        __try
                        {
                            //得到任务条窗口的句柄
        hTaskbar = FindWindow(″Shell_TrayWnd″, NULL);
                            //得到任务条窗口的子窗口句柄
        hReBar = FindWindowEx(hTaskbar, 0, ″ReBarWindow32″, NULL);
                            //得到任务切换窗口的句柄
  hTaskSw = FindWindowEx(hReBar, 0, ″MSTaskSwWClass″, NULL);
                            //得到标签控制窗口的句柄
  hTabCtrl = FindWindowEx(hTaskSw, 0, ″SysTabControl32″, NULL);
//得到当前拥有输入焦点的标签索引,并将它赋予 Sel 变量
  Sel = SendMessage(hTabCtrl, TCM_GETCURSEL, 0, 0);
                            //得到标签的数量
Count = SendMessage(hTabCtrl, TCM_GETITEMCOUNT, 0, 0);
                    Next = !strcmp(order, ″next″);
                            RECT R;
                        GetWindowRect(hTaskbar, & R);
            if (R.bottom - R.top > R.right - R.left)
                                Next = !Next;
  //下面模拟从键盘输入→、←、Home、End、和 空格键.
                            if (Next)
                            {
//当输入焦点在最后一个任务标签时,将切换到第一个任务,这里模拟了从键盘输入 Home 键
                            if (Sel = = Count - 1)
                                    Key = VK_HOME;
                                else
                                    Key = VK_RIGHT;
                            }
                            else
                            {
/* 当输入焦点在第一个任务标签时,将切换到最后一个任务,这里模拟了从键盘输入 End 键. */
                            if (Sel < = 0)
```

屏幕保护程序分析

张 磊

摘 要 本文对屏幕保护程序的工作原理进行了分析，并且给出了一个屏幕保护程序的完整例子。

关键字 屏幕保护程序（Screen Saver），Windows 应用程序接口（Windows API）

一、引言

Microsoft Win32 API 支持特殊的程序：屏幕保护程序。用户可以通过控制面板选择，设置和预览屏幕保护程序。当鼠标和键盘空闲一段时间后，屏幕保护程序即运行。

屏幕保护程序主要有两个作用：

1. 避免长时间显示同一画面，损坏显示器；

2. 避免屏幕上机密信息的泄露。

本文将对屏幕保护程序进行分析并给出一个屏幕保护程序的完整例子。

二、屏幕保护程序的运行过程

屏幕保护程序在 Windows 启动或者用户通过控制面板激活某个屏幕保护程序时将自动地被加载。

Windows 监视键盘输入和鼠标动作，发现用户的键盘输入和鼠标动作空闲一段时间后则启动屏幕保护程序（如果用户选择了某个屏幕保护程序）。然而，以下几种是例外情形：

1. 当前运行的程序不是基于 Windows 的程序；

2. 当前显示的是计算机辅助训练（CBT computer - based training）窗口；

3. 当前运行的程序收到 WM_SYSCOMMAND 消息（wParam 值是 SC_SCREENSAVE）后，并没有将该消息传递给 DefWindowProc 函数处理。

屏幕保护程序启动时，启动代码将创建一个全屏幕窗口。该窗口的窗口类声明如下：

```
WNDCLASS cls;
cls. hCursor        = NULL;
cls. hIcon  = LoadIcon(hInst,  MAKEINTATOM(ID_APP));
cls. lpszMenuName    = NULL;
cls. lpszClassName   = "WindowsScreenSaverClass";
cls. hbrBackground   = GetStockObject(BLACK_BRUSH);
cls. hInstance       = hInst;
cls. style  = CS_VREDRAW | CS_HREDRAW | CS_SAVEBITS
| CS_DBLCLKS;
cls. lpfnWndProc     = (WNDPROC) ScreenSaverProc;
cls. cbWndExtra      = 0;
cls. cbClsExtra      = 0;
```

屏幕保护程序运行后，用户输入任何键或鼠标动作终止其运行。

三、屏幕保护程序的结构

一个完整的屏幕保护程序包括若干部分。屏幕保护程序的 main 函数和必需的其他启动代码均包括在静态连接库

```
                    Key  = VK_END;
                else
                    Key  = VK_LEFT;
                }
        //只有一个任务时，没有必要进行任务切换
            if (!(Sel  = = 0 & &  Count  = = 1))
                {
                //使得任务条窗口成为最前端窗口
                SetForegroundWindow(hTaskbar);
/* 某些时候，调用 SetForegroundWindow()会导致标签控制失去焦点，这里通过发一个设置当前焦点的消息来确保我们能够选择的标签 */
            PostMessage(hTabCtrl,  TCM_SETCURSEL,  Sel,  0);
/* 最后，我们通过发送一条 WM_KEYDOWN 消息到标签控制，模拟从键盘的输入，从而完成任务的切换. */
PostMessage(hTabCtrl,  WM_KEYDOWN,  Key,  0);
PostMessage(hTabCtrl,  WM_KEYDOWN,  VK_SPACE,  0);
                }
                }
                __finally
                    {
                        //释放对互斥对象的拥有权
                        ReleaseMutex(hMutex);
                    }
                }
            }
            __finally
            {
                //关闭互斥对象
                CloseHandle(hMutex);
            }
        }
        return 0;
}
```

收稿日期：2000 年 10 月 31 日）

SCRNSAVE. LIB 中。对程序员而言，为创建一个屏幕保护程序只需要实现必要的函数，定义必要的资源。

1. 函数：

ScreenSaverProc, ScreenSaverConfigureDialog, RegisterDialogClasses;

2. 资源：

图标、字符串。

系统将根据情况，调用相应的程序：

屏幕保护程序设置 - - - >ScreenSaverConfigureDialog；

屏幕保护程序运行（测试） - - - >ScreenSaverProc。

四、有关函数的说明

＊＊＊＊＊＊＊ScreenSaverProc＊＊＊＊＊＊

函数原型：

LONG WINAPI ScreenSaverProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);

说明：

该函数处理屏幕保护程序窗口收到的消息。

返回值表示处理的结果，含义依赖于具体的消息而定。

注意，这里消息的处理有一个特别之处：

窗口函数 ScreenSaverProc 调用函数 DefScreenSaverProc 实现缺省消息处理（注意：不是 DefWindowProc），DefScreenSaverProc 再将不影响屏幕保护程序运行的消息传递给 DefWindowProc 处理。而一般情况下窗口函数是直接调用 DefWindowProc 实现缺省消息处理的。

注意：

该函数必须 EXPORT（在 . DEF 文件的 EXPORTS 节中指明）。

ScreenSaverProc 处理的一些重要消息：

消息	动作
WM_CREATE	从注册表中获取初始化数据。 为屏幕保护程序窗口设置一个定时器。 执行其他需要的初始化操作。
WM_ERASEBKGND	擦除屏幕保护程序窗口，为后面的描画做准备。
WM_TIMER	执行描画工作。
WM_DESTROY	清除定时器。 执行其他需要的清除工作。

＊＊＊＊＊ScreenSaverConfigureDialog＊＊＊＊＊

函数原型：

BOOL WINAPI ScreenSaverConfigureDialog(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

说明：

ScreenSaverConfigureDialog 是屏幕保护程序配置对话框的窗口函数。

如果屏幕保护程序提供配置对话框，那么就需要实现这个函数。如果 ScreenSaverConfigureDialog 处理了消息，则应返回 TRUE，否则返回 FALSE。这里要特别指出的是对于 WM_INITDIALOG 消息的处理：若 ScreenSaverConfigureDialog 调用 SetFocus 将配置对话框的某个控件设置为键盘输入焦点，则应返回 FALSE；否则，如果没有设置键盘输入焦点，应该返回 TRUE，此时系统将自动设置输入焦点。对话框模板的 ID 必须是 DLG_SCRNSAVECONFIGURE（在 SCRNSAVE. H 中已经定义）。

屏幕保护程序给该对话框指定了缺省的窗口类时，将用到该函数。如果对话框模板中没有指明窗口类时，将使用缺省窗口类。尽管对话框函数与窗口函数很象，但是它不需调用 DefWindowProc 函数来处理其他消息（实际上，那些未处理的消息将由缺省的对话框函数处理）。

注意：

该函数必须 EXPORT（在 . DEF 文件的 EXPORTS 节中指明）。

＊＊＊＊＊RegisterDialogClasses＊＊＊＊＊

函数原型：

BOOL WINAPI RegisterDialogClasses (HANDLE hInst);

说明：

该函数用来注册屏幕保护程序的配置对话框所需要的非标准窗口类。

该函数只被 SCRNSAVE. LIB 中的程序调用。如果屏幕保护程序的配置对话框不需注册任何特殊的窗口类，那么应返回 TRUE。

注意：

该函数不需 EXPORT。

＊＊＊＊＊DefScreenSaverProc＊＊＊＊＊

说明：

提供屏幕保护程序未处理的消息处理。屏幕保护程序的 ScreenSaverProc 窗口函数应该调用 DefScreenSaverProc 来提供缺省的消息处理。DefScreenSaverProc 将把不影响屏幕保护程序运行的消息接着传递给 DefWindowProc。

具体的消息处理见下表：

消息	动作
WM_ACTIVATE， WM_ACTIVATEAPP， WM_NCACTIVATE	如果 wParam 参数是 FALSE，终止屏幕保护程序。 表明屏幕保护程序失去输入焦点。屏幕保护程序被通过发送 WM_CLOSE 消息关闭。
WM_SETCURSOR	设置光标为 NULL，其效果是将光标从屏幕上清除。
WM_LBUTTONDOWN WM_RBUTTONDOWN WM_MBUTTONDOWN WM_KEYDOWN　WM_KEYUP WM_MOUSEMOVE	调用 PostQuitMessage 函数以关闭屏幕保护程序窗口。
WM_DESTROY	向屏幕保护程序的窗口 Post 一个 WM_CLOSE 消息。
WM_SYSCOMMAND	如果 WM_SYSCOMMAND 消息的 wParam 参数是 SC_CLOSE 或者 SC_SCREENSAVE 则返回 FALSE；

五、有关资源的说明

图标　屏幕保护程序需要提供一个图标。当屏幕保护程序独立运行时，显示这个图标。该图标的 ID 必须是 ID_APP（ID_APP 在 SCRNSAVE. H 中已经定义）。

字符串　屏幕保护程序的资源文件应该包括一个特殊的说

明性字符串，这个字符串是屏幕保护程序的名字，控制面板将显示这个字符串来表示该屏幕保护程序。该字符串的 ID 是 IDS_DESCRIPTION 。字符串 idsIniFile 指定 . ini 文件的名字。

六、一个屏幕保护程序的完整实例

该屏幕保护程序运行时不断在屏幕上显示字符串。显示位置是随机的，颜色不断变化，用户可以设置位置更新的时间间隔和显示的字符串内容。

修改 ScreenSaverProc 对 WM_TIMER 的处理，可以使屏幕保护程序更加漂亮。

完整的创建过程如下：

1. 创建必要的文件

a. 模块文件

模块文件 SS. cpp 内容：

```
#include <windows. h>
#include <SCRNSAVE. H>
#include "resource. h"
#define MINVEL 1 /* minimum redraw – speed value */
#define MAXVEL 100/* maximum redraw – speed value */
#define DEFVEL 2/* default redraw – speed value */
LONG lSpeed = DEFVEL; /* redraw – speed variable */
CHAR szTemp[100]; /* temporary array of characters */
CHAR szRedrawSpeed[] = "Redraw Speed"; /* . INI speed entry */
CHAR szDisplayContent[ ] = " Display Content"; /* . INI speed entry */
CHAR szHelloMsg[100] = " Screen Saver (ZhangKung @ 21cn. com)";
LONG WINAPI ScreenSaverProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;    /* device – context handle */
    RECT rc;    /* RECT structure */
    BYTE r, g, b;   /* color */
    int x, y;     /* position */
    static UINT uTimer; /* timer identifier */
  switch(message)
  {
    case WM_CREATE:
    /* Retrieve the application name to szAppName */
    /* szAppName will be used by control panel */
    /* to refer to our screen saver ! */
    LoadString(hMainInstance, idsAppName, szAppName, sizeof(szAppName));
    /* Retrieve the . INI (or registry) filename. */
      LoadString(hMainInstance, idsIniFile, szIniFile, sizeof(szIniFile));
/* Retrieve any redraw – speed data from the registry. */
lSpeed = GetPrivateProfileInt(szAppName, szRedrawSpeed, DEFVEL, szIniFile);
 /* Retrieve any display content data from the registry. */
    GetPrivateProfileString(szAppName, szDisplayContent, szHelloMsg, szHelloMsg, sizeof(szHelloMsg), szIniFile);
        /* Set a timer for the screen saver window. */
  uTimer = SetTimer(hwnd, 1, lSpeed * 100 , NULL);
          // seed the random number generator
          srand( uTimer );
          break;
    case WM_ERASEBKGND:
      /* paint the background as appropriate. */
      hdc = GetDC(hwnd);
      GetClientRect (hwnd, & rc);
  FillRect (hdc, & rc, GetStockObject(BLACK_BRUSH));
      ReleaseDC(hwnd, hdc);
      break;
    case WM_TIMER:
//*********************************
//* HERE DO WHAT A SCREEN SAVER SHOULD DO *
//*********************************
//*   TODO:                       *
//*       Add your specific code here  *
//*       to rendor innovative imagination  *
//*********************************
      hdc = GetDC(hwnd);
      GetClientRect(hwnd, & rc);
        {
        SetBkMode(hdc, TRANSPARENT);
            SetTextAlign(hdc, TA_CENTER);
            // set text color to black
            // and draw szHelloMsg
            // producing a effect of erasing!
            SetTextColor(hdc, RGB(0, 0, 0));
  TextOut(hdc, x%GetSystemMetrics(SM_CXSCREEN), y%GetSystemMetrics(SM_CYSCREEN), szHelloMsg, strlen(szHelloMsg));
            // select new random position & color
            x = rand();
            y = rand();
            r = rand()%256;
            g = rand()%256;
            b = rand()%256;
            SetTextColor(hdc, RGB(r, g, b));
      // print szHelloMsg on new position using new color
  TextOut(hdc, x%GetSystemMetrics(SM_CXSCREEN), y%GetSystemMetrics(SM_CYSCREEN), szHelloMsg, strlen(szHelloMsg));
        }
        ReleaseDC(hwnd, hdc);
      break;
    case WM_DESTROY:
      /* do clean_up work */
      if (uTimer)
        KillTimer(hwnd, uTimer);
      break;
  }
   return DefScreenSaverProc(hwnd, message, wParam, lParam);
}
```

```
BOOL WINAPI ScreenSaverConfigureDialog(HWND hDlg,
UINT message, WPARAM wParam, LPARAM lParam)
{
   static HWND hSpeed; /* handle of speed scroll bar */
   static HWND hOK;    /* handle of OK push button */
   switch(message)
   {
      case WM_INITDIALOG:
/* Retrieve the application name from the .RC file. */
      LoadString(hMainInstance, idsAppName, szAppName,
APPNAMEBUFFERLEN);
         /* Retrieve the .INI (or registry) filename. */
LoadString(hMainInstance, idsIniFile, szIniFile, MAXFILELEN);
/* Retrieve any redraw-speed data from the registry. */
lSpeed = GetPrivateProfileInt(szAppName, szRedrawSpeed,
DEFVEL, szIniFile);
/* Retrieve any Display-Content data from the registry. */
      GetPrivateProfileString(szAppName, szDisplayContent,
szHelloMsg, szHelloMsg, sizeof(szHelloMsg), szIniFile);
      /* Initialize the redraw-speed scroll bar control. */
         hSpeed = GetDlgItem(hDlg, ID_SPEED);
SetScrollRange(hSpeed, SB_CTL, MINVEL, MAXVEL, FALSE);
         SetScrollPos(hSpeed, SB_CTL, lSpeed, TRUE);
               /* Initialize the Display Content */
SetDlgItemText(hDlg, IDC_DISPLAYCONTENT, szHelloMsg);
 /* Retrieve a handle of the OK push button control. */
         hOK = GetDlgItem(hDlg, ID_OK);
         return TRUE;
      case WM_HSCROLL:
/* Process scroll bar input, adjusting the lSpeed value as
appropriate. */
               switch (LOWORD(wParam))
               {
               case SB_PAGEUP:
                     --lSpeed;
                     break;
               case SB_LINEUP:
                     --lSpeed;
                     break;
               case SB_PAGEDOWN:
                     ++lSpeed;
                     break;
               case SB_LINEDOWN:
                     ++lSpeed;
                     break;
               case SB_THUMBPOSITION:
                     lSpeed = HIWORD(wParam);
                     break;
               case SB_BOTTOM:
                     lSpeed = MINVEL;
                     break;
               case SB_TOP:
                     lSpeed = MAXVEL;
                     break;
               case SB_THUMBTRACK:
               case SB_ENDSCROLL:
                     return TRUE;
                     break;
               }
               if ((int) lSpeed <= MINVEL)
                     lSpeed = MINVEL;
               if ((int) lSpeed >= MAXVEL)
                     lSpeed = MAXVEL;
   SetScrollPos((HWND) lParam, SB_CTL, lSpeed, TRUE);
         break;
            case WM_COMMAND:
         switch(LOWORD(wParam))
         {
            case ID_OK:
/* Write the current redraw-speed variable and display
content to the .INI file. */
               wsprintf(szTemp, "%ld", lSpeed);
      WritePrivateProfileString(szAppName, szRedrawSpeed,
szTemp, szIniFile);
GetDlgItemText(hDlg, IDC_DISPLAYCONTENT, szHelloMsg,
sizeof(szHelloMsg));
      WritePrivateProfileString(szAppName, szDisplayContent,
szHelloMsg, szIniFile);
            case ID_CANCEL:
   EndDialog(hDlg, LOWORD(wParam) == ID_OK);
               return TRUE;
         }
   }
   return FALSE;
}
BOOL WINAPI RegisterDialogClasses(HANDLE hInst)
{
   // we donot need register any more window classes
   // just returning is ok.
   return TRUE;
}
```

b. **模块定义文件**

模块定义文件 SS.DEF 内容：

```
DESCRIPTION 'SCRNSAVE: Test'
EXETYPE WINDOWS
CODE    MOVEABLE
DATA    MOVEABLE MULTIPLE
HEAPSIZE  1024
STACKSIZE 4096
;* * * * * * * * * * * * * * * * * *
; see the EXPORTS section below
;* * * * * * * * * * * * * * * * * *
EXPORTS
      ; EXPORT ScreenSaverProc
      ScreenSaverProc
        ; EXPORT ScreenSaverConfigureDialog
      ScreenSaverConfigureDialog
;* * * * * * * * * * * * * * * * * *
```

c. 资源文件

资源头文件 resource. h 内容

```
#define ID_SPEED                    1
#define ID_CANCEL                   2
#define ID_OK                       3
#define IDC_STATIC                  1000
#define IDC_DISPLAYCONTENT    1001
```

资源文件 ss. rc 内容

```
#include "resource. h"
#include <windows. h>
#include <SCRNSAVE. H>
// Dialog
DLG_SCRNSAVECONFIGURE DIALOG DISCARDABLE 6, 18, 159, 88
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "X Screen Saver Configurator"
FONT 8, "MS Shell Dlg"
BEGIN
    GROUPBOX        "Redraw Speed", 101, 0, 6, 98, 40
    SCROLLBAR       ID_SPEED, 5, 31, 89, 10
    LTEXT           "Fast", 103, 6, 21, 20, 8
    LTEXT           "Slow", 104, 75, 21, 20, 8
    PUSHBUTTON      "OK", ID_OK, 110, 29, 40, 14
    PUSHBUTTON      "Cancel", ID_CANCEL, 110, 51, 40, 14
    ICON            ID_APP, ID_APP, 121, 3, 20, 20
    GROUPBOX  "Display Content", IDC_STATIC, 0, 48, 97, 31
  EDITTEXT  IDC_DISPLAYCONTENT, 5, 63, 87, 11, ES_AUTO-
HSCROLL
END
// Icon
ID_APP    ICON    DISCARDABLE    "ss. ico"
// String Table
STRINGTABLE DISCARDABLE
BEGIN
    idsIniFile              ". \\SSKung. ini"
END
STRINGTABLE DISCARDABLE
BEGIN
    IDS_DESCRIPTION         "ZhangKung's Screen Saver"
END
```

图标文件 Ss. ico 内容

选择一个你喜欢的图标，将该图标文件名字改为 Ss. ico。

2. 编译

新建 Win32 Application 工程。

将上述 5 个文件加入工程。

修改工程的设置

Project - >Setting - >Link - >Category 选择 General；

Project - >Setting - >Link - >Output file name 设置为 Debug / SS. SCR；

Project - >Setting - >Link - >Object / library modules 增加 SCRNSAVE. LIB。

最后 Build 即可。

3. 屏幕保护程序的安装

将 SS. SCR 拷贝至 Windows 路径（例如：c:\windows）。

注：屏幕保护程序正确的安装后名字将出现在控制面板的屏幕保护程序列表中。为此：

a. 屏幕保护程序的扩展名必须是 . SCR ；

b. 屏幕保护程序必须在 Windows 路径下。

现在，可以将新屏幕保护程序设置为当前的屏保。试一试，不错吧。

（收稿日期：2000 年 11 月 8 日）

“BTI”开拓图书行销新模式，华彩进军中国计算机图书市场

“BTI”是一个概念的名称。对于她来说，2001 年 1 月 9 日是具有历史意义的一个纪念日：华彩对软件的学习、使用与图书营销赋予了一个新生命“BTI”。

国内的软件学用者将得到华彩提供的一个全新的学习模式“BTI”。

新世纪伊始，著名的软件发行商、教育培训机构和图书出版商，来自台湾的华彩网络集团在北京宣布开始进军中国 IT 图书市场，并以其独特的“BTI”（图书 + 培训 + 网络）概念来打造 e 世纪的图书行销模式。

作为全球最大软件开发平台——微软（Microsoft）在港台地区惟一授权的图书出版商，华彩软件在进军中国大陆市场前曾出版微软图书 600 余种，此次华彩软件进军中国市场，是以专业的技术能力，丰富的实践经验，优良的翻译水平和写作品质，为中国大陆的计算机用户提供系列而完整的计算机学习丛书，务使带给每一位读者全新而愉快、轻松的高品质计算机学习经验。

“BTI”作为一个行销概念，是华彩为其图书销售商及消费者提供的具有增值服务功能的行销模式。对于“BTI”的推出，华彩软件学习事业部总监王刚先生如此解释：“BTI”是图书 + 培训 + 网络的三位一体的结合概念，它专注于 IT 书籍的出版发行、对计算机用户的全方位、多层次的培训和基于互联网络的在线学习平台（E - Learning），为企业的系统培训和个人计算机的学习提供最合理化、科学化的解决方案。

据悉，华彩目前已与微软、Micromedia、Caldera、Shade 等全球知名的软件开发商结盟，整合产业资源和优势，在中国大陆全力开拓图书市场，并将先后推行“世纪行”全国计算机应用技术培训项目，在北京、上海、广州三大城市建立教育训练中心，利用因特网打造全语言版的在线学习平台等多项设施，力图打造“华彩软件”（Softchina）在国内的金字招牌。

用 PB 查询 Sybase SQL Server 的表和存储过程信息

孙友军

基于客户/服务器体系的前端开发，PB 前端常常需要知道库端的表和存储过程等的详细信息。而常到库端查询比较麻烦。笔者编制一个小程序可以在客户端方便快捷地查询 Sybase SQL Server 的表和存储过程信息。　表和存储过程是 Sybase 数据库中最常用到的，其它如触发器等，在此不再赘述）方法如下：

一、 建立文本文件 cx. ini

内容如下　不含说明）：

```
[options]
LogID =
logpass =
database =
servername =
dbms =syc sybase system 10
```

说明：应正确填写等号右边的参数。LogID、logpass 分别为登录用户和口令，database 为登录的数据库名，servername 为客户端安装的 Sybase Client 的 SQLEDIT 中配置的 servername

二、建立应用，脚本如下

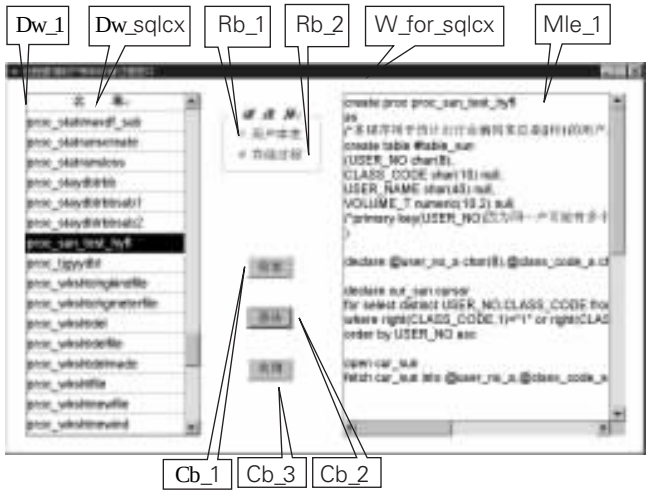
1. OPEN 事件脚本：

```
//打开等待窗口
w_popup_forwait w_wait
open(w_wait)
//检查系统配置文件是否存在
if not fileexists(″. \cx. ini″) then
    messagebox(″系统错误″, ″系统配置文件 cx. ini 找不到或 ~r ~n″ +& ″已被破坏, 无法继续运行″, stopsign!)
    halt close
end if
//连接数据库
sqlca. LogID = ProfileString(″. \cx. ini″, ″options″, ″ LogID ″, ″″)
sqlca. logpass = ProfileString(″. \cx. ini″, ″options″, ″ log-pass ″, ″″)
sqlca. database = ProfileString(″. \ cx. ini″, ″ options″, ″ database ″, ″″)
sqlca. servername = ProfileString(″. \cx. ini″, ″options″, ″ servername ″, ″″)
sqlca. dbms =ProfileString(″. \cx. ini″, ″options″, ″ dbms ″, ″″)
connect using sqlca;
//设置鼠标指针显示类型
SetPointer(HourGlass!)
//如果登录失败, 返回调用
if sqlca. sqlcode <>0 then
  messagebox(″登录″, ″登录数据库失败, 请检查输入的参数″, stopsign!)
      close(w_wait)
      return -1
end if
//关闭等待窗口, 打开主窗口
close(w_wait)
open(w_for_sqlcx)
```

2. CLOSE 事件脚本：

```
disconnect using sqlca;
```

三、建立窗口、控件和脚本如下



1. 建立数据窗口 dw_sqlcx：

数据窗口数据源为 Sybase 系统表 sysobjects 的 name，id 列，where 检索条件为 sysobjects. type = tab_or_pro （ab_or_pro 为定义的检索参数，字符型）。

2. 为 W_for_sqlcx 声明实例变量：

```
string tab_or_pro_name。
```

3. W_for_sqlcx 的 open 事件脚本为：

```
dw_1. settransobject(sqlca)
rb_1. checked =true
cb_1. enabled =true
cb_2. enabled =false
dw_1. retrieve(″U″)。
```

4. Dw_1 的 clicked 事件脚本为：

```
if row < =0 then return
this. selectrow(0, false)
this. selectrow(row, true)。
```

5. Dw_1 的 losefocus 事件脚本为：

```
tab_or_pro_name =dw_1. getitemstring(dw_1. getrow(      ), ″
Name″)。
```

6. Rb_1 的 clicked 事件脚本为：

```
tab_or_pro_name =″″
if this. checked =true then
      cb_1. enabled =true
   cb_2. enabled =false
      mle_1. text =′′
      dw_1. retrieve(″U″)
end if。
```

7. Rb_2 的 clicked 事件脚本为：

```
tab_or_pro_name =″″
if this. checked =true then
      cb_1. enabled =false
      cb_2. enabled =true
      mle_1. text =′′
      dw_1. retrieve(″P″)
end if。
```

8. Cb_1 的 clicked 事件脚本为：

```
if tab_or_pro_name =″″ or isnull(tab_or_pro_name) then
mle_1. text =″请先选择表名!″
return
end if
pointer oldpointer
oldpointer = SetPointer(HourGlass!)
//以下得到记录数
int bb
string sqlstatement
sqlstatement =″SELECT count(*) FROM ″ +tab_or_pro_
name;
DECLARE my_cursor DYNAMIC CURSOR FOR SQLSA;
PREPARE SQLSA FROM : sqlstatement;
OPEN DYNAMIC my_cursor;
FETCH my_cursor INTO : bb;
CLOSE my_cursor;
//以下通过游标得到字段名集
string aaa
string bbb
bbb =″″
DECLARE adf_curs CURSOR FOR
      SELECT syscolumns. name FROM syscolumns WHERE
syscolumns. id =
      (
      select id
      from sysobjects
      where name =: tab_or_pro_name
      )
      using sqlca;
OPEN adf_curs;
FETCH adf_curs INTO : aaa;
aaa =aaa +″ ~r ~n″
DO WHILE SQLCA. sqlcode = 0
bbb =bbb +aaa
FETCH adf_curs INTO : aaa;
aaa =aaa +″ ~r ~n″
loop
CLOSE adf_curs;
//以下结果输出
SetPointer(oldpointer)
mle_1. text =″该表记录个数为: ″ +string(bb) +″ 个″ +″ ~
r ~n″ +″ ~r ~n″ +″ ~r ~n″ +& ″ * * * * * * * * * * * * *
字段名如下 * * * * * * * * * * * * ″ +″ ~r ~n″ +″ ~r ~
n″ +bbb。
```

9. Cb_2 的 clicked 事件脚本为

```
if tab_or_pro_name =″″ or isnull(tab_or_pro_name) then
mle_1. text =″请先选择存储过程名!″
return
end if
pointer oldpointer
oldpointer = SetPointer(HourGlass!)
//以下通过游标得到存储过程内容集
string aaa
string bbb
bbb =″″
DECLARE adf_curs CURSOR FOR
   SELECT syscomments. text FROM syscomments WHERE
syscomments. id =
   (
   select id
   from sysobjects
   where name =: tab_or_pro_name
   )
   using sqlca;
OPEN adf_curs;
FETCH adf_curs INTO : aaa;
DO WHILE SQLCA. sqlcode = 0
bbb =bbb +aaa
FETCH adf_curs INTO : aaa;
loop
CLOSE adf_curs;
//以下结果输出
SetPointer(oldpointer)
mle_1. text =bbb。
```

10. Cb_3 的 clicked 事件脚本为

```
close(parent)。
```

四、编译成可执行文件

编译成 EXE 文件，即可在 Win95 或 Win98 下运行。以上语句均在 PB5. 0 上测试通过。

（收稿日期：2000 年 10 月 23 日）

利用 ADO 扩展管理数据库—ADOX、JRO

陈　省

众所周知，Delphi 最主要的特性之一就是可以用来开发数据库程序。但是在 Delphi4 以前数据库的开发主要是基于 BDE（Borland 数据库引擎）的，但 BDE 存在很多问题，包括过于庞大、不易分发、不稳定等等，Borland 已经决定今后将放弃 BDE 的进一步开发了。幸运的是在 Delphi 5 中包括了一个很重要的新特性那就是可以用 ADO Express 来开发数据库程序了。ADO Express 控件提供了对微软的 ActiveX Data Objects ADO 的封装（ADO 属于微软通用数据存取架构的一部分），给我们提供了一组 BDE 以外的数据库编程机制。ADO 的功能已经是非常强大的了，但除此之外微软还提供了 ADO 的扩展来进一步拓展了它的应用。

接下来我们就来研究一下 ADO 的扩展，第一部分主要是关于 DDL（数据定义语言）、安全机制（ADOX）和 Jet 及 Replication 对象库（JRO）。第二部分我们要来研究一下多维 ADO（ADO MD）。

ADOX 的介绍

ADOX 可以用来执行一系列用 ADO 无法单独实现的功能。例如我们可以用 ADOX 提取数据库的用户信息或创建新的用户账号。ADOX 扩展了 ADO 的对象模型，包括了 10 个新的对象，可以和 ADO 配合使用。比如我们可以用 ADO Connection 对象来连接到一个数据源，并提取出元数据（注意：元数据是数据库的结构描述，象表、字段、索引、关键字段、存储过程等，而不是数据库包含的数据内容）。在现代大多数数据库使用 SQL 来定义元数据。在 ADOX 出现以前，提取元数据的唯一的方法是使用 ADO Connection 对象的 OpenSchema 方法，而且要想创建新的数据库对象，我们还必须使用基于 SQL 的 DDL 对象和 ADO Command 对象。

ADOX 提供了一种不需要懂 SQL 就可以操纵元数据的方式。注意：ADOX 并不支持所有的数据库，它只局限于微软的 Access、SQL Server 以及其它几种数据库。要想详细了解这方面的信息，请参看 http://www.microsoft.com/data。

ADOX 的对象模型如图 1，图 2 显示了表的对象模型。ADOX 的最上层对象是 Catalog。它包括表、视图、过程、用户以及账号。Catalog 对象可以用来打开数据库（通过 ADO Connection 对象），或创建一个新的数据库。到 ADO 2.1 版本为止，我们只能创建 Jet 4.0 数据库，在未来版本有可能扩展到其它数据库。

我们可以通过 Catalog 对象对表、过程以及视图操作。比如，通过枚举表集合，我们可以知道当前数据库有哪些表。

图 1　ADOX 对象模型

图 2　表的对象模型示意图

表 1　ADOX 对象的简单描述

对象	说明
Catalog	包含描述数据源模式目录的集合，提供对表、视图、用户、过程和账号的操作。
Column	表示表、索引或关键字的列。
Group	表示在安全数据库内具有访问权限的帐号。
Index	表示数据库表的索引。
Key	表示数据库表中的主关键字、外部关键字或唯一关键字字段。
Procedure	表示存储过程或查询。
Table	表示包括列、索引和关键字的数据库表。
User	表示在安全数据库中具有访问权限的用户帐号。
View	表示记录或虚拟表的过滤集。

更进一步，我们可以获得一个表的字段、索引、关键字段等元数据信息。通过用户和账号对象集合，我们可以获得数据库的安全信息（注意：这个功能只对安全数据库有效，对于 Access 数据库，我们必须在数据源连接字符串中包括 System.mdw）。

ADOX 另一个强大的功能是我们可以用 Catalog 对象来创建新的数据库，并通过表、字段、索引对象的 Add 方法向数据库添加表，字段，索引和关键字段。

创建一个简单的 ADOX 察看器

接下来我们就演示一下如何在 Delphi 中使用 ADOX。我们将创建一个应用程序，主要功能是：

●显示数据库的元数据。

●显示数据库对象的属性。

●显示视图和存储过程的源代码。

我们先创建一个新的项目，在主窗体上放置下列控件 MainMenu TreeView Memo 和 StatusBar。完成的程序示意如图 3。

图 3 演示程序示意图

接下来，我们必须引入 ADOX 的类型库，选择菜单项 Project | Import Type Library，然后从类型库列表中选择 Microsoft ADO Ext. 2.1 for DDL and Security。为了避免同 VCL 已有控件名冲突，对 ADOX 类要重新命名 比如 TTable 可以改成 TADOXTable ，按 Create Unit 按钮来生成接口文件。Delphi 会生成 ADOX_TLB.pas 文件，我们需要在项目的 Uses 部分引用它。

现在我们创建一个 File | Open Catalog 菜单项，并生成一个 OnClick 事件处理函数，代码如下：

```
procedure TForm1.OpenCatalog1Click(Sender: TObject);
begin
  // Get DataSourceName through standard MS dialog box.
  DS := PromptDataSource(Application.Handle, '');
  // If user selected one...
  if DS <> '' then
    BrowseData(DS);
end;
```

这里我们使用了实现在 ADODB 单元中的 PromptDataSource 方法来显示标准的数据连接属性对话框 见图 4 。

注意：这里我们应该选择 Microsoft Jet 4.0 OLE DB Provider 作为数据源，连接到 Access 的样例数据库 Northwind.mdb 别的库不行），然后程序会调用 BrowseData 过程。这个过程会把从数据源中提取的元数据显示在 TreeView 中，源码如下：

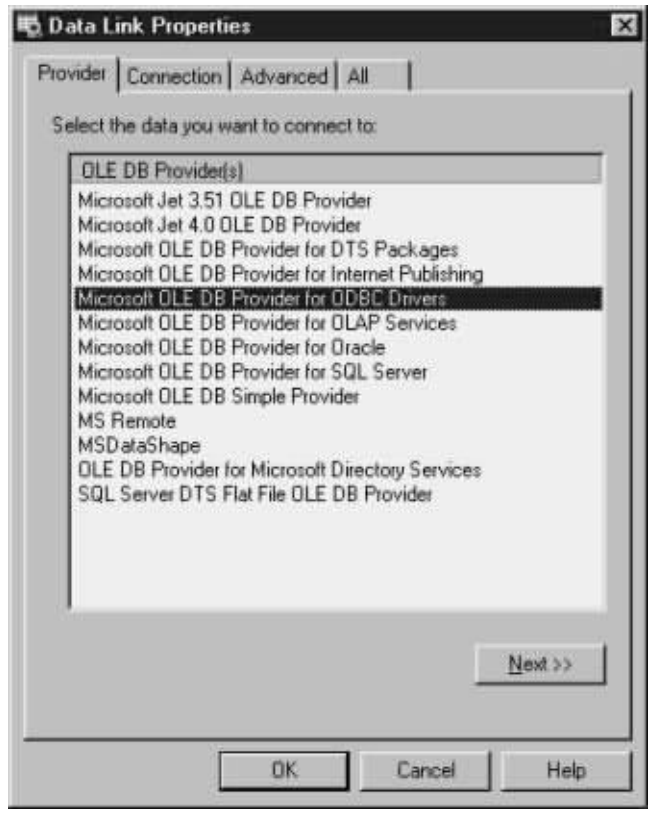


图 4 数据连接属性对话框

```
procedure TForm1.BrowseData(DataSource: string);
var
  RootNode : TTreeNode;
  OneNode  : TTreeNode;
  SubNode  : TTreeNode;
  I        : Integer;
  OldCursor: TCursor;
begin
  // 改变光标为沙漏型
  OldCursor := Screen.Cursor;
  Screen.Cursor := crHourglass;
  StatusBar1.Panels[0].Text :=
    'Extracting metadata, please wait.';
  // 清空 TreeView 和 Memo
  ClearTree;
  Memo1.Lines.Clear;
  Application.ProcessMessages;
  //连接数据源
  Catalog._Set_ActiveConnection(DataSource);
  RootNode := TreeView1.Items.Add(nil, 'Catalog');
  //添加表信息
      OneNode := TreeView1.Items.AddChild(RootNode,
'Tables');
  for I := 0 to Catalog.Tables.Count - 1 do begin
   SubNode := TreeView1.Items.AddChild(OneNode, Cata-
log.Tables[I].Name);
    //处理字段、索引及关键字段
    ProceedTables(Catalog.Tables[I], SubNode);
  end;
  //添加视图信息
```

```
  if CheckViews(Catalog) then begin
       OneNode := TreeView1.Items.AddChild(RootNode,
´Views´);
    for I := 0 to Catalog.Views.Count - 1 do
      SubNode := TreeView1.Items.AddChild(OneNode,
Catalog.Views[I].Name);
   end;
  //添加过程信息
OneNode := TreeView1.Items.AddChild(RootNode, ´Proce-
dures´);
  for I := 0 to Catalog.Procedures.Count - 1 do
    SubNode := TreeView1.Items.AddChild(OneNode, Cat-
alog.Procedures[I].Name);
  RootNode.Expand(False);
  //恢复缺省光标清空状态条
  Screen.Cursor := OldCursor;
  StatusBar1.Panels[0].Text := ´´;
end;
```

通过三个循环我们遍历了表、视图对象。每个对象都被放到了 TreeView 中的适当分支上，每个表都需要处理对应的字段、索引和关键字段列表。我们是通过 ProceedTables 过程来实现的，代码如下：

```
procedure TForm1.ProceedTables(T:Table; N: TTreeNode);
var
  I      : Integer;
  SubNode : TTreeNode;
begin
  //添加字段信息
  if T.Columns.Count > 0 then
   SubNode := TreeView1.Items.AddChild(N, ´Columns´);
  for I := 0 to T.Columns.Count - 1 do
    TreeView1.Items.AddChild(SubNode, T.Columns.Item[I]
.Name);
  //添加索引信息
  if T.Indexes.Count > 0 then
    SubNode := TreeView1.Items.AddChild(N, ´Indexes´);
  for I := 0 to T.Indexes.Count - 1 do
    TreeView1.Items.AddChild(SubNode, T.Indexes.Item[I]
.Name);
  //添加关键字段信息
  if T.Keys.Count > 0 then
    SubNode := TreeView1.Items.AddChild(N, ´Keys´);
  for I := 0 to T.Keys.Count - 1 do
      TreeView1.Items.AddChild(SubNode, T.Keys.Item[I]
.Name);
end;
```

这里我们边历了每个表对应的字段、索引、关键字段信息。

回到 BrowseData 过程，会看到在循环视图对象前，我们执行了下列校验过程：

```
if CheckViews(Catalog) then ...
```

这是为了避免不支持视图的数据源可能带来的错误，CheckView 函数代码如下：

```
function CheckViews(C: _Catalog): Boolean;
var
  I: Integer;
begin
  try
   I := C.Views.Count;
   CheckViews := True;
  except
   CheckViews := False;
  end;
end;
```

为了获得更详细的对象信息，我们通过实现 TreeView 控件的 OnChange 事件来显示元数据，代码如下：

```
procedure TForm1.TreeView1Change ( Sender: TObject;
Node: TTreeNode);
begin
  if Node.Parent.Parent <> nil then
    case Node.Parent.Text[1] of
´C´ : ViewColumns(Node.Parent.Parent.Text, Node.Text);
´I´ : ViewIndexes(Node.Parent.Parent.Text, Node.Text);
      ´K´ : ViewKeys(Node.Parent.Parent.Text, Node.Text);
      ´T´ : ViewTables(Node.Text);
      ´V´ : ViewProps(Node.Text);
      ´P´ : ProcProps(Node.Text);
    end;
end;
```

我们通过调用 ViewColumns、ViewIndexs、ViewKeys 过程来显示被选对象的属性，表 2、表 3、表 4 是相应对象的详细

表 2 字段对象属性

属性	说明
Attributes	描述列特性。
DefinedSize	指示列的规定最大大小。
NumericScale	指示列中数值的范围。
ParentCatalog	指定表或列的父目录以便访问特定提供者的属性。
Precision	指示列中数值的最高精度。
RelatedColumn	指示相关表中相关列的名称 仅关键字列)。
SortOrder	指示列的排序顺序 仅索引列)。
Type	

表 3 索引对象属性

属 性	说 明
Clustered	指示索引是否被分簇。
IndexNulls	指示在索引字段中具有 Null 值的记录是否有索引项。
PrimaryKey	指示索引是否代表表的主关键字。
Unique	指示索引关键字是否必须是唯一的。

表 4 关键字段属性

属 性	说 明
DeleteRule	指示主关键字被删除时将执行的操作。
RelatedTable	指示相关表的名称。
Type	指示列的数据类型。
UpdateRule	指示主关键字被更新时会执行的操作。

信息。

ViewProps 和 ProcProps 过程显示视图和存储过程的源代码。ProcProps 代码如下，ViewProps 过程类似：

```
procedure TForm1.ProcProps(Name: string);
var
  S       : string;
  Disp    : IDispatch;
  Command : _Command;
begin
  S := ´PROCEDURE : ´ + Catalog.Procedures.Item[Name].Name;
  S := S + ^M^J + ´Created : ´ +
  VarToStr(Catalog.Procedures.Item[Name].DateCreated);
  S := S + ^M^J + ´Modified : ´ +
  VarToStr(Catalog.Procedures.Item[Name].DateModified);
  if CmdSupported(Catalog.Procedures.Item[Name]) then
begin
  Disp := Catalog.Procedures.Item[Name].Get_Command;
    Command := Disp AS Command;
    S := S + ^M^J^M^J + Command.Get_CommandText;
  end;
  Memo1.Text := S;
end;
```

存储过程可以通过 ADO Command 对象获得。我们这里使用 Get_Command 方法来获得 Command 对象的 IDispatch 接口，然后调用接口的 Get_CommandText 方法来获得存储过程的源代码。

现在我们知道如何使用 ADOX 来获得数据源的元数据了。ADOX 的另外一个强大的功能就是不需要复杂的 SQL DDL 语句就可以创建数据库。

创建数据库

首先我们需要创建 Catalog 对象的一个实例。通过 CataLog 对象我们不仅可以指定要创建的数据库的类型，还可以指定数据库文件的位置。代码如下：

```
const
  BaseName = ´c:\data\demo.mdb´;
 DS = ´Provider=Microsoft.Jet.OLEDB.4.0;Data Source=´
 + BaseName;
var
  Catalog : TADOXCatalog;
  ...
  //创建 Catalog 对象的一个实例
  Catalog := CoCatalog.Create;
  //如果数据库已经存在，就删除它
  if FileExists(BaseName) then
    DeleteFile(BaseName);
  //创建新的 .mdb 文件
  Catalog.Create(DS);
  //指定活动连接
  Catalog._Set_ActiveConnection(DS);
  ...
```

创建好数据库后，我们要向库中添加表和字段：

1 创建表对象的一个实例。

2 创建字段对象的一个实例。

3 设定新字段的属性。

4 继续添加新的字段

5 重复第 3、4 步。

6 添加新的表。

代码如下：

```
//第一步创建表对象的一个实例
Table := CoTable.Create;
//指定表名
 Table.Name := ´Customers´;
// ... 设定表所属的 Catalog
 Table.ParentCatalog := Catalog;
//第二步创建字段对象的一个实例
Column := CoColumn.Create;
with Column do begin
  ParentCatalog := Catalog;
  //第三步设定新字段的属性
  Name := ´CustID´;
  Type_ := adInteger;
  Properties[´Autoincrement´].Value := True;
  Properties[´Description´].Value := ´Customer ID´;
end;
//第四步添加新的字段
Table.Columns.Append(Column, 0, 0);
Column := nil;
//第五步重复第三、四步
with Table.Columns do begin
  Append(´FirstName´, adVarWChar, 64);
  Append(´LastName´, adVarWChar, 64);
  Append(´Phone´, adVarWChar, 64);
  Append(´Notes´, adLongVarWChar, 128);
end;
//第六步添加新的表
Catalog.Tables.Append(Table);
Catalog := nil;
```

接下来我们就可以添加索引和关键字段了。下面代码演示了如何给 LastName 字段创建索引：

```
Index := CoIndex.Create;
with Index do begin
  Name := ´LastNameIndex´;
  IndexNulls := adIndexNullsDisallow;
  Columns.Append(´LastName´, adVarWChar, 64);
  Columns[´LastName´].SortOrder := adSortAscending;
end;
Table.Indexes.Append(Index, EmptyParam);
```

原理很简单，先创建一个 Index 对象，然后设定它的 Name 属性，指定 NULL 索引如何处理，同字段相关联，最后添加到表中，关键字段实现原理类似。注意 生成的 Access 数

据库是无法用 Access97 打开，因为只有 Access2000 才支持 Jet 4.0 生成的数据库，要想生成 Access97 的数据库，必须用 Jet 3.5 才行。

到目前为止，我们还没有提到如何使用用户和账号对象，这是由于我们当前的项目是基于 Access 数据库的，而这两个对象是 Access 不支持的。

使用 Jet 和 Replication 对象

另一个 ADO 的扩展是 Jet 和 Replication 对象 JRO。ADOX 能对不同的数据源操作，而 JRO 对象只对 Jet 数据库起作用。也就是只能操作 Access 数据库。

JRO 提供了一组对象，可以创建、修改和同步数据库复制。核心对象是 Replica 主要用来创建新的 Replica，获得 Replica 的属性，同步其它 Replica 的改变。

JRO 对象框架包括有 JetEngine 对象，JetEngine 提供了一些 Jet 引擎的特性，特别是能够压缩数据库，设定数据库密码，对数据库加密和从内存缓冲中刷新数据。对象模型如图 5 所示：

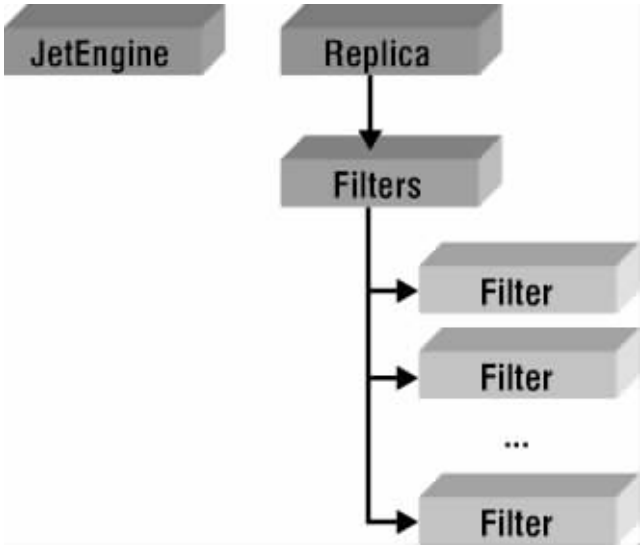


图 5 JRO 对象模型

这里我们只简单介绍一下 JRO 复制数据库。

第一步，我们要创建一个 design master（设计原版），表明数据库将用来作为一个复制的数据源，可以用来复制。我们首先需要调用 Replica 对象的 MakeReplicable 方法。然后可以用 GetObjectReplicability 和 SetObjectReplicability 方法来改变数据库的 replicability 状态。根据情况我们可以创建或者部分或者完全的设计原版的副本。

第二步，我们可以利用 Filter 对象来定义更新规则。最后，可以在两个副本之间保持同步，可以在 Internet 上进行直接或间接的同步。如果是间接同步，我们需要使用微软的 Office Developer 带的复制管理器。

要想在 Delphi 中使用 JRO 库，我们需要引入 Microsoft Jet and Replication Objects 2.1 Library 2.1 版本 的类型库。

使用 JetEngine 对象

下面代码显示了如何使用 JetEngine 对象来压缩 Northwind.mdb 数据库，并创建一个新的压缩后的 Northwind.mdb 数据库的副本 NewNorth.mdb。

```
const
  Provider = 'Provider=Microsoft.Jet.OLEDB.4.0;';
  //替换下面的路径为微软 Access 例子库的真实路径
  SrcMDB   = 'c:\data\northwind.mdb';
  DstMDB   = 'd:\data\newnorth.mdb';
procedure TForm1.Button1Click(Sender: TObject);
var
  JetEng: JetEngine;
  Src   : WideString;
  Dest  : WideString;
begin
  //创建 JetEngine 对象
  JetEng := CoJetEngine.Create;
  //设定数据源
  Src   := Provider + 'Data Source=' + SrcMDB;
  //设定目的源
  Dest  := Provider + 'Data Source=' + DstMDB;
  //如果目标数据库存在,就删除它
  if FileExists(DstMDB) then
    DeleteFile(DstMDB);
  //压缩数据库
  JetEng.CompactDatabase(Src, Dest);
  //释放 JetEngine 对象
  JetEng := nil;
end;
```

让我们稍微深入地理解一下压缩数据库的过程。

表的页面首先被重组。压缩后，原来碎片的数据页面被放在相邻的数据库页面中，这可以极大地改进数据库的性能。

通过删除原来只是被标记为删除的记录来回收未被使用的空间。

自增加字段被重置，以便下一个分配的值可以保持连续。

用来优化查询的表的统计信息被更新。因为统计信息被改变了，所有的查询都会被标记，下一次运行查询时，查询会被重新编译。

结论

本文介绍了两个 ADO 扩展：DDL 和 Security ADOX 以及 Jet 和 Replication 对象 JRO。我们简单研究了一下如何使用 ADOX 对象来获得数据源的元数据、如何创建新数据库和如何使用 JRO 来压缩 Jet 数据库，并简单介绍了数据库复制过程。

第二部分我们将要介绍 ADO Multi-Dimensional ADO MD，它可以用来操作多维数据存储。

本文源程序可以从 http://lgc.delphibbs.com 下载。

（收稿日期：2000 年 10 月 18 日）

网络数据传输方案与 VB 异步数据通信程序的实现

李 祥 陆 玲

摘 要 本文以海南人口 MIS 系统为例，先介绍了有关网络数据传输的技术实现方案与系统设计，再具体介绍了在 VB 中如何通过 Inet 控件实现异步数据通信及集成 Windows 系统拨号网络的 API 应用技术，供大家参考。

关键词 分布式数据处理，Inet 控件，FTP 协议，拨号网络，MIS 信息管理系统）

一、 绪论

随着计算机网络技术与数据库技术的发展，数据仓库与分布式数据处理是当前网络数据库系统设计中的热门话题，出现了很多应用方案与技术。对于每一个 MIS 系统来说，网络间的系统远程数据传输，是一个共同面临的问题。由于不同的 MIS 信息的软、硬件环境及系统需求的不同，对网络远程数据传输也就提出了不同的要求，因而如何处理分布式数据的通信仍然是一个让系统分析员与程序设计人员深感头痛的事情。在参与开发海南人口 MIS 系统的过程中，笔者在设计非实时异步数据通信模块时，也遇到了类似的问题。在探讨实际解决方案时，必须从海南人口 MIS 系统异步数据通信的背景要求谈起。

海南人口 MIS 系统是受海南公安厅委托进行开发的新型的人口信息管理系统。主要是要用来实现对全省人口信息的统一管理。系统采用后台数据库系统由省厅、市局、派出所三级构成。在省厅与各市局通过高速光纤连接，数据库之间进行数据实时通信，同步更新，而派出所到市局间的通信方式则是多元化的。由于各个派出所与市局之间距离及派出所的软硬件条件不同，一部分派出所可与市局数据库直接进行数据通信，一部分派出所与市局数据库间的连接是通过电话拨号方式上网的，甚至有一部分派出所暂时不能上网。因此，我们在进行派出所与市局通信系统的设计时，针对各派出所的特点，考虑了如下几种不同的网络数据通信方案。

二 、网络数据传输方案

方案一：实时数据通信方案

实时数据传输方案，这是一种直接操作服务器端数据库的一种方案。即派出所在进行人口数据管理工作 如录入、修改、更新、删除等操作）时，不是对本地数据库进行操作，而是直接存取市局服务器端的数据库中的记录。这样，可以实时保持派出所与市局数据的一致性，也减轻了数据维护及网络数据传输方面的设计难度。但这只是针对一部分通信条件比较好、离市局较近的一些派出所所采用的网络数据传输方案，时效性强。

方案二：异步数据通信方案

在这种方案中，派出所端存在一个本地 Access 数据库，派出所进行的人口信息的数据管理操作，如插入、修改、更新、删除等，是对本地的 Access 数据库中的记录进行操作，同时将变化的记录内容输出到一个更新数据库中，等到可以联网通信时，将更新数据库经过编码后，传输到市局数据库服务器，经确认后，再通过解码程序进行解码后，自动对市局数据库记录进行更新。市局端的下载数据，也通过相同的方法与派出所端进行通信。这样，就可以实现派出所与市局间数据的一致性及信息通信的需要了。这种方案，虽然时效性稍差，但可以满足大部分派出所 通过网卡或 Modem 可以上网，但不能进行实时通信）与市局间的通信需求，这也是我们设计的通信模块中的主体。

方案三：报盘维护方案

这种方案与方案二有相似之处，只不过它的更新数据库不通过网络传输，而是通过磁盘寄送的方式人工进行与市局数据库之间的数据更新与维护工作。时效性差，是为边远地区的派出所设计的一种迫不得已的工作模式。

接下来，我们将详细介绍 VB 异步通信程序的具体实现方法。

三、VB 异步通信程序的具体实现

由于派出所更新信息上报的时间与得到市局审批信息并下载的时间的不一致性，我们将异步通信模块的功能分为数据上传与数据下载两个程序分别来完成。程序完成的最后界面如下页图所示：

从图中可以看出，不论上传还是下载，其程序的功能都是由传输数据预处理 打包或解包）、电话拨号、数据传输控制 止传或下载）组成。下面，我们分别介绍三部分功能的 VB 程序实现方法。

1. 传输数据预处理的实现

传输数据的预处理，主要是完成对更新数据库的数据压

缩的打包操作及对下载审批信息的数据解压缩的解包操作，通过调用一个第三方压缩类控件（ZIP 压缩动态库文件）来实现。它由五个文件组成：CGUnzipFiles. cls CGZipFiles. cls CodeModule. bas zip32. dll 及 unzip32. dll 。

首先，将 zip32. dll 及 unzip32. dll 两个文件拷到 windows h system 中，并进行注册；其次，将上述两个类模块文件及一个标准模块文件添加到工程中；最后，在‘数据打包”按钮的 click 事件中加入下列代码：

```
Private Sub CMD_ZIP_Click()
. . . . . .
myfile = Dir $ (App. Path & ″\ data\″ & ZIPFILE & ″. MDB″)
. . . . . .
zip (myfile) ′完成数据打包工作
. . . . . .
End Sub
```

Zip myfile 为一个用户自定义函数，可写在程序文件代码后或新建一标准模块文件来存放，其代码如下：

```
′压缩操作子过程
Private Sub zip(filename As String)
    Dim oZip As CGZipFiles
On Error GoTo vbErrorHandler
    Set oZip = New CGZipFiles
    With oZip
ZipFileName = App. Path & ″\data\w″ & ZIPFILE & ″. zip″
′ ″\ZIPTEST. ZIP″
UpdatingZip = False ′ ensures a new zip is created
AddFile App. Path & ″\data\″ & filename ′App. Path & ″\ *. *″
If . MakeZipFile <> 0 Then
        MsgBox . GetLastMessage ′ any errors
    End If
End With
Set oZip = Nothing
    MsgBox ″\data\″ & ZIPFILE & ″. ZIP 文件创建成功!″
    Exit Sub
vbErrorHandler:
    MsgBox Err. Number & ″ ″ & ″Form1: : cmdZip_Click″ & ″ ″ & Err. Description
End Sub
```

同样，解码过程类似，只不过是换成 unzip filename 即可。其解码的子过程就不详述了。

2. 电话拨号的实现

本功能是实现是通过 Windows API 函数间接操纵 Windows 系统的拨号网络来完成拨号设置与拨号连接功能的。

首先，在标准模块文件中加入如下 API 函数声明：

```
Declare Function RasCreatePhoneBookEntryA Lib ″ Rasapi32. dll″ Alias ″RasCreatePhonebookEntryA″ (ByVal hwnd As Long, ByVal lpszPhonebook As String) As Long ′新建拨号连接
Declare Function RasEditPhoneBookEntryA Lib ″ Rasapi32. dll″ Alias ″RasEditPhonebookEntryA″ (ByVal hwnd As Long, ByVal lpszPhonebook As String, ByVal lpszEntryName As String) As Long ′编辑拨号连接
```

然后，再在‘新建连接”按钮的 Click 事件代码中加入下面语句：

```
Private Sub CMD_CREATE_Click()
Dim LDLD
LDLD = RasCreatePhoneBookEntryA(hwnd, ″″)
End Sub
```

在‘开始拨号”按钮的 Click 事件代码中加入下面语句：

```
Private Sub cmd_bh_Click()
. . . . . .
lbl_info. Caption = ″现在进行拨号连接″
strname = InputBox$(″请输入你要拨号的″ + Chr(13) + Chr(10) + ″连接名称, 再按确定. ″, ″输入框″, ″我的连接″, 2300, 2500)
LX = Shell( ″ rundll32. exe rnaui. dll, RnaDial ″ + Trim (strname), vbNormalFocus)
. . . . . .
End Sub
```

3. 数据传输控制（上传或下载）功能实现

在我们设计的系统中，是通过 Microsoft Internet Transfer 控件中 FTP 文件传输的方式来完成数据传输控制的。

首先，由于 Microsoft Internet Transfer 控件不是缺省控件，所以在使用前应先通过‘工程”菜单中的‘部件. . . . ”来把 Microsoft Internet Transfer Control6. 0 装入工程项目中，再从工具栏拖到工程窗口中。

其次，编写程序代码。对于数据下载，则在‘数据下载”按钮的 Click 事件中，填写如下代码：

```
Private Sub cmd_getdown_Click()
. . . . . .
′取得 FTP 服务器的地址
```

```
ip_address = Trim(txt_input1.Text)
Dim LX_STR1 As String
Dim lx_str4 As String
Inet1.RequestTimeout = 30
'判断客户端是否已经有了下载的审批包,如有,就提示先进行处理,再下载,如没有,就执行下载的程序
If Inet1.StillExecuting = False And Dir(App.Path & "\data\s" & ZIPFILE & ".zip") = "" Then
lbl_info.Caption = "现在开始检查服务器审批数据"
    If Dir(App.Path & "\DATA\s" & ZIPFILE & ".zip") = "" And Inet1.StillExecuting = False Then
       lx_str4 = "get s" & ZIPFILE & ".zip " & App.Path & "\DATA\s" & ZIPFILE & ".zip"
    With Inet1
       .Protocol = icFTP
       .URL = Trim(getdown_data.txt_input1.Text)
       .UserName = Trim(txt_input2.Text)
       .Password = Trim(txt_input3.Text)
       .RemoteHost = "FTP://" & Trim(txt_input1.Text)
    End With
     Inet1.Execute ip_address, lx_str4
   Do Until Inet1.StillExecuting = False
   lbl_info.Caption = "现在正在传输数据,请稍侯!!"
    DoEvents
    Loop
       '延时操作
     For i = 0 To 1000
     Next i
   cmd_ok1.Enabled = True
 End If
 End If
.....'传输正确性检测
End Sub
```

对于数据上传，则只需在 '数据上传 " 中输入下列代码：

```
Private Sub cmd_ok1_Click()
ip_address = Trim(txt_input1.Text) '取得服务器地址
With Inet1'控件属性设置
      .Protocol = icFTP
   .  .RemoteHost = "10.0.1.98"
      .URL = Trim(txt_input1.Text)
      .UserName = Trim(txt_input2.Text)
      .Password = Trim(txt_input3.Text)
'       .RemoteHost = "FTP://" & Trim(txt_input1.Text)
End With
Dim LX_STR1 As String
Dim lx_str2 As String
Inet1.Execute ip_address, "size W" & ZIPFILE & ".zip"
Do While Inet1.StillExecuting = True
   DoEvents
Loop
Dim getsize As String
getsize = CStr(Inet1.GetChunk(1024))
LX_STR1 = "send " & App.Path & "\DATA\W" & ZIPFILE & ".ZIP W" & ZIPFILE & ".zip"
Inet1.Execute ip_address, LX_STR1
    Do Until Inet1.StillExecuting = False
    lbl_info.Caption = "现在正在进行数据传输,请稍侯!!!"
    CMD_UP.Enabled = False
    cmd_ok1.Enabled = True
    DoEvents
    Loop
End Sub
```

四、结束语

对于没有实时要求或不能达到实时要求的数据传输应用场合，网络间数据的异步通信，的确是一种非常有效的方法。当然，在 VB 中，除了上述办法可以实现数据异步通信外，还有很多其它实现的方法，如用 E - MAIL，HTTP 或用 Winsock 控件都可以实现，只要灵活使用，总可以得到自己想要的结果。

所有源代码均在 Windows98 系统 VB6.0 中文企业版，NT 网络 装有 IIS4.0）下调试通过）

收稿日期：2000 年 10 月 11 日）

BMC 与 PricewaterhouseCoopers 和 Sun 联手推出服务级别管理（SLM）公用网站

该网站将提供有关 SLM 先进理念、研究、文章信息及自我评估工具

2000 年 11 月 6 日美国休斯顿和帕罗阿尔托讯——著名电子商务系统管理软件供应商 BMC 软件公司、全球最大专业服务机构 PricewaterhouseCoopers 和著名网络计算软硬件供应商 Sun Microsystems 今天联合宣布，它们将共同为市场建立一个服务级别管理（SLM）的公用学习网站，即：www.nextslm.org。该网站将着重介绍各种工具，帮助访问者掌握 SLM 知识；网站内容包括文章、研究活动、在线研讨会、相关的销售商、与其他网站的链接及自我评估工具信息。

BMC 软件公司行销部高级副总裁 Wayne Morris 认为尽管 SLM 在业界几乎算不上是新概念，但是实际运用起来毕竟没有像理解它那么容易。该网站为 IT 用户提供一个学习论坛，使用户不仅可以掌握 SLM 的理论，而且可以懂得如何将他们自己的做法与业界其他公司的先进做法进行对比。

服务级别管理的提出充分说明服务水平已成为顾客的首选配置项目，在许多方面甚至比特性和功能更重要。协助 IT 用户学会如何在创建内部机制的过程中解决复杂的业务问题。对于希望在今天快速发展的电子商务环境中赢得一席之地的企业来说，服务级别管理是关键。BMC、Sun 和 PricewaterhouseCoopers 充分意识到了 SLM 的重要性和市场对 SLM 的需求，决心积极协助客户掌握更多的 SLM 应用范例。

为了有效地帮助客户应用推荐的范例来衡量他们目前采用的 SLM 程序，该网站提供一套 SLM 评估工具 - 在线自我评估程序，可用来评估企业为关键业务 IT 企业和应用软件提供 SLM 的能力。

利用 VB6.0 建立基于 Interent 的多层应用模型

胡朝晖

摘 要 计算机体系结构已经从主机系统、文件服务器系统、客户服务器系统发展到基于 Web 的服务器浏览器体系结构。而我们的应用程序的开发也从简单的基于文件的处理系统发展到客户服务器两层结构，而随着应用系统业务逻辑的复杂化、多样化和易变化的趋势，建立一个三层甚至多层的应用系统也就变得十分重要，这里我们讨论了如何基于 VB 这个开发工具，建立一个基于 Internet 的多层的应用系统 并且通过一个详细的例子说明如何建立一个基于 Internet 的多层的应用系统，为开发人员把系统开发从面向 C/S 的两层结构转向基于 Internet 的多层结构提供了一定的帮助。

关键字 Visual Basic Internet NTier RDS ADO Web DBMS

一、简介

当前很多应用系统的开发都是基于 C/S 结构进行的，而采用这种模式的话，实际上，客户机不仅要进行表示层的处理，而且还要进行逻辑层的处理，这样当应用的逻辑需要变化的时候，客户端程序必须要重新修改、编译和安装，对于业务流程比较复杂而且多变的企业来说，系统开发和维护人员的工作就会比较繁杂，而对于基于 Internet 的多层应用来说，我们可以单独剥离出业务逻辑层，把它和 Web Server 放在一起，这样当业务逻辑改变的时候，只需要改变逻辑层的对应的模块，而客户端就不需要做任何的改动，方便的应用的升级和维护。

建立三层的数据库应用，可以使用基于 Internet 的远程数据服务 (RDS) 通过 HTTP 协议进行数据的传输。一般来说三层结构分别包含的内容如下：

1. 客户层：

● 通过 4GL 开发工具建立前端的表示层，具体可以用 VB，DELPHI 等工具建立。在我们的例子中采用的是 VB。

● RDS DataSpace 对象：该对象使数据能够通过 HTTP 协议进行传输。

2. Web 服务器/中间层

● ActiveX DLL - 主要用来存放业务规则，三层体系结果使它很容易独立的更新你的业务逻辑，而不需要重新发布你整个应用程序。

● Microsoft IIS - 当使用 RDS 的话，你可以通过 HTTP 协议在 Internet 网上进行数据的传输。

● ADO - 微软最新的数据连接技术，几乎可以和任何类型的数据源进行连接，并不仅仅局限于关系数据库。

3. 数据库

● 任何 OLEDB 支持的数据库都可以。比较常见的有 ORACLE，SYBASE，SQL SERVER 等等，当然也可以应用到象 ACCESS 这样的桌面数据库上。

一般来说，建立一个三层结构的应用在逻辑上至少需要两台机器

● 和 Internet 相连的存放业务对象的 Web 服务器。服务器必须在 Windows NT 下运行，并且必须安装 Pack 4 以上的补丁。因为安装了 Pack 4 以后，RDS 缺省的也被装到了机器上。而数据库服务器可以和 Web 服务器在一台机器上也可以在不同的机器上。

● 安装客户端的机器，除非你是在 Intranet 范围内创建应用系统，不然的话，计算机必须连接到 Internet 为了安装客户端应用，要使用 VB Package 和 Deployment wizard 确保 ADO 和 RDS 被安装到客户机器上。

注意，在实际进行程序调试的时候，我们也可以在一台安装有 Windows 98 的机器上进行调试，但是必须安装 Personal Web Server.

二、建立基于 Internet 应用所需要注意的问题

通过 HTTP，你实际上是创建了一个无状态的应用，也就是说，当客户端每次发送一个请求到服务器的时候，就被认为是一个新的事务。一个 HTML 页面也是无状态的，使用浏览器，你点击某一个链接，服务器就返回该链接相关的页面。当数据传递结束的时候，服务器和浏览器都在本质上忽略了对方的存在。

对每一个事务来说，用户对商业对象 (business object) 和数据库来说，就是一个“陌生人”，比如，当需要一个事务的时候，用户必须登入到 DBMS 中，对用户而言，多次登入是另人厌烦的。当然实际上，你可以让用户登入一次，然后把

用户的名字和密码缓存在客户端，然后在每一个接下来的事务中，自动传递这些信息。

三、一个简单的客户/业务对象的建立和实现

一个简单的客户/服务器应用的例子，包括一个客户端的表示层，一个商业对象和一个数据库系统。商业对象将处理来自客户端 Form 的所有的请求，并且以 ADO 记录集的形式返回数据或者更新数据库。

为了说明如何建立一个三层的结构，我们要建立一个数据库，一般说来对于大中型应用程序来说，都需要采用 ORACLE，SQL SERVER 等功能完备的数据库，这里为了说明问题的方便并且为了方便用户进行实际的测试，我们建立了一个基于 ACCESS 的数据库，但是其基本的原理也可以应用到其他数据库上。这里我们建立一个名称为 PersonalInfo 的数据库，为了说明问题的方便，我们建立一个表，该表名称为 Person。表的结构如下：

字段名称	字段类型	字段长度	备注
PersonID	长整型		主键
Name	字符串	10	必填字段
Email	字符串	30	
MobilePhone	字符串	15	
BP	字符串	15	
HomePhone	字符串	15	
OfficePhone	字符串	15	
OfficeAddress	字符串	50	
OfficeZip	字符串	6	
HomeAddress	字符串	50	
HomeZip	字符串	6	

在建立了数据库表以后，我们就可以建立商业对象，建立商业对象的过程如下：

1. 在服务器上，至少需要安装 NT Option Pack 4 以上。

2. 创建一个 Active DLL 或者是 Active EXE 项目。

3. 命名项目和类的名称，项目和类的名称将成为该应用的 ID 标志，而且这个名字会被用来作为得到商业对象的一个引用。这里的例子类被命名为 hzhProject，项目被命名为 hzhClass。

注意：为了初始化商业对象，客户端需要使用 CreateObject 方法。可以采用下面的语句：

```
Set objProxy = dsDataSpace.CreateObject _
    ("hzhProject.hzhClass", "HTTP://WebServerName")
```

4. 在项目菜单上，点击引用子菜单，然后建立对 Microsoft ActiveX Data Objects 2.0 Library 的引用。

5. 创建一个公共的过程，通过给定的连接字符串来初始化 ADO Connection 对象，因为应用是无状态的，连接必须在商业对象初始化的时候被创建。

我们的例子建立一个称为 MakeConnection 的过程：

```
Option Explicit
Public cn As ADODB.Connection ' 公共的连接对象.
Public Function MakeConnection(Optional UserID As String,
Optional Password As String) As String
  ' 创建这个连接对象.
  Set cn = New ADODB.Connection
  On Error GoTo ConnectErr
  With cn
    .CursorLocation = adUseClient
  .ConnectionString = "Driver={Microsoft Access Driver "
& "(*.mdb)}; Dbq=" & app.path & "\db\PersonInfo.mdb;" & "Uid=" & UserID & "; Pwd=" & Password
    .Open
  End With
  ' 如果没有错误出现的话
  MakeConnection = True
  Exit Function
  ' 如果发生错误
ConnectErr:
  Dim i As Integer
  Dim sErrors As String
  If cn.Errors.Count > 0 Then
    For i = 0 To cn.Errors.Count - 1
      ' 处理错误信息.
    sErrors = sErrors & cn.Errors(i).Number & ": " &
cn.Errors(i).Description & vbCrLf
    Next i
  End If
  MakeConnection = sErrors
  Exit Function
End Function
```

当一个记录集必须从 DBMS 中取出的时候，MakeConnection 函数被激活来创建一个全局的可得到的连接对象，假设你只使用一个数据库的话，当设定一个 recordset 的 ActiveConnection 的时候，该全局对象 cn 被使用。所以该全局对象只要被创建一次就可以了。

6. 创建函数返回一个记录集到客户处，返回一个记录集的代码返回数据库 PersonalInfo 的表 Person 的限定的客户端所需要的记录集合。该函数如下：

```
Public Function GetLimitedPersonInfo(UserID As String,
Password As String, sCondition As String) As Recordset
  ' 连接对象.
  MakeConnection UserID, Password
  Set rsPersons = New ADODB.Recordset
  With rsPersons
    .CursorLocation = adUseClient
    If Trim(sCondition) = "" Then
      .Source = "SELECT * FROM Person"
    Else ' 表示选择某个范围的记录集
      .Source = "SELECT * FROM Person where Name
like '" & sCondition & "%'"
    End If
    Set .ActiveConnection = cn
    .CursorType = adOpenStatic
    .LockType = adLockOptimistic
    .Open
```

```
    End With
    Set rsPersons. ActiveConnection = Nothing
    Set GetLimitedPersonInfo = rsPersons
End Function
```

为了保证能正常工作，需要设置 Recordset 对象的几个属性，CursorLocation 需要被设定为客户端游标 adUseClient 因为客户端游标引擎使记录集能够以交叉进程的形式返回，如果选择 adUseServer 的话，你必须确信数据提供者支持这些属性。

7. 创建一个函数用来更新数据库服务器上的记录。该函数是为了对数据库中的某条记录进行更新。

```
Public Function UpdateTblRS(UserID As String, Password As String, tblName As String, KeyFldValue As String, UdfldCol( ) As String, UdfldTypeCol( ) As String, UdValueCol( ) As String) As String
    Dim sUpdateStr As String
    Dim sTempStr As String
    Dim nAffectedRow As Long
    Dim nCount1 As Integer
    Dim i As Integer
    On Error GoTo Err1:
    MakeConnection UserID, Password
    nCount1 = UBound(UdfldCol) – 1
      sUpdateStr = "update " & tblName & " set "
      For i = 0 To nCount1
        If UdfldTypeCol(i) = 1 Then
 sTempStr = UdfldCol(i) & "='" & UdValueCol(i) & "',"
        Else
    sTempStr = UdfldCol(i) & "=" & UdValueCol(i) & ","
        End If
        sUpdateStr = sUpdateStr & sTempStr
      Next
      sUpdateStr = Left(sUpdateStr, Len(sUpdateStr) – 1)
       sUpdateStr = sUpdateStr & " where personID =" & KeyFldValue
      cn. BeginTrans
        cn. Execute sUpdateStr, nAffectedRow
      cn. CommitTrans
      If nAffectedRow > 0 Then
        UpdateTblRS = "True"
        Exit Function
      Else
      UpdateTblRS = "数据库中不存在该记录,不能更新记录"
        Exit Function
      End If
Err1:
    cn. RollbackTrans
    UpdateTblRS = Err. Description
End Function
```

注意 也可以使用在是数据库上使用存储过程来更新数据库，把数据传递给存储过程，让它来更新数据库，这会使性能有所提高。

8. 删除数据库中的某条记录 以下是相关的代码。

```
Public Function DeleteRS(UserID As String, Password As String, tblName As String, KeyFldValue As String) As String
  Dim sDeleteSQL As String
  Dim nAffectedRow As Long
  On Error GoTo Err1:
  MakeConnection UserID, Password
  sDeleteSQL = "delete from " & tblName & " where personID =" & KeyFldValue
  cn. BeginTrans
    cn. Execute sDeleteSQL, nAffectedRow
  cn. CommitTrans
  If nAffectedRow > 0 Then
     DeleteRS = "True"
     Exit Function
  Else
     DeleteRS = "数据库中不存在该记录,不能删除该记录"
     Exit Function
  End If
Err1:
    cn. RollbackTrans
    DeleteRS = Err. Description
End Function
```

9. 增加一条记录到数据库中 以下是相关的代码 . 注意该函数实际是通过 SQL 语句实现记录的插入的。

```
Public Function InsertPersonRecord(UserID As String, Password As String, tblName As String, fldCol( ) As String, fldTypeCol( ) As String, valueCol( ) As String) As String
    Dim nCount1 As Integer
    Dim i As Integer
    Dim sCombinSQL As String
    'Create connection object.
     On Error GoTo Err1:
     MakeConnection UserID, Password
     nCount1 = UBound(fldCol) – 1
        sCombinSQL = " Insert Into " & tblName & " ("
        For i = 0 To nCount1
          sCombinSQL = sCombinSQL & fldCol(i) & ","
       Next
sCombinSQL = Left(sCombinSQL, Len(sCombinSQL) – 1)
        sCombinSQL = sCombinSQL & ") values ("
        For i = 0 To nCount1
          If fldTypeCol(i) = 1 Then '1 表示字段为字符串类型
sCombinSQL = sCombinSQL & "'" & valueCol(i) & "',"
        ElseIf fldTypeCol(i) = 2 Then '2 表示字段为数值类型
         sCombinSQL = sCombinSQL & valueCol(i) & ","
          End If
        Next
      cn. BeginTrans
sCombinSQL = Left(sCombinSQL, Len(sCombinSQL) – 1)
        sCombinSQL = sCombinSQL & ")"
        cn. Execute sCombinSQL
      cn. CommitTrans
      InsertPersonRecord = "True"
      Exit Function
Err1:
    cn. RollbackTrans
    InsertPersonRecord = Err. Description
End Function
```

10. 编译这个项目。在编译商业对象的时候，在工程属性对话框中，选择部件标签，然后对远程服务器文件栏打上勾，这样在商业对象生成 .dll 文件的同时，也会生成一个 .tlb 文件，然后可以在客户端引用这个 .tlb 文件。当然客户端也可以不引用该 .tlb 文件，也就是所说的采用动态绑定的方法来创建对象，因为商业对象实际是存放在 WEB 服务器中的。

11. 注册业务对象

1 在任务条的开始菜单中 点击运行，在运行文本框中键入 RegEdit 并点击 OK.

使用 Registry Editor 并找到下面的键：

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\ADCLaunch

2 增加一个新的键并且用业务对象的 ID 来描述它，ID 的形式为 ProjectName.ClassName。比如，对我们的这个简单的示例程序而言，它应该是 hzhProject.hzhClass. 注意建立的新的键应该是 "主键" 或者也可以称为 "项"，相当于一个子目录这样的概念 而不是文件的概念。

在创建了业务对象以后，我们需要创建客户端。实际上，一旦业务对象被创建以后，你可以在客户的 Form 中调用它，客户端能够：

- 通过 RDS DataSpace 对象得到对商业对象的一个引用。
- 激活商业对象的函数、方法和属性。

创建一个客户端的 Form

1. 建立一个标准的 EXE 项目

2. 在项目中增加对 Microsoft ActiveX Data Objects 2.0 Library 和 Microsoft Remote Data Services 2.0 Library 的引用

3. 在 Form 上放置控件，并增加相应的代码。

4. 初始化 DataSpace 对象，使用 DataSpace 对象的 CreateObject 方法来创建一个业务对象的 Proxy 具体代码如下：

```
Option Explicit
Private dsDataSpace As RDS.DataSpace
Public objProxy  '用 DataSpace 对象创建的对象
Public Sub GetProxy()
    Set dsDataSpace = New RDS.DataSpace
    '设置一个合理的时间
    dsDataSpace.InternetTimeout = 30000
    Set objProxy = dsDataSpace.CreateObject ("hzhProject.hzhClass", "HTTP://WebServerName")
    '注意对一个基于 Intranet 的应用来说，可以采用下面的调用形式:
    ' Set objProxy = dsDataSpace.CreateObject _
    ("prjServer.clsServer", "\\MyComputerName")
End Sub
```

同时需要注意的是：注意 CreateObject 方法需要两个参数：业务对象的 ID 和服务器的地址。

5. 使用一个登入的 Form 激活 MakeConnection 业务对象的函数 从 Form 的 txtUserName 和 txtPassword 文本框中得到用户名和密码。

```
Private Sub cmdOK_Click()
  Dim strConnectResult As String
  strConnectResult = frmPersons.objProxy.MakeConnection
(txtUserName.Text, txtPassword.Text)
  If strConnectResult = "True" Then
      frmPersons.strUserID = txtUserName.Text
      frmPersons.strPassword = txtPassword.Text
      LoginSucceeded = True
      Me.Hide
  Else '如果因为一些原因没有成功的话
      '显示失败的原因.
      MsgBox strConnectResult
      Unload Me
      Exit Sub
  End If
End Sub
```

如果 MakeConnection 函数返回 True 的话，两个文本框中的值被保存在模块级的变量中，被称为 strUserID 和 strPassword. 因为在业务对象中每一个事务都是无状态的，所以对所有的调用重新发送这些值是必要的，客户端的 Form 只需要在最开始的时候，输入用户名和密码，然后在整个 session 过程中，它们的值都会被缓存起来。

6. 使用新的 proxy 对象来激活业务对象的方法，可以增加，删除，修改和返回记录。这里举一个返回记录集合的例子。下面的例子表示通过传递被缓存起来的用户名和密码来调用业务对象的方法 GetLimitedPersonInfo。

```
Set rsPersons = objProxy.GetLimitedPersonInfo(strUserID, strPassword, sCondition)
```

商业对象 objeProxy 从数据库中提取表 Person 的记录集并把它返回给客户端 Form。

当你运行应用程序的时候，Form 被装载，然后建立一个连接，确定需要返回的数据集合的范围，然后提取数据。

四、开发多层结构需注意的事项

在具体开发基于 Internet 的多层结构的时候，为了加快开发的进度，我们一开始可以把客户端和商业对象放在一个 Project 中，即把商业对象作为相应的类 (class)，然后当所有的调试都正确的时候，可以把相关的类剥离出来，作为一个单独的 Active DLL 在安装了 IIS Web Server 的机器上进行注册。注意到该商业对象要进行更新的时候，必须先关闭相应的进程或者是重启装有 Web Server 的机器。同时需要注意的是当客户端和商业对象进行交互的时候，并不是所有的数据结构都可以进行传递，比如当前在 VB 6.0 中还不支持把 COLLECTION 对象作为参数在表示层和商业对象之间进行传递。

五、小结

本文详细的描述了基于 Internet 的多层开发的方法，并且利用 VB 这个开发工具，描述了一个详细的实现过程。对于基于 Internet 的多层开发提供了一个很好的说明和借鉴。

收稿日期：2000 年 10 月 16 日）

通过代理服务器访问网页

侯永森 董 超

在当前 IT 业如火如荼飞速发展的今天，VisualC + + 网络编程也越来越受到青睐。对于通过 HTTP、FTP 协议访问指定网站的指定路径内容的方法也受到重视。但一些文献大都是对直接访问网页给出例子并作出解释，很少提及通过代理服务器访问 INTERNET 的方法。本文作者根据实际情况利用 HTTP 协议实现了该方法，提供给广大编程爱好者，以供参考。本程序在 WIN98 操作环境中，在 VC + + 6.0 中调试通过。

一、WinInet 类

微软 MFC 提供了 WinInet 类来实现对网页的访问。通过 CInternetSession，CHttpConnection 和 CHttpFile 类来具体实现。CInternetSession 类主要进行初始化一个 Internet 会话；CHttpConnection 则打开一个与服务器的连接；从 CInternetFile 类派生的 CHttpFile 主要进行文件的读取。

二、直接读取网页的一般过程

当不涉及代理服务器时（即计算机直接连接到 Internet 上），访问网页的过程如下：

1. 创建一个对话：

首先，为了访问网页我们必须创建一个 Internet 对话，最简单的办法是构建一个新的 CInternetSession 对象。以后我们利用它来开始一个对话。CInternetSession（CInternetSession 指出操作环境、访问的类型是否通过代理服务器、代理服务器的地址及访问方式。

2. 存取文件：

一旦创建了一个 Internet 对话，我们即可以直接通过 CInternetSession：：OpenURL（）来读取文件。不需要建立一个连接。这种情况最为简单也很实用。

三、通过代理服务器访问网页的过程

当必须通过代理服务器连接到 Internet 上时，情况就复杂了许多，我们必须显式建立连接，并通过代理服务器的身份验证。这种情况下的网页访问过程如下：

1. 创建一个 Internet 对话：

如同直接访问情况一样，这是必须要做的第一步。

2. 建立 HTTP 连接：

MFC 提供 CInternetSession 类用来与 HTTP 服务器实现连接。通过 GetHttpConnection 来完成。它指出要连接的服务器和端口，以及用户和密码。

3. 创建一个新的 CHttpFile：

通过 CHttpConnection：：OpenRequest，我们可以从该连接上创建一个 CHttpFile 对象。该函数指出请求所要完成的动作、路径以及连接的方式。

4. 发送请求：

通过 CHttpFile：：SendRequest 将请求发送到服务器，一旦请求被成功发送，可以通过 ChttpFile：：Read 来读取从服务器返回的信息。

5. 进行代理服务器验证：

只有通过代理服务器验证，我们才能访问指定网页。和 IE 类似，当我们利用 IE 上网（往：特指通过代理服务器的情况），很快，就有一个对话框弹出来，要求输入用户和密码。只有用户和密码通过验证，我们才有权访问网页。

我们通过检查 SendReques 的返回信息，当其值是 407 时，即代理服务器要求身份验证。通过 SetOption 设置用户和密码，然后再一次调用 SendReques 发送请求。当用户和密码通过验证，我们即可访问网页。

6. 读取网页：

利用 ChttpFile：：Read 读取指定路径的内容。

四、程序源代码

本程序通过代理服务器验证，读取 http：//www.sohu.com 网站的源 HTML 文件。主要代码在 void CMyWininetView OnButtonGo 中。源码如下：

```
//1. 建立一个 Internet 会话. 其中, 参数 "192. 168. 0. 1 "是代理服务器地址
  CInternetSession * pSession = new CInternetSession(
"Raw HTML Reader", 0, INTERNET_OPEN_TYPE_PROXY,
"192. 168. 0. 1", 0, INTERNET_FLAG_DONT_CACHE);
  if (pSession = = NULL)
  {
    : : MessageBeep(MB_ICONEXCLAMATION);
    AfxMessageBox("Internet session initialization failed!",
      MB_OK | MB_ICONEXCLAMATION);
    return;
  }
 //2. 建立和服务器的连接.
    CHttpConnection * pConnect = pSession - >GetHttp-
Connection(m_strServer);
      //m_strServer 指所要访问的网页, 在这里是
WWW. SOHU. COM
  if (pConnect = = NULL)
  {
    : : MessageBeep(MB_ICONEXCLAMATION);
    AfxMessageBox("Internet connection failed!", MB_OK |
MB_ICONEXCLAMATION);
    // Close session
```

（下转第 92 页）

Windows 下不规则图形的动画技术

徐 磊

虽然 Windows 的 GDI 只支持静态图形，不具备通常的动画支持，但我们还是可以利用 Windows 功能强大的 API 位块传输函数 Bitblt 来实现图形的动画，遗憾的是 BitBlt 函数只能操作矩形图形区域，而实际工作中，更实用和更吸引人的往往是不规则图形的动画技术。但简单地使用 BitBlt 函数，由于参与动画的矩形区域会破坏背景，因而使得动画效果看起来索然无味，鉴于此，笔者通过摸索，利用 BitBlt 函数和其丰富的 ROP 不是绘图方式的 ROP2 码 运算方式，实现了不规则图形动画，现介绍如下。

细心的读者一定注意到，作为一个矩形的图标，当你在桌面拖动它时，它变成了一幅黑白图象，并且看起来并不总是矩形的 有兴趣的读者不妨拖动一下书写器图标试一试 ，据 Windows 文档，图标实际上是由两种格式的位图构成的，一种为 “与掩码”，一种黑白位图象，另一种为 “或掩码”，一种彩色位图象，为了画一个图标，Windows 总是先用 “与掩码”与背景进行与运算，由于单色位图象只有 0、1 RGB 0 0 0 和 RGB 1 1 1 两种情况，所以，只有黑色像素可以影响背景，然后 Windows 再把 ‘或掩码”以或的方式画到同样的位置，得到平常我们所看到的正常图标图案。当拖动图标时，Windows 仅仅是不断地在背景上绘制 ‘与掩码”图象，从而可以看到一个不规则的图形在桌面上移动。

根据以上思路，为实现不规则图象的动画，首先准备两种格式的位图，一种为单色不规则位图，如图 1 所示，另一种为彩色不规则图形，如图 2 所示。图 1 的图象除了要动画的部分涂成黑色外，其余部分涂成白色，图 2 的图象除要动画的部分保留彩色外，其余部分涂成黑色。动画时，首先将

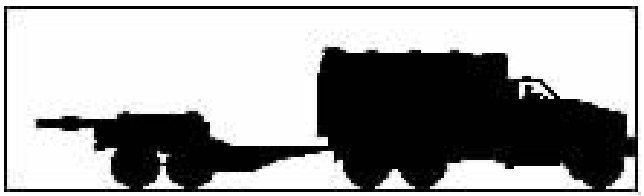


图 1 单色位图 不规则图象的剪影

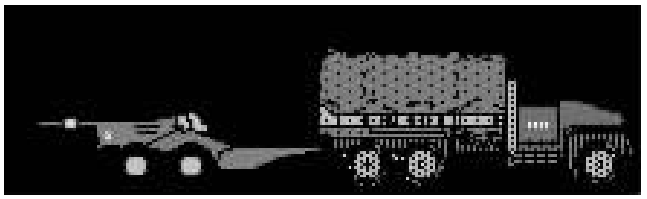


图 2 彩色位图 其余部分用刷子涂黑

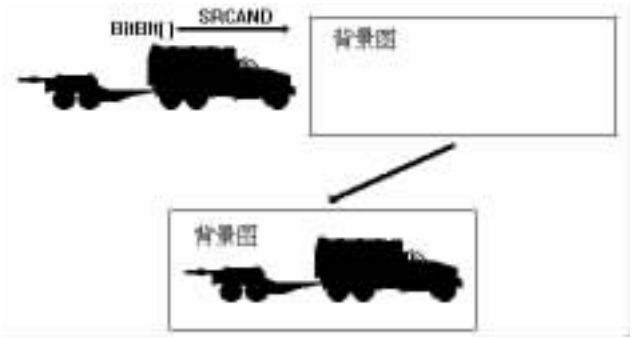


图 3 将单色图用 BitBlt 函数 SRCAND 方式传输到背景图上

图 4 将彩色图用 BitBlt 函数 SRCPAINT 方式传输到背景图上

图 1 所示的单色位图用 BitBlt 传输到背景上，ROP 码置为 SRCAND 与）方式，这样只有黑色像素可以传输到背景上，产生图 3 所示的效果。然后将图 2 的彩色位图再用 BitBlt 传输到图 3 上，ROP 码置为 SRCPAINT 或）方式，得到图 4 所示的最终效果，即不规则的彩色图形显示在背景上，而背景完好无损。下面是笔者采用这种方法用 Borland C + + 4. 0 编制的一个演示程序清单。程序演示了一个不规则的图形从窗口的左侧移动到右侧的过程，程序在 WM_SIZE 消息响应函数中拉伸背景图，使得背景图总是刚好填充整个客户区，在定时器 WM_TIMER 消息中作动画处理。

```
#include <owl\owlpch.h>
#include <owl\applicat.h>
#include <owl\framewin.h>
#include <owl\gdiobjec.h>
#include <owl\dc.h>
#include "animate.rh"
class TAnimateWindow : public TFrameWindow {
  public:
   TAnimateWindow(TWindow * parent, const char * title);
    ~TAnimateWindow()  {
```

```
        delete      BackGrd;   //删除位图对象
        delete      MonoCar;
        delete      Car;
        delete      BackGrdBak;
        delete      CelBak;
        KillTimer(1);          //删除定时器
                                }
    protected:
      TPoint        point;
      TBitmap *     BackGrd;          //背景位图
      TBitmap *     MonoCar;          //单色动画位图
      TBitmap *     Car;              //彩色动画位图
      TBitmap *     BackGrdBak;       //用于备份背景位图
      TBitmap *     CelBak;
      int           x;
      void          SetupWindow();
      void          Paint(TDC& dc, BOOL erase, TRect& rect);
      void          EvSize(UINT sizeType, TSize& size);
      void          EvTimer(UINT);
DECLARE_RESPONSE_TABLE(TAnimateWindow);
};
DEFINE_RESPONSE_TABLE1(TAnimateWindow, TFrameWindow)
  EV_WM_SIZE,
  EV_WM_TIMER,
END_RESPONSE_TABLE;
TAnimateWindow::TAnimateWindow(TWindow * parent,
const char * title):
TFrameWindow(parent, title){
  BackGrd=new TBitmap(*GetModule(), BACKGRD);
//声明背景位图对象
   MonoCar=new TBitmap(*GetModule(), MONOCAR);
//声明单色位图对象
  Car=new TBitmap(*GetModule(),CAR);  //声明彩色位
图对象
  x=-215;          //从客户区的-215处开始
}
void
TAnimateWindow::SetupWindow(){
  TFrameWindow::SetupWindow();
  TClientDC dc(*this);
  BackGrdBak=new TBitmap(dc, 1000,1000); // 声明一个
1000*1000面积的位图区域用于备份背景
  CelBak=new TBitmap(dc, 220, 65);          //动画作图区
  SetTimer(1, 100);                         //开定时器
}
void
TAnimateWindow::EvSize(UINT, TSize& size){
  TClientDC dc(*this);
  TMemoryDC memDCBak(dc);
  TMemoryDC memDC(dc);
  point.x=size.cx;                          //保存客户区尺寸
  point.y=size.cy;
   memDCBak.SelectObject(*BackGrdBak);//将1000*
1000的位图选进内存
  memDC.SelectObject(*BackGrd); //将背景位图选入另
一内存
  memDCBak.StretchBlt(0, 0, point.x, point.y, memDC, 0, 0,
BackGrd->Width(), BackGrd->Height(), SRCCOPY);
//将背景位图拷贝到备份位图上并按客户区的大小进行拉伸
        Invalidate();       //产生一条WM_PAINT消息
}
void
TAnimateWindow::Paint(TDC& dc, BOOL, TRect&){
  TMemoryDC memDC(dc);
  memDC.SelectObject(*BackGrdBak);    //将备份过的位
图选入内存
  dc.BitBlt(0, 0, point.x, point.y, memDC, 0, 0, SRCCOPY);
                                      //将背景图拷贝到客户区
}
void
TAnimateWindow::EvTimer(UINT)
{
  TClientDC dc(*this);
  TMemoryDC memDC(dc);
  memDC.SelectObject(*BackGrdBak);  //得到备份位图
  TMemoryDC MemCelBak(dc);
  MemCelBak.SelectObject(*CelBak);  //将动画作图区选
进内存
  MemCelBak.BitBlt(0, 0, CelBak->Width(),
     CelBak->Height(), memDC,  //将背景的一部分拷贝到
     x, (80.0/100)*point.y,    //内存动画作图区
     SRCCOPY);
  memDC.SelectObject(*MonoCar);     //将单色动画位图
选进内存
  MemCelBak.BitBlt(
   0, 0, CelBak->Width(),  //将单色位图与背景进行
  CelBak->Height(), memDC, //SRCAND(与)运算后拷贝到
  0, 0, SRCAND);          //动画作图区
  memDC.SelectObject(*Car); //将彩色动画位图选入内存
  MemCelBak.BitBlt(
   0, 0, CelBak->Width(),  //将彩色位图与背景进行
   CelBak->Height(), memDC, //SRCPAINT(或)运算后
   0, 0, SRCPAINT);        //拷贝到动画作图区
dc.BitBlt(x, (80.0/100)*point.y, CelBak->Width(), Cel-
Bak->Height(), MemCelBak, 0, 0,SRCCOPY); // 将动画作
图区拷回客户区
  if(x>point.x) x=-215;
  else x++;
}
class TAnimateApp : public TApplication {
  public:
   TAnimateApp() : TApplication()      {}
   void InitMainWindow()    {
     SetMainWindow(new TAnimateWindow(0, "Windows下
不规则图形的动画技术"));
   }
};
int OwlMain(int /*argc*/, char* /*argv*/ [])
{
  return TAnimateApp().Run();
}
//end of file
```

收稿日期 2000年11月7日

在 VC6.0 中定制 ObjectARX2000 开发环境

刘良华 袁英战

摘 要 本文结合 ObjectARX 开发特点，介绍了对 Visual C++6.0 环境设置进行定制的方法和技巧，以提高 AutoCAD2000 二次开发的效率。

关键字 二次开发，定制，ObjectARX

ObjectARX 是 Autodesk 公司针对 AutoCAD（3.0 或以上版本）平台上的二次开发而推出的一个开发软件包，它支持面向对象编程（Object-Oriented Programming OOP），同时也向下兼容 ADS C。ObjectARX 的早期版本称为 ARX（AutoCAD Runtime eXtension），意为 AutoCAD 运行库扩展。目前，针对 AutoCAD2000 已发展为 ObjectARX2000。由于其本身的优点和特点，采用 ObjectARX 接口进行 AutoCAD 二次开发越来越普遍。ObjectARX2000 的编译环境采用 Visual C++6.0，使用 Windows 98/NT 操作系统。

Visual C++6.0 功能强大，为大多数一般应用开发提供了缺省的环境设置，但对于 ObjectARX 开发而言，该缺省环境并没有太多的便利，往往需要（或必须）对 Visual C++6.0 环境定制，以使得二次开发快捷便利。例如，在编程过程中开发者往往希望及时获得 ObjectARX 的联机帮助（如源代码中函数的原型说明等），特别是 F1 形式的联机帮助，即首先选定一个关键字，然后按 F1 键，以获得相关的帮助主题；又例如，开发者希望 ObjectARX 特有的函数名、常量等关键字高亮度地显示在源窗口中（如同C/C++ 语言的 int、float 等关键字），以方便源代码的编辑。

诸如上述问题的解决，必须对 Visual C++6.0 的环境设置进行修改。本文将介绍定制开发环境的方法和技巧，以使采用 ObjectARX 进行 AutoCAD2000 二次开发来得更方便、更快捷。

一、设置资源引用路径

在进行 ObjectARX 开发时，显然需要使用 ObjectARX 开发工具包的资源，并且在每一个工程项目都要使用到，因此不妨将这些资源的引用路径包括在 Visual C++6.0 环境配置中。为此，单击 Tools|Options 菜单命令，选择 Directories 属性页。

首先，设置头文件引用路径。选择 Show directories for 下拉列表框的 Includes files 项，设置头文件的引用路径；然后，在 Directories 列表框的空白项双击，输入 ObjectARX 头文件的路径名，如 D:\ObjectARX 2000\inc（如图 1 所示）。

其次，设置库文件引用路径。同理，选择 Library files 项，添加 ObjectARX 库文件的路径名，如 D:\ObjectARX 2000\lib（如图 2 所示）。

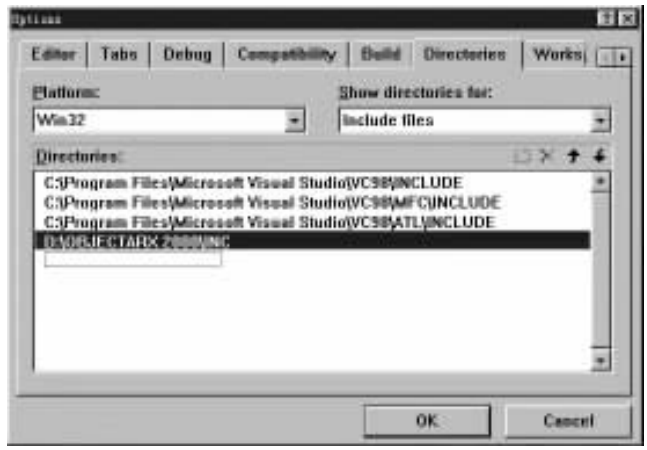


图 1 添加 ObjectARX 头文件引用路径

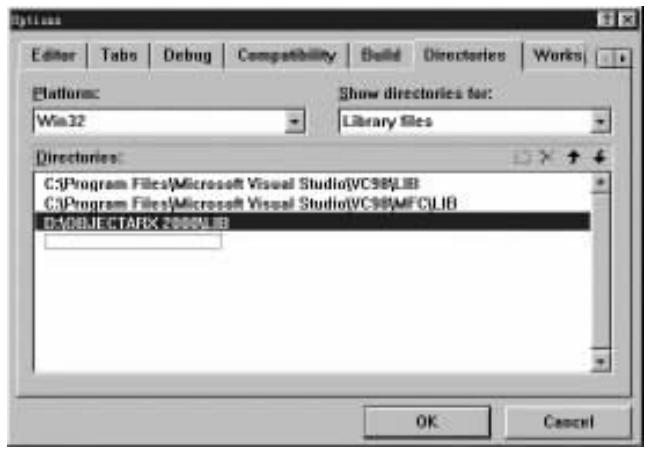


图 2 添加库文件引用路径

二、使用扩展联机帮助

Visual C++6.0 提供了一个菜单命令即 Help Use Extension Help，它可以将定制的联机帮助作为扩展帮助集成到 Visual C++6.0 系统中。如果不核选该菜单项，那么打开的将是 Visual C++6.0 缺省的联机帮助，否则打开扩展帮助系统。

按照下面的步骤，可以将 ObjectARX2000 的联机帮助定义成 Visual C++6.0 的扩展帮助。这样，在 Visual C++6.0 开发环境中就可以很方便地使用 ObjectARX 帮助了。

1. 单击“开始|运行”菜单命令，运行注册表编辑程序 Regedit. exe，它是 Windows9. x 操作系统自带的工具软件，存放在系统安装目录下。如图 3 所示，单击“确定”按钮。

图 3　运行注册表编辑程序

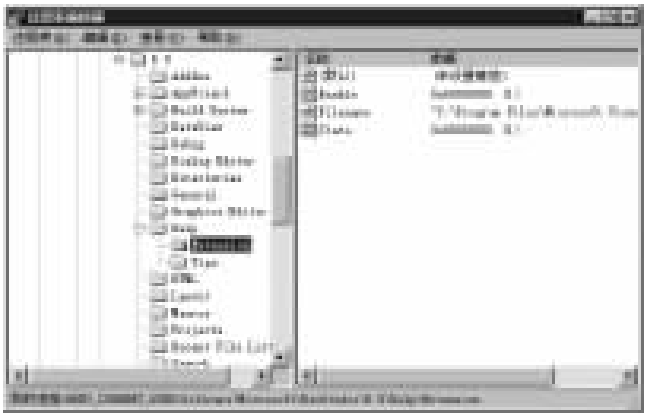


图 4　打开注册表

2. 打开主键 hHKEY_CURRENT_USER hSoftware hMicrosoft hDevStudio h6. 0 hHelp hExtension。该主键下有三个键值，即 Enable、Filename 和 State，如图 4 所示。

在该主键下，有一个键值 Enable，它控制 Visual C + + 6. 0 中 Help|Use Extension Help 菜单项显示与否。如果该键值为 1，那么可以在 Visual C + + 6. 0 开发环境中使用扩展联机帮助，否则不行。Enable 的当前值（即缺省值）为 0x00000001（十六进制），如果不为 1 的话，则将其修改为 1。

键值 Filename 指定扩展联机帮助的文件名（包括完整路径）。将其修改为 ObjectARX2000 联机帮助（文件名为 arxdev. hlp）所在的路径即可，如 D hObjectARX 2000 hDocs h arxdev. hlp。注意：只有当 Enable = 1 且键值 Filename 的值为一合法文件时，Help|Use extension help 菜单项才会显示出来。如果扩展帮助需要由多个帮助文件组成，那么可以定义一个帮助内容文件（cnt），将所有的帮助文件组织在一起，而该文件则作为键值 Filename 的值。限于篇幅，帮助内容文件的编写不作介绍。

键值 State 控制 Help|Use Extension Help 菜单项在 Visual C + + 6. 0 启动后的初始状态，即是否为核选。当 State 为 0，表示初始状态为不核选，而为 1 则表示核选。此时可以不修改该键值。

最后，完成修改后的键值如图 5 所示。

3. 关闭注册表，重新启动 Visual C + + 6. 0 即可。

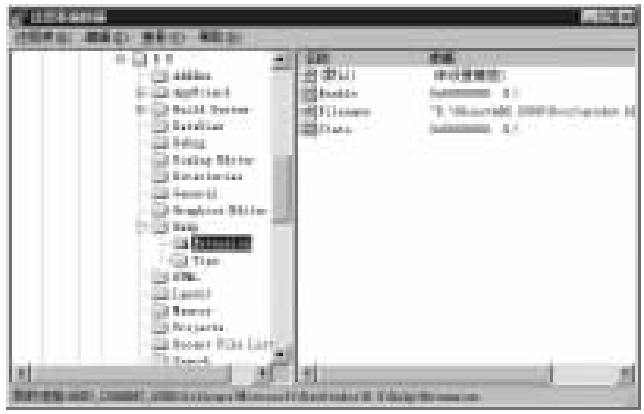


图 5　修改后的各键值

需要说明的是，如果需在 Visual C + + 6. 0 中使用 ObjectARX2000 的联机帮助，那么应该必须核选 Help|User Extension Help 菜单项。之后的其它操作方法与通常的联机帮助使用完全相同。例如，需要迅速获得下述代码中 kForWrite 关键字的联机帮助信息。

```
void changeLayerARX(const AcDbObjectId & entId, const char * pNewLayerName)
{
    AcDbEntity *pEntity;              //定义一个实体对象指针
  acdbOpenObject(pEntity, entId, AcDb::kForWrite); //打开图形数据库并获取实体
    if(pEntity)                       //实体指针不为 NULL
pEntity->setLayer(pNewLayerName); //设置实体的新图层
    pEntity->close();                 //关闭图形数据库
    ads_retvoid();
}
```

首先，选择 kForWrite 字符串，如图 6 所示。

图 6　选定关键字符串

然后，按 F1 键，那么 Visual C + + 6. 0 将打开 ObjectARX2000 的联机帮助，并显示与该关键字相关的帮助主题（除非没有找到相关的帮助主题），如图 7 所示。

三、添加高亮度显示关键字

使用过集成开发环境（如 Visual C + +、Visual Basic 等）的用户应该知道，在源代码文件中，关键字（如 C/C + + 语言的 sizeof、float、double 等，是高亮度显示的，这样就方便了源代码的编辑和修改，从而提高程序开发效率。同理，也可以将 ObjectARX 中的关键字包括函数、类等类型标识符添加到 Visual C + + 6. 0 系统中，使它们也呈高亮度显示。

具体做法是，新建一个文本文件（取名必须为 Usertype. dat），将需要高亮度显示的关键字符添加到文件中，每

图 7　显示相关的帮助主题

个关键字字符串都必须单独占一行。例如，

```
kForwardBranch
kForWrite
kFullGraphics
kGovernedByOrthoMode
RTCAN
RTENAME
RTERROR
```

然后，将该文件复制到 Visual C + + 6.0 安装目录的 hCommon hMSDev98 hBin 子目录中。如果该子目录已经存在一个同名的文件，那么只需将新建 Usertype.dat 文件中的内容添加到当前的同名文件中即可。

完成上述两个步骤后，重新启动 Visual C + + 6.0 之后可以发现，上述字符串在源代码编辑窗口中是高亮度显示的，其效果与 C/C + + 关键字完全相同。

最后需要说明的是，ObjectARX2.x 软件包提供了 Usertype.dat 文件，它存放在 ObjectARX 安装目录的 hUtils hDatfiles 子目录中。但不知什么原因，ObjectARX2000 并没有提供 Usertype.dat 文件。因此，用户可以在卸载 ObjectARX2.0 之前保留该文件备用，按照上述方法将其复制到指定位置即可。

四、使用 ObjectARX 开发模板

为方便 AutoCAD 二次开发，ObjectARX2000 提供了一个模板，它的安装文件存放在 ObjectARX2000 安装目录的 hUtils hObjARXWiz 子目录中，运行目录中的 WizardSetup.exe 安装程序即可安装。这样，在 Visual C + + 6.0 新建工程　使用 File | New 命令）时，就多了一项工程类型　如图 8 所示）。

除此之外，安装完成后还可以激活该模板的工具栏。为此，单击 Tools | Customize 菜单命令，选择 Add - ins and Macro files 属性页。然后，核选其中所有的选项，Visual C + + 6.0 将生成的一个名为 Toolbar1 的工具栏，如果需要更改该工具栏

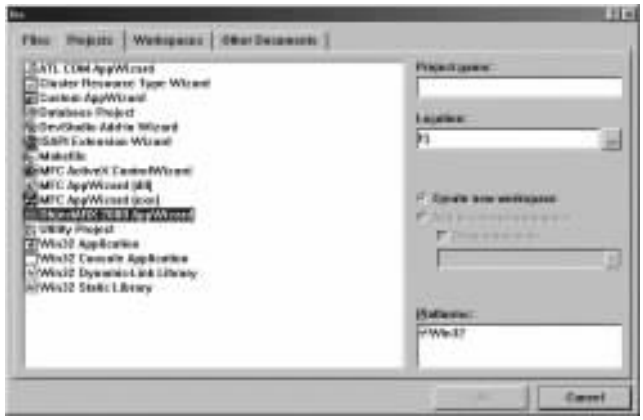


图 8　ObjectARX2000 工程类型

图 9　定制的 ObjectARX 工具栏

名称　如 ObjectARX），那么可以在同样的对话框中选择 Toolbars 属性页，然后在相应的位置修改即可。激活的工具栏如图 9 所示。

五、结束语

通过上述四个方面的定制和修改，ObjectARX 与 Visual C + + 6.0 紧密地结合在一起，大大提高了 AutoCAD 二次开发的效率。

收稿日期：2000 年 7 月 20 日）

中洲 :金仕达 MIS 系统推动企业进步

企业管理软件系统作为一种先进的辅助管理工具,已经被越来越多的现代企业所重视。近日,出于业务发展需要，中洲对外经济贸易公司选择金仕达多媒体公司进行企业管理系统（MIS 系统)的开发建设。

中洲是一家以进出口业务为主的贸易公司，近年来公司扩张的速度很快，不断涉足新的业务领域。由于业务多样化和运作规模不断扩大，中洲公司迫切希望通过公司管理软件协助业务部门规范、高效的实现业务流程，同时也能为公司信息集中管理和共享提供帮助，通过定制的管理软件完善管理流程，达到避免公司资源流失、业务操作的随意性和信息交流的不畅通。

金仕达多媒体作为一家以软件开发见长的高科技公司，在此项目中受到中洲公司的认可，将根据中洲公司业务现状及未来发展定制开发适用的 MIS 系统。

在 PowerBuilder 中利用动态链接库实现数据加密

梁伦发

在数据库应用系统中，有些重要的信息须以加密的形式保存，以防止除用户本人以外的其他用户获知。本文简单介绍了 MD5 单向散列算法以及在VC + + 环境中采用 DLL 技术创建字符串加密函数的方法，阐述了如何在 PowerBuilder 程序中调用 DLL 函数，从而实现数据库中重要信息的 MD5 散列算法加密保存。

一、MD5 散列算法简介

MD5 是一种用来单向变换用户口令和对信息签名的单向散列算法。对于任意长度的输入信息，通过 MD5 散列算法，可以得到一串 128 位（32 位十六进制字符）的标识数据。该算法设想：不可能通过计算使两个不同的输入信息具有相同的标识数据，也不可能根据指定的标识数据推断出变换前的输入信息。因此它是一种单向散列算法，它只能加密数据，而不能进行解密运算。它具有比 DES 算法更强的安全性。

举例来说，在密码认证应用中，数据库里保存了用户密码的标识数据，除用户本人外，任何人无法根据该标识数据知道用户密码。要判断登录用户的密码是否正确，应用系统会根据用户登录时输入的密码进行 MD5 散列表运算，然后将运算的结果和数据库里保存的表示数据进行比较，如果相同，则可以证明该用户是合法用户，否则该用户是非法用户。

MD5 算法包含了以下几个主要步骤：

1. 对输入的数据进行补位，使得如果数据位长度对 512 求余的结果是 448；
2. 补数据长度：用一个 64 位的数字表示数据的原始长度，附加在第 1 步骤生成的数据后；
3. 初始化 MD5 参数；
4. 处理位操作函数；
5. 四轮变换数据处理；
6. 输出结果。

详细的 MD5 算法请参见 RFC 1321。

二、在 VC + + 中创建实现 MD5 算法函数的动态链接库（DLL）程序

PowerBuilder 是目前公认的、最佳的数据库前端开发工具之一，但是它在处理位运算时却显得力不从心，而 MD5 算法又主要是通过一系列的移位和变换实现的，因此要想使用它编写 MD5 散列算法函数几乎是不可能的。幸运的是，PowerBuilder 可以通过 DLL 调用外部函数，这弥补了它本身的不足。

DLL 是一种过程库，应用程序可以在运行时链接并使用它，DLL 库的更新可以独立地进行，而且许多个应用程序可以共享同一个 DLL。从而使应用程序可以共享代码，减少内存占用。

在 VC + + 6.0 中可以用 MFC AppWizard 自动生成 DLL 文件。启动 VC 之后，从 File | New 菜单项选择 New 对话框中的 Projects 标签，选择新项目为 MFC AppWizard（dll），输入工程名 md5，点击 OK 按钮，在其后弹出的对话框上选择 Regular DLL using shared MFC DLL，然后点击 Finish 按钮。这样，一个 DLL 程序框架已经创建，剩下的事情就是手工编写函数代码。

关于实现 MD5 散列算法的具体编码在这里就不详细介绍了，有兴趣的读者可以参考 RFC 1321 中的 C 源代码。需要提起注意的是：在 MSDN 联机帮助以及不少VC + + 参考图书中都提到——在 VC + + 中可以采用两种办法输出函数，一是通过在源程序中使用 _declspec dllexport 关键词输出函数，一是通过编辑 .DEF 文件输出函数。这在 VB 和 VC + + 编程环境中是可以的，但是在 PowerBuilder 调用 DLL 函数时，单纯使用其中一种办法都是行不通的，成功的做法是在 .CPP 程序中做如下声明：

```
extern ″C″ _declspec(dllexport) char * _stdcall MDString (unsigned char *string, int string_length)
```

同时编辑 .DEF 文件，使其内容如下：

```
; md5.def: Declares the module parameters for the DLL.
LIBRARY ″md5″
DESCRIPTION ′md5 Windows Dynamic Link Library′
EXPORTS
  ; Explicit exports can go here
MDString @1
```

即在 .DEF 文件中的 EXPORTS 标签下标明输出函数的顺序值。

编译无误后，一个可以从中调用 MD5 加密函数的 .DLL 文件就完成了。

三、在 PowerBuilder 中调用 MD5 散列算法函数

要在 PowerBuilder 中调用外部函数，必须先声明它。有两种类型的外部函数：全局外部函数和局部外部函数。全局外部函数可应用程序的任何地方声明使用。局部外部函数可以为窗口，菜单，用户对象等定义，它们只能在声明该函数的对象的范围内使用。

下面以一个非常简单的应用程序来说明一下 MD5 函数的调用。该应用程序在 PowerBuilder7.0 下开发，我们做一个窗口对象，然后在选择 Declare / Local External Function 菜单后弹出的窗口中输入下面的字符串，声明该函数：

Function string MDString(string arg1, int arg2) LIBRARY ″md5.dll″

接下来就可以象调用任何标准的 PowerScript 函数一样在该窗口范围内使用该函数，代码如下：

```
sle_2.text = MDString(sle_1.text, len(sle_1.text))
```

运行的效果如下图：

要注意的是：当 PowerBuilder 应用程序调用外部函数时，动态链接库装入内存。如果按上例声明被调用函数，则操作系统会在如下位置查找动态链接库：（1）EXE 文件运行的目录。（2）Windows 的系统目录。如果没有找到 .DLL 文件，应用程序将产生运行错误。如果按下面的方法声明被调用函数，则系统会在指定的存放目录中去查找 .DLL 文件：

Function string MDString(string arg, int length) LIBRARY ″d:/work/md5.dll″

四、结束语

MD5 为我们加密数据提供了有力的武器，而 DLL 为 PowerBuilder 和 VC++之间搭起了桥梁，弥补了 PowerBuilder 在位运算和其他底层程序开发方面的不足，在 PowerBuilder 中轻松实现了数据的 MD5 散列算法加密保存。同样的思路，在 PowerBuilder 环境中也可以调用由 C、PASCAL、VB 等语言开发的 .DLL 程序，大大拓展了 PowerBuilder 的编程处理能力。

（收稿日期：2000 年 10 月 31 日）

（上接第 85 页）

```
    pSession->Close();
    delete pSession;
    return;
  }
/*3. 创建一个 ChttpFile 对象，打开请求句柄，注意：OpenRequest 的最后一个参数必须是 INTERNET_FLAG_KEEP_CONNECTION，否则，程序会莫名其妙地进入死循环。*/
  CHttpFile* pHttpFile = pConnect->OpenRequest(″GET″, m_strPath, NULL, 0, NULL, NULL, INTERNET_FLAG_KEEP_CONNECTION);
  // See if HTTP request handle is valid
  if (pHttpFile == NULL)
  {
    ::MessageBeep(MB_ICONEXCLAMATION);
     AfxMessageBox(″HTTP request failed!″, MB_OK | MB_ICONEXCLAMATION);
    // Close session handles and clean up
    pConnect->Close();
    pSession->Close();
    delete pConnect;
    delete pSession;
    return;
  }  //显示处于等待状态时的鼠标.
  CWaitCursor wait;
  // 4. 发送请求:
  BOOL bSendRequest = pHttpFile->SendRequest();
 // bSendRequest = pHttpFile->SendRequest();
 //5. 验证返回值，验证用户和密码
  DWORD dwRet;
    pHttpFile->QueryInfoStatusCode(dwRet);
    if (dwRet == HTTP_STATUS_PROXY_AUTH_REQ) //代理服务器要求验证
    {
        BOOL i;
i=pConnect->SetOption(INTERNET_OPTION_PROXY_USERNAME, ″dongchao″, 9); //设置用户名
        if(i)
        {
            MessageBox(″OK″);
        },
        pConnect->SetOption
(INTERNET_OPTION_PROXY_PASSWORD
        ″dongchao″, 9); //设置密码
        pHttpFile->SendRequest(); //重新发送请求，进行代理服务器验证.
  }// 读取 SOHU 网页信息
  if (bSendRequest)
  {// Get the size of the requested file
    char achQueryBuf[16]; //    int achQueryBuf;
    DWORD dwFileSize;
    DWORD dwQueryBufLen = sizeof(achQueryBuf);
BOOL bQuery=pHttpFile->QueryInfo(HTTP_QUERY_CONTENT_LENGTH, achQueryBuf, &dwQueryBufLen, NULL);
    if (bQuery)
{// The query succeeded, specify memory needed for file
      dwFileSize = (DWORD)atol(achQueryBuf);
    }
    else
    { // The query failed, so guess at a max file size
      dwFileSize = 10 * 1024;
    }// Allocate a buffer for the file data
    char* lpszBuf = new char[dwFileSize + 1];
   memset(lpszBuf, ′\0′, dwFileSize+1);
    // Read the file
UINT uBytesRead=pHttpFile->Read(lpszBuf, dwFileSize+1);
  //将结果显示在对话框中
     CEdit * pEdit_RESULT=(CEdit *) GetDlgItem(IDC_EDIT_RESULT);
    pEdit_RESULT->SetWindowText((CString)lpszBuf);
    // Clean up buffer
    delete [] lpszBuf;
    // Close all open Internet handles
    pHttpFile->Close();
    pConnect->Close();
    pSession->Close();
    // Clean up
    delete pHttpFile;
    delete pConnect;
    delete pSession;
  }
```

（收稿日期：2000 年 10 月 18 日）

博士信箱 QUESTION&ANSWER

电脑硬盘系统使用与维护常见问题解答

如何全面设置硬盘参数

对个人用户来讲，常用的是 EDIE 接口硬盘（以下简称 EDIE 硬盘）。在微机用的 EDIE 硬盘中，不同厂商、品牌、规格采用的技术各异，使用的参数也不同，如何根据机型、软硬件配置情况设置好硬盘参数，对微机性能的优化、软件的安装使用起着至关重要的作用。下面以 EIDE 硬盘参数的设置为例加以说明。

硬盘参数主要有：记录密度、存储容量、平均寻道时间、数据传输率、硬盘的间隔因子（Interleave）等。记录密度以道密度和位密度来确定，道密度、位密度以及平均寻道时间与采用的生产技术、工艺有关，一般不能修改；硬盘的存储容量是由柱面数、磁头数以及每磁道扇区数决定，硬盘的物理参数（实际的存储容量、柱面数、磁头数、每磁道扇区数等）决定硬盘的类型，主机必须正确知道这些参数，才能正确控制硬盘；由于接口技术的限制，对大容量的 EDIE 硬盘（大于 504MB）还需选择硬盘读写工作模式（MODE），才能正确使用硬盘；数据传输率与接口的传输模式（PIO MODE）、DMA 方式及一些其他 CMOS 参数有关，硬盘的间隔因子（Interleave）需根据微机情况进行设置。

486 以上微机大都使用 EDIE 硬盘，硬盘参数保存在主板上的 CMOS 存储器中，可用 setup 程序进行设置调整，大多数微机在 ROM BIOS 中均有 setup 程序，可在启动微机时运行（AMI 及 AWARD 的 BIOS 可按“Del”键运行），根据所用的硬盘情况选择一个适当的类型或填入相应的参数，也可用高级诊断程序（如 QAPLUS）完成这一工作。

（1）硬盘容量、柱面数、磁头数、每磁道扇区数等参数的设置

硬盘容量（B）= 柱面数 × 磁头数 × 每磁道扇区数 × 每扇区字节数

只要确定了柱面数、磁头数和每磁道扇区数，硬盘容量也随之而定，EIDE 硬盘容量都比较大，硬盘类型设置时都选“user”（用户自定义）。一般用 BIOS 中的 setup 程序对硬盘类型、容量、柱面数、磁头数、每磁道扇区数等参数进行设置。

1）用“IDE HDD DELECTION”项可自动查找硬盘参数，显示可用读写工作模式，并推荐最优的模式。

2）在“STANDARD CMOS SETUP”项中可手动选择（填入）TYPE（类型）、CYLS（柱面数）、HEAD（磁头数）、PRECOMP（写补偿）、LANDZ（着陆区）、SECTOR（每磁道扇区数）、MODE（读写工作模式）等项。“TYPE”项一般选择“USER”，柱面数、磁头数、每磁道扇区数等项填入硬盘物理参数（硬盘面上的标签写有），硬盘容量也随之而定，EIDE 硬盘不需要写补偿，一般该项填“0”，“LANDZ”项是指磁头的“停车”位置，应填入最大柱面数，“MODE”项根据需要选择。

（2）设置硬盘读写工作模式（HDD MODE）

EIDE 支持 3 种硬盘读写工作模式：NORMAL、LBA 和 LARGE 模式，在 CMOS 参数的“MODE”项下有 4 个设定值：NORMAL、LBA、LARGE 和 AUTO。

1）NORMAL，普通模式：这是原来的 IDE 硬盘工作模式，支持的最大柱面数为 1024，最大磁头数为 16，每磁道最大扇区数为 63，该模式可管理的最大磁盘容量为：1024 × 16 × 63 × 512 = 528482304B，即 504MB（MB = 1024 × 1024）

在该模式下，即使硬盘的实际物理容量更大，也只能访问 504MB 空间。

2）LBA（Logial Block Addressing）逻辑块地址模式：在该模式下设置的柱面数、磁头数、每磁道扇区数并不是实际的物理参数，在访问硬盘时，由 EIDE 控制器把柱面数、磁头数、每磁道扇区数等参数确定的逻辑地址转换为实际的物理地址。支持的最大磁头数为 255，其他参数与普通模式相同。因此，该模式可管理的最大硬盘容量为：1024 × 255 × 63 × 512 = 8422686720B，约 7.8GB（GB = 1024MB）

3）LARGE，大模式：当硬盘的柱面数超过 1024 而又不为 LBA 所支持时，可采用该模式。LARGE 模式参数设置方法是把柱面数除以 2，把磁头数乘以 2，使柱面减少容量不变。例如，硬盘物理柱面数为 1460，磁头数为 16，进入 LARGE 模式则为柱面数 730，磁头数 32，这样在一些软件看来柱面数小于 1024，该模式可管理的最大硬盘容量为：1024 × 32 × 63 × 512 = 1056964608B，约 1GB。

在设置 MODE 项参数时，可根据需要进行设置。选择“AUTO”，微机自动根据最优原则选择硬盘支持的工作模式；对于大于 504MB 支持 LBA 模式的硬盘，选择“LBA”比较合适；如系统要安装不支持 LBA 模式的一些软件，如低版本的 DOS、SCO Xenx V3.22、Unix R3.24 等，只能选择“NORMAL”。

（3）影响硬盘数据传输率的 CMOS 参数的设置

在 CMOS 参数中，与硬盘数据传输率有关的主要参数有：IDE HDD BLOCK MODE（IDE 硬盘块模式）、32bit Transfer（32 位数据传送）、PIO MODE/DMA（PIO 传输模式/DMA 方式）等。

1）设置 IDE HDD MODE

IDE 设备块模式是在每次中断时，都传送指定的若干个扇区的数据。硬盘支持块模式时，允许该模式工作，可提高访问硬盘速度，提高数据传输率。参数选项有：Auto、Optimal、Disabled 或 Enabled、Disabled。设定“Enabled”允许此模式，设定“Disabled”禁止此模式，参数“AUTO”将按照硬盘自动检测功能的报告值作为数据传送的扇区数，“Optimal”则以最优缺省值为传送扇区数。有的 BIOS 还给出传送的扇区数，如 2、4、

6、8、16、32、HDD MAX 等，选什么值合适，可多试几次，一般选“HDD MAX”。

2） 设置 32bit Transfer

该参数的选项为：Enabled 和 Disabled（允许和禁止）。该项允许与否直接影响 Windows 3.1 以上版本提供的 32bit 文件访问功能的使用。在 Windows 3.1 以上版本，32bit 磁盘访问功能配以 32bit 文件访问功能（可在 Windows 控制面板中进行设置），可大幅度提高硬盘访问速度，提高系统性能。因此如系统装有 Windows 3.1 以上版本，此参数应设置为“Enabled”。

3） 设置 PIO MODE

PIO（Programmed Input/Output）MODE/DMA 直接影响硬盘的数据传输率，应根据设备特性进行设置 PIO 模式。设定值有：AUTO、MODE 0、MODE 1、MODE 2、MODE 3 和 MODE 4，DMA 方式设定值有 DMA0、DMA1、DMA2。不同的设定值，其对应的数据传输率差别较大，PIO MODE 0、PIO MODE 1、PIO MODE 2 中，DMA0 一般是普通的 IDE 接口使用的，数据传输率仅为 3MB/s～8MB/s。EIDE 支持新的宿主传输标准，如 PIO MODE 3 和 PIO MODE 4，数据传输率达 11.1MB/s 和 16.6MB/s，也支持 DMA1 和 DMA2，数据传输率为 13.3MB/s 和 16.6MB/s。通常对 EIDE 硬盘为了说明不同的传输标准，把支持 PIO MODE 3 或 DMA1 的称为 FAST ATA，支持 PIO MODE 4 或 DMA2 的称为 FAST ATA-2，在设置时应予注意。还要注意硬盘本身所支持的 PIO 模式及 DMA 方式，如不了解可先设成“Auto”或“Enabled”，然后再根据情况进行调整。

此外，还有新型的 EIDE 接口，即 Ultra/DMA33，Ultra/DMA66 等。数据传输率高达 33MB/s 和 66MB/s，在 CMOS 参数中有“IDE Ultra mode”或“IDE...UDMA”项，设定值有 Auto、Disabled 或 Enabled、Disabled，当硬盘支持该传输标准时，应选择“Auto”或“Enabled”。

4）硬盘间隔因子（Interleave）的设定

硬盘间隔因子为硬盘读/写周期中按逻辑顺序读/写扇区之间的物理扇区号间隔值，即已操作物理扇区号与紧接读/写物理扇区号之间的差值，如间隔因子为 3，则在一磁道上读/写物理扇区号顺序应为 1,4,7,10,13,... 间隔因子并不是越小传输率越高，这与硬盘如何与计算机匹配有关。间隔因子一般在对硬盘进行低级格式化时需进行设定，选择适当的数值以使硬盘有较高的性能。在一些微机 ROM BIOS 中，存有供对硬盘进行低级格式化程序菜单和一些高级诊断程序菜单（选项），它们有自动确定间隔因子一项，在给出的间隔因子及对应的数据传输速度中，可选择最合适的间隔因子（对应数据传输速度应最快）。当然，对 EIDE 大容量硬盘，间隔因子一般可采用默认值，通常认为 3 是最优的。

如何解决主板不认大硬盘的情况

现在硬盘容量已大大超过了 8.4GB，为了超越这个容量限制，人们又定义了新的扩展 INT13。新的 INT13 不使用操作系统的寄存器传递硬盘的寻址参数，它使用存储在操作系统内存里的地址包。地址包里保存的是 64 位 LBA 地址，如果硬盘支持 LBA 寻址，就把低 28 位直接传递给 ATA 界面；如果不支持，操作系统就先把 LBA 地址转换为 CHS 地址，再传递给 ATA 界面。通过这种方式，能实现在 ATA 总线基础上 CHS 寻址最大容量是 136.9GB，而 LBA 寻址最大容量是 137.4GB。因此，要正常使用大容量硬盘，可以从软、硬件两方面来加以解决。

（）硬件解决方法

1）升级主板或主板 BIOS

新的主板 BIOS 对磁盘读写中断 INT13H 进行了扩展，一般主板升级 BIOS 后即可支持 8.4GB 以上的磁盘。此外还可以使用 BIOS 扩展卡（它对大容量硬盘提供正确的 LBA 寻址支持）。比如 Pormise 生产的多功能 I/O 卡，它自带的 BIOS 能识别大容量硬盘。升级主板 BIOS 的具体方法，可参看相关的文章。

2）购买自带 LBABIOS 的多功能 I/O 卡

自带 LBABIOS 的多功能 I/O 卡，它自带的 BIOS 能识别大容量硬盘，如：Pormise 生产的多功能 I/O 卡。或者是单个只带 LBABIOS 的 ISA 插卡。

（）软件解决方法

1）升级 BIOS

① Award1997 年 11 月及其以后的 BIOS 支持容量大于 8.4GB 的硬盘。

② AMI1998 年 1 月及其以后的 BIOS 支持容量大于 8.4GB 的硬盘。

③ Phoenix 基础版本 4（Version4），修改版本是 6（Revision6）和更高的版本支持容量大于 8.4GB 的硬盘。而如果 BIOS 的修改版本（Revision）是 5.12 它就不支持扩展 INT13。因为所有的 Phoenix BIOS 基础版本都是 4 所以升不升级主要看它的修改版本号。

2）用硬盘自带的 DM 分区软件分区。使用特殊的驱动程序（一般是硬盘自带的分区软件 DM），也提供 INT13H 的扩展功能，从而在不动主板的情况下支持大硬盘。比如对 BIOS 不支持 LBA 寻址的机器来说，迈拓公司（Maxtor）提供了 MaxBlast 的软件，它能有效地转换大容量硬盘的各个参数，达到全容量使用硬盘的目的。MaxBlast 不是在操作系统启动后才加载的，而是在 BIOS 启动后、操作系统启动前。最新的 MaxBlast 软件可从 www.maxtor.com 处下载。

另外，Western Digital 的 EZdrive（最新 9.0 版本）也是类似的软件。它界于操作系统和 BIOS 之中，既能符合老式 BIOS 限制硬盘容量的要求，也能保证操作系统正确地访问整个硬盘。西部数据 WD 硬盘的最新配套工具 wd906w.zip 中的 ez.exe 文件，运行后将帮助用户快速简单地并代替 FDISK 和 FORMAT 程序完成分区和格式化，如果主板 BIOS 不支持大容量硬盘，EZdrive 会安装 EZ-BIOS 支持大容量硬盘。

3）使用 Win97 以上的操作系统，使用 FAT 32，并合理分区。