电脑编程技巧与维护月刊

2001 年第 4 期　　总第 82 期　1994 年 7 月创刊）　　每月 3 日出版

主管部门　中华人民共和国信息产业部
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

新技术追踪

编程与应用起步

编程语言

专家论坛

可视化专栏

数据库

网络技术

图形图像处理

计算机维护

计算机安全

博士信箱

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?** _____

_____

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

Maxtor 公司弃 FreeBSD 转用 Windows

Maxtor 公司将在新的 Max Attach 4100 存储系统中放弃 Free BSD，转而采用 Microsoft 的 Windows 2000。Maxtor 之所以将它的网络附加存储设备从开放的 Free BSD Unix 转向 Microsoft 的操作系统，是为了提高互操作性和可管理性，同时降低编程成本。

Bluetooth Version 1.1 确定 3 月中旬公开

据从事近距离无线通信技术“Bluetooth”标准化工作的 Bluetooth SIG Special Interest Group 的有关人员透露，Bluetooth 的最新标准“Version 1.1”已经于 2001 年 2 月底通过了 SIG 上层组织的投票。该标准的草案版已于 2000 年 10 月份向会员企业公开。根据该标准的规定，各会员企业今后将不能再使用支持此前的标准“Version 1.0b”或者“Version 1.0b + Critical Errata”，而必须采用支持 Version 1.1 的收发信号模块来推动产品化进程。据有关人士介绍“最早在 3 月 15 日，最晚在 3 月底向普通用户公开”。

美国微软发布 Windows XP 的 ACPI 封装

微软日前公开了将于 2001 年下半年推出的最新操作系统“Windows XP”的 ACPI 封装。该产品没有来得及完全采用最新标准的 ACPI2.0 封装，而是在 ACPI1.0 的基础上嵌入了 ACPI2.0 的若干标准。在此前推出的 Windows XP 试用版中，Power Scheme 电源设定 被设定为“Portable / Laptop”时，AC 适配器就会自动选择“None”模式，而在最新版本中则被改变为“Adaptive”。因为目前的笔记本电脑在最高性能下工作时容易引发散热方面的问题。此外，ACPI 2.0 的 Processor Performance State 的封装仅限于 i815 芯片组等以 ICH 为基础的情形，440BX / MX 等无法封装。

SONY 构筑可播放动态图像 Web 3 维空间

SONY Marketing 近日开始公开发售可在 Web 的 3 维虚拟空间内连续播放流式动态图像的软件服务产品“Space stream”。Space Stream 可根据 Web 上的 VRML 虚拟现实模型语言 所构成的 3 维空间内显示屏幕，并能在该屏幕上播放动态图像的相关内容。通过鼠标的操作，动态图像可在 3 维空间内自由地来回移动，并且还可根据空间内的虚拟位置和屏幕的距离及角度自动地调节音量。播放动态图像的引擎可对应 QuickTime Player、RealPlayer、Windows Media Player，浏览所需的软件为 PlugIn 方式，可以免费获得。

微软发布 Visual Studio.NET Open Tools Platform

微软日前发布了用于 Web 服务和应用的开发工具平台“Visual Studio.NET Open Tools Platform”。该平台的主要用户为从事 Web 服务和应用功能构筑的开发人员、中立软件开发商 ISV 以及语言开发商。Visual Studio.NET Open Tools Platform 由作为核心的面向 .NET Platform 的开发工具“Visual Studio.NET”和软件开发工具包 SDK“Visual Studio Integration SDK”以及“Visual Studio for Applications SDK”组成。

苹果发布 Mac OS X“黄金代码”

日前，苹果公司公开了新一代苹果电脑操作系统的最终代码 专业人士称之为“黄金代码”。苹果公司将利用公开的“黄金代码”生产可供零售的版本。部分分析人士指出，苹果公司这么做，完全是为了赶在已经确定好的 3 月 24 日新操作系统发布日这一天将 Mac OS X 推向市场，但是由于缺乏对 DVD 和笔记本电脑休眠功能的支持，新操作系统的成熟还需要一段时间。Mac OS X 售价为 129 美元。凡是购买了测试版的用户可享受 7 折优惠。任何一款 iBook、iMac、Power Mac G3、Power Mac G4、G4 Cube 和 1998 年 9 月以后出厂 PowerBook 都可安装使用，但是内存不得小于 128 兆。

新款 Linux 文件管理工具发布

日前，美国 New Planet 新行星 软件公司推出了 System G 工具软件测试版。根据新行星软件公司以前的声明，该版本将在新的许可证模式下发行。System G 提供了一流的图形操作界面，使得电脑用户在 Linux 操作系统下可以简单地操作文件和目录。System G 2.0.0 第一个测试版的最新特性：如果你想得到更多有关 System G 的信息，请访问它的官方站点：www.newplanetsoftware.com，并免费下载一个有时间限制的测试版。

黑客新工具可盗用 IBM 电商服务器密码

日前，IBM 在网站上张贴了一则通告，提醒用户当心一个新发现的黑客工具。这个黑客工具名叫“SUQ.DIQ”，版本为 1.00。据称，如果用户的服务器中使用了某些 IBM 的电子商务软件，那么，黑客就能利用上述工具以及宏来破译存储在该服务器上的管理员及用户的密码。受影响的 IBM 电子商务服务器包括：v3.1，v3.1.1，v3.1.2 和 v3.2 的 Net.Commerce；v4.1 和 v4.1.1 的 WebSphere 商务套装软件；v3.1.1，v3.1.2 和 v3.2 的 Net.Commerce 主机服务器；v3.2 的服务提供商版 WebSphere 商务套装软件；以及 v4.1 的市场版 WebSphere 商务套装软件。受影响的操作系统包括 IBM 的 AIX，微软的 Windows NT，以及 Sun 微系统公司的 Solaris OS。

NEC 上市配备无线 LAN 功能面的路由器

NEC 日前推出了配备无线 LAN 功能、面向 SOHO 用户的路由器“AtermWL20R”。该产品将与个人电脑用无线 LAN 适配器配套，以开放式价格销售。估计实际销售价格，在附带1

台通过 USB 个人电脑的无线 LAN 适配器的 “AtermWL20R & WL11U”的情况下将低于 5 万日元；而附带 1 台 PC 卡无线 LAN 适配器的 “AtermWL20R & WL11C”则将在 4 万 5000 日元左右。上述产品将从 3 月中旬开始供货。AtermWL20R 配备一个用来连接 ADSL Modem 及有线电视 Modem 的 10BASE - T 的外部 LAN 接口，据悉该产品的用户对象为将多台个人电脑同时通过 ADSL 及有线电视接驳因特网的用途。内部 LAN 的接口则只有无线 LAN 和一个 USB 端口。

英特尔 CPU 又向 10GHz 迈进了一步

日前，英特尔公司在微处理器制造技术开发方面又跨越了一个重要的里程碑。这种新的微处理器制造技术，可以用于生产速度为目前 5 倍快的微处理器。据称，上述技术是第一种可用于尖端紫外线光刻技术 EUV 的标准格式光掩膜。这一技术可以使微处理器制造商在硅晶片上蚀刻出更小的图案，使微处理器制造工艺达到 70 毫微米这一等级。目前的微处理器大多是采用 180 毫微米的工艺而制造的。由于 EUV 技术的开发比较顺利，许多行业人士认为，这种技术将取代现有的深度紫外线光刻技术 DUV ，用于制造速度可达 10GHz 或者更高的微处理器。

微软准备开放 Win 2000 和 Win XP 的源代码

微软最近出台了一项新的政策，即允许大企业客户选择使用 Windows 2000 专家版、服务器版本、高级服务器版本以及数据中心版本 Windows XP 已经发布的 beta 版本 以及所有相关服务软件包的开放源码版本。这项新的措施十分类似 IBM 开放其操作系统源码的做法，即允许客户在调试自己应用程序的时候咨询 Windows 的源码，以更好的将 Windows 整合到自己的企业环境之中。但是，值得注意的是，新协议并不允许客户修改源码或对源码进行一些个性化处理。据微软公司的发言人说，一旦客户在 Windows 中发现问题或臭虫将直接报告微软公司由他们通过正常的技术支持渠道负责提供解决办法。

汉语拼音方案的通用键盘表示规范发布

由北京语言文化大学和北京信息工程学院专家历时两年设计的 《汉语拼音方案的通用键盘表示规范》，已于近日发布，并将从今年 6 月 1 日起实施。 《汉语拼音方案》是给国家通用语言文字拼写和注音的工具，自 1958 年公布以来，在许多领域得到了广泛应用。 《汉语拼音方案》采用国际通用的拉丁字母符号，为在通用键盘上直接输入汉语拼音提供了方便。但是，《汉语拼音方案》中声调符号和隔音符号在通用键盘上没有相应的键位表示。各种汉语拼音输入方法对此采用了各自的表示方法。由于各种输入法替代表示方法的不同，给用户使用带来了不便，也影响了中文信息的处理。为此，国家语委于 1999 年开始组织制定 《汉语拼音方案的通用键盘表示规范》。

卡西欧、NEC 将推出克鲁索笔记本电脑

一批使用全美达克鲁索处理器的笔记本电脑产品将在后两个月登陆美国市场，其中一款将包含视窗和 Linux 两种操作系统。卡西欧公司将在春季在美国市场推出使用克鲁索处理器的 Fiva 笔记本电脑。Fiva 在提供 Windows 2000 操作系统的同时还使用了 Linux 系统。NEC 也将推出低价使用克鲁索芯片的笔记本电脑。其中一个版本将使用节省空间的锂聚合电池。其他的笔记本电脑制造商也将陆续在北美推出使用全美达芯片的产品。新机器也标志着全美达进一步扩展了市场。目前仅有索尼在美国推出了全美达笔记本电脑。克鲁索笔记本产品正越来越走进商业客户，这种芯片耗能量较其他笔记本芯片少，能提供更长的电池使用时间，体积也可以做的更小些。

三星率先开发 0. 1 微米芯片制造技术

韩国三星电子公司日前宣布，该公司已率行开发领先全球的芯片加工技术，可以生产四个 10 亿位元 gigb bit 的动态随机存取存储器 DRAM 芯片。三星电子公司发表的声明指出，已完成一项 4 - gigabit 的全密度回路设计，可以批量生产这种新 DRAM 芯片。这家全球最大电脑存储器制造商在声明中说，每颗测试的 4 - gigibit DRAM 芯片可以储存 42. 9 亿位元，相当于 5 亿个英文字、一般报纸 3. 2 万页、1600 张静态照片或是 64 小时的录音资料。三星公司发言人说，最受人瞩目的是，这是全球首次使用 0. 10 微米加工技术。该公司预期，0. 10 微米加工新技术能将目前市场广泛使用的 128 个百万位元 megabit 或 156 个百万位元的 DRAM 生产成本降低 60%。

VC + +系列讲座（四）

徐 亮 张博强

第四讲 对话框的应用

在 Windows 中有两种类型的对话框：预定义对话框和自定义对话框，预定义对话框的外观已经被预先定义好，只能对它进行很小的改动；自定义对话框则是由用户自己创建，用户可以定义它的外观及行为。

MFC 的 CDialog 类提供了管理对话框的功能。为了使用对话框，我们要做两部分的工作，首先是创建对话框资源，即利用各种控件完成对话框的外观设计；然后，编辑对话框的值及类成员函数，以便实现用户与对话框之间的通信。

作为对话框的基类，Cdialog 定义了一个构造函数和一个 Create 函数。构造函数常用于构造基于资源的有模式对话框，所谓有模式对话框，即当它运行时，将不能对该应用程序的其它部分进行操作；Create 函数常用于创建无模式对话框，无模式对话框打开时不会影响你对应用程序其它部分的操作。同时，Cdialog 类还定义了三个虚函数：

OnInitDialog 默认情况下，在对话框调用时将焦点设在对话框的第一个控件上。

OnCancel 在其中调用 DestoryWindow 函数删除对话框窗口。

OnOK 在其中调用 EndDialog 函数，如果有模式对话框重载了 OnOK 函数，则该函数应调用基类的 OnOK 函数，确保调用 EndDialog 函数。

下面的通过一个简单的对话框应用实例，来说明创建对话框的步骤和方法。

在这个例子中我们通过点击 “Getdata ”按钮，打开数据输入对话框，在编辑框内输入数据后，点击 OK 按钮返回，当点击 “Display ”按钮的时候新的数据将被显示在编辑框内。

首先进行对话框资源的设计，打开 AppWizard，创建一名为 “Mydialog ”的对话框工程，并保留默认选项。在建立好的对话框中删除已存在的控件，并加入新的控件，设置如下：

Static Text ID ID_STATIC
Capition 数据将在下面显示
Edit Box ID IDC_DATA
Button ID IDC_DISPLAY
Capition & Display
Button ID IDC_GETDATA
Capition & Getdat
Button ID IDC_EXIT
Capition E& xit

接下来，我们还要创建一个用来输入数据的对话框。

从 ResourceView 中扩展 Mydialog resource，用鼠标右键点击 Dialog 项，在弹出菜单中选择 “Insert Dialog ”，这时一个新的对话框被建立，设置它的属性并为它增加新的控件：

Dialog ID IDD_GETDATA
Capition 输入数据
Static Text ID IDC_STATIC
Capiticon 在下面输入数据
Edit Box ID IDC_INPUT

下一步进行编码工作，这时当你打开 ClassWizard，由于此时对话框还没有与任何类相连，所以，你会看到弹出 “Adding a Class ”的对话框，点击 OK 按钮，进入 “New Class ”对话框，在 Name 中输入 “CGetdata ”，这样就创建了 CGetdata 类，同时还创建了两个文件，Getdata. h 和 Getdata. app。

虽然，我们已经知道两个对话框之间的关系，但 VC 并不清楚，我们还要在两者间建立起联系。

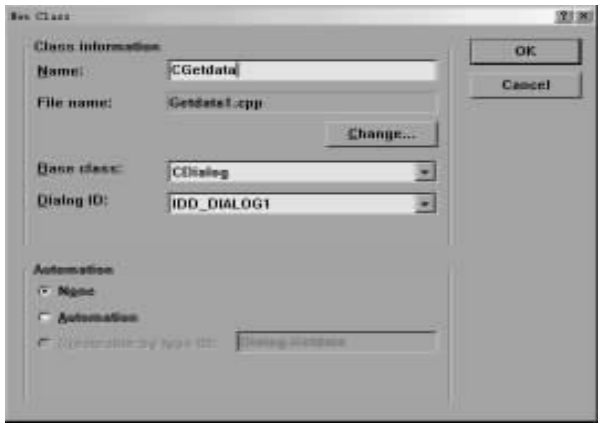


图 1 新类对话框

使用 FileView 扩展头文件项，选择 MydialogDlg. h，双击打开后，在其类定义前加入

```
#include “Getdata. h ”
```

在公有成员声明部分声明 CGetdata 类对象 m_pGetdata

```
……
public:
CMydialogDlg(CWnd * pParent = NULL);    // standard constructor
CGetdata m_pGetdata;
……
```

然后，我们为 Exit、Getdata 按钮加入代码。打开 ClassWizard， “Class Name ”选择 CmydialogDlg， “Object ID ”选择 IDC_EXIT， “Message ”选择 ON_CLICKED，为它添加 OnExit 函数如下：

```
void CMydialogDlg: : OnExit()
{
OnOK();
}
```

再选择 IDC_GETDATA，“Message”为 ON_CLICKED，为它添加 OnGetdata 函数：

```
void CMydialogDlg: : OnGetdata()
{
    m_pGetdata. DoModal();
}
```

DoModal 函数将唤醒一个模式对话框，并且在对话框运行结束时，返回运行结果。

现在你可以编译运行程序，看一看结果。点击 Getdata 按钮，数据输入对话框将弹出，如果点击 Exit 按钮，程序将退出。可是现在的程序还不能显示输入数据，接下我们需要增加两个变量并为 Display 按钮编写代码，并且在 OnInitDialog 函数中为对话框的编辑框初始数据。

图 2 数据显示主对话框

打开 ClassWizard，在 MemberVariables 中选择 CGetdata 类，为编辑框 IDC_INPUT 引入一个 CString 型的成员变量 m_input。再从 Clsss Name 下拉列表中选择 CMydialogDlg，为 IDC_DATA 引入变量，Member Variable Name 为 m_data，Category 为 Control，Variable Type 为 CEdit。

再回到 Messaage Maps，从“Class Name”下拉列表中选择 CMydialogDlg，“Object ID”选择 IDC_DISPLAY，“Message”为 ON_CLICKED，点击添加函数按钮，添加 OnDisplay 函数：

```
void CDialogDlg: : OnDisplay()
{
    m_data. SetWindowText(m_pGetdata. m_input);
}
```

SetWindowText 函数将用来在编辑框中显示输入的字符串。

在 ClassView 中扩展 CmydialogDlg，双击 OnInitDialog 函数，编辑它的代码，

```
BOOL CDialogDlg: : OnInitDialog()
{
    CDialog: : OnInitDialog();
        //下面是加入的代码
    m_data. SetWindowText(″没有数据″);
    //上面是加入的代码
    // Add ″About. . . ″ menu item to system menu.
//IDM_ABOUTBOX must be in the system command range.
    ASSERT(( IDM_ABOUTBOX& 0xFFF0 ) = = IDM_ABOUT-
BOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
    CMenu * pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu ! = NULL)
    {
        CString strAboutMenu;
        strAboutMenu. LoadString(IDS_ABOUTBOX);
        if (! strAboutMenu. IsEmpty())
        {
            pSysMenu - >AppendMenu(MF_SEPARATOR);
 pSysMenu - >AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }
    }
    // Set the icon for this dialog.  The framework does this
automatically
    //  when the application´s main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon
        // TODO: Add extra initialization here
        return TRUE;  // return TRUE  unless you set the
focus to a control
}
```

图 3 数据输入子对话框

现在编译运行程序，你可以看到，我们已经得到了所需要的功能，但如果你细心会发现一个问题，就是当按下“Enter”或“Esc”键时，程序窗口将自动关闭，这是由于 VC 将“Enter”和“Esc”作为默认键，分别控制 OK 和 CANCEL 按钮。虽然我们在程序的开始已经将它们都删除了，但 Window 仍然会自动搜寻程序或资源指定的缺省按钮，当我们不需要这项功能时有必要对对话框的虚函数 OnOK 和 OnCancel 进行处理。

首先使用 ClassView 为 CMydialogDlg 加入两个成员函数。用鼠标右键点击 CmydialogDlg，选择“Add Member Function”，在 Functionu Typ 填入 Void，Function Declaration 为 OnOK，点击 OK 确定。以同样的方法加入 OnCancel 函数。但我们并不需要为它们加入任何代码，接下来我们还要修改一下 OnExit 函数，修改后代码如下：

```
void CMydialogDlg: : OnExit()
{
Cdialog: : OnOK();
}
```

Word 文档反像打印技术

洪志全

Microsoft Word 以其强大的图文排版和所见即所得功能，是各单位、出版印刷行业使用最广泛的文字排版软件。对出版行业而言，为了便于排版文档制版、提高印刷质量、降低印刷成本往往采用硫酸纸反像（镜像）发排输出。对于 Word 编排的书稿，因 Word 无反像打印功能，出版社一般只能打印成胶片，这样增加了印刷成本。

本文将介绍如何利用 Visual Basic 6.0 编写针对 Word 文档的反像打印技术和方法，由于程序较长，文中只能给出部分关键处理程序。

一、反像打印原理

为了将 Word 文档反像打印输出，一般有两种方法：一是在 Microsoft Word 中用 Visual Basic 编制宏程序，对当前编辑的 Word 文档进行反像处理输出，这要求用户对 Word 中的各种对象格式有相当了解，在实现上有一定困难；二是把 Word 文档转换为打印映像文件，再对映像文件分析处理，然后反像输出。

本文就是采用后一种方法，首先将 Word 文档转换为 HP6L PCL 命令集，再进行数据处理，实现反像打印功能，打印原理如图 1 所示。该反像打印可保证打印输出完全与 Word 的排版打印风格相同，无需添加硬件即可直接实现 Word 文档反像输出的优点，输出效果非常好 600DPI 。

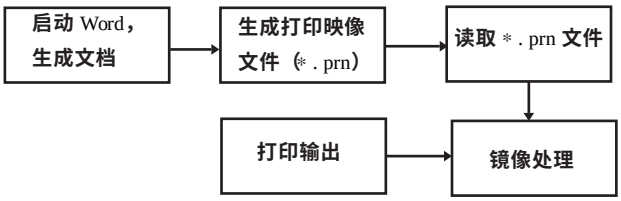


图 1 反像打印原理

在 Word 文档反像打印处理中，首先在 Word 中用 HP 6L PCL 打印一个 PCL5 命令集的映像文件，然后通过程序对该映像文件读取、处理、反像打印输出。

二、PCL5 数据压缩方法

HP6L 采用 PCL5 命令集，PCL（Printer Command Language）命令集是利用打印时点与点之间、行与行之间数据的相似性，对数据压缩后再传送到打印机，以提高打印速度。PCL 是 HP 公司开发的用于打印机命令语言，PCL5 命令集常用命令如表：

PCL5 命令	命令功能说明
ESC% - 12345X	打印页开始、结束标志
@PJL SET ……	激光打印参数设置命令
ESC * c0t####x####Y	打印页面大小说明，#### = 页宽（页高）×分辨率
ESC * p0x0Y	打印坐标回原点
ESC * p####X	打印光标到当前行####位置
ESC * p####Y	打印光标####行位置
ESC * t###R	设置打印分辨率
ESC * b#M	设置数据压缩方式
ESC * b###W	一行需传送的数据字节数
ESC * rB	结束光栅图像
ESC * r0A、ESC * r1A	设置光栅图像输出位置

实现 Word 文档反像打印的关键是如何将 PCL5 的压缩数据还原为非压缩数据，PCL5 有四种数据压缩方法，PCL5 通过压缩方法命令 ESC * b#M 来告知激光打印机当前所传送的数据使用什么压缩方法，其中#代表压缩方法编号，# = 0 表示数据位未经压缩编码；# = 1 表示数据是用行程编码压缩；# = 2 时表示数据是用间或编码压缩；# = 3 时表示是用行间差异编码

我们再来试一下，“Enter”和“Esc”键已经不再起作用了。

小结

本讲主要介绍了建立对话框的基本过程，通过本讲的学习我们应该掌握：

1. 对话框的组成，即对话框资源和对话框的类。
2. 对话框的分类，有模式对话框和无模式对话框的。
3. CDialog 类的几个主要函数，OnInitDialog 、OnOK 、OnCancel 及 Create 等。
4. 对话框间进行数据传递的方法，及用于显示有模式对话框的 DoModal 函数。

本讲主要讲述了有模式对话框的建立方法，读者可以试着建立无模式对话框。只有不断练习当中，才能学会使用一门语言。

参考资料

1. 王晖等编著．精通 Visual C++ 6.0. 电子工业出版社
2. 张金山、廖果编著．VC++ 5.0 易学活用
3. Davis Chapman 著．学用 VC++ 6.0

收稿日期：2000 年 7 月 25 日）

压缩。PCL 5 中的压缩方法都是针对某一行数据而言，而所谓一行数据指打印机在纸张上打印的一行像素用八位　一个字节二进制编码形成的数据。计算机一旦对一行数据处理完就用这样一种指令传送给打印机：ESC ＊ b#W，这里的#号表示这一行数据共有多少个字节。

1）方法 0　ESC ＊ b0M

压缩方法 0 是非压缩数据传输方式，它所送出的数据是按打印点的顺序以二进制形式编码后形成的。如要打印如下图所示的两行图形，根据对一行数据单独传送打印的特点，对这两行数据进行其点的排列进行二进制编码：

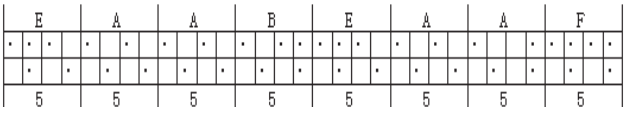


图 2　未压缩打印数据

第一行形成的二进制数据为：EA AB EA AF 共四个字节；第二行形成的二进制数据为：55 55 55 55 共四个字节。这样用方法 0 传送这两行数据的指令为：

ESC ＊ b0M

ESC ＊ b4W EA AB EA　AF 传送第一行数据　；

ESC ＊ b4W 55 55 55 55 传送第二行数据　；

把指令全部用十六进制表示即为：

1B 2A 62 00 4D 1B 2A 62 04 57 EA AB EA AF 1B 2A 62 04 57 55 55 55 55 55，这 23 个字节的数据就是计算机向打印机传送的全部数据。

2）方法 1　ESC ＊ b1M

先分析一下图 2 中欲打印数据的特点：对第一行而言无规律，但第二行数据就不同了：后面数据只是前面数据的重复，这就是说可传送一块被重复图形的数据及其重复次数来代替原来传送数据。通过下面指令说明压缩方法 1 压缩处理方式：

ESC ＊ b1M

ESC ＊ b4W 03 AA 01 DE

ESC ＊ b2W 04 5A

第一条指令 ESC ＊ b1M 表示以后所传送的数据是用方法 1 压缩，第二条指令中的 ESC　＊ b4W 表示后面的 4 个数据 03 AA 01 DE 是本行欲传送的打印数据，其中 03 AA 表示将具有 AA 特点的图形重复打印 3 次，而 01 DE 表示将具有 DE 特点的图形重复打一次；第三条指令中的 ESC ＊ b2W 表示后面的 2 个数据 04 5A 表示将具有 5A 特点的图形重复打印 4 次，打印结果如图 3 所示。从传送的数据可以看出第一行数据压缩前后的数据量相同，第二行的压缩率可达 50%。

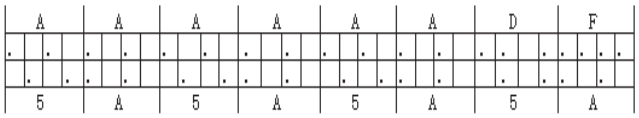


图 3　方法 1 打印图形编码

3）方法 2　ESC ＊ b2M

方法 2 是方法 1 的改进。方法 1 是建立在重复数据传送基础上的，当所传送数据重复性很小时方法 1 不但不能减少数据量，还会增加数据量，如传送这样一行数据 11 22　33 44，则数据传送指令为 ESC ＊ b8W 01 11 01 22 01 33 01 44，所传送的数据量增加一倍。方法 2 的原理就是 对能压缩的数据则压缩，不能压缩的则在不改变方法的前提下通知打印机为非压缩数据。具体规定为 :在传输一段数据之前必须给出这段数据的解释码 N，N 为 1 个字节，当 N 的值小于 128 时，表示后面应传输 N + 1 个非压缩数据；当 N 大于 128 时，它后面只跟一个字节的数据，该字节应重复 257 - N 次，以图 2 数据为例，用方法 2 传送时的指令如下：

ESC ＊ b2M

ESC ＊ b5W 03 EA AB EA AF

ESC ＊ b2W FD 55

这里 ESC ＊ b2M 表示以后所传送的数据是用方法 2 所压缩；ESC ＊ b5W 表示这一行共有 5 个数据 03 EA AB EA AF 被传送，由于 03 小于 128，所以 03 后面的 3 + 1 = 4 个数据 EA AB EA AF 为非压缩数据；ESC ＊ b2W 表示这一行共有 2 个数据 FD 55，由于 FD 十进制为 253　大于 128，所以它后面的数据 55 要重复 257 - 253 = 4 次。用 VB 6. 0 处理压缩方法 2 的程序如下：

```
Private Sub M2_Proc(X As Long, Y As Long, Data_ll As Long)
    Dim i As Long, x_count As Long, c_char As Byte
    Dim j As Integer, p_count As Long, XX As Long
    x_count = 1: p_count = 0: XX = X
n1:
    Get #1, , c_char: p_count = p_count + 1
    If c_char < 128 Then          ´控制码 c_char 小于 128
      char_count = c_char + 1    ´则打印 c_char +1 个未经压缩的数据
       For j = 1 To char_count
          Get #1, , lin(x_count): p_count = p_count + 1
          x_count = x_count + 1
       Next
    Else
     char_count = 257 – c_char   ´若控制码 c_char 大于 128
       Get #1, , c_char: p_count = p_count + 1 ´则传 257 – c_char 个重复数据
       For j = 1 To char_count
          lin(x_count) = c_char
          x_count = x_count + 1
       Next
    End If
    If p_count < Data_ll Then GoTo n1  ´若数据未传送完转向 n1 继续读数据
    If PrintFlag = 0 Then Display_Pixel XX, Y, x_count ´判断打印标志决定是预览还是打印输
    If PrintFlag = 1 Then M3_MirrorPrint XX, Y, x_count
End Sub
```

4）方法 3　ESC ＊ b3M

方法 3 是通过行与行之间的数据关系进行压缩，也是 PCL

5 最常用的数据压缩方式。基本原理是在数据传送的开始先传送一行完整的数据作为种子行，第二行数据只传送与第一行数据不同的部分，同样第三行数据只传送与第二行数据不同的部分，以此类推。具体方法是对于非种子行其传送数据是由数个子记录块组成，每个子记录块最多为 8 个字节的打印数据。每个子记录块又是这样定义的：首先是一个控制信息字节，该控制字节的 8 个比特又分为两个部分：高三位表示该字记录块中有多少个欲被传送的差异数据 因为它只有三位，故最多表示 8 个数据，0 代表 1 个数据，7 代表 8 个数据 ，而低五位用以说明该子记录块所携带数据在该打印行所处的相对位置。有一点需强调指出的是当 5 位全是 1 时，表示上一子记录块后面不只 31 个未变化数据，这时就需在它后面跟一个表示附加距离的数值，该数值加上 31 即为该子记录块离开上一子记录块的具体位置。紧接在控制后面的是欲传送的数据。若预传送下列两行图形打印数据：

00 11 22 33 44 55 66 77 88 99 AA BB CC

DD EE FF

00 11 11 33 44 55 66 61 62 63 AA BB CC

DD EE AA

第一行数据可作种子行直接传送，而第二行数据与第一行相差不大，可用方法 3 压缩传送。第二行数据用方法 3 传送时的指令为：ESC * b8W 02 11 61 62 63 05 AA。VB 6. 0 读取处理压缩方法 3 程序如下：

```
Private Sub M3_Proc(X As Long, Y As Long, Data_ll As Long)
   Dim i As Long, x_count As Long, c_char As Byte
   Dim j As Integer, p_count As Long, XX As Long
   Dim off As Long, cc As Byte
   x_count = 1: p_count = 0: XX = X
   If Data_ll = 0 Then GoTo b0w   ´重复上一行数据点
m3_n1:
   Get #1, , c_char: p_count = p_count + 1
    cc = 0: If (c_char And & HE0) > 0 Then cc = Int ((c_char And & HE0) / 32) ´数据块字符数 =(c_char and 11100000B) SHR 5
   off = (c_char And & H1F)   ´数据块偏移 =(c_char and 00011111B)
   If off = & H1F Then   ´判断低五位是否为全 1
      Get #1, , c_char: p_count = p_count + 1
      off = off + c_char   ´如果是则加上 31
      If c_char = & HFF Then
         Get #1, , c_char: p_count = p_count + 1
         off = off + c_char
         If c_char = & HFF Then
            Get #1, , c_char: p_count = p_count + 1
            off = off + c_char
         End If
      End If
   End If
   char_count = cc + 1
   x_count = off + x_count
   For j = 1 To char_count
      Get #1, , lin(x_count): p_count = p_count + 1
      x_count = x_count + 1
   Next
   If p_count < Data_ll Then GoTo m3_n1
b0w:
   x_count = P_Width
   For i = P_Width To 1 Step -1
      If lin(i) =0 Then x_count =x_count - 1 Else GoTo disp
   Next
disp:
   If PrintFlag = 0 Then Display_Pixel XX, Y, x_count
   If PrintFlag = 1 Then M3_MirrorPrint XX, Y, x_count
End Sub
```

三、反像处理方法

反像打印处理首先需要将每个字节数据相对页面左右颠倒，然后对字节内部高低位互换，即字节 D7D6D5D4D3D2D1D0 八位，反像处理成 D0D1D2D3D4D5D6D7。相应的 VB 处理程序如下：

```
If Print_win. MirrorPrint. Value Then
  XX = PageWidth - X - S_Width * 8 - 600
  For i = S_Width To 1 Step -1
     If lin(i) >0 Then
        For j = 0 To 7 Step 4
           If (lin(i) And mi(j)) <>0 Then Display_win. PSet (XX * XSca Y * XScale)
           XX = XX + 4
        Next
     Else
        XX = XX + 8
     End If
  Next
End If
```

在打印输出时，为了避免 Windows 打印驱动程序对打印数据的处理，不能采用 VB 的 PrintForm、Print 等打印命令，本程序中采用直接写 LPT1 设备的方法，程序如：

```
Open ″LPT1: ″ For Binary As #2   ´打印数据送打印端口 LPT1
   Put #2, , Char
……
Close(2)
```

系统的映像文件读取、反像打印输出程序如下：

```
Private Sub Mirror_Print(Offset As Long, SaveFlag As Byte)
  Dim SS As String * 6, Char As Byte, den As String * 3
  Dim Ctrl_data As Byte, Wh As String * 4, Data_h8 As Byte
  Dim Data_l8 As Byte, mm As Byte, X As Long
  Dim Y As Long, XX As Long, YY As Long
  Dim FileLenght As, Data_ll As Long
  P_Width = 600: ff = 0
  For i = 1 To P_Width
    lin(i) = 0: lin1(i) = 0
  Next
Open FileName For Binary As #1   ´从映像文件读取数据
Get #1, Offset, Char
If Print_Win. PrintFile. Value = 0 Then   ´若打印按钮被激活
```

```
Open "LPT1:" For Binary As #2'则打印数据送打印端口 LPT1
Put #2, , Char
End If
If Print_Win. PrintFile. Value = 1 Then '若保持到文件复选框被选种
If SaveFlag = 1 Then '则把数据写到文件保持 SaveFile
If Trim(OutPutFile) = "" Then GoTo ll
Open OutPutFile For Binary As #2
Put #2, 1, Char
    Else
      Open OutPutFile For Binary As #2
      FileLenght = LOF(2)
      Put #2, FileLenght + 1, Char
    End If
  End If
  PrintDisplay. Show
  PrintDisplay. MousePointer = 11
  Do While True
    If EOF(1) Then GoTo ll
    Get #1, , Ctrl_data
    If Ctrl_data <> 27 Then
      Put #2, , Ctrl_data
      GoTo nn
    End If
    If Ctrl_data = 27 Then
      Get #1, , Ctrl_data
If Chr$(Ctrl_data) = "%" Then '每页开始介绍标志
        Get #1, , Ctrl_data
        If Chr$(Ctrl_data) = "-" Then
          Get #1, , SS
If EOfOutPutFlag Then Put #2, , Chr$(27) + "%-" + SS
GoTo ll
        End If
      End If
If Chr$(Ctrl_data) = "*" Then '图形数据结束标志
        Get #1, , Ctrl_data
        If Chr$(Ctrl_data) = "r" Then
          Get #1, , Ctrl_data
          Put #2, , Chr$(27) + "*r"
          Put #2, , Ctrl_data
           If Chr$(Ctrl_data) = "B" Then
            For i = 1 To P_Width
              lin(i) = 0
            Next
          End If
          GoTo nn
        End If
If Chr$(Ctrl_data) = "c" Then '读取页宽页高
          Get #1, , Ctrl_data
          If Chr$(Ctrl_data) = "0" Then
            Put #2, , Chr$(27) + "*c0"
            Get #1, , Ctrl_data
            Put #2, , Ctrl_data
            Get #1, , Wh
            PageWidth = Val(Wh)
            Put #2, , Wh '传送页宽
            Get #1, , Ctrl_data
            Put #2, , Ctrl_data
            Get #1, , Wh
            PageHeight = Val(Wh) '传送页高
            Put #2, , Wh
            GoTo nn
          End If
        End If
If Chr$(Ctrl_data) = "p" Then '读取 x, y 的偏移量
          Get #1, , Data_h8
          Y = Data_h8 - 48
          If Y = 0 Then
Put #2, , Chr$(27) + "*p" '偏移量标志"*p"送打印端口
            Put #2, , Data_h8
            GoTo nn
          End If
          For i = 1 To P_Width
            lin(i) = 0
          Next
          For i = 1 To 4
            Get #1, , Data_h8
            If Data_h8 < 58 Then
              Y = Y * 10 + Data_h8 - 48
            Else
              If Chr$(Data_h8) = "X" Then XX = Y
              If Chr$(Data_h8) = "Y" Then
                YY = Y
Put #2, , Chr$(27) + "*p" + Format$(Y) + "Y"'传送偏移量 Y
              End If
              GoTo nn
            End If
          Next
        End If
If Chr$(Ctrl_data) = "b" Then '读取数据压缩方式
          Get #1, , Data_h8
          Get #1, , Data_l8
          If Chr$(Data_l8) = "M" Then
            Proc_Flag = Data_h8 - 48
            Put #2, , Chr$(27) + "*b0M"
            GoTo nn
          End If
          If Chr$(Data_l8) = "W" Then
            Data_ll = Data_h8 - 48
          Else
            Get #1, , mm
            If Chr$(mm) = "W" Then
      Data_ll = (Data_h8 - 48) * 10 + Data_l8 - 48
            Else
      Data_ll = (Data_h8 - 48) * 10 + Data_l8 - 48
              Get #1, , Data_l8
              If Chr$(Data_l8) = "W" Then
                Data_ll = Data_ll * 10 + mm - 48
              End If
            End If
          End If
          '由压缩标志决定数据压缩方式
          If Proc_Flag = 0 Then M0_Proc XX, YY, Data_ll
```

运用JSP与数据库交互

杜文峰 蔡自兴

JSP 是一种很容易学习和使用的在服务器端编译执行的 Web 设计语言，它是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准。在传统的网页 HTML 文件 ＊.htm ＊.html 中加入 Java 程序片段 Scriptlet 和 JSP 标记 tag，构成 JSP 网页 ＊.jsp。Web 服务器在遇到访问 JSP 网页的请求时，首先执行其中的程序片段，然后将执行结果以 HTML 格式返回给客户。程序片段可以操作数据库、重新定向网页以及发送 email 等等，这就是建立动态网站所需要的功能。所有程序操作都在服务器端执行，网络上传送给客户端的仅是得到的结果，对客户浏览器的要求很低。本文将以新建的中南大学信息工程学院智能控制研究所同学录来介绍一下如何利用 JSP 与数据库交互，从而实现动态网页。

由于 JSP 页面的内置脚本语言是基于 Java 编程语言的，而且所有的 JSP 页面都被编译成为 Java Servlet，JSP 页面就具有 Java 技术的所有好处，包括健壮的存储管理和安全性。

作为 Java 平台的一部分，JSP 拥有 Java 编程语言“一次编写，各处运行”的特点。随着越来越多的供应商将 JSP 支持添加到他们的产品中，您可以使用自己所选择的服务器和工具，而更改工具或服务器并不影响当前的应用。一个 JSP 原文件的处理分为两个阶段：一个是 HTTP 的编译时候，一个是请求的处理时间。

1. HTTP 编译的时候，当 JSP 页面是第一次被调用时，如果它还不存在，就会被编译成为一个 Java Servlet 类，并且存储在服务器的内存中。这使得在接下来的对该页面的调用有非常快的响应。 这避免了 CGI - BIN 为每个 HTTP 请求生成一个新的进程的问题，或是在服务器端引用时所引起的运行时语法分析。）HTML 标签和 JSP 标签在这个时候同时被处理了，这之前用户还没有任何的请求被提交。

2. 请求处理时间是当用户在 JSP 页面中提交了一个请求，这时请求由客户端被 request 对象传到了服务器端，接着 JSP 引擎把存放在 request 对象中的数据发到 JSP 页面指定的服务器端的组件（JavaBeans 组件 servlet 或者 enterprise-bean），组件收到这些数据以后，有可能再把这些数据存入到数据库或者其他的地方存放起来，同时，返回一个 response 对象给 JSP 引擎。JSP 引擎再把 response 对象传给 JSP 页面，这时的页面包含了定义好的格式和从服务器端得到的数据。这时 JSP 引擎和 Web 服务器再发送一个整理好的完整的页面给客户，也就是这们在浏览器上看到的结果。

```
            If Proc_Flag = 2 Then M2_Proc XX, YY, Data_ll
            If Proc_Flag = 3 Then M3_Proc XX, YY, Data_ll
            YY = YY + 1
            GoTo nn
          End If
          Put #2, , Chr$(27) + "*"
          Put #2, , Ctrl_data
        GoTo nn
nn: Loop
ll: Close (1): Close (2)
  PrintDisplay. MousePointer = 0
  PrintDisplay. Hide
End Sub
```

四、结论

本反像打印程序是针对 HP6L 激光打印机的 PCL 5 格式数据进行处理，而实现 Word 文档的反像打印输出时，输出的稿样能保持 Word 的原有风格，无需添加任何硬件，可直接实现反像打印输出，也可正像输出。此外，用户还可根据需要设置打印参数：如打印浓度、图形设置、打印质量、是否启用分辨率增强等，程序界面友好、交互性强、操作简单，如图 4 所示。

图 4 程序运行界面

由于本程序是针对 HP6L 激光打印机编写的反像输出程序，若添加命令集转换功能也可支持其他型号的激光打印设备。程序对处理后的反像打印数据未进行压缩传送，所以打印速度比 Word 打印慢，如对反像数据压缩后再输出，可提高其打印速度。本程序在 Windows 98 下用 Visual Basic 6.0 编写，因源程序较长，需要的读者可与笔者联系。

（收稿日期：2000 年 10 月 25 日）

与数据库的连接对于动态网站来说是最为重要的部分，Java 中连接数据库的技术是 JDBC Java Database Connectivity 。很多数据库系统带有 JDBC 驱动程序，Java 程序就通过 JDBC 驱动程序与数据库相连，执行查询、提取数据等等操作。Sun 公司还开发了 JDBC - ODBC bridge，用此技术 Java 程序就可以访问带有 ODBC 驱动程序的数据库，目前大多数数据库系统都带有 ODBC 驱动程序，所以 Java 程序能访问诸如 Oracle、Sybase、MS SQL Server 和 MS Access 等数据库。对于一个同学录来说，当有同学登录，或查询，修改同学录里的内容时，与数据库的交互是不可避免的，在这里为了示例方便，我使用的数据库是 MS Access。

在数据库里建了一个名为 personinfo. mdb 的数据库，并在 Windows 里的 ODBC 数据源里把它设为系统 DSN，命名为 personinfo，登录名为 dwf，登录密码为 lmh，并让它指向 personinfo. mdb。在本同学录里，数据库所设的字段有 entryname （文字型，长度 50）、 password （文字型，长度 50）、 birth_year （数字型，长度 4 、 birth_month 数字型，长度 2 、 birth_day （数字型，长度 2 、 e_mail （文字型，长度 50）。在本同学录里，为了实现相应的功能，制作了一个名为 beanfordenlu. java 的 JavaBean 文件。它使用了 JDBC - ODBC 桥与数据库连接。

下面是这个 Bean 的代码清单：

```
//导入必要的 package
import java. sql. *;
public class beanfordenlu{ // 此处 beanfordenglu 为 JavaBean 类的名称
//声明变量
public Connection Conn;
public Statement Stmt;
public ResultSet Rs;
private String SQL;
//构造函数
public void beanfordenlu(){
}
//得到一个与数据库连接的 Statement
public void getStmt(){
String SEVER = "jdbc: odbc: personinfo";
String User = "dwf";
String Password = "lmh";
try{
Class. forName( "sun. jdbc. odbc. JdbcOdbcDriver"); //先得到一个 DriverManage
//得到一个与数据库连接的 Connection
Conn = DriverManager. getConnection(SEVER, User, Password); Conn. setAutoCommit(false);
//通过 Conn 的到一个 Statement
Stmt = Conn. createStatement();
}
catch(Exception e){
System. out. println("can't get a statement");
}
}
//设置 SQL 语句
public void setinsertSQL(String SQL){
this. SQL = SQL;
}
//由 name 和 password 来产生一条 SQL 语句, 用于确定用户的身份
public void setchangeSQL(String name, String password){
this. SQL = "select * from personinfo where entryname = ''
+name + ''andpassword = '' +password + ''";
}
//确定查询结果是否为空
public boolean isEmpty() throws SQLException{
Rs = Stmt. executeQuery(SQL);
if(Rs. next())
return false;
return true;
}
//返回一个 ResultSet
public ResultSet getRs(){
return Rs;
}
//执行 SQL 语句
public void doinsert(){
try{
Stmt. executeUpdate(SQL);
Conn. commit();
};
catch(Exception e){
System. out. println("insert err")
}
}
//从数据库里得到数据
public String getdate(String datename) throws SQLException
{
String date = Rs. getString(datename). toString();
if(date = =null) date = "";
return date;
}
//关闭数据库
public void dbclose(){
try{
if(Rs! =null)
Rs. close();
if(Conn! =null)
Conn. close();
}
catch(SQLException e){
System. out. println("close err");
System. out. println(e. getMessage());
}
} }
```

JSP 网页吸引人的地方之一就是能结合 JavaBean 技术来扩充网页中程序的功能。JavaBean 是一种 Java 类 class ，通过封装属性和方法使其成为具有某种功能或者处理某个业务的对象。在 JavaBean 中可以用 Java 语言来编写一些功能强大的方法，通过这些方法来实现与数据库的交互，当然你也可以把一些要在 JSP 页面里实现的功能放在 JavaBean 里，从而简化 JSP

脚本的编写。而且，由于 JavaBean 的通用性很强，你可以在多个页面里使用同一个 JavaBean 来实现一些相同的功能。比起 ASP 里使用 COM 技术来说，JavaBean 编写简单，而且可以实现 COM 同样的功能。

在 JSP 页面里可以通过调用 JavaBean 实现与数据库的交互，通过 JavaBean 把从页面里得到的数据存到数据库里，同时从数据库里提取数据。在 JSP 里使用 JavaBean 的脚本为：

```
<jsp: useBean id = ″mytongxuelu″ scope = ″page″ class = ″beanfordenlu″ >
Java 代码
</jsp: useBean>
```

通过在这里使用标记 <jsp useBean>把 JavaBean 引到 JSP 页面里，有效范围为本页。Beanfordenlu. class 在本页里的标志为 id，即 mytongxuelu，然后就可以在本页里通过 mytongxuelu. getStmt 等方法来调用 JavaBean 里的方法。

产生 SQL 语句内容页的代码清单如下： 对于一些重复的部分，笔者进行了适当的删减）

```
<html>
<head>
<title>智能所同学录 neirong </title>
</head>
<body>
<%@ page contentType = ″text/html; charset = gb2312″%>
<%@ page language = ″java″ %>
<%@ page import = ″beanfordenlu″ %>
<! – –调用 JavaBean – – >
<jsp: useBean id = ″ mydenglu″ scope = ″ page″ class = ″beanfordenlu″>
<! – –使用 Session 得到第一页里的变量 – – >
<% String entryname = (String) session. getValue( ″h_entryname″);
<! – –使用 request 得到第二页里用户所填入的数据 – – >
String password = request. getParameter(″h_password1″);
String name = request. getParameter(″h_name″);
String e_mail = request. getParameter(″h_e_mail″);
String sqlforword = ″insert into personinfo(″;
String sqllast = ″values(″;
%>
<font color = ″#0000FF″ size = ″5″ face = ″华文彩云 ″>您的档案: </font>
<p> </p>
<table border = ″ 1″ width = ″ 100% ″ bordercolorlight = ″ #0000FF″ bordercolor = ″#0000FF″>
<tr>
<td width = ″50%″>& nbsp; & nbsp; & nbsp; & nbsp; 项& nbsp; 目& nbsp; 名& nbsp; 称 </td>
<td width = ″50%″ colspan = ″4″>& nbsp; & nbsp;
& nbsp; & nbsp; & nbsp; & nbsp; & nbsp; & nbsp; & nbsp; & nbsp; & nbsp; & nbsp;
内& nbsp; & nbsp; & nbsp; & nbsp; 容 </td>
</tr> <! – –产生 SQL 语句 – – >
<%
if(entryname! = ″″){
entryname = new String(entryname. getBytes( ″8859_1″));
//处理 JSP 页面里的中文问题
sqlforword = sqlforword + ″entryname, ″;
sqllast = sqllast + ″′″ + entryname + ″′, ″; %>
<tr>
<td width = ′50%′>& nbsp; & nbsp; & nbsp; & nbsp; 登录名称: </td>
<td width = ′50%′ colspan = ″4″><% = entryname %></td>
</tr>
<% }
if(name! = ″″){
name = new String(name. getBytes(″8859_1″)); %>
<tr>
<td width = ′50%′>& nbsp; & nbsp; & nbsp; & nbsp; 您的姓名: </td>
<td width = ′50%′ colspan = ″4″><% = name%></td>
</tr>
<% }
if(password! = ″″){
sqlforword = sqlforword + ″password, ″;
sqllast = sqllast + ″′″ + password + ″′, ″;
};
<td width = ′28%′> </td>
</tr>
<%
if(e_mail! = ″″){
sqlforword = sqlforword + ″e_mail, ″
sqllast = sqllast + ″′″ + e_mail + ″′, ″; %>
<tr>
<td width = ′50%′>& nbsp; & nbsp; & nbsp; & nbsp; 您的e – mail: </td>
<td width = ′50%′ colspan = ′4′><% = e_mail%></td>
</tr>
<% }
if(name! = ″″){
name = new String(name. getBytes(″8859_1″));
sqlforword = sqlforword + ″name″;
sqllast = sqllast + ″′″ + name + ″′″;
}
sqlforword = sqlforword + ″) ″; sqllast = sqllast + ″)″;
String SQL = sqlforword + sqllast;
%>
</table>
<p> & nbsp; & nbsp; <font size = ″4″> & nbsp; <font color = ″#0000FF″ face = ″华文行楷 ″>祝贺您已经加入信息工程学院智能控制研究所的同学录 </font></font>
</p> <p>
<font color = ″ # 0000FF″ face = ″ 仿 宋 _GB2312″><a href = ″AIPAGE1. htm″>返回同学录首页 </a></font>
</p> <p>
</p>
<! – –JavaBean 的方法调用 – – >
<%
mydenglu. getStmt();
```

下转第 17 页

Visual C++ 中几种情况下实现打印的方法

朱立新

摘　要　本文介绍了在几种情况下，Visual C++ 语言中实现打印的方法，简要叙述了其实现过程。

关键词　打印，设备环境　方法

在一般的应用程序或者商业软件中　实现打印是程序中经常要用到的功能。本人在应用 Visual C++ 语言的工作过程中　总结出了一些经验技巧 希望对需要在自己的应用程序中实现打印功能的编程者有所启发。

（）一般情况下，在 MFC 中，打印工作是由应用程序类和视图类共同完成，视图和应用程序框架实现各自的功能。它的主要过程有以下几步：

A　框架显示打印对话框，为打印机创造设备环境，并调用 CDC 类中的 StartDoc 函数。

B　对于文档的每一页，应用程序框架调用 CDC 类中的 StartPage 函数，通知视类打印，并用 EndPage 结束该页的打印。

C　若开始打印某一页之前，更改打印机的模式，视类调用 CDC 类中的 Escape 函数。

D　文档打印完毕后，应用程序框架调用 CDC 类中的 EndDoc 函数结束本次打印工作。

流程图简要表示如下：

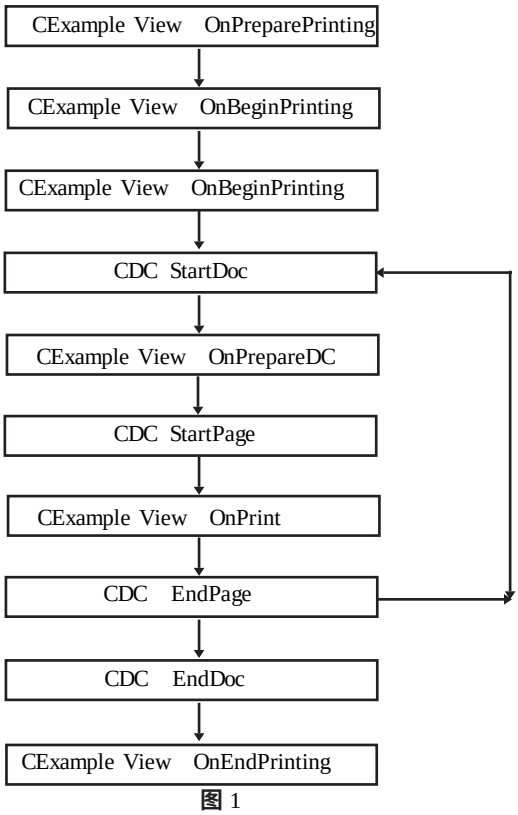


图 1

在打印的过程中，自始至终都用到了 CPrintInfo 对象。CPrintInfo 对象包括打印作业的有关信息，如打印份数，打印预览，打印页码范围等选项。同时注意　要想实现自己的打印任务 可以重载 CView 类中的 OnPrint 函数。

在由 ClassWizard 生成的视图和应用程序框架实现文件中，实现打印较简单，只要在视类中的虚函数中填充代码即可，无需其它的准备工作。

在这里以打印一个简单的矩形为例说明打印的具体步骤。

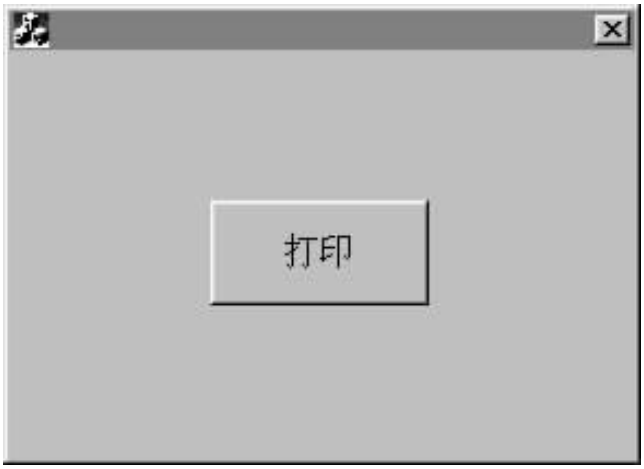
```
BOOL CMyPrintView: : OnPreparePrinting(CPrintInfo * pInfo)
{
  return DoPreparePrinting(pInfo);
}
void CMyPrintView: : OnPrint(CDC * pDC, CPrintInfo * pInfo)
  {
        //添加实现打印的代码
       int x_Scale  = pDC ->GetDeviceCaps(HORZRES);
       int y_Scale  = pDC ->GetDeviceCaps(VERTRES);
       pDC ->Rectangle(0, 0, x_Scale, y_Scale);
       CView: : OnPrint(pDC,  pInfo);
}
```

查看在 OnPreparePrinting CPrintInfo * pInfo 中调用缺省的 CView 类中的 DoPreparePrinting CPrintInfo * pInfo 源代码就会发现，DoPreparePrinting CPrintInfo * pInfo 将 CPrintInfo 类中的成员如 m_pPD 中的各项参数，m_bPreview，m_bDirect 等都进行了设置。在其中已经获取了当前打印机的设备环境句柄。Classwizard 产生的标准打印命令 ON_COMMAND ID_FILE_PRINT CView　OnFilePrint 中，OnFilePrint 获取当前打印机的所有缺省信息，搜集要打印文档的信息，如文件的标题，长度等。调用 CDC 类中的 StartDoc，StartPage 开始打印，EndPage EndDoc 结束打印，AbortDoc　放弃打印。

（）在很多情况下，程序中不具备视和框架之间的交互，必须直接获取打印机的设备环境句柄，这时可以设置 CPrintDialog 类中的构造函数的参数，获取打印机的设备环境句柄。利用这个句柄，转化为指针，再按上述流程图的步骤，实现打印任务。

例如，如下图所示，在一对话框中，鼠标左击一按钮，弹出打印对话框，按取消返回，按确定开始打印，在打印纸上打印出一个矩形。实现这一过程，可编码如下：

```
void CMyPrintDlg: : OnPrint()
{
```




```
    CDC * pDC;
    CPrintDialog myprintdlg(false);
     if (myprintdlg.DoModal() == IDCANCEL) //弹出打印对话框
        return;
     pDC = CDC::FromHandle(myprintdlg.GetPrinterDC());
//获取当前打印机设备环境指针
MyPrintTask(pDC); //在 MyPrintTask 函数中实现打印任务
    pDC->Detach();
}
void CMyPrintTestDlg::MyPrintTask(CDC * pDC)
{
//具体的打印任务
   int x_Scale, y_Scale;
   DOCINFO m_docinfo = {sizeof(DOCINFO), " PRINT", NULL};
  x_Scale = pDC->GetDeviceCaps(HORZRES);
  y_Scale = pDC->GetDeviceCaps(VERTRES);
  if( pDC->StartDoc(& m_docinfo)>0)
{
      if(pDC->StartPage()>0)
      {
            pDC->Rectangle(0, 0, x_Scale, y_Scale);
           if(pDC->EndPage()>0)
                pDC->EndDoc();
           else return;
      }
}
else return;
}
```

CPrintDialog 类的构造函数中，将参数 bPrintSetupOnly 设置为 FALSE 后 实质上标志 PD_RETURNDC 被自动调用，调用 DoModal GetDefaults GetPrinterDC 获取打印机设备环境句柄。CDC 类的静态函数 FromHandle 转化为指针。将指针作为参数传递给 MyPrintTask CDC * pDC 函数。在 MyPrintTask CDC * pDC 函数中，就要自己调用 StartDoc StartPage EndPage EndDoc 等打印的函数，从而实现打印任务。

§）在特殊情况下，程序中既不具备视和框架之间的交互，又不需打印对话框，直接实现打印任务。要想直接获取打印机设备环境指针，实现打印操作，可采取以下方案。

a 利用 EnumPrinters 搜索打印机和 SDK 中的 CreateDC 函数。

HDC CreateDC LPCTSTR lpszDriver LPCTSTR lpszDevice LPCTSTR lpszOutput CONST DEVMODE * lpInitData 。第一个参数 LPCTSTR lpszDriver，lpszDriver 在早期的 Windows 版本中字符串包含设备驱动程序的文件名。在现在的 Windows95 中一般可缺省为 NULL。第二个参数 LPCTSTR lpszDevice lpszDevice 字符串包含打印机设备名称，如“HP LaserJet 4L”，第三个参数 LPCTSTR lpszOutput，在 Win32 程序中，为了与早期的 Windows 版本相兼容，必须设置成 NULL。第四个参数 CONST DEVMODE。DEVMODE 包含打印机初始化信息的具体数据。若使用缺省的初始化值，可设置成 NULL。

要填充这四个参数，就要搜寻当前使用的打印机的有关信息。EnumPrinters 可用来获取打印机的信息。它共有 7 个参数。

BOOL EnumPrinters DWORD Flags LPTSTR Name DWORD Level LPBYTE pPrinterEnum DWORD cbBuf LPDWORD pcbNeeded LPDWORD pcReturned

Flags 参数用来指定使用打印机的方式。如本地打印机，网络打印机等。

Name 参数 Flags 和 Level 的组合形式不同，有多种数值。这里不在一一列举。

Level 参数指明由 pPrinterEnum 指针所指的结构类型。pPrinterEnum 指针所指的结构有四种形式。有 PRINTER_INFO_1 PRINTER_INFO_2 PRINTER_INFO_4 and PRINTER_INFO_5 四种结构。对应的 Level 值 1，2，4，5。

pPrinterEnum 是指向存储打印机信息结构缓冲区的指针。cbBuf 参数表明 pPrinterEnum 指向的缓冲区的大小。

pcbNeeded 该功能实现后，被拷贝的字节数用一个变量来表示。pcbNeeded 是指向这个变量的指针。

PcReturned 是指向存储打印机信息结构个数的变量的指针。

假设我们仍然采用上图的界面，单击鼠标左键，使用缺省打印机，不出现打印对话框，直接打印。可以如下编码：

```
void CMyPrintDlg::OnPrint()
  {
   CDC * pDC;
     pDC = CDC::FromHandle(GetPrinterDC());      //获取当前打印机设备环境指针
   ASSERT(pDC! =NULL);
MyPrintTask(pDC); //在 MyPrintTask 函数中实现打印任务
   pDC->Detach();
  }
HDC CMyPrintDlg::GetPrinterDC()
```

```
{
PRINTER_INFO_5 info[3];
DWORD m_Needed, m_Returned;
if (EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5,
(LPBYTE)info, sizeof(info), & m_Needed, & m_Returned))
return CreateDC(NULL, info[0] .pPrinterName, NULL,
NULL);
else
return 0;
}
void CMyPrintTestDlg: : MyPrintTask(CDC * pDC)
{
//具体的打印任务, 同上.
}
```

在此要注意一点，在该实现文件中，要加上#include "winspool. h"。

b 采用搜索 Windows 配置文件的方法获取打印机的信息和 CreateDC 函数。

例如，假设 WIN. INI 文件中有以下几行：

```
[windows]
load =
run =
NullPort = None
device = HP LaserJet 4L, HPPCL5MS, LPT1:
```

device 一行中，HP LaserJet 4L 是设备名，HPPCL5MS 是设备驱动程序名，LPT1 端口名。

在这里可以利用 GetProfileString 函数将这三段提取出来。GetProfileString 函数的各个参数含义如下：

DWORD GetProfileString LPCTSTR lpAppName LPCTSTR lpKeyName LPCTSTR lpDefault LPTSTR lpReturnedString DWORD nSize

lpAppName 包含段名的字符串指针，段名即如以上所示的[windows]。lpKeyName 包含键名的字符串指针，键名即如以上所示的 load，run，device 等。

lpDefault 指向缺省字符串的指针。lpReturnedString 指向存储字符串缓冲区的指针。

nSize 由 lpReturnedString 指向的缓冲区的大小。

函数 GetPrinterDC 中代码改写如下：

```
HDC CMyPrintDlg: : GetPrinterDC()
{
 char printinfo[100];
 char * p_DeviceName, * p_DriveName, * p_OutPut;
 char * sectinon = "windows";
 char * key = "device";
 char * defstr;
 GetProfileString(sectinon, key, defstr, printinfo, 100);
 p_DeviceName = strtok(printinfo, ", ");
 p_DriveName = strtok(NULL, ", ");
 p_OutPut = strtok(NULL, ", ");
 if((p_DeviceName! = NULL) & & (p_DriveName! = NULL)
& & (p_OutPut! = NULL))
return CreateDC(p_DriveName, p_DeviceName, p_OutPut,
NULL);
 else
return 0;
}
```

其余的打印函数无需更改。

小结

从以上打印的各种情况不难总结出这样的结论，对打印机的操作最关键的部分是要根据程序的具体要求，想方设法获取其设备环境句柄。然后按照图 1 的流程，实现具体打印任务。读者可根据不同的程序环境，按上述方法实现自己想要的打印任务。

以上所有例子在 Visual C + + 5. 0 中全都调试通过。

参考文献

David J . Kruglinski 著 . Inside Visual C + + 5. 0

收稿日期：2000 年 11 月 9 日）

上接第 14 页

```
 mydenglu. setinsertSQL(SQL);
 mydenglu. doinsert();
 mydenglu. dbclose();
%>
<! - - 停止 JavaBean 的调用 - - >
</jsp: useBean>
</body>
```

当执行了这段代码后，JSP 页面就把用户输入的数据通过 JDBC - ODBC 桥存入到 Access 数据库里。我们登录时就能使用此次登录的用户进行登录。对于登录的用户，我们可以使用 JavaBean 里的 SetchangeSQl String name String password 方法来产生查询 SQL 语句，并调 JavaBean 里的 isEmpty 方法来确定用户的身份，从而引导不同的操作。

事实证明， Java Servlet 是一种开发 Web 应用的理想构架。 JSP 以 Servlet 技术为基础，又在许多方面作了改进。JSP 页面看起来象普通 HTML 页面，但它允许嵌入执行代码，在这一点上，它和 ASP 技术非常相似。利用跨平台运行的 JavaBean 组件， JSP 为分离处理逻辑与显示样式提供了卓越的解决方案。 JSP 必将成为 ASP 技术的有力竞争者。

收稿日期：2000 年 10 月 25 日）

理解 Application 对象与 Session 对象

雷劲 雷超

我们知道，HTTP 协议是无状态协议，Web 服务器把每一个来自浏览器的请求视为一个独立的请求，Web 服务器不保留以前请求的任何信息。所以当用户从应用程序中的一页转到另一页时，要维护整个访问或会话期间的用户信息便成为了一个难题。

Active Server Pages 中 Application 对象和 Session 对象的引入一定程度上解决了 HTTP 协议无状态的缺陷，同时也成为了 ASP 区别于传统 CGI 的显著特征之一。Application 对象用于在同一应用程序的所有用户会话中共享信息，一个用户改变了 Application 对象中变量的值，会影响所有用户对该变量的引用；而 Session 对象则用于在一个用户会话中共享信息，一个用户改变了 Session 对象中变量的值，不会影响其它用户对该变量的引用。在这种意义上 Application 对象就像是一个类的静态成员变量 i，而 Session 对象则该类的非静态成员变量 j。所有该类的实例 对象 共享同一个 i 但却各自保留着 j 的不同的副本。

下面的例子将说明 Application 对象和 Session 对象的作用：设有 A、B、C 三台计算机，A 为服务器。以下是 A 上的几个 ASP 脚本文件。

例一、Application 对象的使用

```
SetValue.asp
<%
    Application.Lock()
    Application("Value") = 0
    Application.Unlock()
    Response.Write "The Value has been setted to "&
CStr(Application("Value"))
%>
ShowValue.asp
<%
    Response.Write "The Value is " & CStr(Application("
Value"))
%>
ChangeValue.asp
<%
    Application.Lock()
    Application("Value") = Application("Value") + 1
    Application.Unlock()
    Response.Write "The Value has been changed to "&
CStr(Application("Value"))
%>
```

首先在 B 上浏览 SetValue.asp，结果为

The Value has been setted to 0

接着在 C 上浏览 ShowValue.asp，结果为：

The Value is 0

接着在 B 上浏览 ChangeValue.asp，结果为：

The Value has been changed to 1

最后在 C 上浏览 ShowValue.asp，结果为：

The Value is 1

例二、Session 对象的使用

```
SetValue.asp
<%
  Session("Value") = 0
  Response.Write "The Value is setted to " & CStr(Session
("Value"))
%>
ShowValue.asp
<%
Response.Write "The Value is " & CStr(Session("Value"))
%>
ChangeValue.asp
<%
  Session("Value") = Session("Value") + 1
  Response.Write "The Value has been changed to " &
CStr(Session("Value"))
%>
```

首先在 B 上浏览 SetValue.asp，结果为

The Value has been setted to 0

接着在 C 上浏览 SetValue.asp，结果为：

The Value has been setted to 0

接着在 B 上浏览 ChangeValue.asp，结果为：

The Value has been changed to 1

最后在 C 上浏览 ShowValue.asp，结果为：

The Value is 0

以上的例子说明了 Application 对象和 Session 对象的用法，看上去使用还很方便，也很有效。但是在使用 Session 对象却存在着一个问题，以至于在很多的情况下我们都不得不放弃使用它，这正是本文要特别指出的。

我们在计算机 B 上按下面这个步骤来运行例二：

首先打开 IE，浏览 SetValue.asp 结果为：

The Value has been setted to 0

接着我们再浏览 ChangeValue.asp，注意不是再双击打开一个 IE，而是在已有的 IE 窗口中选择 “文件/新建/窗口”，并在新建的窗口中浏览 ChangeValue.asp，结果为：

The Value has been changed to 1

最后，我们在最初打开的那个窗口中浏览 ShowValue.asp，

利用 VB 实现基于 Office2000 的报告自动生成器

郑平泰

摘 要 本文介绍了利用 VB6.0 对 Office2000 进行二次开发的方法，并结合一个报告自动生成器的例子进行了具体阐述。

关键词 VB6.0，Office2000，二次开发

一、 引言

随着信息技术的不断发展和计算机的迅速普及，越来越多的计算机用户使用各种应用软件和编程工具来完成日常工作。中文版 Office2000 正式发布以来，得到了广大计算机用户的普遍认同和好评，用户可以利用它的强大功能来完成几乎所有的日常工作。

对每一个公司而言，每天根据其生产销售情况都有大量的报告、表单需要编制，而在大多数情况下仅仅是其中的数据不同而已。如此周而复始的繁杂劳动，如果依靠人工通过计算机编辑表格、录入数据，不仅劳动量巨大，而且容易出错。能否对 Office2000 进行二次开发，根据数据自动生成表单和报告呢？回答是肯定的。Office2000 中集成的 VBA 就提供了进行二次开发的手段。

Visual Basic for Application （VBA）是由 Visual Basic 发展而来的，它用于 Microsoft Office 的应用编程。VBA6.0 是最新版本的 VBA，它适用于 Office2000 的开发，它增加了适合 Office2000 的上百个新特性，其中包括支持新的文档类型、新的万维网选项设置、HTML 出版等。

二、 Office2000 的对象模型

Office2000 软件中使用的所有功能和在屏幕上所创建的所有可视内容，在 VBA 中都可由相应的对象来代表，这些对象是可编程的，因此我们可以开发出操纵这些对象及其属性的应用程序。

操纵对象的方式有以下三种：

- 改变对象属性；
- 激活与对象有关的方法，使对象完成某一任务；
- 定义一个过程，一旦某一特定事件发生在该对象时，此过程运行。

2.1 创建新应用程序对象

使用 CreateObject 函数可以创建新对象：

```
Dim WordApp As Object
Set WordApp = CreateObject(″Word.Application″)
WordApp.Visible = True
```

使用 New 关键字可以创建新对象

```
Dim WordApp As New Word.Application
WordApp.Visible = True
```

2.2 引用应用程序对象

对象被创建以后，可以使用 GetObject 函数来引用已创建的对象实例：

```
Dim WordApp As Word.Application
On Error Resume Next
Set WordApp = GetObject(″Word.Application″)
If WordApp Is Nothing Then
Set WordApp =
CreateObject(″Word.Application″)
EndIf
WordApp.Visible = True
```

2.3 释放对象变量

将内存中不再需要的应用程序对象和对象变量释放出来，使用 Nothing 关键字：

结果为：

The Value is 1

也就是说在后一窗口中对 Session “Value” 的改变影响了前一窗口中 Session “Value” 的值。按照微软的说法，Session 对象用以在一个会话期间共享信息。这说明了什么呢？举个例子，如果一个网页聊天室用 Session “ID” 来保存聊天者的昵称，而一个用户想要以两个或以上不同的身份同时访问聊天页面时，就会出现问题：用户以不同身份的讲话都将会被认为是出至用户最后一个进入该页面的身份。因为 Session “ID” 的值会在每次用户进入该页面中被替换。

解决的方法有两个：一是在进入聊天页面时检查 Session “ID” 是否已设有值，若有则给出警告，防止用户在不同的窗口中同时访问该页面；二是不使用 Session 对象，用其它的方法来保存用户的昵称，如在 URL 后跟上 “ID = Carling”。

具体使用哪种方案与你的程序有关，希望这篇文章会给你一点帮助。

（收稿日期：2000 年 10 月 24 日）

Set WordApp = Nothing

三、报告自动生成器编制方法

本报告自动生成器实现以下功能：从外面读入数据文件（可以读入多个数据文件，本程序以读入一个数据文件为例），在报告中自动生成文本、表格和曲线。具体编制方法如下：

1. 启动 VB6.0

在控件箱中添加 Microsoft Common Dialog Control 6.0 控件和 ProgressBar Control 控件。

2. 窗体及控件属性

编制如下图所示的界面，窗体及控件的属性见下表。

图 窗体界面

表 控件属性表

控件类型	控件名称	主要属性
Form	Form1	Caption = **"欢迎使用"** BorderStyle = 3
Label	Label1	Caption = **"输入数据文件："**
	Label2	Caption = **"输出文档文件："**
	Label3	Caption = **"报告自动生成器"** Font = **"黑体,粗体,斜体,小五号"**
	Lbl	Caption = ""
CommandButton	CmdData	Caption = ""
	CmdDest	Caption = ""
	CmdOK	Caption = "& OK"
	CmdFinish	Caption = "& Finish"
	CmdCancel	Caption = "& Cancel"
ProgressBar	ProgBar	Min = 0,Max = 100
CommonDialog	ComDialog1	**输入数据文件**
	ComDialog2	**输出文档文件**

3. 选择 Reference

在工程的 Reference 中，添加如下几项： Microsoft Office Object Library，Microsoft Word Object Library，Microsoft Excel Object Library。

4. 添加代码

在窗体中添加如下代码：

```
Dim bDataFile As Boolean
Dim bDestFile As Boolean
Dim bWordStatus As Boolean
Dim bExcelStatus As Boolean
Dim DataFile As String
Dim DestFile As String
Private Sub CmdData_Click()
With ComDialog1
    .DefaultExt = "*.DAT;*.TXT"
    .DialogTitle = "打开数据文件"
    .Filter = "数据文件
(*.DAT;*.TXT)|*.DAT;*.TXT"
    .Flags = &H280000
    .ShowOpen
    If Not .FileName = "" Then
    CmdData.Caption = .FileName
    DataFile = .FileName
    bDataFile = True
    Call CmdOKStatus
    End If
End With
End Sub
Private Sub CmdDest_Click()
Dim msgPrompt As String, msgTitle As
String
Dim msgButtons As Integer, msgResults As
Integer
With ComDialog2
    .DefaultExt = "*.DOC"
    .DialogTitle = "保存目标文件"
    .Filter = "文档文件(*.DOC)|*.DOC"
    .Flags = &H280000
    .ShowSave
    If Not .FileName = "" Then
      msgPrompt = "文件" & .FileName &
"已经存在.是否替换原有文件?"
      msgButtons = vbYesNoCancel + vbExclamation +
vbDefaultButton2
      msgTitle = "报告生成器"
      If Not Dir(.FileName) <> "" Then
        CmdDest.Caption = .FileName
        DestFile = .FileName
        bDestFile = True
        Call CmdOKStatus
      Else
msgResults = MsgBox(msgPrompt, msgButtons, msgTitle)
        If msgResults = vbYes Then
          CmdDest.Caption = .FileName
          DestFile = .FileName
          bDestFile = True
          Call CmdOKStatus
          Form1.Show
        End If
```

```
        End If
    End If
End With
End Sub
Private Sub CmdOKStatus()
If bDataFile = True And bDestFile = True
Then
CmdOK. Enabled = True
End If
End Sub
Private Sub CmdCancel_Click()
Unload Me
End Sub
Private Sub CmdOK_Click()
CmdOK. Enabled = False
Lbl. Caption = "正在进行文档生成,请等待..."
Dim WordApp As Word. Application
Dim WordDoc As Word. Document
If WordApp Is Nothing Then
    Set WordApp = CreateObject("word. application")
    WordApp. Visible = False
    bWordStatus = False
Else
    Set WordApp = GetObject(, "word. application")
    bWordStatus = True
End If
Set WordDoc = WordApp. Documents. Add
/////////以下是文本生成的代码/////////////
WordDoc. Paragraphs(WordDoc. Paragraphs. Count)
. Alignment = wdAlignParagraphCenter
WordDoc. Paragraphs(WordDoc. Paragraphs. Count)
. Range. Text = "第一章 视频音频编辑的知识基础"
WordDoc. Paragraphs. Add
WordDoc. Paragraphs(WordDoc. Paragraphs. Count)
. Alignment = wdAlignParagraphLeft
WordDoc. Paragraphs(WordDoc. Paragraphs. Count)
. Range. Text = "自 90 年代中期以来,随着计算机硬件环境和
视音频编辑软件的完善和发展,一种新型的演播室出现了,录象
节目的后期制作进入到一个崭新的阶段.其特点是采用了在线
和非在线的非线性编辑,视频编辑的观念发生了重大的变化,压
缩技术的实用化使脱机编辑和非线性编辑称为可能."
Set Range01 = WordDoc. Range(Start: =WordDoc. Paragra-
phs(1). Range. Start, End := WordDoc. Paragraphs (1)
. Range. End)
    Range01. Style = WordDoc. Styles("标题 1")
    Range01. Bold = True
    Range01. InsertParagraphAfter
 WordDoc. Paragraphs. Add
WordDoc. Paragraphs. Add
'/////////文本生成的代码结束/////////
ProgBar. Value = 30
'/////////以下是表格生成的代码/////////
Dim fn As Integer
Dim j As Integer
Dim d As Data01
fn = FreeFile
Open DataFile For Input As #fn
Dim TableRange As Word. Range
Set TableRange = WordDoc. Range(
Start: =WordDoc. Paragraphs(5). Range.
Start, End: =WordDoc. Paragraphs(5). Range.
Start)
WordDoc. Tables. Add Range: =TableRange,
numrows: =1, numcolumns: =2
With WordDoc. Tables(1)
    With . Rows. Last
        . Cells(1). Range. Text = "时间(ms)"
        . Cells(2). Range. Text = "压力
(MPa)"
    End With
End With
j = 1
Do While Not EOF(fn)
     Input #fn, d. dTime, d. dPressure
    With WordDoc. Tables(1)
    . Rows. Add
        With . Rows. Last
          . Cells(1). Range. Text = d. dTime
          . Cells(2). Range. Text = d. dPressure
            j = j + 1
        End With
    End With
Loop
Close #fn
'/////////表格生成的代码结束/////////
ProgBar. Value = 70
'/////////以下是图形曲线生成的代码/////////
Dim ExcelApp As Excel. Application
Dim ExcelBook As Excel. Workbook
Dim ExcelSheet As Excel. Worksheet
Dim ExcelChart As Excel. Chart
If ExcelApp Is Nothing Then
    Set ExcelApp = CreateObject("excel. application")
    ExcelApp. Visible = False
    bExcelStatus = False
Else
    Set ExcelApp = GetObject(, "word. application")
    bExcelStatus = True
End If
Set ExcelBook = ExcelApp. Workbooks. Add
Set ExcelSheet = ExcelBook. Worksheets(1)
With ExcelSheet. Rows(1)
. Cells(, 1) = ""
. Cells(, 2) = "压力(Mpa)"
End With
Open DataFile For Input As #fn
j = 1
Do While Not EOF(fn)
```

```
    With ExcelSheet. Rows(j + 1)
        Input #fn, d. dTime, d. dPressure
        . Cells(, 1) = d. dTime
        . Cells(, 2) = d. dPressure
        j = j + 1
    End With
Loop
Close #fn
Set ExcelChart = ExcelBook. Charts. Add
With ExcelChart
    . ChartType = xlLine
   . SetSourceData Source: =ExcelSheet. Range("A1: B11"),
PlotBy: =xlColumns
. Location Where: =xlLocationAsNewSheet, Name: ="压力
图"
    With . Axes(xlCategory, xlPrimary)
        . HasTitle = True
        . AxisTitle. Characters. Text = "时间(ms)"
        . AxisTitle. AutoScaleFont = False
    End With
    With . Axes(xlValue, xlPrimary)
        . HasTitle = True
        . AxisTitle. Characters. Text = "压力(MPa)"
        . AxisTitle. AutoScaleFont = False
    End With
End With
ExcelChart. PlotArea. Interior. ColorIndex = 2
ExcelChart. Axes(xlValue). TickLabels. AutoS
caleFont = False
With ExcelChart. Axes(xlCategory). TickLabels
    . Offset = 100
    . Orientation = xlHorizontal
    . AutoScaleFont = False
End With
ExcelChart. ChartArea. Copy
If bExcelStatus = False Then
Set ExcelChart = Nothing
Set ExcelSheet = Nothing
Set ExcelBook = Nothing
ExcelApp. DisplayAlerts = False
ExcelApp. Quit
Set ExcelApp = Nothing
End If
'//////////图形曲线生成的代码结束//////////
ProgBar = 100
WordDoc. Paragraphs. Add
WordDoc. Paragraphs(WordDoc. Paragraphs. Count)
. Range. Paste
WordDoc. Paragraphs. Add
WordDoc. Paragraphs(WordDoc. Paragraphs. Count)
. Alignment = wdAlignParagraphCenter
WordDoc. Paragraphs(WordDoc. Paragraphs. Count)
. Range. Text = "压力图"
WordDoc. SaveAs DestFile
WordDoc. Save
If bWordStatus = False Then
WordApp. DisplayAlerts = False
WordApp. Quit
Set WordApp = Nothing
End If
CmdFinish. Enabled = True
Lbl. Caption = "文档已经生成, 请打开. "
End Sub
Private Sub CmdFinish_Click()
Unload Me
End Sub
```

插入一个模块 添加如下代码

```
Type Data01
    dTime As Single
    dPressure As Single
End Type
Dim bFinish As Boolean
Sub main()
Load Form1
Form1. Show
End Sub
```

四、结束语

该程序在 Office2000 和 VB6. 0 的环境中调试通过。本文仅仅起到了抛砖引玉的作用，对 Word 中常用的文字、表格、曲线图形进行了处理。

读者可以根据实际需要对该程序进行改编，可以输入多个数据文件，处理多个表格、曲线。

收稿日期：2000 年 10 月 24 日）

新业务接踵而来，金仕达多媒体并蒂花开

金仕达多媒体在 2000 年实现了跳跃性发展，企业的各项业务都快速成长，所承接的上海证券交易所、古北新区、罗氏等一系列项目，由于较高的质量得到客户的认可。

2001 年伊始，金仕达多媒体将再攀新高峰。近日，上海联合基因公司基因谷项目、中原经济园区都市型工业基地项目先后与金仕达多媒体签约，委托金仕达多媒体进行项目多媒体演示的开发工作，拟通过多媒体这一高科技演示手段，对项目进行宣传。其中，上海联合基因公司的基因谷是科技含量极高的工程项目，对光盘制作也提出了很高要求，金仕达多媒体以精美的制作质量得到客户的青睐，顺利成为该项目的承接商。

2000 年，金仕达多媒体已经承接了数十项多媒体演示项目制作，这表明，多媒体作为一种先进、新颖的演示手段，已经为越来越多的企业所认识和接受。

编程控制报表输出

刘遵雄　郑淑娟

摘　要　本文介绍了在 C + + Builder 中在使用 Quickreport 控件制作报表时进行适当编程的方法　由此可以制作满足特定要求的报表。

关键词　C + + Builder，Quickreport 控件，编程，报表输出

一、前言

任何数据库应用报表系统都十分重要，它是主要的输出系统。目前市场上颇受软件开发人员欢迎的数据库应用系统的开发工具有 INPRISE 公司　原宝兰公司）的快速开发工具（RAD）Delphi 和原 POWERSOFT 公司　现已并到大型数据库提供商 Sybase 公司）的 POWERBUILDER，各自具有不同的优点。

在众多可视化数据库开发工具中　Delphi 以其真正的面向对象、高效率、支持多层结构应用开发、支持多层 B / S 结构开发等优良特性脱颖而出，成为广大编程人员的首选开发工具。C + + BUILDER　BCB）与 Delphi 同为 BORLAND 公司所提供，两者共用可视化类库　VCL），差别在于它们的语言内核不一样，Delphi 基于 Pascal 语言，BCB 以 C　C + +）语言为基础，由于 C 语言编程方面的优越性以及受 TC　Turbo C）和 BC　Borland C + +）以前应用深远的影响，BCB 越来越受到大家的推崇。

BCB 和 Delphi 中的 quickreport 控件是挪威的 QuSoft AS 公司开发的用于制作报表的控件组，使用它可以制作一些相当复杂、功能完善的报表，具有很强的访问数据库的能力。通常编程人员都是在设计期间设计好报表的所有格式，并在运行时连上数据源 即 delphi 中的 dataset 运行程序的，这种方法缺乏灵活性。而且简单地用 quickreport 控件生成的报表更适合于西方的应用形式，不能满足我们现实工作的需求。譬如下面的报表：

分析此报表知道，这是一个打印校园运动会比赛成绩的报表，它要求按比赛项目进行分组，列出各参赛部门及单项运动员姓名、成绩和具体名次。用前面所提到那种简单的报表制作方法解决不了问题，需要通过编程来开发。这里假设该报表的数据源取自某 Query 控件，它使用 SQL 语句从数据库不同的表中提取数据，并按 “名次” 字段排序。

二、Quickreport 控件应用

制作此报表，其关键就是要知道 quickreport 控件组中提供的控件具有自己的相关属性和对应的事件，与一般的 VCL 控件不太一样，quickreport 控件没有方法。其属性既可以在报表设计阶段进行设定，也能够在程序运行时动态地修改，为 quickreport 控件修改属性编写的代码将置于有关控件对象的事件中，从而达到动态控制报表输出的目的。

quickreport 控件组中 TQRDBText、TQRLabel 等控件的属性及 quickreport 控件的 BandType 属性等方面问题在此不再赘述，需要提及的是 TQRShape 使用来在报表上画简单图形如长方形，圆和线条的控件，有两种方法可以画线条，一是修改 TQRShape 控件的属性 Shape 设定值 qrsVertLine 代表垂直线，qrsHorLine 代表水平线　即可　二是分别设置 TQRShape 的 Height、Width 属性值，Height 值为 1，将画水平线；Width 属性值为 1，将画垂直线；如果设 Height 属性值为 0，该 TQRShape 控件将不可见。需要特别提到的是 TQRBand 的 BeforePrint 事件，它在对应的 Band 将要打印之前调用，在此编码改变 quickreport 控件的属性值，将输出符合一定要求的报表。

三、报表实现示例

下面就以前面提出的报表实现方法为例，阐述如何使用 BCB 动态控制报表的输出。

首先在 C + + Builder 中新建工程存为 RptPrint. BPR，工程中将包含两个 Unit 单元。新建 Form1　放置 quickreport 元件的窗体　其中放入 quickrep 元件 quickrep1 和两个 qrband 元件，其 bandtype 分别为 rbPageHeader　rbDetail；对应的 Name 分别为 PageHeaderBand1，DetailBand1. 向该窗体上添加两 Query 控件，Name 分别为 Query1，Query2　设置 quickrep1 其 DataSet 属性为 Query1 ，设置它们的 DatabaseName 属性值为

指定的数据库别名，其中如前所述 Query1 用于选取用于输出到报表中的数据记录，Query2 用于读出每个比赛项目的参加部门的项次数。假设 Query1 的数据都取自表 ForGame. DB，Query1 的 SQL 属性为 “select * from ForGame order by Gameid Result”。 ForGame. DB 字段如下表：Gameid、GameName、Depart、Name 、Grade、 Measure 和 Result；对应说明是：项目编号、项目名称、参赛部门、运动员、成绩、计量单位和获得名次。

使用 TQRLabel 制作报表的标题、报表的栏目名称，它们的 Caption 分别为 ‘序号”、 ‘项目名称”、 ‘单位”、 ‘参赛部门”、 ‘运动员”、 ‘成绩”和 ‘名次”；TQRShape 控件画线条，方法是向 PageHeaderBand1 上拖放 TQRShape 控件，设置它们的 Height、Width 属性值使之满足要求，它们包含 8 条短竖线和长的顶线，用 ‘Ctrl”键 + 箭头键移动，使之对齐。PageHeaderBand1 上的内容将在报表的每页上都会出现。

现在要设置 DetailBand1 了，向 DetailBand1 拖放三个 TQRLabel 控件对象，分别与上面的 “序号”、 “项目名称”、 “单位”对齐；拖放四个 TQRDBText 对象到 DetailBand1，设置它们的 DataSet 属性值为 Query1，DataField 属性值分别为 ‘Depart”、 “Name”、 “Grade”和 ‘Result”字段同样。同样添加 TQRShape 控件对象，分隔各栏数据的短竖线与上面的对齐。DetailBand1 的顶线有二，一为 QRShape10，置于三个 TQRLabel 控件对象上方；另者 QRShape11，置于四个 TQRDBText 对象上方；两者对齐，设计它们的 Height 属性值皆为 1。DetailBand1 的底线为 QRShape21 将编程改变 Height 属性值的 TQRShape 对象即 QRShape10 和 QRShape21。改变此 Form 的 Name 为 Form_Report，保存 Unit 为 Report1，将形成文件 Report1. Cpp、Report1. h 和 Report1. dfm。

建立 form1；unit1 （调用窗体），其中放入 button1 caption 为 ‘预览’，用以查看报表，form1 的 caption 为 ‘报表打印’， 保存 Unit1 为 Print 形成文件有 Print. Cpp、Print. h 和 Print. dfm。该窗口主要用于打开调用上面制作的包含 quickreport 元件的窗体，在 Print. Cpp 中加入 # include ‘Report1. h”，编写 button1 的 Click 事件代码如下：

```
void __fastcall TForm1: : Button1Click(TObject * Sender)
{
//此处可加入 Form_Report 的 Query1 的控制,以取得不同的数据
Form_Report - >QuickRep1 - >Preview();
}
```

于是该报表示例的准备工作基本完成，下面主要的是在 Report1. cpp 和 Report1. h 添加代码控制报表的动态输出，讨论 Report1. cpp 的编程了。其思路是：①每次打印 Form_Report 的 QuickRep1 在 DetailBand1 前，因为 Query1 记录的打印是一条一条进行的。先取出每一条将打印记录的 ‘Gameid”的值，保存了上一记录的 ‘Gameid”的值为 OldStr （对于第一条记录，其 OldStr 为 “”），如果 ‘Gameid”的值与 OldStr 不等，说明将打印另一比赛的情况，QRShape10 的 Height 属性值等于 1，否则为 0，从而动态地改变 QRShape10 的属性，使 QRShape10 代表的线条在适当的时候打印，这样实现了报表数据按 ‘比赛项目”分组。②是打印报表每页的最后记录或者报表数据打印完成时，都应该画直线，使 QRShape21 的 Height 属性值等于 1，别的时候此线不画。③控制比赛项目分组中记录共同关于项目的信息打印一次，并且打印在分组中对应为中间记录的数据行上，关键是记住报表已打印的行数，取出下个项目对应的记录数，再判断将打印项目信息的数据行。

Report1 单元的具体实现，首先在 Report1. h 中类的 Public 下定义

```
public:
int LineCount, K, J, RecordCount1, Id;
    AnsiString OldStr; //上条记录的 GameId 值
    void __fastcall EmptyLables(); // 将记录项目信息序号的 TQRLabel 的 Caption 为空
    int __fastcall IsPageFoot(); //判断页末函数
    int __fastcall IsLastRecord(); //判断是否为最后记录
```

其中 LineCount 记录报表一页的记录数；Id 记录项目序号；K 表示报表已打印的记录数；RecordCount1 表示比赛项目信息将打印的数据行；J 表示报表记录累加各分组将达到的数据行数

Report1. cpp 的源代码如下：

```
#include <vcl. h>
#pragma hdrstop
#include "Report1. h"
#pragma package(smart_init)
#pragma resource "*. dfm"
TForm_Report * Form_Report;
__fastcall TForm_Report: : TForm_Report(TComponent * Owner)
      : TForm(Owner)
{
OldStr = "";
}
void __fastcall TForm_Report: : DetailBand1BeforePrint (TQRCustomBand * Sender,
    bool & PrintBand)
 {
 AnsiString NewStr;
 int I, RecordLast, PageFoot, Quey2Count;
   NewStr = Query1 - >FieldByName( " Gameid") - >AsString; //读取当强将打印记录的 Gameid 值
 K = K + 1; //报表已打印的总记录数加 1
 I = AnsiStrComp( NewStr. c_str(), OldStr. c_str());
//比较当前记录的 Gameid 值是否与前记录的 Gameid 值相同否,相同函数返回 1
 if (I = = 1)// 如果当前记录的 Gameid 值是否与前记录的 Gameid 值不同,即读入新 Gameid 值
```

```
    {
    QRShape10->Height = 1;  //画比赛项目信息的顶线
     OldStr = Query1->FieldByName("Gameid")->AsString; //保存本记录的 Gameid 值
    Query2->Close(); //Query 控件动态编程
    Query2->SQL->Clear();
     Query2->SQL->Add("SELECT Count(Gameid) C1 FROM ForGame where Gameid=StrGameid"); //通过 Query2 读取对应新 Gameid 值的比赛情况记录数
Query2->ParamByName("StrGameid")->AsString = NewStr; //给 Query2 的参数赋新 Gameid 值
    Query2->Prepare();
    Query2->Open();
     Quey2Count =Query2->FieldByName("C1")->AsInteger; //取得新 Gameid 值对应的记录数
    J=J+ Quey2Count; //记录每个分组的最后记录是数据集的那条记录
    RecordCount1=J-Quey2Count/2 ; //设置每个分组比赛共同信息将打印的数据行的位置
    }
    else
    {//取得的 Gameid 值是否与前记录的 Gameid 相同,记录间共用信息部分分隔线不画
    QRShape10->Height = 0;
    };
if (K==RecordCount1)
//如果报表打印的数据行到了打印分组公用信息对应的行,则打印分组公用信息
    {
     Id=Id+1; //比赛项目的序号递增
     QRLabel9->Caption=IntToStr(Id); //打印对应的序号
      QRLabel10->Caption = Query1->FieldByName("GameName")->AsString;
     QRLabel11->Caption = Query1->FieldByName("Measure")->AsString;
    }
    else
    {
     EmptyLables(); //否则不打印分组公用信息
    };
     RecordLast=IsLastRecord();
     if (RecordLast>0)//如果报表打印到最后记录,画报表的最后底线(QRShape21 线)
     {
       QRShape21->Height = 1;
       OldStr ="";//需要重新初始化 OldStr
     }
     else
     {
      QRShape21->Height = 0;// 如果报表打印不到最后记录,QRShape21 代表的线不画
     };
     PageFoot=IsPageFoot();
    if (PageFoot=0)//每次报表打印到本页末,画报表底线
    {
      QRShape21->Height = 1;
      OldStr = "";//初始化 OldStr,使下页如前重新开始
      LineCount = 0;//初始化 LineCount
    };
}
int __fastcall TForm_Report::IsPageFoot()//判断报表打印是否到了本页末
{
 int PageFoot;
 LineCount = LineCount+1;//每次调用,代表报表一页记录数的 LineCount 加 1
     if (QuickRep1->Height -
        QuickRep1->Bands->DetailBand->Height -
        (QuickRep1->Bands->DetailBand->Top +
         (LineCount * QuickRep1->Bands->DetailBand->Height)) <
        QuickRep1->Bands->DetailBand->Height )// 如果报表打印到了本页末
     {
       PageFoot = 1;//函数返回 1
     }
     else
     {
    PageFoot = 0;//如果报表打印未到了本页末,函数返回 0
     }
     return PageFoot; }
int __fastcall TForm_Report::IsLastRecord()// 判断报表数据是否为最后记录
{
int RecordLast;
Query1->Next();//Query1 记录指针下移
if (Query1->Eof)//如果数据集的末尾,函数返回 1
     {
       RecordLast = 1;
     }
     else
     {
      Query1->Prior();//否则数据集指针移回到原位置,函数返回 0
       RecordLast = 0;
    }
return RecordLast;
}
//
 ---------------------------
void __fastcall TForm_Report::EmptyLables()
{
     QRLabel9->Caption = "";//使不该显示项目及序号信息的记录显示空
     QRLabel10->Caption = "";
     QRLabel11->Caption = "";
 }
void __fastcall TForm_Report::FormActivate(TObject *Sender)
{
K=0;//在报表窗体激活时初始化这些便变量
J=0;
Id=0;
}
```

下转第 33 页

Linux 下用 C + + 进行 OOP 窗口编程

李宋琛

摘 要 本文介绍了使用 C + + 在 Linux 下开发 X Window 窗口程序的方法。
关键字 Linux，C + + ，OOP，窗口编程

GUI 编程一直是人们非常感兴趣的一个课题。长期以来，图形化的人机界面一直是 Mach 和微软 Windows 的天下。特别是 MS - Windows，由于窗口程序编写十分容易，因而吸引了大批的程序员为其编写程序，其编程工具也层出不穷，可视化的开发工具如，Visual Basic，Visual C + + ，C + + Builder 等已经十分普遍。而正是这些工具吸引了大批的程序员投向了 Windows。传统 Unix 系列的操作系统虽然功能强大，技术成熟，但由于使用不便，很难吸引普通的 PC 用户涉足，而 Unix 下虽然也可以进行图形界面的窗口程序设计，但是却令程序员十分头痛。像基于 XLib 或者 Motif 进行编写都是面向过程的开发形式，开发周期长，难度大，维护困难 只有少数程序员才能掌握，因此没有被广泛使用。

由于近年来 Linux 的快速发展和普及，这种现象正在发生着变化。实际上，当前所有的 Linux 发布版几乎都带有一套甚至更多的桌面系统。Linux 正在悄悄的进军 PC 桌面系统，并且可能凭其开放性和优越的性能而影响微软在此领域的霸主地位。在 Linux 下进行图形界面开发的方法多种多样，一般都是以某种库代码为基础而进行。本文将介绍使用 C + + 进行面向对象的窗口编程方法和实例。

本文将着重讲述一种 Linux 下的窗口编程方法，读者需要一定的 Linux 使用经验和 C + + 编程的知识。

一、X Window 系统

X Window 是由麻省理工学院 MIT 设计的一套开放性的口系统，当前版本为第六版，即 X11R6。这个系统实际上是由一系列的协议组成，因此有时候也称为 X Window 协议。与 Windows 的一个重要区别是，所有 X Window 下的程序都是 Client / Server 模型，其中的 Server 是控制实际绘图的程序，在 Linux 中一般就是图形驱动程序完成，并在系统安装时就已配好。而所有的用户程序是客户端 通过一系列的函数调用请求服务器完成绘图功能。实际上，X Window 协议是基于 TCP / IP 之上的，使用默认的 6000 端口进行通信。因此，X Window 系统本身就是分布式的，在其中一台计算机上运行，而在另外一台计算机上显示和在同一台计算机上进行并没有什么实质上的差别，程序员也不需要了结其中的细节。

进行 X Window 程序设计最基本的方法就是调用 Xlib 库，它是对 X Window 协议的直接描述。因此，使用其它库能写出的程序都可以用 Xlib 进行重写。然而，由于 Xlib 库中的函数过于初级，大都是画点，画线一类的非常初等的函数，用 Xlib 直接进行一般的应用程序开发工作量太大，并不实际。

由于上述的原因，一些公司或者组织开发了许多构建在 Xlib 等之上的库，提供一系列的高级函数，如 OSF 的 Motif Athena 等。这些库的共同特点是 面向过程编程，提供画窗口，对话框等基本函数。Motif 在一定程度上获得了成功，并一度成为 Unix 下的标准 GUI 开发包，但是和九十年代中后期迅猛发展的 Windows 相比，差距还是十分明显，在 Unix 中急需一个类似 Microsoft MFC 的类库包使程序员能进行 OOP 的程序设计，提高代码的可重用性和好的维护性。

二、Qt 库

Qt 库是由挪威 Troll 公司设计开发。是一个直接用于 Xlib 的，比较成熟的 OOP 开发包。使用 Qt 进行窗口程序设计非常类似用 Visual C + + 的 MFC。因而自从其发布后，出现了一大批用 Qt 写成的自由软件 其中最有名的就是 KDE K Desktop Environment 。虽然 Qt 作为商业软件并不遵从 GPL 许可证，但是，只要是用于非 Windows 平台的应用，都是免费的。而且正如上面提到的，基于 Qt 的程序可以方便地移植到 MS Windows 中。安装 Qt 非常简单，也许你的 Linux 系统中已经安装了。RedHat 6.0 以上都带有，你可以用 rpm 命令安装编译好的二进制包。如果你没有，可以到 Troll 公司的站点（www. trolltech. com）下载。具体安装方法不在此赘述，需要提醒的一点是要正确的设置环境变量 PATH，QTPATH，LD_LIBRARY_PATH 的值。在启始目录的 . bashrc 文件中加上如下语句：

```
export QTDIR = /usr/local/qt
export PATH = $PATH: $QTDIR/bin
export LD_LIBRARY_PATH = $QTDIR/lib
```

其中 QTDIR 应该设置成你 Qt 库的实际安装位置。

三、第一个 “Hello World ”程序

本节将描述一个最简单的 可运行的窗口程序 以使读者能够有一个初步认识。用任何你喜欢的编辑器编辑以下的 hello. cpp 和 Makefile 文件 可以省略掉 / / 开始的注释），注意，你必须修改 Makefile 中的 QTINC 和 QTLIB 宏，使它们和你的安装路径相匹配，否则编译 2 无法通过。然后在目录下输入 “make ”，如果没有什么错误的话，应该生成一个叫 “hello ”的程序，输入 . /hello 运行一下，看看效果。

hello. cpp

```
#include <qapplication.h>
#include <qpushbutton.h>
int
main(int argc, char * * argv)
{
    //QApplication 是应用程序类,由它创建程序实例,并完成命令行的参数分析
    //X Window 程序和其它的 Unix 程序一样,可以接受参数.并且,这些参数在所有
    //的 XWindow 程序中通用,比如 -geometry,可以把这些参数全部传给 QApplication
    //进行分析.所有的应用程序都应该有该类的实例.
    QApplication a(argc, argv);
    //构建一个 PushButton 实例
    QPushButton hello("Hello World!");
    //进行窗口的缩放,第一个参数是长,第二个参数是宽
    hello.resize(100, 30);
    //设置"主窗口";
    a.setMainWidget(&hello);
    //显示窗口并执行应用程序.
    hello.show();
    return a.exec();
}
Makefile
CC = g++
QTINC = /usr/lib/qt-1.45/include
QTLIB = /usr/lib/qt-1.45/lib
hello: hello.o
    ${CC} -o hello hello.o -lqt -L${QTLIB}
hello.o: hello.cpp
    ${CC} -c hello.cpp -I${QTINC}
```

四、Signal - Slot 机制

要实现一个好的 OOP 库必须解决对象间通信的问题。实际对象间通信的问题一直是面向对象程序设计的一个重要内容，而实现解决这个问题的方法也多种多样，Qt 中的对象通信机制非常有特点。在 Qt 中的对象间通信方法被称作"Signal - Slot"，这也是 Qt 与其它一些方法的区别之一。

在 MS - Windows 中 程序通过消息机制和事件循环来实现对图形对象的行为进行触发。而在 Qt 中采用了另外的一种实现方法。一个类可以定义多个 Signal 和 Slot，Signal 就好像是"事件"，而 slot 则是响应事件的"方法"，并且和一般的成员函数没有太大的区别。而需要实现它们之间通信时，就将某个类的 Slot 与另外一个类的 Signal "连接"起来 从而实现"事件驱动"。举个例子，我们需要实现这个功能：当用户按了某个按钮，程序弹出一个对话框。我们的做法就是 当按钮按下时，发出一个 Signal 而这个 Signal 是事先和某个类的弹出对话框的方法，也就是 slot 相连接的。需要注意的是 Signal 和 Slot 函数的返回类型必须是 void，并且不支持缺省参数。

有了 Signal - Slot 机制，进行事件响应编程就变得十分的轻松。比如对于鼠标事件，在 Windows 中是通过发送鼠标左击事件给相应的窗口，而在 Qt 中，鼠标左击事件被设计成某个对象的 Signal。这样，如果需要响应鼠标左击事件，只需要简单的把需要的函数与该鼠标事件连接起来即可。

让我们来看一段简单的代码。以下的 3 个文件的编译方法同上。

```
test.h
#ifndef _TEST_H_
#define _TEST_H_
//包含必要的头文件
#include <iostream.h>
#include <qobject.h>
//在该类中有几个显著的特点:
//1. 有一个 Q_OBJECT 宏,所有派生于 QObject 并且要使用 Signal-Slot 机制的类都必须
//   定义这个宏
//2. public slots 下定义的是响应"事件"的"方法",这些成员函数可以通过 connect
//和 slog 相连接,但其本质上就是一般的公用成员函数.
//3. signals 下定义"事件",注意,这里定义的函数没有实现,只是原型
class classA: public QObject
{
  Q_OBJECT
  public:
      say(){
          emit sayHello();
      }
  public slots:
      void hello(){
          cout << "hello, world\n";
      }
  signals:
      void sayHello();
};
#endif

test.cpp
#include "test.h"
//程序中通过 QObject::connect 函数将 sayHello 与 hello 连接
//QObject::connect 为 QObject 的静态成员(static public member)原型为:
//bool connect)(const QObject * sender, const char * signal, const QObject *
// receiver, const char * member)
//当执行 a.say()时,发生了 sayHello()事件,触发了 hello 响应
//产生了输出.
//注意一下输出的结果,只有一行"hello, world"而不是两行,原因在于
//执行 b.say()的时候虽然也发出了信号,但是 b 的 signal 并没有和 a 的
//slot 相连接,所以没有输出.
void main()
{
  classA a, b;
  QObject::connect(&a, SIGNAL(sayHello()), &b, SLOT
```

```
(hello()));
    a.say();
    b.say();
}
```

Makefile

#由于 test.h 中定义类的特殊性(含有 Q_OBJECT, public slots, signals 等关键字)
#在编译前必需进行特殊的宏处理,这通过一个程序 moc 来完成.

```
CC = g++
QTINC = /usr/lib/qt-1.44/include
QTLIB = /usr/lib/qt-1.44/lib
MOC = /usr/lib/qt-1.44/bin/moc
test: test.o test.moc.o
    ${CC} test.o test.moc.o -o test -lqt -L$(QTLIB)
test.o: test.cpp
    ${CC} -c test.cpp -I$(QTINC)
test.moc.o: test.moc.cpp
    ${CC} -c test.moc.cpp -I$(QTINC)
test.moc.cpp: test.h
    ${MOC} test.h -o test.moc.cpp
```

由于 Signal - Slot 机制的特殊性,当我们在使用这个机制时可能会碰到不少问题,但是,用 Qt 编程又不可能回避之,因此在使用 Signal - Slot 机制和 moc 时应该要注意以下几点:

1. 所有的头文件必须经过 moc 的处理后才能进行编译,使用方法 以 hello.h 为例 :

moc hello.h - o hello.moc.cpp

2. moc 生成的文件必须由 g++ 进行编译和连接

3. moc 不支持模板 template

4. moc 不对#include 等宏进行扩展 简单跳过

5. 使用多重继承时,moc 假设第一个类是 QObject 或其派生类,而其它的类不能是 QObject 或其派生类

6. signal,slot 可以接受参数,但不能是函数的指针

7. moc 不支持 friend 关键字

8. moc 不支持嵌套类

9. signal 和 slot 不能是构造函数

10. 不支持缺省参数

11. 所有的 signal 和 slot 不能有返回值 必须是 void

12. slot 可以为虚函数

13. signal 的访问权限和 protected 相同 即只有本身及其派生类可以 emit signal。

五、使用控件 widget

前面介绍一个简单的例子和 Signal - Slot 机制。但是要进行完整的图形界面的程序设计还需要大量的高级控件,这些控件在 Qt 中被称为 "widget"。与 Windows 下的程序开发过程类似,Qt 中也有若干类似的控件可以使用,这对我们进行快速应用开发提供了条件。这些控件包括常用的各种按钮,进度条,菜单,工具条,状态条等,甚至还包括了 OpenGL 控件。为了解释一般的开发过程,下面分步描述如何实现一个带菜单,工具条,状态条和模态对话框的程序框架。

同其它的 Unix 程序一样,代码从 main 开始编写。在 main 中主要做两件事情,其一是设置应用程序实例,这通常由构造 QApplication 来实现,另外一个是确定主控件,或者叫 "主客户窗口 。看下面这段代码:

```
int
main(int argc, char * * argv)
{
  QApplication a(argc, argv);
  MainWidget w;
  w.setGeometry(150, 150, 300, 200);
  a.setMainWidget(& w);
  w.show();
  return a.exec();
}
```

以上的这段代码十分典型。首先构造一个 QApplication 的实例,并把命令行参数进行分析。然后构造一个 "主控件" MainWidget。执行 w.show 和 a.exec 是必须的,否则程序不能正常的显示和运行。

有 Windows 编程经验的读者可能会问了:为什么 setMainWidget 设置的是 "主客户窗口" 而不是 "主窗口" 呢?其实这是由 X Window 协议本身的特点决定的。在 X Window 中的窗口程序,它们的 "边框" 的样式和风格是应用程序自身无法控制的,而是由另外一个叫 "窗口管理器" 的程序来进行控制。应用程序只能控制 "边框" 里面的部分,而不能实现,像在 Windows 中那样可以设置边框风格的功能。如果程序一定要实现这个功能,就必须和这个 "窗口管理器" 进行 "协商"。当然,由于这并非强制性的,所以窗口管理器可以不理会你的程序提出的要求。

以上的程序还不完整,因为 MainWidget 这个类并不是由 Qt 库提供的,而是我们定义在 mainwidget.h 这个头文件里面的。所以我们必须实现之。这个 MainWidget 可以是任何 Qwidget 的派生类,由于我们常常要使用的是普通的应用程序框架,所以我们在这里从 QmainWindow 派生。

```
class MainWidget: public QMainWindow{
  Q_OBJECT
  public:
MainWidget(QWidget * parent = 0, const char * name = 0);
  public slots:
      void setColor();
      void exitMain();
  private:
      QMenuBar * menu;
};
```

上面是 MainWidget 的定义。首先是 Q_OBJECT 宏,上面曾经提到过,所有要使用 Signal - Slot 机制的类都必须使用这个宏定义。另外有两个 Slot:setColor 和 exitMain 。先看看这个类的实现部分。

```
#include "mainwidget.h"
MainWidget::MainWidget(QWidget * parent, const char *
```

```
name)
: QMainWindow(parent, name)
{
setCaption(″Example″);
//设置窗口的标题
QPopupMenu * option = new QPopupMenu;
//构造一个弹出式的菜单
option ->insertItem(″& Set color″, this, SLOT(setColor()));
    option ->insertSeparator();
    option ->insertItem(″E& xit″, this, SLOT(exitMain()));
    menu = new QMenuBar(this);
    menu ->insertItem(″& Option″, option);
}
void
MainWidget: : setColor()
{
    MainDialog m;
    m. exec();
}
void
MainWidget: : exitMain()
{
    QApplication: : exit();
}
```

在构造函数中主要是新建一个菜单。在 Qt 中的菜单实际上有两个部分组成，一个是菜单条 QMenuBar 另外一个是弹出式的菜单 QPopupMenu，这两个部分必须单独构建。需要注意的是 insertItem 方法的第 2，3 个参数，这实际上是调用了 connect 方法把菜单被选择这个事件与某个类的 Slot 方法连接起来，这是很典型的用法。

在 exitMain 方法中简单地调用了 QApplication 类的静态方法 exit 这也是退出应用程序的一般途径。而在 setColor 方法中，我们又定义了一个新的类：MainDialog，并使用和 QApplication 类似的 exec 方法使之运行。以下是 MainDialog 的定义和实现：

```
class MainDialog: public QDialog{
    Q_OBJECT
    public:
MainDialog(QWidget * parentl = 0, const char * namel = 0);
};
MainDialog: : MainDialog(QWidget * parent, const char *
name)
    : QDialog(parent, name, TRUE)
{
    setCaption(″Choose color″);
    QPushButton * r, * b;
    r = new QPushButton(″Red″, this);
    b = new QPushButton(″Blue″, this);
this ->setFixedSize(140, 80);   //设置对话框的大小
    r ->setGeometry(20, 20, 40, 30);
    b ->setGeometry(80, 20, 40, 30);
    connect(r, SIGNAL(clicked()), SLOT(accept()));
    connect(b, SIGNAL(clicked()), SLOT(reject()));
}
```

这个类的实现很简单，安排两个按钮。最后两个 connect 调用需要注意。每个按钮的 Signal - clicked 分别和一个 Slot 相连接。而 accept 和 reject 都是 Qdialog 的成员函数，作用是关闭对话框并返回 “真” 或者 “假”。由此就可以判断对话框中选择的内容。

如果实际应用中是个很复杂的对话框，含有各种按钮，列表等，这些数据不能简单的用 “真” 或者 “假” 来表示怎么办呢？方法也很简单，就是在 MainDialog 的私有变量中保留这些选择的数据，并在对话框完成后通过公共接口进行访问即可。

五、响应鼠标事件和在窗口中画图

在一般的 GUI 程序中，鼠标是最主要的输入工具。如果不能够很好的处理鼠标事件，应用程序的友好性将大打折扣。幸运的是，Qt 在这方面提供了完整的解决方法，使我们在进行程序设计时难度大大降低了，实际上和在 MS - Windows 下进行的程序设计已经十分类似。通常，鼠标事件分成三类：左键单击，右键单击和左键双击。而这三个事件则对应了 Qwidget 中的一个虚方法，不同的鼠标事件有不同的入口参数，这是和 Windows 不同的地方。捕获鼠标事件首先要重载这些虚方法，使鼠标事件发生时能够调用适当的过程。

先看段代码。

```
class MainWidget: public QMainWindow{
  Q_OBJECT
  public:
MainWidget(QWidget * parent = 0, const char * name = 0);
  public slots:
      void setColor();
      void exitMain();
  protected:
      void paintEvent(QPaintEvent * );
      void mousePressEvent(QMouseEvent * );
  private:
      QMenuBar * menu;
      QColor color;
      QPoint point;
};
```

下面是对上一个例子的扩充，其中增加了一个保护的 mousePressEvent()方法. 我们看看它的实现部分.

```
void
MainWidget: : mousePressEvent(QMouseEvent * e)
{
if(e ->button() = = LeftButton){      //看是否按的是左键
point = e ->pos(); //把鼠标的当前位置存入私有成员变量中
repaint();                            //强制进行重绘
  }
}
```

从上面的代码可以看出，在 Qt 中，所有的鼠标事件都调用 mousePressEvent 方法，而要区分它们，就必须比较入口参数 QmouseEvent * 所提供的信息。

与处理鼠标事件类似，要在窗口中画图，需要重载 Qwidget 的虚方法 paintEvent 。以下是 paintEvent 的实现。

```
void
MainWidget: : paintEvent(QPaintEvent * )
{
QPainter p;                              //构造一个图形引擎
QPen pen(color);                         //构造“一支笔”
p. begin(this);                          //开始画图
p. setPen(pen);                          //选择一只笔
p. drawText(point, ″Hello, World! ″);    //在 point 处写上文字
p. end();                                //结束画图
}
```

这段代码的功能是在鼠标单击的位置用对话框中设置的颜色写出“Hello World!”语句。在进行绘图时，首先必须构造一个 Qpainter 对象，这个对象是 Qt 的图形引擎，所有的画图语句都是 Qpainter 的一个成员函数。画图从 begin 语句开始，注意 begein 的参数，这实际上是把当前的这个控件（MainWidget）引入到图形场景中。

Qt 的图形引擎也有“笔”，“刷子”的概念，在这一点上是和 Windows 相同的。通常的画图过程是调用 begin 选择适当的笔和刷子，使用 Qpainter 的成员函数进行化点，线，矩形，坐标变换等过程，最后调用 end 结束绘图过程。

由于加入了图形代码，我们的 setColor 方法也有所变化。

```
void
MainWidget: : setColor( )
{
    MainDialog m;
    color = m. exec( )?red: blue;
}
```

显然，颜色将设置成所选的值。

五、一个完整的应用程序框架

上面的所讨论的代码讲述了一个基本的框架，但这对普通的程序还不够。我们希望在程序中能做到和 Windows 同样的界面和功能。要实现这些，工具条和状态条必不可少。在 Qt 中添加工具条和状态条十分简单。构造一个 QtoolBar 需要若干的 QtoolButton 这些小按钮需要从位图生成，读者可以用 Gimp 在 Linux 下绘制 16x16 的图形，并存为以 . xpm 为后缀的文件，这种图形文件类似 Windows 中的 BMP 文件，是 X Window 的“标准”图形文件，Qt 中也支持。而如果你的主控件是从 QmainWindow 派生的，那么生成状态条只需要一句代码。StatusBar 成员函数会在没有状态条的情况下生成，如果已有，则返回状态条对象的指针。但如果你的 MainWidget 不是从 QmainWindow 派生，使用状态条就要麻烦一些。在 MainWidget 的构造函数中，我们添加了以下代码：

```
MainWidget: : MainWidget(QWidget * parent, const char * name)
    : QMainWindow(parent, name), color(red), point(0, 0)
{
    //前面的代码在此省略. . .
    QToolBar * tools = new QToolBar(″example″, this);  //构造一个 ToolBar
QPixmap colorIcon(setcolor_xpm), exitIcon(exit_xpm); //从文件生成一副位图
QToolButton * setcolor = new QToolButton(colorIcon, ″Set Color″, 0,
this, SLOT(setColor( )),
tools, ″set color″);                     //在 ToolBar 上放按钮
QToolButton * exitmain = new QToolButton(exitIcon, ″Exit″, 0,
  this, SLOT(exitMain( )),
  tools, ″exit″);
  statusBar( ) ->message(″Ready″, 2000);
}
```

到此为止，一个基本的应用程序框架已经讲述完成。下面贴上窗口图像和完整的程序代码供读者参考。全部程序包括 4 个文件：Makefile main. cpp mainwidget. h mainwidget. cpp。

另外，读者必须自己准备两幅 16x16 的 XPM 格式图形，分别存为“setcolor. xpm”和“exit. xpm”。然后在目录下输入“make”会产生一个“example”的可执行文件。

Makefile:

```
CC = g + +
QTINC = /usr/lib/qt - 1. 44/include
QTLIB = /usr/lib/qt - 1. 44/lib
MOC = /usr/lib/qt - 1. 44/bin/moc
example: main. o mainwidget. o mainwidget. moc. o
  ${CC} main. o mainwidget. o mainwidget. moc. o -o example -lqt -L${QTLIB}
main. o: main. cpp
  ${CC} -c main. cpp -I${QTINC}
mainwidget. o: mainwidget. cpp
  ${CC} -c mainwidget. cpp -I${QTINC}
mainwidget. moc. o: mainwidget. moc. cpp
  ${CC} -c mainwidget. moc. cpp -I${QTINC}
mainwidget. moc. cpp: mainwidget. h
  ${MOC} mainwidget. h -o mainwidget. moc. cpp
```

main. cpp

```
#include ″mainwidget. h″
int
main(int argc, char * * argv)
{
  QApplication a(argc, argv);
  MainWidget w;
  w. setGeometry(150, 150, 300, 200);
  a. setMainWidget(& w);
  w. show( );
  return a. exec( );
}
```

mainwidget. h

```
#ifndef _MAINWIDGET_H_
#define _MAINWIDTET_H_
#include <qmenubar. h>
#include <qapplication. h>
#include <qdialog. h>
#include <qpushbutton. h>
#include <qpainter. h>
#include <qpen. h>
```

```
#include <qtoolbar.h>
#include <qpixmap.h>
#include <qtoolbutton.h>
#include <qmainwindow.h>
#include <qstatusbar.h>
class MainWidget: public QMainWindow{
    Q_OBJECT
    public:
MainWidget(QWidget * parent=0, const char * name=0);
    public slots:
    void setColor();
    void exitMain();
    protected:
    void paintEvent(QPaintEvent *);
    void mousePressEvent(QMouseEvent *);
    private:
    QMenuBar * menu;
    QColor color;
    QPoint point;
};
class MainDialog: public QDialog{
Q_OBJECT
public:
MainDialog(QWidget * parentl=0, const char * namel=0);
};
#endif
mainwidget.cpp
#include "mainwidget.h"
#include "setcolor.xpm"
#include "exit.xpm"
MainWidget:: MainWidget(QWidget * parent, const char *
name)
    : QMainWindow(parent, name), color(red), point(0, 0)
{
    setCaption("Example");
    setBackgroundColor(white);
    QPopupMenu * option=new QPopupMenu;
    option->insertItem("&Set color", this, SLOT(setColor
()));
    option->insertSeparator();
    option->insertItem("E&xit", this, SLOT(exitMain()));
    menu=new QMenuBar(this);
    menu->insertItem("&Option", option);
    //add toolbox
    QToolBar * tools=new QToolBar("example", this);
    QPixmap colorIcon(setcolor_xpm), exitIcon(exit_xpm);
    QToolButton * setcolor=new QToolButton(colorIcon, "
Set Color", 0,
    this, SLOT(setColor()),
    tools, "set color");
QToolButton * exitmain=new QToolButton(exitIcon, "Exit", 0,
    this, SLOT(exitMain()),
    tools, "exit");
    statusBar()->message("Ready", 2000);
}
void
MainWidget:: paintEvent(QPaintEvent *)
{
  QPainter p;
  QPen pen(color);
  p.begin(this);
  p.setPen(pen);
  p.drawText(point, "Hello, World!");
  p.end();
}
void
MainWidget:: mousePressEvent(QMouseEvent * e)
{
  if(e->button() == LeftButton){
statusBar()->message("You pressed mouse key", 2000);
      point=e->pos();
      repaint();
  }
}
void
MainWidget:: setColor()
{
  statusBar()->message("You opened a dialog", 2000);
  MainDialog m;
  color=m.exec()?red: blue;
}
void
MainWidget:: exitMain()
{
  QApplication:: exit();
}
MainDialog:: MainDialog(QWidget * parent, const char *
name)
  : QDialog(parent, name, TRUE)
{
  setCaption("Choose color");
  QPushButton *r, *b;
  r=new QPushButton("Red", this);
  b=new QPushButton("Blue", this);
  this->setFixedSize(140, 80);
  r->setGeometry(20, 20, 40, 30);
  b->setGeometry(80, 20, 40, 30);
  connect(r, SIGNAL(clicked()), SLOT(accept()));
  connect(b, SIGNAL(clicked()), SLOT(reject()));
}
```

七、进一步讨论

Qt 的功能十分的强大，远远不只是上面的介绍的功能，感兴趣的读者可以参考 Qt 附带的电子文档，Qt 的主页上也有很多关于此方面的讨论。另外，网上已经有不少开放源代码的程序，对我们进一步学习也很有帮助。

收稿日期：2000 年 11 月 7 日）

Delphi 图像演播技巧

吴勇军

本人曾用 Dlephi 为一家 LED 图文广告公司编制图文广告演播软件 （‘所见即所得”，即利用取显存技术使显示器屏幕特定区域内的图像内容即为 LED 广告屏上的内容）。虽然有复杂场景变化的广告确实需要专门的动画制作或者视频录制来完成，但是如何使一副静态的图像的演播更精彩更引人注目确实有很多技巧。在完成图文广告演播软件的时候，我收集了一些图像演播技巧以及根据日常的所见自己琢磨了几种方法。下面我介绍我所使用的图像演播方法，我想对如今日益追求美观华丽甚至花哨的软件界面制作应该有所帮助。

一、 ‘拉”的效果

‘拉”的效果可分为两种：垂直方向的 ‘拉”和水平方向的 “拉”。垂直方向的 “拉” （包括上 “拉”和下 ‘拉”）主要用于宽度小于等于显示窗口的宽度而长度大于等于显示窗口的长度的静态图像的显示。水平方向的 ‘拉” （包括左 ‘拉”和右 ‘拉”）主要用于宽度大于等于显示窗口的宽度而长度小于等于显示窗口的长度的静态图像的显示。这种情况在一行字或一列字图像的广告中经常遇到。

这种效果的实现主要利用 Delphi 中的时钟控件和画布的 CopyRect 方法。下面是上 ‘拉”效果的源程序：

```
procedure TForm1. Timer1Timer(Sender: TObject);
begin
  //Bitmap1 中图像宽度与 Image1 相等长度大于 Image1 的长度
  x: =Image1. Width;
  y: =Image1. Height;
  i: =i +10; //全局变量初始值为 0
  Image1. Canvas. Brush. Color: =clBlack; //每次重显均将显示控件初始化成全黑
  Image1. Canvas. FillRect(Rect(0, 0, x, y));
  if i < =y then
Image1. Canvas. CopyRect(Rect(0, y –i, x, y), Bitmap1. Canvas, Rect(0, 0, x, i))
  else
    if i < =Bitmap1. Height then
Image1. Canvas. CopyRect(Rect(0, 0, x, y), Bitmap1. Canvas, Rect(0, i –y, x, i))
    else
      if i < =Bitmap1. Height +y then
   Image1. Canvas. CopyRect( Rect(0, 0, x, Bitmap1. Height –i +y), Bitmap1. Canvas, Rect(0, i –y, x, Bitmap1. Height))
      else
        Timer1. Enabled: =False;
end;
```

其它各个方向上的 ‘拉”效果的实现均与此相似。

二、 ‘展开”的效果

‘展开”效果有两类：矩形展开和不规则展开。它主要用于长度和宽度均不大于显示窗口的静态图像的显示。矩形展开的实现利用时钟控件和画布的 CopyRect 方法，使出现在显示窗口上的图像矩形区域 （与显示窗口矩形同中心）由小变大逐步展开。不规则展开 （如圆及多圆展开、扇形旋转展开）的实现就较为复杂。具体参见如下源程序 （圆展开）。其中 FloodFill 使 Bitmap2 （中间位图）上产生一个黑底绿边的白色实心圆。然后利用 cmSrcAnd 与拷贝的特性 （黑）0 and X = 0 （白）1 and X = X 仅在白色实心圆内出现图像内容，而其它地方仍为黑色。随着圆半径的增长，就产生了圆展开的效果。

```
procedure TForm1. Timer1Timer(Sender: TObject);
begin
  i: =i +5;
  x: =Bitmap2. Width;
  y: =Bitmap2. Height;
  if (i < =x/2) and (i < =y/2) then
    begin
 //Bitmap2 为中间位图, 原图在 Bitmap1
Bitmap2. Canvas. Brush. Color: =clBlack;
Bitmap2. Canvas. FillRect(Rect(0, 0, x, y));
Bitmap2. Canvas. Pen. Color: =clGreen;
Bitmap2. Canvas. Ellipse(Round(x/2) –i, Round(y/2) –i,
Round(x/2) +i, Round(y/2) +i);
Bitmap2. Canvas. Brush. Color: =clWhite;
Bitmap2. Canvas. FloodFill(Round(x/2),      Round(y/2),
clGreen, fsBorder);
Bitmap2. Canvas. CopyMode: =cmSrcAnd;
Bitmap2. Canvas. Draw(0, 0, Bitmap1);
Image1. Canvas. Draw(0, 0, Bitmap2);;
    end
  else
     begin
       Image1. Canvas. Draw(0, 0, Bitmap1)
       Timer1. Enabled: =False;
     end;
end;
```

用同样的方法在不同位置多画几个圆就可实现多圆展开。对于以左上角或者显示窗口中心为圆心的扇形旋转展开，基本的图像处理方法与圆展开的相同。只不过圆展开限定显示图像的区域用 Ellipse，而扇形旋转展开则需先用 PenPos 定位然后用 LineTo 画出显示图像的不规则多边形，最后随

便找出多边形内的一点用 FloodFill 使之成为白色实心多边形。随着扇形角度的增加就实现了扇形旋转展开。其中需要用到 Tan 和 ArcTan 数学函数，要在 implementation 下添加 uses Math。

三、由远及近的效果

由远及近效果的实现主要利用 StretchDraw 方法在一个较小的矩形内画出原图的缩图。再利用时钟控件使矩形区域逐步增大，这样就出现了由远及近的效果。当然，由于画缩图的小矩形的中心位置的不同，同样使用 StretchDraw 而得到的视觉效果是不同的：中心在显示窗口的中心时，看起来更像膨胀（或爆炸）的效果；中心在显示窗口的左（或右）上角看起来才更像由远及近的效果。

当然，这种效果可以与圆展开或扇形展开相结合，实现以圆形膨胀或扇形展开膨胀。

由于使用了 StrechDraw 方法，因此这种方法可以用于长、宽均比显示窗口大或小的静态图像的演播，当然该静态图像的长宽比应该与显示窗口的长宽比相近，否则 StrechDraw 出来的图像就会扭曲变形。

四、“雨滴”效果

“雨滴”效果的实现相当简单，不过对它的理解颇有点窍门。下面是垂直方向上的“雨滴”效果的源程序：

```
procedure TForm1.Timer5Timer(Sender: TObject);
var
  a: integer;
begin
  //Slidey(全局变量)初始值为 Image1.Height
  //dh(全局变量)为“雨滴”的厚度，设为 Trunc(Image1.Height/40)
  for a:=1 to Trunc(Slidey/dh) do
    Image1.Canvas.CopyRect(Rect(0,(a-1)*dh,Image1.Width,a*dh),
Bitmap1.Canvas, Rect(0, Slidey-dh, Bitmap1.Width, Slidey));
  Slidey:=Slidey-dh;
  if Slidey<=0 then
    begin
      Image1.Canvas.Draw(0,0,Bitmap1);
      Timer5.Enabled:=False;
    end;
end;
```

利用 CopyRect 将待显示的“雨滴”（小厚度 dh 的矩形区域）从原图拷贝至显示窗口，并使该“雨滴”从上到下移动（for 循环），直到它在显示窗口中的准确位置。并且在移动的过程中不进行黑底的填充刷新，这样累积的效果就像雨水连续下落。

这种方法适用的图像一方面长宽要小于等于显示窗口，另一方面图像内容的颜色反差要比较大，否则“雨滴”效果就不明显。

当然，还有横向的“雨滴”效果，总的来看横向的“雨滴”效果要比垂直方向的好看。

五、“百叶窗”效果

“百叶窗”效果的实现相当简单。如果是水平的“百叶窗”效果，那就在 y 方向不同位置上多次使用 CopyRect 方法，并通过时钟使拷贝的矩形区域由小变大，这样看起来的效果就像百叶窗逐渐下拉的过程。如果是垂直的“百叶窗”效果，那只需在 x 方向不同位置上多次使用 CopyRect 方法。

六、“马赛克”效果

“马赛克”效果的实现主要利用 CopyRect 方法和随机函数 Random，在不同位置一次产生多个矩形小方块，并且利用时钟多次产生。这样看起来的效果就是一副图像不断以块状涌现。

静态图像的演播主要就是以上几种效果。当然还有其它较为复杂的“渐隐”、“渐现”，以及和“拉”效果类似的“拼接”，在此就不作介绍了。以上几种效果的实现中使用的时钟触发间隔为 100 - 200ms 可以根据各自 PC 的配置进行调整。使用时钟控件的一个好处是它的触发是后台式的，也就是说你可以在程序完成主要功能的同时，利用时钟控件来演播美观华丽的背景图案。

（收稿日期：2000 年 11 月 16 日）

上接第 25 页

四、结束语

该报表打印实例在 BCB5 下编译通过，它给出了在 BCB 中编程控制报表输出的范例，能较好地解决一些特殊报表打印的问题。当然，在 Delphi 中同样可以做到这些。以上是本人在编程中的一些心得，希望能解决广大朋友遇到的类似的问题。

参考文献

1. 刘前进等著 . DELPHI 数据库编程技术 . 北京：人民邮电出版社，1999

2. Charlie Calvert，et al 著，徐可等译 . Borland C++ Builder 应用开发大全 . 北京：清华大学出版社，1999

3. C++ Builder5 联机帮助文件

（收稿日期：2000 年 11 月 3 日）

Unix 下的调度程序的编写

柳 玉 郭吉平 谷艳玲

在 Unix 系统中，有一个程序，它维护进程的执行优先级，进程的行为和用户对优先级的要求，称为 scheduler 调度进程 。然而，在有些应用中，也需要类似调度进程的程序，自动根据一定的条件，程序不断运行一些进程，作一些作业，同时检测并杀死挂死的进程。系统提供的调度就不能满足要求了。这时就需要我们编写一个调度程序，负责启动进程，杀死挂死进程。

调度进程的主要任务就是：按需要启动进程，杀死挂死进程。

先谈度进程如何启动进程：

先看一下进程的作业环境：进程产生时，其进程图像会包括三大部分，如图：

图 1

在文本区 Text Region ，存放需要让 CPU 执行的命令。

在用户数据区 User Data 存放进程本身的数据 系统数据区 System Data 存放有关系统需要知道的信息。比如：进程的拥有者的用户标识符，进程标识符 PID ，进程的父进程标识符等等。

FORK 系统调用会产生一个与原进程相同的新进程，称为子进程，并会响应子进程的 PID 给父进程，而响应 0 给子进程，子进程会去执行父进程的剩余部分 因为它继承了父进程的图中所有三部分数据 ，待结束后回到原父进程。

如果 fork 的返回值为正数，那么就是在父进程中运行，并且该返回值为子进程的进程号 PID；如果返回值为 0，那么是在子进程中执行，如果返回值为负数，则表示有错误产生，需检查错误代码 errno。

而 EXEC 系统调用会产生一个新进程取代原进程的执行，新进程将保留原进程的系统数据区，故其 PID 与原进程相同，原先的 file descriptor table，环境变量，优先级都保持与原进程相同。

EXEC 系统调用包括 execl ，execlp ，exec ，execle ，execv ，execvp 等库函数，当 EXEC 调用返回时，如果有错误发生，错误代码 errno 标记了错误原因。

这样 FORK，EXCL 两者结合使用，调度进程利用 fork 派生出子进程，在此子进程中可以用 exec 系列函数来运行想启动的进程，在父程序中利用循环启动子进程，在子进程中运行指定的程序，从而实现多个进程的并发

```
for (jhjx = 1; jhjx + +; 30)
{
    pid = fork();
    printf("pid = %d\n", pid);
    if (!pid)    /* 在子进程中启动进程 */
     {
      execl("/yhsj/yhsj/bin/center", jhjx);
/* 如果 exec 函数返回, 表明没有正常执行命令, 打印错误信息 */
      perror('progress is failed! \n');
      exit(1);
     }
 else if (pid>0) /* 在父进程中记录启动信息 */
 {
sprintf(s_id, "%d", pid);   /* 此 pid 是子进程的 ID 号 */
            fprintf(fp, s_id);
}
else
{
      perror('fork failed!');
      exit(errorno);
}
}
```

另外，有一个更简单的执行其他程序的函数 system，它是一个较高层的函数，实际上相当于在 SHELL 环境下执行一条命令，而 exec 系统调用则是较低层的调用。

如何查询调度进程启动的进程中，哪些正在运行，哪些已经结束运行，哪些已经 ‘吊死 ”

利用 UNIX / LINUX 的系统调用 ps - u user 就可以得到此 user 用户启动正在运行的进程

```
ps - u user
PID  TTY  TIME COMMAND
69    01   0 01    csh
76    01   0 01    ps
102   02   0 01    . /center
```

在调度进程中，仍然使用 fork 和 exec 系统调用，来获得正在运行的进程的信息

```
if (!pid)    /* 在子进程中启动进程 */
{
      execl("/bin/ps", " – u yhsj >ly");
/* 如果 exec 函数返回, 表明没有正常执行命令, 打印错误信息 */
```

```
        perror(´progress is failed! \n´);
        exit(errorno);
          }
         else if (pid>0) / * 在父进程中等待子进程返回后, 处理ps 命令返回的结果 * /
         {
              wait(& rtn);
        fp = open(ly, ˝r˝); / * 获得正在运行的进程的信息 * /
while () / * 循环读取, 将进程名写进˝运行进程字符串˝中 * /
              {
              }
        }
        else
        {
              perror(´fork failed! ´);
              exit(errorno);
        }
```

注意 上面的程序中有个父程序和子程序的同步问题。父程序需要等待子程序运行结束后继续运行。在这里用 wait & rtn 调用，rtn 是子程序的返回值。

但是，这个办法比较麻烦，比较简单的办法是利用文本文件记载每次启动的信息，供下次使用。

对于正在正常运行的进程，无疑不需再次启动它。但首先应该检查一下进程是否在正常运行，是不是挂死了。

判断挂死进程的办法有很多，这里介绍一简单办法：让进程打开一文本文件 也可以是数据库文件），定时写些信息用于表明它在运行。调度进程只需要定时查询这些信息。如果此信息不再随时间的推移而改变，那么此进程就可以认为是死进程。例如进程运行时，定时向文件中写数字或字符，调度进程只需要读取文件中的标志，并比较相邻两次的读取结果，如果调度进程读取的时间间隔大于进程运行时向文件写标志的时间间隔，那么在这个时间间隔内，进程应该改变文本文件中的标志，如果文件中的标志不变，可以认为进程死掉了。

在进程中

```
while (1)  / * 每次循环写一次运行标志 * /
{
. . .
fp = open(˝run. ini˝, wr);
fgets(fp, ˝& bz);
if (bz = 10000)
       bz = 0;
else
       bz = bz + 1;
fputs(fp, bz);
fclose(fp);
}
```

在调度进程中

```
 if (bz_old = = bz_now)  / * 两次读取的标志相同 * /
 kill(kill_id, SIGKILL);       / * 杀死死进程 * /
```

bz_old 是上次读取的进程运行时写的标志，bz_now 是本次调度进程运行得到的标志，两者不同，那么程序在运行着，如果相同，说明被控进程没有正常运行，即是死进程，可以杀掉。调度进程的读标志 2 个周期内，标志不能循环出现，以防止出现错误判断!

注意：有些时候调度进程需要运行许多进程，发送许多消息，会运行较长一段时间，而在这段时间内，有的子进程已经结束运行退出了，这时候就会产生 ‘僵尸 ” 进程。

一但一个进程处于 ‘僵尸 ” 状态，它不享受任何资源也不去做任何事。一直等到其父进程接到 exit 信号并加以确认后，才真正死亡，这些僵尸将自动被系统释放。因此建议调度进程每次运行完毕后，正常退出。等需要时，再次启动，而不采用循环等待处理的办法。

对挂死的进程应该在 ‘运行进程字符串 ” 中，把它删除。并且在启动每个进程前，检查它的名字是否在 ‘运行进程字符串 ” 中。

调度进程结束后，它产生的子进程很可能还在运行。这些子进程就成了 ‘孤儿 ” 进程，由 init 进程认领，这些孤儿进程还在正常运行，直到子进程正常结束。

调度进程一般应该在后台自动运行，类似精灵进程，它应该与控制台无关，我们可以利用 UNIX 的 CRON 机制，定时启动调度进程。例如

将 00 05 10 15 20 25 30 35 40 45 50 55 * * * * * sh - c /yhsj /yhsj /bin /getdata 写入脚本，用crontab + 脚本文件名登记作业后，系统会每隔 5 分钟启动一次调度我们编写的进程。

有时候，一次调度时间很长，可能大于调度进程的启动周期，没等上次调度完成，下次调度又起了，为防止这种情况还得防止进程的重入。

如何防止进程的重入

这里介绍个简单的办法：首先让进程去锁定某一文本文件，在第一次运行时，调度进程无疑锁定这个文件，在第二次运行此进程时，如果再锁定那个文本文件时，因第一个进程没退出，文件没有解锁，第二次运行，进程就不能再次锁定，不能锁定，说明有此进程在运行，就不需要运行此程序了。

调度进程的运行之初，先锁定指定文件，若不成功，说明调度进程正在运行，调度进程应该向指定设备发布消息，告诉维护人员这种情况并返回。如果这种情况总发生，就应该考虑延长调度进程的启动周期，或检查一下什么原因使得调度进程不能正常结束运行。

```
  fp = open(FileName, O_WRONLY | O_NDELAY, 0777);
      if (flock(fp, LOCK_EX | LOCK_NB) < 0) exit(0) ;  / *
Already Lock * /
```

在调度进程退出前，应该解锁它锁定的文件。这样，调度进程下次可以正常运行。

```
       flock  fp  LOCK_UN       / * 解锁文件 * /
       close  fp      / * 关闭文件 * /
```

在下面列出一个调度进程的简单例子

```
#define   PROC_NUM   100
#include  <stdio. h>
#include  <string. h>
#include  <stdlib. h>
#include  <unistd. h>
```

```
#include <fcntl.h>
#include <signal.h>
#include <sys/file.h>
extern errno;
int main(int argc, char * argv[])
{
    int bz_old[PROC_NUM], bz_new[PROC_NUM], proc_id
[PROC_NUM];
    int id, i, j, run_num, old_num, fd;
    char old_file[PROC_NUM] [50], run_file[PROC_NUM]
[50], tmp[256], buf[256];
    pid_t pid=1;
    FILE * fp;
    /* 防止调度进程重入 */
if ((fd=open("old.ini", O_WRONLY|O_NDELAY, 0777))<0)
    {
      printf("文件 old.ini 不存在!\n");
      exit(0);
    }
    id=flock(fd, LOCK_EX | LOCK_NB);
    if (id<0)
      {
        printf("调度进程因重入而关闭!\n");
        close(fd);
        exit(0);
      }
    flock(fd, LOCK_UN);
      close(fd);
/* 取上次进程的状态信息 格式: PROC_ID, BZ; NAME */
    if ((fp = fopen("old.ini", "r"))==NULL)
    {
      printf("文件 old.ini 不存在!\n");
      exit(0);
    }
      i=0;
      while(!feof(fp))
      {
          fgets(tmp, 256, fp);
          if (strchr(tmp, ';')==NULL) break;
     tmp[strlen(tmp)-1]='\0'; /* 去掉尾部回车 */
          strcpy(old_file[i], strchr(tmp, ';')+1);
          strcpy(buf, strchr(tmp, ',')+1);
          proc_id[i]=atol(buf);
        bz_old[i]=atol(tmp);
          i++;
          if (i==PROC_NUM)
          {
          printf("文件 old.ini 中进程数目过多!\n");
          exit(0);
        }
        strcpy(tmp, "");
      }
      fclose(fp);
      old_num=i;
      /* 取上次运行进程的当前运行的标志 */
      for (i=0; i<old_num; i++)
      {
          if ((fp=fopen(old_file[i], "r"))==NULL)
          {
          printf("文件%s 不存在!\n", old_file[i]);
          exit(0);
          };
          fgets(tmp, 10, fp);
          bz_new[i]=atol(tmp);
  if (bz_new[i]==-1)/* 子进程调用结束时, 标志为 -1 */
          {
          strcpy(old_file[i], "");  /* 复位 */
          bz_new[i]=0;
          }
          fclose(fp);
    }
     /* 读取要运行的进程 */
    i=0;
     if ((fp=fopen("run.ini", "r"))==NULL)
     {
        printf("文件 run.ini 不存在!\n");
        exit(0);
     };
     while(!feof(fp))
     {
          fgets(tmp, 256, fp);
          if (strlen(tmp)<2) break;
            tmp[strlen(tmp)-1]='\0';  /* 去掉尾部回车 */
          strcpy(run_file[i], tmp);
          i++;
          if (i==PROC_NUM)
          {
          printf("文件 run.ini 中进程数目过多!\n");
          exit(0);
       }
       strcpy(tmp, "");
     }
    fclose(fp);
    run_num=i;
    /* 杀死挂死进程 */
    for (i=0; i<old_num; i++)
    {
  if (bz_old[i]==bz_new[i])  /* 两次读取的标志相同 */
          {
          kill(proc_id[i], SIGKILL);
          strcpy(old_file[i], "");
        }
     }
        /* 执行欲运行的进程 */
    for (i=0; i<run_num; i++)
    {
          /* 与正在运行的进程比较 */
          for(j=0; j<old_num; j++)
          {
              if (strcmp(run_file[i], old_file[j])==0)
              break;
          } /* END FOR */
          if (strcmp(run_file[i], old_file[j])!=0)
          {
              pid=fork();
            printf("pid=%d\n", pid);
```

在 Win95 / 98 / ME 下物理硬盘的访问

王　京

摘　要　直接读写硬盘的操作在 Windows 系列的操作系统中可以说是一个“神秘”的话题，但又是磁盘工具所不可缺少的模块。本文说明了如何在 Windows 95 / 98 / ME 的应用程序中对硬盘进行直接读 / 写的操作。在其中，还涉及到 Thunk Layer 技术的讨论，对于 Windows 磁盘工具编程及其它涉及到 Thunk Layer 的应用具有一定的参考意义。

关键词　Thunk Layer　DLL　Int 13h

一、直接访问物理硬盘的意义

直接访问物理硬盘，对磁盘的绝对扇区的读 / 写是很多磁盘工具软件实现的基本模块。在 DOS 时代，磁盘的绝对读写并不是什么难事，磁盘工具软件只要调用 BIOS 的 INT13H 功能就可以了。时过境迁，随着 Windows 95 / 98 的应用以及最近 Windows ME 的推出，工具软件也渐渐转移到了 Windows 平台上。但是，磁盘工具却有着一定的特殊性。由于 16 位内核代码的实际存在和 32 位驱动程序代码的使用，直接访问磁盘变成了一个比较复杂的问题。

在这个问题上，很多著名的磁盘工具公司都采取了回避的态度。出品 Ghost 的 Symantec 就是一例。直到如今，Ghost 也没有 for Windows 的版本 （Windows 下的数据包管理器不能算）。鼎鼎大名的 Patition Magic 也是直到 4. 0 才有 for Windows 的版本，而且某些功能还要返回 DOS 后才能工作。这个问题真有那么困难吗？

二、CreateFile 的局限性

要在 Windows 中实现一个操作，我们当然应该先来查一查 API。毕竟 API 是 Windows 程序的立足之本嘛。首先映入我们眼帘的是一个“看起来很美”的函数：CreateFile。虽然名字看起来怪怪的，但它可以用一个很好听的参数：“PHYSICALDRIVEx”。没想这么简单！不过先别急，再看一看，后面还有一行小字：

> Windows 95 这个技术不能打开一个物理驱动器。在Windows 95 中使用这个字符串将导致 CreateFile 产生一个错误。

还说明了在 Win 95 中使用这个函数来达到访问物理硬盘的目的是不可行的。同样地，这个方法在 Win98 / ME 也不行。但是，在 Windows NT / 2000 下，使用这个函数倒是可以的。可是，我们的目标并不是 NT 平台，所以还得另想办法。

三、回归 INT 13H

可以看出，Windows 95 / 98 / ME 下直接访问物理磁盘不是一件容易的事。这是由于操作系统的设计决定的。Windows 95 / 98 / ME 在内核结构上与和它们的长相相似的兄弟 Windows NT / 2000 有着本质的区别。NT / 2000 系统全是 32 位的，在操作系统建立起来以后，所有的驱动程序都是 32 位的，没有 16

```
            if (!pid)    /* 在子进程中启动进程 */
        {
            execl("/yhsj/yhsj/bin/center", "");
/* 如果 exec 函数返回, 表明没有正常执行命令, 打印错误信息 */
            perror("progress is failed! \n");
                exit(1);
        }
        else if (pid>0) /* 在父进程中记录启动信息 */
            {
    bz_new[i] =0;    /* 计下进程初始标志 */
    proc_id[i] =pid;  /* 此 pid 是子进程的 ID 号 */
        }
            else
            {
            perror("fork failed!");
            exit(errno);
            }
        }
        else
        {
bz_new[i] =bz_old[j];  /* 将旧的进程运行标志计下来 */
        }
    } /* END FOR */
     /* 写下进程的运行信息 */
     fp =fopen("old. ini", "wt");
    for(i =0; i <run_num; i + +)
    {
 fprintf(fp, "%d, %d; %s\ n", bz_new[i], proc_id[i], run_file
[i]);
    }
    fclose(fp);
   exit(0);
}
```

说明：本程序在 Redhat Linux 6. 2 和 Digital UNIX 下调式通过。

收稿日期：2000 年 11 月 24 日）

位的驱动程序，16 位的 BIOS 代码也没有任何作用，不会被执行到。而 Windows 95 /98 /ME 系列则由于和“老祖宗”Windows 3. 1 有着一脉相承的血统，所以直到如今，内核的 16 位代码一天也没有消失过。还记得 95 时的实模式光驱驱动吗？那只是浮在水面的冰山一角。不要相信那些迷人的广告语，16 位的代码在 95 /98 /ME 的操作系统中存在着并将一直存在下去。

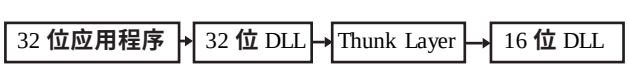
说了这么多，笔者只是想陈述这样一个观点：在 Windows 95 /98 /ME 中，使用 16 位的代码是一件非常正常的事情，尤其是在像磁盘操作这样的地方。因为这里本身就有很多 16 位的代码在运行。接受了这样的事实，使用 INT 13H 就名正而言顺了。但是，我们的程序界面还是要用 32 位的代码来写，所以下面问题就集中在了：怎样在 32 位的代码中使用 16 位代码，从而调用 INT 13H？这就涉及到了下一个主题：Thunk。

四、Thunk Layer

在 32 位的程序中使用 16 位的功能代码并不简单轻松。问题主要在于 32 位的函数在调用 16 位函数时，如何传递参数？如何进行必要的转换？这些如果都由我们来完成，显然将会是一件很麻烦的事。不过，幸好在于一个前面已经提到过的事实：Windows 自身就包含有很多的 16 位代码。这意味着什么呢？这就是说，Microsoft 必然会有一种解决方案，使他自己的大量代码的转换工作不至于太辛苦。这个方案就是 Thunk Layer。在 Win95 中，Thunk Layer 可以使 32 位代码调用 16 位代码，也可以使 16 位代码调用 32 位代码。顺便说一句，NT 中也有 Thunk Layer，不过只能从 16 位代码调用 32 位代码。不在我们这里讨论之列。

Thunk Layer 实际上是一个 Thunk 编译器 （Thunk Compiler）和一个函数库 （Library）。Thunk 编译器的作用是把 Thunk 脚本编译为相应的接口函数的汇编语言代码。所以，从理论上来说，Thunk 编译器不是必要的，只要能够写出必要的汇编代码就可以。不过，比较来说，还是写 Thunk 脚本要容易得多。

总的来说，含有 32 位到 16 位调用的程序的结构如下图所示：



五、我们需要的工具

为了实现这个程序结构，我们需要的工具是比较复杂的。如果想都用微软的产品的话，以下所有的工具必须齐备：

Microsoft Visual C + + 6. 0 （4. 0 以上即可，用于 32 位代码部分）

Microsoft Visual C + + 1. 5x （用于写 16 位的 DLL）

Microsoft Windows 98 DDK （用 Win 95 DDK 也可以）

最后的 98 DDK 里面有三样有用的工具，分别是：

MASM 6. 1x （用于汇编 Thunk Compiler 的结果）

Thunk Compiler （编译 Thunk 脚本）

BIN16 hRC （这个 RC 是很不一样的！它有一个 - 40 的参数！）

当然，如果你用其它公司的编程工具，也是可行的，而且可能还会减少一些工具的使用。比如使用 Borland C + + 5. 0 时，由于它既可以输出 32 位代码，也可以输出 16 位代码，所以一个编译器就够了。而且其中的 Tlink 程序直接支持一个等价于上面 RC - 40 的参数，所以这个特别的 RC 也不需要了。但是 Thunk Compiler 好像没有什么可以替代的。为了方便，下面将统一以 Microsoft 的工具为例，毕竟是标准嘛……。

六、Thunk 脚本

在准备齐工具之后，我们可以开始了。首先，写出 Thunk 脚本：

```
// 32to16. thk
enablemapdirect3216 = true;
typedef unsigned long   DWORD;
typedef unsigned char   BYTE, * LPBYTE;
typedef bool            BOOL;
BOOL ReadPhysicalSectorMBR(BYTE  bDrive, LPBYTE lp-
Buffer, DWORD  cbBuffSize)
{
  lpBuffer = inout;
}
```

这个脚本很简单，意思很容易看懂。第一行是说，允许从 32 位代码调用 16 位代码。下面是定义了几个数据类型，最后是我们这个例子中唯一的一个 16 位函数的接口说明。这个接口有三个变量，Bdrive 和 cbBuffSize 是单纯的用于输入，而 lpBuffer 用于输入输出。Thunk 编译器规定：对变量不加说明的确省情况是输入。所以，在这里只有 lpBuffer 进行了说明。

关于怎样编译、汇编脚本，我们下面再介绍。这里要说明的是，Thunk 脚本经过编译、汇编之后，会生成两个 . obj 文件——32 位部分和 16 位部分，分别与 32 位 DLL 和 16 位 DLL 连接。容易猜到，实际的接口就藏在这两个 . obj 文件之中。

实际上，在 32 位部分的 . obj 文件中，有一个接口函数名叫 xxx_ThunkConnect32 ，作用是初始化时调入 16 位的 DLL。其他的接口函数与 16 位 DLL 中的输出 （DLL Export）函数一一对应，并且同名，与 32 位的用户 DLL 连接在一起，被用户程序调用。另一方面，16 位部分的 . obj 文件，也同样地与 16 位的用户 DLL 连接在一起，也相应的有一个接口函数名叫 xxx_ThunkConnect16 ，在初始化时会被调用。

七、32 位 DLL 部分

关于怎样在 32 位应用程序中调用 32 位 DLL 的问题各种 Win32 编程的资料中都很常见，就不在这里讨论了，下面直接介绍怎样写 32 位 DLL 部分。

```
// DLL32. c
// prototype for function in . obj file from the thunk script
```

```
BOOL WINAPI thk_ThunkConnect32(LPSTR lpDll16, LPSTR lpDll32, HINSTANCE hDllInst, DWORD dwReason);
BOOL WINAPI DllMain(HINSTANCE hDLLInst, DWORD dwReason, LPVOID lpvReserved)
{
  if (!thk_ThunkConnect32("DLL16.DLL", "DLL32.DLL", hDLLInst, dwReason))
  {
    return FALSE;
  }
  switch (dwReason)
  {
    case DLL_PROCESS_ATTACH:
      break;
    case DLL_PROCESS_DETACH:
      break;
    case DLL_THREAD_ATTACH:
      break;
    case DLL_THREAD_DETACH:
      break;
  }
  return TRUE;
};
// Prototype for function in 16-bit DLL.
BOOL FAR PASCAL ReadPhysicalSectorMBR(BYTE bDrive, LPBYTE lpBuffer, DWORD cbBuffSize)
_declspec(dllexport) BOOL WINAPI TestReadMBR()
{
  char lpBuff[512];
  BOOL fResult;
  // Read the first sector of the first hard disk.
  fResult = ReadPhysicalSectorMBR(0x80, lpBuff, 512);
  if (fResult)
    {
     // lpBuff contains the sector data. Use it here
    }
  return fResult;
}
```

这里，thk_ThunkConnect32 函数上面已经提到过。这个函数的主体处在由脚本生成的 .obj 文件中。调用它时使用的参数“DLL16.DLL”和“DLL32.DLL”分别是 16 位 DLL 和 32 位 DLL 的文件名。ReadPhysicalSectorMBR 函数是在 16 位 DLL 中实现的，实际并不能在这里直接调用。但是，Thunk 脚本编译后产生的 32 位 .obj 中含有与这个函数同名的另一个函数（接口代理函数），从程序员的角度看，这样就可以调用这个函数了。TestReadMBR 函数是 32 位 DLL 的主体，当然，在这里它只是简单的调用了 ReadPhysicalSectorMBR 这个函数。

八、16 位 DLL 部分

16 位代码是我们的目的所在，自然是有着重要的意义。还是让我们先看代码：

```
// DLL16.c
// prototype for function in .obj file from the thunk script
BOOL WINAPI _export thk_ThunkConnect16(LPSTR lpDll16, LPSTR lpDll32, WORD hInst, DWORD dwReason);
BOOL WINAPI _export DllEntryPoint(DWORD dwReason, WORD hInst, WORD wDS, WORD wHeapSize, DWORD dwReserved1, WORD wReserved2)
{
  if (!thk_ThunkConnect16("DLL16.DLL", "DLL32.DLL", hInst, dwReason))
  {
    return FALSE;
  }
  return TRUE;
}
// Converts two BYTEs into a WORD. This is useful for working with a RMCS, but was not provided in WINDOWS.H.
//#define MAKEWORD(low, high) \((WORD)((((WORD)(high)) << 8) | ((BYTE)(low))))
#define SECTOR_SIZE 512// Size, in bytes, of a disk sector
#define CARRY_FLAG 0x0001
typedef BYTE FAR *LPBYTE;
typedef struct tagRMCS
{
  DWORD edi, esi, ebp, RESERVED, ebx, edx, ecx, eax;
  WORD wFlags, es, ds, fs, gs, ip, cs, sp, ss;
} RMCS, FAR* LPRMCS;
BOOL FAR PASCAL SimulateRM_Int (BYTE bIntNum, LPRMCS lpCallStruct);
BOOL FAR PASCAL _export ReadPhysicalSectorMBR(BYTE bDrive, LPBYTE lpBuffer, DWORD cbBuffSize)
{
  BOOL fResult;
  RMCS callStruct;
  DWORD gdaBuffer;// Return value of GlobalDosAlloc().
  LPBYTE RMlpBuffer;  // Real-mode buffer pointer
  LPBYTE PMlpBuffer;  // Protected-mode buffer pointer
  /*
   Validate params:
bDrive should be int 13h device unit -- let the BIOS validate this parameter -- user could have a special controller with its own BIOS. lpBuffer must not be NULL cbBuffSize must be large enough to hold a single sector
  */
  if (lpBuffer == NULL || cbBuffSize < SECTOR_SIZE)
    return FALSE;
  /*
   Allocate the buffer that the Int 13h function will put the sector data into. As this function uses DPMI to call the real-mode BIOS, it must allocate the buffer below 1 MB, and must use a real-mode paragraph-segment address. After the memory has been allocated, create real-mode and protected-mode pointers to the buffer. The real-mode pointer will be used by the BIOS, and the protected-mode pointer will be used by this function because it resides in a Windows 16-bit DLL, which runs in protected mode.
  */
```

```
  gdaBuffer = GlobalDosAlloc (cbBuffSize);
  if (!gdaBuffer)
    return FALSE;
    RMlpBuffer = (LPBYTE) MAKELONG(0, HIWORD (gdaBuffer));
    PMlpBuffer = (LPBYTE) MAKELONG(0, LOWORD (gdaBuffer));
  /*
    Initialize the real-mode call structure and set all values needed to read the first sector of the specified physical drive.
   */
  _fmemset (lpCallStruct, 0, sizeof (RMCS));
  callStruct.eax = 0x0201;          // BIOS read, 1 sector
  callStruct.ecx = 0x0001;          // Sector 1, Cylinder 0
callStruct.edx = MAKEWORD(bDrive, 0); //Head 0, Drive #
 callStruct.ebx = LOWORD(RMlpBuffer); //Offset of sector buffer
callStruct.es = HIWORD(RMlpBuffer); //Segment of sector buffer
  /*
 Call Int 13h BIOS Read Track and check both the DPMI call itself and the BIOS Read Track function result for success. If successful, copy the sector data retrieved by the BIOS into the caller´s buffer.
   */
  if (fResult = SimulateRM_Int (0x13, &callStruct))
    if (!(callStruct.wFlags & CARRY_FLAG))
      {
      _fmemcpy (lpBuffer, PMlpBuffer, (size_t)cbBuffSize);
      fResult = TRUE;
      }
    else
      fResult = FALSE;
  // Free the sector data buffer this function allocated
  GlobalDosFree (LOWORD(gdaBuffer));
  return fResult;
}
BOOL FAR PASCAL SimulateRM_Int (BYTE bIntNum, LPRMCS lpCallStruct)
{
  BOOL fRetVal = FALSE;     // Assume failure
  _asm {
       push di
mov ax, 0300h        ; DPMI Simulate Real Mode Int
mov bl, bIntNum      ; Number of the interrupt to simulate
mov bh, 01h          ; Bit 0 = 1; all other bits must be 0
xor cx, cx           ; No words to copy
les di, lpCallStruct
int 31h              ; Call DPMI
jc END1              ; CF set if error occurred
mov fRetVal, TRUE
END1:
pop di
       }
  return (fRetVal);
}
```

让我们先看这段程序的接口部分。前面已经讲过，thk_ThunkConnect16 函数的主体处在由脚本生成的 .obj 文件中。与 thk_ThunkConnect32 一样，调用它时使用的参数“DLL16.DLL”和“DLL32.DLL”分别是 16 位 DLL 和 32 位 DLL 的文件名。

在实现部分，同样也有一些技巧。虽说是 16 位的程序，调用 INT 13H 也不是直接写上 INT 13H 这条语句就可以的。由于保护模式的限制，我们只能通过 DPMI 来模拟实模式的中断。SimulateRM_Int 函数就是这个功能。由于 DPMI 的使用已经超出本文的讨论范围，所以就不再多说了。我们还是来看一下关于 INT 13H 的使用。

在上面的例子里，我们只使用了 INT 13H 的 02 号子功能。实际上，随着大容量硬盘的使用，02 号子功能以及其它原始的 INT 13H 调用已经被 41 - 49 号子功能所替代。这些子功能被称为 INT 13H Extension。要在程序中使用 INT 13H Extension，要先检查它是否存在。41H 号子功能就是这个作用。42H 和 43H 分别对应硬盘扇区的读和写。不过一般来说，不应该直接对硬盘作读、写操作，而应该先使用 48H 号子功能得到硬盘的参数。在这里，只作一点粗略的介绍，细节请查阅相关资料。

九、目标代码生成

所有代码已经写成之后，我们要开始进行编译、连接的操作。这种 32 位—16 位混合代码的编译还是比较复杂的。步骤很多，我们一步一步来：

1. 对 Thunk 脚本的编译

thunk - t thk 32to16.thk - o 32to16.asm

2. 汇编 Thunk 脚本

32 位部分：ml /DIS_32 /c /W3 /nologo /coff /Fo thk32.obj 32to16.asm

16 位部分：ml /DIS_16 /c /W3 /nologo /Fo thk16.obj 32to16.asm

3. 在 32 位 DLL 的 .DEF 文件 EXPORTS 段中加入一行

thk_ThunkData32

4. 把应该导出的自定义函数添加到 .DEF 文件的 EXPORTS 段中，连接 32 位 DLL，注意在连接时应加入 Thunk 的静态库部分 Thunk32.lib

5. 在 16 位 DLL 的 .DEF 文件 IMPORTS 段中加入

C16ThkSL01 = KERNEL.631

ThunkConnect16 = KERNEL.651

6. 在 16 位 DLL 的 .DEF 文件 EXPORTS 段中加入

THK_THUNKDATA16 @1 RESIDENTNAME

DllEntryPoint @2 RESIDENTNAME

其中的@1，@2 可以是任意的数值。

7. 把应该导出的自定义函数添加到 .DEF 文件的 EXPORTS 段中，连接 16 位 DLL

在 VB 中实现树/列表视界面

李 越

树/列表视是 VC 应用程序中最为常见的一种界面，界面左边一般为树型窗口，右边是列表窗口，两个窗口之间有一个分割条，用户可以通过拉动分割条来调整两个窗口大小。树/列表视界面具有层次清楚、操作方便等特点，在许多专业应用程序中得到广泛应用。

在 VC 程序开发中，这种界面是通过树/列表视类（Tree/List View）和分割条（Splitter）来实现的，而在 VB 中，却没有视类的概念，也没有实现这种界面的现成方法，但是，我们可以利用 VB 所提供的树、网格等控件，通过编程的方法模拟出这种界面，下面具体介绍实现的方法和步骤。

一、创建程序框架

启动 VB6.0，选 Standard EXE 创建新工程，在 IDE 环境中，选菜单 Project→Components...，在弹出的控件对话框中分别选定 Microsoft Windows Common Controls 6.0 和 Microsoft Data Bound Grid Control5.0 SP3，按确定按钮后，就将常用控件组和数据网格控件加入到开发环境的控件面板中，然后将控件面板中的工具条控件、状态条控件、树型控件、数据网格控件、数据库控件、图像列表控件（2个）拖入表单窗体中，其中树型控件用来实现树型窗口，数据网格控件用来实现列表窗口，最后从控件面板中拖入分界控件（Frame），用来实现分割条。表单窗体中各控件的名称和有关属性如附表所示。

控件的其他属性取默认值或自行设定，表单窗体的菜单也可根据需要自行设置。

二、树/列表视界面的实现代码

程序主要要实现当窗口尺寸发生变化时，树/列表窗口的尺寸要随之变化，以及当用户拖动分割条时，树/列表窗口尺寸也应作相应的调整，其中实现树/列表窗口尺寸随主窗口尺寸变化的函数如下（以下两函数置于表单窗体的 General/Declaration 中）：

```
Private Sub Resizebar()
On Error Resume Next
//确定树窗口左上角坐标
TreeView1.Left = 20
TreeView1.Top = Toolbar1.Height + 20
//确定树窗口高度
TreeView1.Height = Form1.ScaleHeight - Toolbar1.Height
 - StatusBar1.Height - 40
//确定分割条的位置和尺寸
Frame1.Width = 60
Frame1.Top = TreeView1.Top
Frame1.Left = TreeView1.Width
Frame1.Height = TreeView1.Height
//确定列表窗口的位置和坐标
listview1.Top = TreeView1.Top
```

附表

控件类型	控件名称	属性名称	属性值
图象列表	ImageList1	General	16×16
图象列表	ImageList2	General	16×16
工具条	Toolbar1	ImageList	ImageList1
状态条	Statusbar1	Panels	SbrTime、sbrNum
树型视窗	TreeView1	ImageList	ImageList2
		Width	2184
数据库控件	Data1	DatabaseName	Nwind.mdb
		RecordSource	Products
数据网格控件	listview1	DataSource	Data1
分界控件	Frame1	BorderStyle	None

8. 把 DLL 标记为 4.0 版，这时，开始提到的特别的 RC 用上了：

bin16\rc -40 DLL16.DLL。

十、完成

这里所列出的只是一些实验用的代码，只能读取硬盘的 MBR 扇区。想把此技术应用于实际程序中，还有一段路要走。也许在整个进行这个过程之中您会遇到很多麻烦，这时，您可以根据下面的参考文献查阅 MSDN 中的相关文章。

当然，这只是一种直接访问硬盘扇区的一种方法，很可能有其它更简单、更快捷的方法，欢迎您提出不同的思路。如果您有什么问题，您也可以通过 E-mail 与我联系。

参考文献

MSDN HOWTO Call 16-bit Code from 32-bit Code Under Windows 95 (ID Q155763)

MSDN PRB DeviceIoControl Int 13h Does Not Support Hard Disks (ID Q137176)

MSDN Platform SDK Win32 API - Thunk Compiler

MSDN Windows 95 DDK Int 13 Extension APIs

（收稿日期：2000 年 10 月 10 日）

```
listview1. Left = TreeView1. Width + Frame1. Width
listview1. Height = TreeView1. Height
listview1. Width = Me. ScaleWidth – listview1. Left – 40
End Sub
```

其中实现树/列表窗口尺寸随分割条移动而变化的函数如下：

```
Private Sub Movebar()
On Error Resume Next
//根据分割条位置调整树窗口的尺寸
TreeView1. Move 20, TreeView1. Top, Frame1. Left – 20,
TreeView1. Height
//根据分割条位置调整列表窗口的尺寸
listview1. Move Frame1. Left + Frame1. Width, listview1.
Top, Me. ScaleWidth – Frame1. Left – Frame1. Width –
40, listview1. Height
End Sub
```

程序运行的主要原理如下：

当程序运行时，表单窗体被载入，以下过程被调用。

```
Private Sub Form_Load()
Resizebar//调用 Resizebar()函数初始化程序主窗口界面
//往树型窗口中添加内容
TreeView1. Nodes. Add , , "R", "Red", 2
TreeView1. Nodes. Add , , "G", "Green", 2
TreeView1. Nodes. Add , , "B", "Blue", 2
TreeView1. Nodes. Add "R", 4, , "1234", 1
TreeView1. Nodes. Add "R", 4, , "1234", 1
TreeView1. Nodes. Add "R", 4, , "1234", 1
TreeView1. Nodes. Add "G", 4, , "1234", 1
TreeView1. Nodes. Add "G", 4, , "1234", 1
TreeView1. Nodes. Add "G", 4, , "1234", 1
TreeView1. Nodes. Add "B", 4, , "1234", 1
TreeView1. Nodes. Add "B", 4, , "1234", 1
TreeView1. Nodes. Add "B", 4, , "1234", 1
End Sub
//当主窗口尺寸发生变化时,以下过程被调用
Private Sub Form_Resize()
Resizebar //调用 Resizebar()函数根据变化了的主窗口而调整相应树/列表窗口尺寸
End Sub
```

响应 Frame1 的 MouseMove 事件以移动工具条.

```
Private Sub Frame1_MouseMove(Button As Integer, Shift As
Integer, X As Single, Y As Single)
 Frame1. MousePointer = 9 //将鼠标形状置为双箭头形
If Button = 1 Then //假如鼠标左键按下
 Frame1. Left = X + Frame1. Left //移动 Frame1(分割条)
End If
End Sub
```

响应 Frame1 的 MouseUp 事件停止移动 Frame1（分割条），并调用 Movebar 函数调整树/列表窗口尺寸。

```
Private Sub Frame1_MouseUp(Button As Integer, Shift As
Integer, X As Single, Y As Single)
Movebar
End Sub
```

响应窗体菜单事件。

```
Private Sub mcopy_Click()
    TreeView1. SelectedItem. Expanded = Not TreeView1.
SelectedItem. Expanded
End Sub
```

响应窗体按钮事件，退出程序运行。

```
Private Sub mquit_Click()
Unload Me
End Sub
```

响应工具条按钮事件，展开或收缩树形窗口分支。

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSCom-
ctlLib. Button)
    TreeView1. SelectedItem. Expanded = Not TreeView1.
SelectedItem. Expanded
End Sub
```

以上程序主要通过 Resizebar 和 Movebar 两个函数以及对分界控件（Frame1）的两个主要鼠标事件 MouseMove 和 MouseUp 的响应来模拟树/列表视界面有关特性，可以看出 在在 VB 中实现树/列表视界面是比较容易的。

将上述数据网格控件换成文本框、图形框等控件，就可以实现树/文本视、树/图形视等程序界面。

利用同样原理，还可以在 VB 中方便地实现三个或三个以上分割窗口的程序界面。

（收稿日期：2000 年 10 月 11 日）

软件保护新套装与 RG - DL 软件狗

近日，彩虹天地在现有加密狗产品原基础上，又研制、开发并最新推出一款新产品 RG - UMH 套装加密狗和一款升级产品 GS - DL 型软件狗。

新的 RG - UMH 套装有微狗、USB 狗和一根 USB 线组成，这款产品的最大特点是突破以往并口加密产品与 USB 口产品不兼容的桎梏。在使用时，用户或软件开发商无须重复安装驱动程序，即可在并口狗或 USB 狗间自由切换，体验彩虹田地以用户为中心的理念。

RG - UMH 套装产品是由 GS - MH 微狗和 USB 狗两大部分组成的。其中 GS - MH 微狗是使用在计算机并口上的用于软件的硬件产品。USB 狗是使用在计算机 USB 口上的硬件加密产品。使用 RG - UMH 套装程序，只需安装一次驱动，就可以完成对微狗与 USB 够的加密操作，从而轻而易举的满足最终客户的需求。

对于正在使用微狗或 USB 狗进行软件保护的用户，也只需简单的将现有的软件驱动升级，即可得到相互兼容的并口狗或 USB 狗。

另外，彩虹天地 RG - DL 型软件狗也是使用在计算机并口上的用于软件保护的硬件产品，具有 100 个字节的数据存储区。

RG - DL 型软件狗内部所采用的硬件不同与原 GS - DL 型软件狗，其稳定、可靠性能有了较大的提升；软件方面，RG - DL 型软件狗解决了原 DJ/DK 型软件存在的 DOS16、WIN16 模块在 WINDOWS NT/2000 下使用不稳定的问题。

彩虹天地为软件开发商提供网上免费升级服务，无论是 RG - UMH 套装还是软件狗 RG - DL，您都可以在网上获得相应的升级软件。

用 ActiveX 控件实现目录遍历

余 锋 祝晓鹰

所谓目录遍历，是指检索指定目录及其子目录下的所有文件，比较常用的算法是递归。本文利用 ActiveX 技术，用 Visual C++ 编了一个目录遍历的控件 BrowDir.ocx。ActiveX 控件的优点在于建立在组件对象模型 COM 的基础上，实现了软件模块的二进制连接，封装性强，可广泛用于各种支持 ActiveX 的开发平台，如 VC、VB、VBA Visual Basic for Application 、Delphi 等，用户所要做的只是掌握控件的属性、方法和事件。以下详细介绍控件 BrowDir.ocx 的结构及编写步骤，通过本文，读者可以初步了解编写控件的一般方法。

遍历目录的递归算法很多书上都有，这里不再赘述。

一、控件结构

任何一个 ActivcX 控件，不论其内部结构如何复杂，它与用户的交互都要借助于属性、方法和事件：属性是控件的一个值，可以被设置，也可以被获取；方法跟 C++ 的类成员函数类似；事件从功能上说，有点象 Windows 系统中的消息，控件用它来通知用户，准确地说，事件是由控件调用的用户端函数。本文中的 BrowDir 控件有 2 个属性、1 个方法及 2 个事件：

1. 属性

Dir 字符串类型，设置或返回要遍历的目录，支持相对路径和网络路径 如 “hhhawk hshare h” 。缺省值为空 如果为空，则遍历当前目录 ；

FileType 字符串类型，设置或返回要检索的文件类型，支持通配符 ‘*’和 ‘’，缺省值为 “*.*”。注意：FileType 属性不能带路径，如 “t hhawk h*.cpp”是错误的。

2. 方法

BeginBrowse 使用指定的目录和文件类型开始遍历目录，无参数，返回值类型 Long：返回 0，表示已成功遍历完目录；返回 1，表示用户取消了遍历 参见下文 ‘事件”部分 ；返回 2，说明 Dir 属性给出的目录不存在。

3. 事件

FindFile 控件每找到一个符合 FileType 所指定类型的文件，就会产生此事件。它有两个参数：

strFileName 字符串类型，文件名存放在其中，使用绝对路径；

bCancel 布尔类型，用来确定是否取消遍历。初始值为 False，如果设置成 True，则取消遍历，BeginBrowse 方法返回 1。

FindSubDir 控件每找到一个子目录，就会产生该事件。它有 3 个参数

strSubDir 字符串类型，存放子目录名，使用绝对路径；

strParentDir 字符串类型，存放该子目录的上一级目录名，使用绝对路径；

bCancel 布尔类型，用来确定是否取消遍历。初始值为 False，如果设置成 True，则取消遍历，BeginBrowse 方法返回 1。

顺便提一下，这两个事件都发生在方法 BeginBrowse 的调用过程中。

二、编写步骤

BrowDir 控件是用 MFC 实现的， 类向导 ClassWizard 为我们自动生成了大部分代码，即便是不了解 COM 的用户， 也能在很短的时间内编出一个控件。以下是具体步骤：

1. 创建一个新项目，项目类型 MFC ActiveX ControlWizard，项目名称为 BrowDir，其余使用缺省设置即可。类向导会自动生成 3 个类：

CBrowDirApp 应用程序类；

CBrowDirCtrl 控件类，控件的属性、方法和事件代码都在该类中；

CBrowDirPropPage 用来管理控件的属性对话框。

2. 添加属性

运行类向导，选择 Automation 标签，在 Class name 下拉列表框中选择类 CBrowDirCtrl， 单击 Add Property 按钮，会弹出一个对话框，在 External name 组合框中输入 Dir，Type 设置为 CString，其它保持不变，单击 OK 确认输入并关闭对话框。现在我们已经创建了 Dir 属性，同时类向导还会为类 CBrowDirCtrl 自动生成一个 CString 类型的成员变量 m_dir 及成员函数 OnDirChanged：m_dir 用来保存属性值，OnDirChanged 通知控件属性已改变。

再次单击 Add Property 按钮，在 External name 组合框中输入 FileType，Type 设置为 CString，单击 OK，我们将得到 FileType 属性及成员变量 m_fileType 和通知函数 OnFileTypeChanged。为了通知容器及时刷新属性浏览器，比如 VB 集成开发环境中的属性窗口，需要在函数 OnDirChanged 和 OnFilTypeChanged 中调用 BoundProertyChanged，

代码如下：

程序清单 1 BrowDirCtrl.cpp - 修改后的 OnDirChanged 和 OnFilTypeChanged:)

```
void CBrowDirCtrl::OnDirChanged(
{;
    BoundPropertyChanged(dispidDir)
    SetModifiedFlag();
```

```
}
void CBrowDirCtrl: : OnFileTypeChanged( )
{
BoundPropertyChanged(dispidFileType);
SetModifiedFlag( );
}
```

3. 创建属性对话框

用 VB 设计程序时，定制控件 如工具条 都有自己的属性对话框。本例中，类向导已经为我们生成了一个空白的属性对话框，打开对话框资源 IDD_PROPPAGE_BROWDIR，删除标有 TODO 的静态文本控件，加入两个静态文本控件，标题设为 “初始目录”、 “文件类型”。再加入两个编辑控件，命令 ID 改为 IDC_DIR 和 IDC_FILETYPE，分别放在那两个静态控件的旁边。

接下来创建对话框成员变量：运行类向导，选择 Member Variables 标签，选 CBrowDirPropPage 类，Control IDs 列表框选择 IDC_DIR，单击 Add Variable 按钮，Member variable name 编辑框中输入变量名 m_strDir，为了把该变量和控件的 Dir 属性关联起来，在 Optional property name 框输入 Dir，其它保持不变，单击 OK 确认。用同样的方法为 IDC_FILETYPE 创建成员变量 m_strFileType，并和 FileType 属性相关联。

缺省情况下，类向导生成的属性对话框只有一个标签，如果用户想添加标签，参看联机帮助主题：Visual C + + Programmer s Guide、 Adding User Interface Features、Details、ActiveX Control Topics、 ActiveX Controls、 ActiveX Controls Adding Another Custom Property Page。

4. 添加事件

打开类向导，选择 ActiveX Events 标签，选择 CBrowDirCtrl 类，单击 Add Event 按钮，在 External name 框中输入事件名 FindFile，在参数列表框中添加两个参数 strFileName、bCancel，类型为 LPCTSTR、BOOL * 。

同样再创建事件 FindSubDir，它有 3 个参数：strSubDir、strPrentDir 和 bCancel，类型依次为 LPCTSTR、 LPCTSTR、BOOL * 。类向导自动生成两个类成员函数 FireFindFile 和 FireFindSubDir，用于触发事件。

5. 添加方法

打开类向导，选择 Automation 标签，选 CBrowDirCtrl 类，单击 Add Method 按钮，在 External name 框输入方法名 BeginBrowse，Return Type 选 long，单击 OK 确认，类向导自动生成成员函数 BeginBrowse 的框架，加入下面的代码：

程序清单 2 BrowDirCtrl. cpp - 修改后的 BeginBrowse:

```
. . .
#include <sys\stat. h>
#include <io. h>
. . .
long CBrowDirCtrl: : BeginBrowse( )
{
    //把目录转换为绝对路径
    char szFullDir[_MAX_PATH];
    if (_fullpath(szFullDir, m_dir, _MAX_PATH) = = NULL)
        //目录不存在
        return BD_ERROR;
    //去除路径结尾的´\´
    int len;
    len = strlen(szFullDir);
    if (szFullDir[len - 1] = = ´\\´)
    szFullDir[len - 1] = 0;
    //判断目录是否存在
    struct _stat fileinfo;
    if (_stat(szFullDir, & fileinfo) = = 0)
    {
        //进一步判断是目录还是文件
        if (! (fileinfo. st_mode & _S_IFDIR))
            //不是目录
            return BD_ERROR;
    }
    else
    {
//在路径结尾加上´\´后，再判断一次. 注意，加上´\´后，就不必再判断是否是文件了.
        strcat(szFullDir, ″\\″);
        if (_stat(szFullDir, & fileinfo) = = -1)
            //目录不存在
            return BD_ERROR;
    }
//如果路径的最后一个字母不是´\´，则在最后加上一个´\´
    len = strlen(szFullDir);
    if (szFullDir[len - 1] ! = ´\\´)
        strcat(szFullDir, ″\\″);
//调用自定义函数 BrowseDir 遍历目录，当遇到子目录时，BrowseDir 会对自己进行递归调用.
    return BrowseDir(szFullDir, m_fileType);
}
```

方法 BeginBrowse 用到一个成员函数 BrowseDir 和一些枚举常量，需要自己定义：

程序清单 3 BrowDirCtrl. h - 函数 BrowseDir 和枚举常量的声明：

```
class CBrowseDirCtrl : public COleControl
{
. . .
protected:
enum {BD_OK, BD_CANCEL, BD_ERROR};
//遍历目录 dir 下由 filetype 指定的文件. 对于子目录，采用迭代. 如果返回 BD_CANCEL，表示用户中止遍历文件.
long BrowseDir(const char * dir, const char * filetype);
. . .
```

程序清单 4 BrowDirCtrl. cpp - 函数 BrowseDir：

```
long CBrowDirCtrl: : BrowseDir(const char * dir, const char * filetype)
{
//首先查找 dir 中符合要求的文件
char fullfiletype[_MAX_PATH];
strcpy(fullfiletype, dir);
strcat(fullfiletype, filetype);
long hFile;
```

```
 _finddata_t fileinfo;
if ((hFile = _findfirst(fullfiletype, & fileinfo)) ! = -1)
{
do
{
//检查是不是目录,如果不是,则进行处理
 if (!(fileinfo. attrib & _A_SUBDIR))
{
char filename[_MAX_PATH];
strcpy(filename, dir);
strcat(filename, fileinfo. name);
//触发 FindFile 事件.
 //如果 bCancel 被置为 TRUE,结束遍历
BOOL bCancel = FALSE;
FireFindFile(filename, & bCancel);
if (bCancel)
return BD_CANCEL;
}
} while (_findnext(hFile, & fileinfo) = = 0);
 _findclose(hFile);
}
//查找 dir 中的子目录
strcpy(fullfiletype, dir);
strcat(fullfiletype, "* . * ");
if ((hFile = _findfirst(fullfiletype, & fileinfo)) ! = -1)
{
do
{
//检查是不是目录,如果是,再检查是不是 . 或 . .
//如果不是,进行迭代
if ((fileinfo. attrib & _A_SUBDIR))
{
if (strcmp(fileinfo. name, " .") ! = 0 & & strcmp
(fileinfo. name, ". . ") ! = 0)
{
char subdir[_MAX_PATH];
strcpy(subdir, dir);
strcat(subdir, fileinfo. name);
strcat(subdir, "\\");
//触发 FindSubDir 事件.
//如果 bCancel 被置为 TRUE,结束遍历
BOOL bCancel = BD_OK;
FireFindSubDir(subdir, dir, & bCancel);
if (bCancel)
return BD_CANCEL;
 if (BrowseDir(subdir, filetype) = = BD_CANCEL)
return BD_CANCEL;
 }
}
} while (_findnext(hFile, & fileinfo) = = 0);
  _findclose(hFile);
}
return BD_OK;
}
```

6. 加入持久性支持

所谓持久性，通俗地说，就是让控件“记住”用户在设计时指定的属性值，并在程序运行时体现出来。MFC 中控件持久性的实现非常简单，只需在控件类的成员函数 DoPropExchange 已由类向导自动生成 中，为每个属性添加一行代码即可：

程序清单 5 BrowDirCtrl. cpp – 修改后的函数 DoPropExchange:

```
void CBrowDirCtrl: : DoPropExchange(CPropExchange * pPX)
{
        ExchangeVersion ( pPX, MAKELONG (_wVerMinor,
_wVerMajor));
 COleControl: : DoPropExchange(pPX);
// TODO: Call PX_ functions for each persistent custom
property.
PX_String(pPX, "Dir", m_dir, "");
PX_String(pPX, "FileType", m_fileType, "* . * ");
}
```

实现持久性时，可以为每个属性指定一个缺省值。上面的代码中， 属性 Dir 的缺省值为空串，表示当前目录；FileType 缺省值为 “* . * ”，表示所有类型。

7. 绘制控件

绘制代码在控件类的成员函数 OnDraw 已由类向导自动生成 中，这点和普通的 MFC 应用程序挺相似。BrowDir 控件没有用户界面，甚至都不必显示出来，因此绘制很简单，只是显示一句话，提醒用户把控件的 Visible 属性设为 False。类向导自动生成的 OnDraw 函数仅仅在控件窗口。

画了一个椭圆，把它修改如下：

程序清单 6 BrowDirCtrl. cpp - 修改后的函数 OnDraw：

```
void CBrowDirCtrl: : OnDraw(CDC * pdc, const CRect &
rcBounds, const CRect& rcInvalid)
            {
               pdc – >FillRect(rcBounds, CBrush: : FromHan-
dle((HBRUSH)GetStockObject(WHITE_BRUSH)));
              pdc – >TextOut(0, 0, "请设置 Visible = False");
            }
```

至此，代码已全部输入完毕，现在要做的就是编译连接，生成 OCX 文件。

切记，编译时，一定要选 Win32 Debug 或 Win32 Release，不能选 Win32 Unicode Debug 或 Win32 Unicode Release。因为本文中的字符串都是以单字节方式处理的，如果按 Unicode 版编译，将导致错误。

三、测试

为了方便起见，本文使用 Visual Basic 做测试平台。以下是一个统计指定目录下的文件个数和子目录个数的例程：

1. 新建一个工程，类型为‘标准 EXE”；

2. 把 BrowDir 控件加入工程：先用 Windows 系统提供的 Regsvr32. exe 注册控件 其实在编译生成 OCX 文件时，VC 已经替我们做了 ，然后在 VB 中选择主菜单 “工程”的‘部件”，弹出 “部件”对话框，勾选 BrowDir ActiveXControl Module，单击‘确定”按钮返回。VB 的工具窗口将增加一个

巧用 VB 的 ActiveMovie 控件制作视频播放器

陈业斌

编程思路

在 Visual Basic 6.0 系统中，能够用来播放视频文件的 ActiveX 控件有：Mmcontrol、MediaPlayer、ActiveMovie 等，相比之下各有所长，但用 ActiveMovie 控件播放扩展名为 AVI 的视频文件，则显得更加灵活。因为它不仅可以控制视频大小，还可以实现全屏播放。

本人所设计的“视频播放器”主要有以下功能：用“CommonDialog”控件调出“打开”对话框，并找到某个特定的文件；用“ActiveMovie”控件进行播放和控制；用“Slider”控件显示和控制播放进度；从下拉列表框中选择视频尺寸可改变视频大小；窗体以及其中的控件尺寸的大小随视频大小的改变而改变。

创作步骤

一、 窗体界面设计

1. 启动 Visual Basic 6.0，创建一个“标准 EXE”工程。VB 自动创建一个名为“Form1”的工程窗口，将窗口调整到适当的大小。

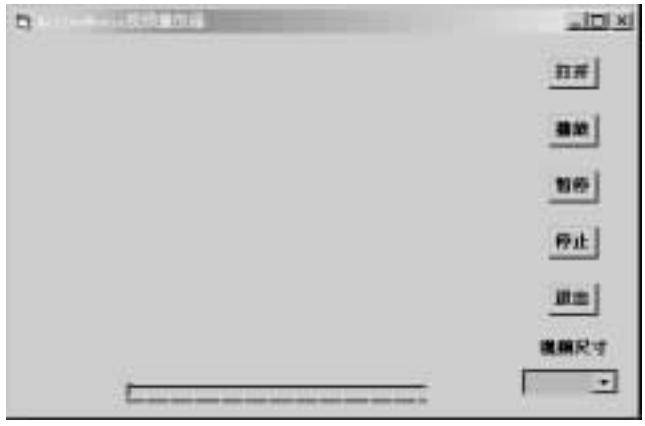
2. 右击工具箱中的空白区，在弹出的快捷菜单中选择“部件”选项，启动“部件”对话框窗口，选择“控件”标签，选中“Microsoft ActiveMovie Control”、“Microsoft Common Dialog Control6.0”和“Microsoft Windows Common Controls 6.0”（用于调用滑块控件）三个选项。单击“确定”按钮，将所选控件添加到 VB 的工具箱中。

3. 双击工具箱中的“ActiveMovie”控件，将其添加到窗体中，用于播放视频文件，并调整其合适的大小和位置。

4. 双击工具箱中的“CommonDialog”控件，将其添加到窗体中，用于显示打开对话框，其大小不可变，位置任意。

5. 双击工具箱中的“Slider”控件，将其添加到窗体中，放在“AcitveMovie”控件的下面，用于显示播放进度。

6. 向窗体中添加 5 个命令按钮控件：Command1、Command2、Command3、Command4、Command5；一个标签控件：Label1；和一个下拉列表框控件：Combo1；其大小和位置如下图。



图标，代表 BrowDir 控件；

3. 在窗体 Form1 上放置一个 BrowDir 控件，如果一切正常的话，控件是一个白色矩形，上面有一行黑字“请设置 Visible = False”。把控件的 FileType 属性设为“*.*”，Visible 设为 False，其余保持不变。再放置一个按钮和一个编辑框，均使用缺省属性，加入下列代码：

程序清单 7 例程：

```
Dim nFileCount As Integer, nDirCount As Integer
Private Sub BrowDir1_FindFile(ByVal strFileName As String, bCancel As Boolean)
nFileCount = nFileCount + 1
End Sub
Private Sub BrowDir1_FindSubDir(ByVal strSubDir As String, ByVal strParentDir As String, bCancel As Boolean)
nDirCount = nDirCount + 1
End Sub
Private Sub Command1_Click()
nFileCount = 0
nDirCount = 0
BrowDir1.Dir = Text1.Text
BrowDir1.BeginBrowse
Print "文件数:"; nFileCount
Print "子目录数:"; nDirCount
End Sub
```

运行程序，在编辑框内输入要统计的目录名，然后单击按钮，调用控件的 BeginBrowse 方法开始遍历目录。统计代码放在事件 FindFile 和 FindSubDir 中。遍历完毕后，输出统计结果。

以上程序均在 Visual C++ 5.0 和 Visual Basic 5.0 上调试通过。

（收稿日期：2000 年 11 月 20 日）

二、 设置控件属性

1. “Slider1”控件的属性在程序代码中进行设置，在窗体界面上不宜静态设置。

2. “Command”和“Label”控件的“Font”的设置为：宋体、粗体、字号为 9，余下控件的属性如下表所示。

	Caption	ActiveMovie 视频播放器
Form1	AutoRedraw	True
	BorderStyle	2 - Sizable
	StartupPosition	2 - 屏幕中心
	Moveable	True
	Height	4680
	Width	7155
Command1	名称	Cmdopen
	Caption	打开
	Heigh	375 其它与此相同）
	Width	615 其它与此相同）
Command2	名称	Cmdplay
	Caption	播放
Command3	名称	Cmdpause
	Caption	暂停
Command4	名称	Cmdstop
	Caption	停止
Command5	名称	Cmdexit
	Caption	退出
Label1	Caption	视频尺寸
	Autosize	True
Combo1	Text	“”
	Style	0 - DropDown Combo
CommonDialog1	Filename	*.avi
	Filter	*.avi
ActiveMovie1	ShowControls	False
	ShowDisplay	False
	MovieWindowSize	3 - amvOneFourthScreen

三、 编写事件驱动代码

1. 用 ActiveMovie 的属性值给进度条属性赋值，以实现用进度条显示视频文件的长度，以及视频文件的播放进度：

```
Private Sub ActiveMovie1_Timer()
Slider1. Max = ActiveMovie1. SelectionEnd
Slider1. Value = ActiveMovie1. CurrentPosition
End Sub
```

2. 从下拉列表框中选择视频文件的尺寸，以及窗体随视频大小的改变而改变：

```
Private Sub Combo1_Click()
If Combo1. Text = "原始尺寸" Then
ActiveMovie1. MovieWindowSize = amvOriginalSize
Form1. Height = ActiveMovie1. Height / 3 * 4. 5
Form1. Width = ActiveMovie1. Width / 3 * 4. 5
End If
If Combo1. Text = "1/2 屏幕" Then
ActiveMovie1. MovieWindowSize = amvOneHalfScreen
Form1. Height = ActiveMovie1. Height / 3 * 4 3
Form1. Width = ActiveMovie1. Width / * 4. 5
End If
If Combo1. Text = "1/4 屏幕" Then
ActiveMovie1. MovieWindowSize = amvOneFourthScreen
Form1. Height = ActiveMovie1. Height / 3 * 4 3
Form1. Width = ActiveMovie1. Width / * 4. 5
End If
If Combo1. Text = "全屏幕" Then
ActiveMovie1. FullScreenMode = True
ActiveMovie1. Run
End If
End Sub
```

3. 打开一个视频文件，并将文件名赋给 ActiveMovie 控件：

```
Private Sub cmdopen_Click()
CommonDialog1. Action = 1
ActiveMovie1. FileName = CommonDialog1. FileName
End Sub
```

4. 播放视频文件：

```
Private Sub cmdplay_Click()
ActiveMovie1. Run
End Sub
```

5. 暂停播放：

```
Private Sub cmdpause_Click()
ActiveMovie1. Pause
End Sub
```

6. 停止播放：

```
Private Sub cmdstop_Click()
ActiveMovie1. Stop
End Sub
```

7. 退出播放器：

```
Private Sub cmdend_Click()
End
End Sub
```

8. 为下拉列表框赋初值：

```
Private Sub Form_Load()
Combo1. AddItem "原始尺寸"
Combo1. AddItem "1/2 屏幕"
Combo1. AddItem "1/4 屏幕"
Combo1. AddItem "全屏幕"
End Sub
```

9. 当窗体大小改变时，调整窗体上各控件的大小：

```
Private Sub Form_Resize()
ActiveMovie1. Top = Form1. Height / 180
ActiveMovie1. Left = Form1. Width / 1
cmdopen. Top = Form1. Height / 18
cmdplay. Top = Form1. Height / 18 * 2 + cmdopen. Height
cmdpause. Top = Form1. Height/18 * 3 + cmdopen. Height * 2
cmdstop. Top = Form1. Height/18 * 4 + cmdopen. Height * 3
cmdend. Top = Form1. Height/18 * 5 + cmdopen. Height * 4
Label1. Top = Form1. Height/18 * 6 + cmdopen. Height * 5
Combo1. Top = Form1. Height / 18 * 7 + cmdopen. Height *
6 - 300
cmdopen. Left = Form1. Width - 1100
```

（下转第 61 页）

用 VC 实现的实时曲线类的改进

董国亮

在工业控制中，我们常常需要使用实时曲线监测某些状态量（如温度，速度等）的变化，MFC 没有直接提供这样的类。Jan Vidar Berger 编写的 clPlot 类可完成此功能（在 http://www.vchelp.net/source/realtime_plot.zip 上可下载到该软件的源代码及示例程序）。结合现实中的实际需要，本文在原来代码的的基础上又增加了一些新功能。

Jan Vidar Berger 所写的类 clPlot 是专用于画实时曲线的类，派生于 CWnd。X 轴为时间，左右 Y 轴可用于标示不同变化范围的变量。演示程序（即应用程序）是在视图中显示曲线，并且曲线区域的大小可根据视图区域的大小自动调整。类 clPlot 以扩展 DLL 的形式提供给应用程序，这意味着应用程序可导出整个 clPlot 类。在应用程序中可以指定曲线区域的大小、曲线的颜色等特性（类 clPlot 提供了相应的成员变量和函数实现这些功能）。但 Jan Vidar Berger 的类 clPlot 没有实现采样数据的存盘和加载、打印和打印预览、标 X 轴及 Y 轴名称等。改进的 clPlot 类增加了这些功能，如下：

- 采样数据存盘和加载
- 演示程序中可以实现图形的打印预览和打印功能
- 标 X、Y 轴名称（请别小看此项。调试中你会发现，在原先 clPlot 类代码中上直接添加 CDC::DrawText 或 CDC::TextOut 显示名称，执行时会导致严重错误。详细请看第四部分。）
- 实现曲线从原点（而不是从右端）开始显示。

图 1 为 Jan Vidar Berger 的演示程序界面，图 2 为增加了新功能后的演示程序界面。

为简单起见，此处我们仅讨论左 Y 轴的情况（右 Y 轴处理相同）。下面分别说明其实现。

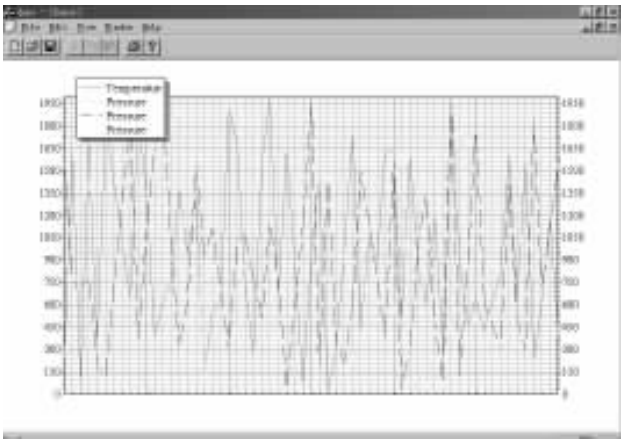


图 1 原来在视图中显示实时曲线

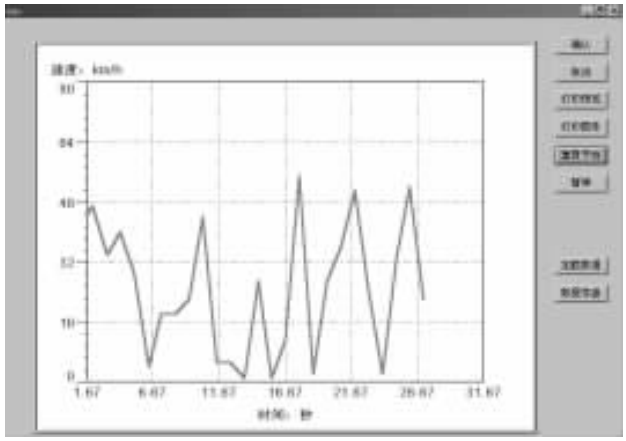


图 2 在对话框中实现改进的实时曲线

一、采样数据的存盘和加载

考察源代码可知，clPlot::m_series[]保存各条曲线的采样值（时间，实测值），所以我们可以通过为 series 类添加 serialize 实现序列化。实现代码如下：

```
void serie::Serialize(CArchive & ar)
{
  if(!m_bIAmInUse) return;
  long  i, j, k;
  int nos;
  if(ar.IsStoring())
  {
//为防止无限制消耗内存，此处用 clPlot::m_lMaxDataPrSerie 限制每条曲线最
//多可以存储的数据个数，超限后数据循环存放
nos = (m_lend - m_lbegin + clPlot::m_lMaxDataPrSerie) % clPlot::m_lMaxDataPrSerie;
      ar << nos; //数据个数
      for(i = m_lbegin, j = 0; j < nos; i++, j++)
      {;
      k = i%( clPlot::m_lMaxDataPrSerie)
      //保存采样值(时间，度)
  ar << m_pvalues[k].ValueTime << m_pvalues[k].dValue;
      }
  }
  else
  {
      ar>>nos; //数据个数
      if(!nos) return;
      m_lbegin = 0;
      m_lend = nos;
      if(m_lNoValues >0)
```

```
                //重新调整数组大小
m_pvalues = (value * )realloc(m_pvalues, (nos + 1) * sizeof
(value));
            else
m_pvalues = (value * )malloc((nos + 1) * sizeof(value));
//创建数组
            m_lNoValues = nos;
            for(i = m_lbegin, j = 0; j < nos; i + + , j + + )
            {;
                k = i% ( clPlot: : m_lMaxDataPrSerie)
                //装入采样值(时间, 速度)
ar>>m_pvalues[k]. ValueTime>>m_pvalues[k]. dValue;
            }
        }
}
```

同样在使用 series 类的 clPlot 类中也要加入 serialize ，代码如下：

```
void clPlot: : Serialize(CArchive& ar)
{
for(int s = 0; s < MAXSERIES; s + + )
{
if(m_series[s]. m_bIAmInUse)
m_series[s]. Serialize(ar);
//依次存各条曲线的采样值(时间: ms, 速度: km/h)
        }
}
```

然后在类 clPlot 中加入新成员函数 clPlot StoreToFile 。当该函数被调用时，它将先弹出一文件选择对话框。然后 CFile 对象和 CArchive 对象相关联，最后调用 clPlot serialize 即完成了采样数据的写盘保存。

```
BOOL clPlot: : StoreToFile()
{
 CfileDialog fd ( false, "AsmDat", "asmfile", OFN_HIDEREAD-
ONLY| OFN_OVERWRITEPROMPT, " * . AsmDat");
        if(fd. DoModal()! = IDOK)// 文件选择对话框
            return false;
        CString lpszPathName = fd. GetPathName();
        CFile pFile;
if( ! pFile. Open(lpszPathName, CFile: : modeCreate | CFile: :
modeReadWrite| CFile: : shareExclusive))
return FALSE; // 文件创建失败
        CArchive saveArchive( & pFile, CArchive: : store |
CArchive: : bNoFlushOnDelete);
        saveArchive. m_pDocument = NULL;
        TRY
        {
        CWaitCursor wait;
        Serialize(saveArchive);     // 写盘保存
        saveArchive. Close();
            pFile. Close();
        }
        CATCH_ALL(e)
        {
        pFile. Abort();
        return FALSE;
    }
    END_CATCH_ALL
    return TRUE;        // 成功
}
```

clPlot LoadFromFile 加载历史数据，其实现与 clPlot StoreToFile 基本相同：先显示文件对话框，然后 CFile 对象和 CArchive 对象关联，最后将磁盘中的数据加载到内存。

```
    BOOL clPlot: : LoadFromFile()
{
CFileDialog fd ( true, "AsmDat", "asmfile", OFN_HIDEREAD-
ONLY|     OFN_OVERWRITEPROMPT, " * . AsmDat");
    if(fd. DoModal()! = IDOK)
        return false;
    CString lpszPathName = fd. GetPathName();
    CFile pFile;
    if( ! pFile. Open(lpszPathName, CFile: : modeReadWrite |
CFile: : shareExclusive))
        return FALSE; //文件不存在
    CArchive loadArchive(& pFile, CArchive: : load|CArchive: :
bNoFlushOnDelete);
    TRY
    {
        CWaitCursor wait;
        Serialize(loadArchive);     // 加载历史数据到内存
        loadArchive. Close();
        pFile. Close();
    }
    CATCH_ALL(e)
    {
        pFile. Abort();
        return FALSE;
    }
    END_CATCH_ALL
    Invalidate();
    return TRUE;        // 成功
}
```

二、在演示程序中实现实时曲线的打印预览

考虑通常的情况，演示程序中我们在对话框 CAsmDlg 类实例 的某区域 如果在 CView 中则很容易实现打印预览功能显示实时曲线。首先，构造函数 CAsmDlg CAsmDlg CView * pview 将视图对象指针传递过来：

```
CAsmDlg: : CAsmDlg(CView * pView)
{
    m_pView = pView; // m_pView 是类 CAsmDlg 的成员变量, CView * m_pView
    . . . . . .
}
```

然后在 CAsmDlg OnAsmPrintPreview 命令消息响应函数对应命令按钮 IDC_ASMPRINTPREVIEW 中向视图窗口发送自定义消息 ASMPRINTPREVIEW。代码如下：

```
void CAsmDlg: : OnAsmprintpreview()
{
    // TODO: Add your control notification handler code here
```

```
    m_pView->PostMessage
(WM_ASMPRINTORPREVIEW, ASMPRINTPREVIEW);
}
```

相应地，在与 CAsmDlg 对象关联的视图派生类 CPowerView 中处理 CAsmDlg OnAsmPrintPreview 发送来的自定义消息。代码如下：

```
LRESULT CPowerView:: OnAsmPrintOrPreview(WPARAM
wParam, LPARAM lParam)
{
    if(m_pAsmDlg->GetSafeHwnd()!=0)
    {
        switch(wParam)
        {
    case ASMPRINTPREVIEW:
     CView::OnFilePrintPreview(); //实时曲线打印预览
     break;
    case ASMPRINT:
     CView::OnFilePrint();break; //实时曲线打印
    case ASMEND://
     m_pAsmDlg->DestroyWindow();
     break;
    default:break;
        }
    }
    return 0L;
}
```

实时曲线打印和打印预览时 CPowerView OnAsmPrintPreview 将调用 OnPrint 。OnPrint 才是实现打印的地方，其代码如下：

```
void CPowerView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
// TODO: Add your specialized code here and/or call the
base class
if(m_pAsmDlg->GetSafeHwnd()!=0)
{
CRect oldrect;
clPlot & plot=m_pAsmDlg->m_asmrealtime; //引用
plot.GetDiagramRect(oldrect); //保存原来设置图形的大小
CSize sizePrint(pDC->GetDeviceCaps(HORZRES), pDC->
GetDeviceCaps(VERTRES));
sizePrint.cy/=2; //设置为半张打印纸大小
pInfo->m_rectDraw.SetRect(0, 0, sizePrint.cx, sizePrint.cy);
pInfo->m_rectDraw.DeflateRect(sizePrint.cx/20,
sizePrint.cy/20); //留 margin
plot.SetDiagramRect(pInfo->m_rectDraw, pDC); //设置打
印图形的大小(象素)
        long oldfrom, oldto;
        long from=0, to;
plot.GetXFromToTime(oldfrom, oldto); //保存原来图形的 X
轴起始值(实际值, ms)
        to=oldto;
plot.SetBXRange(from, to, false); //设置新的 X 轴起始值(画
整张图)
plot.Print(pDC); //在打印机上打印
plot.SetBXRange(oldfrom, oldto); //恢复原来图形的 X 轴起
始值(实际值, ms)
plot.SetDiagramRect(oldrect); //恢复原来设置图形的大小
(象素)
  }
}
```

其中，m_pAsmDlg 为 CPowerView 的公有成员变量，它是指向实时曲线所在包容器的对话框类实例对象的指针。在 CPowerVeiw.H 中声明如下：

```
CAsmDlg * m_pAsmDlg
```

CAsmDlg 实例的构造和销毁分别在 CPowerView 的构造函数和析构函数中完成：

```
CPowerView::CPowerView():CFormView(IDD_FORMVIEW)
{
  // TODO: add construction code here
  m_pAsmDlg=new CAsmDlg(this);
}
CPowerView::~CPowerView()
{
  delete m_pAsmDlg;
}
```

这样通过调用 CView OnFilePrintPreview 实现了 clPlot 实时曲线的打印预览。

三、在演示程序中实现实时曲线的打印

实时曲线的打印实现方法同上，只要在 CAsmDlg 中加入如下代码即可。

```
void CAsmDlg::OnAsmprint()
{
  // TODO: Add your control notification handler code here
  m_pView->PostMessage(WM_ASMPRINTORPREVIEW,
ASMPRINT);
}
```

四、标 X Y 轴名称

原 clPlot 类中没有标 X、Y 轴名称。如果我们只是在 clPlot DrawXAxisGrid 和 clPlot DrawYAxisGrid 中添加 CDC DrawText 或 CDC TextOut 进行名称显示，执行演示程序进行打印预览时，放大预览，反复拖拽上下滚动条若干次后，出现资源无法找到等错误。去掉 CDC DrawText 或 CDC TextOut 等代码则一切正常，问题出在字体上。改进后的 clPlot 类将构造函数中的 m_font.CreateFontIndirect 去掉，而是在 clPlot OnDraw 内创建字体 该字体大小可根据打印区域的大小自动调整，由 clPlot ComputeRece 实现)。而且在 ComputeRect 、SetDiagramRect 参数中加入了标示是显示设备还是打印设备的 CDC * printerDC。默认为显示设备，printerDC=NULL；当为打印设备时，printerDC 为指向打印设备的 CDC 指针。clPlot ComputeRect 根据不同的设备类型重新计算实时曲线图形所占的矩形区域坐标。同时，新添加了 SetDiagramRect 成员函数，用于设置实时曲线矩形区域的新坐

用底层设备接口函数回放声音

余　锋　祝晓鹰

在编程中，回放声音常用的方法一般有两种：API 函数 PlaySound 或多媒体控制接口 MCI ，这两者简单易用，可以满足大部分情况下的需求。它们有一个共同的特征，即对声音采样数据的读取处理都是在内部进行的，我们接触不到这些数据。如果想对声音进行简单的处理，比方说加一个增益，或者倒放，它们就无能为力了。这时就需要我们自己从 WAV 文件中读取声音采样数据，然后通过底层的声音设备接口函数进行回放。为此先要弄清楚采样数据是如何在 WAV 文件中存放的。

一、WAV 文件格式

请注意，以下的讨论都是针对 PCM 格式 一种非压缩存储方式 的 WAV 文件。

1. 文件结构

一个 WAV 文件由若干个不同类型的数据块组成：格式数据块包含声音的格式信息，如采样率；波形数据块是实际的采样数据。这二者是 WAV 文件必须有的，其它数据块都属于可选项，它们描述了诸如设备参数等内容。

在排列顺序上，要求格式数据块必须位于波形数据块之前，但可以不相邻。

WAV 文件的具体结构如下 省略号代表可选数据块 ：

“RIFF”　　标志，占 4 字节

一个长整型数、文件剩余的字节数，等于文件长度 - 8，占 4 字节。

“WAVE”　　标志，占 4 字节

……

格式数据块

……

波形数据块

……。

2. 采样点和采样帧

一个采样点就是声音的一个采样数据，按采样位数的不同，可能占 1、2、4 个字节。要注意采样数据的正负：采样位为 16 时，一个采样点是一个短整型数，占两字节，取值范围 - 32768 至 32767；采样位为 8 或更少时，一个采样点是一个无符号字符型数据，取值范围 0 至 255。

如果声音是多通道的，每个通道的采样点交错排列。以立体声 两通道 为例：先存储左声道的第一个采样点，然后是右声道的第一个采样点，然后左声道的第二个采样点，右声道的第二个采样点……。

每个通道相同位置的采样点合称为一个采样帧，如立体声左声道的第一个采样点和右声道的第一个采样点构成一个采样帧，同理它们的第二个采样点又构成一个采样帧。如果是单声道，一个采样点就是一个采样帧。

3. 格式数据块 format chunk

格式数据块可以用一个结构描述：

```
typedef struct {
  char          chunkid[4];
  long          chunksize;
  short         wformattag;
  unsigned short wchannels;
  unsigned long  dwsamplespersec;
  unsigned long  dwavgbytespersec;
  unsigned short wblockalign;
```

标。其参数 printerDC 意义同前。限于篇幅，不再给出实现代码。有兴趣的读者可与编辑部联系索取修改后的完整源代码。

五、实现曲线从原点　而不是从右端）开始显示

将 clPlot. cpp 中的

```
clPlot: : clPlot( )
{   . . . . . .
    SetBXRange(CTime: :  GetCurrentTime( ) – CTimeSpan
(60), CTime: : GetCurrentTime( ));
    . . . . . .
}
```

改为:

```
    clPlot: : clPlot( )
{
. . . . . .
long totime, fromtime;
  fromtime = 0; totime = 30000; / / ms
  SetBXRange(fromtime, totime);
  . . . . . .
}
```

即可。

六、其它

clPlot 是一个扩展 DLL，程序员可导出整个 clPlot 类。但是 clPlot 成员变量太多，为便于使用又新添了一些辅助成员函数。限于篇幅，此处不再一一列出。

收稿日期：2000 年 10 月 17 日）

```
    unsigned short wbitspersample;
  } formatchunk;
```

chunkid 的前 3 个字符总是格式数据块的标志字“fmt”。

chunsize 存放格式数据块剩余的字节数，等于 sizeof formatchunk - sizeof chunkid - sizeof chunksize 。

wformattag 指明采样数据是否使用了压缩格式：如果为 1，表示使用非压缩格式 PCM ；不为 1，表示使用压缩格式，这时需要额外的字段来描述压缩算法。本文不讨论压缩格式的 WAV 文件，有关资料可在微软官方站点获得。

wchannels 存放声音的通道数：1 为单声道；2 为立体声……

dwsamplespersec 存放声音的采样率，常见的有 11025、22050、44100。

dwavgbytespersec 存放采样数据每秒的数据量，等于 dwsamplespersec * wblockalign。

wblockalign 为一个采样帧占用的字节数。

wbitspersample 为声音的采样位数。

4. 波形数据块 data chunk

采样帧都存放在该块中，用结构定义如下：

```
typedef struct {
  char          chunkid[4];
  long          chunksize;
  unsigned char  waveformdata[1];
} datachunk;
```

chunkid 总是波形数据块的标志字“data”。

chunsize 存放数据块剩余的字节数，即采样数据的大小，不包括 chunkid 和 chunksize 占用的 8 个字节，也不包括为了使波形数据块长度成为偶数而在末尾补齐的空白字节。

waveformdata 存放采样数据，按采样帧存放，帧数等于 chunksize 除以格式数据块的 wblockalign 字段。

5. 其它数据块

如 playlist chunk、label chunk 等，跟回放声音关系不大，这里不再赘述，感兴趣者可去微软官方站点查询。所有这些可选数据块都有一个共同点，即以这么一个结构开头：

```
typedef struct {
  char  chunkid[4];
  long  chunksize;
} chunk_hdr;
```

chunkid 存放数据块的标志字，如“plst”、“labl”等。

chunksize 存放数据块剩余的字节数，不包括结构 chun_hdr 本身占用的 8 个字节，也不包括为了使数据块长度成为偶数而在末尾补齐的空白字节。

有了这个结构，我们在读取 WAV 文件时，就能跳过不需要的数据块。另外，格式和波形这两种数据块也可以看做以 chunk_hdr 开头。

为了方便读者，本文提供一个 C + + 类 CPCMWaveFile 用于读取 WAV 文件，使用方法见后面的例程及其注释。

二、数据传输

从 WAV 文件中得到采样数据后，接下来要做的就是通过底层接口函数把数据传给音频设备，其大致顺序是：打开设备；向设备发送采样数据开始回放；回放完毕后关闭设备。所有这些函数的名称都以 waveOut 开头，需要包含头文件 mmsystem. h，并且在编译连接时加入库文件 winmm. lib。

1. 回调 Callback

因为底层接口函数在执行完后立刻返回，这样就存在一个问题：比如向设备发送了一批采样数据后，函数返回，与此同时声音开始回放，那么我们怎么才能知道声音何时回放完呢？Windows 系统通过回调机制解决了这个问题：

在打开音频设备时，用户向系统传递一个函数地址，类型见 waveOutProc 的联机帮助，也可以传递一个事件或窗口句柄。

如果指定的是函数地址，系统将在设备打开、采样数据回放完毕以及设备关闭后，调用该函数，通过函数参数来区分这三种情况。

如果指定的是事件句柄，系统将在设备打开、采样数据回放完毕以及设备关闭后，把该事件置为有信号状态。因为事件没有参数，因此只能通过发生顺序来区别三种情况。

如果指定的是窗口句柄，系统将在设备打开、采样数据回放完毕以及设备关闭后，分别向该窗口发送消息 MM_WOM_OPEN、MM_WOM_DONE 及 MM_WOM_CLOSE。实际编程中，只需处理 MM_WOM_DONE 即可。这是因为另外两个消息都是在接口函数的执行过程中发送的，函数返回后，设备必然已经打开或关闭，再处理这两个消息就有点多此一举了。顺便提一下，对于回调函数或事件，在打开/ 关闭设备时，其调用或状态的修改也是在接口函数的执行过程中进行的。

事件比较适合编写控制台程序或工作线程，窗口则适用于 Windows 编程。至于回调函数，笔者认为还是不用为好。为什么这么说呢？假设现在正在播放声音，为了在回放完毕时得到通知，有两种方法，一是设置一个全局变量做标志，程序把采样数据发送给设备后，进入一个循环，不停地判断标志变量是否改变，如果改变，则结束循环，标志变量的修改放在回调函数中。另一种方法是用一个全局事件做为标志，程序发送完数据后，用 WaitForSingleObject 等待该事件被置为有信号状态，全局事件的位置放在回调函数中。前者会导致 CPU 的占用率达到 100%，而后者跟直接使用回调事件其实是一回事。

本文中的例程采用事件作为回调方式。

2. 打开音频设备

用函数 waveOutOpen，调用时要指定：设备 ID，通常用 WAVE_MAPPER，即标准音频设备；采样数据格式，如采样率，采样位数等；回调方式。如果调用成功，我们将得到一个设备句柄，以后的函数调用需要这个句柄。

3. 传送采样数据

首先用函数 waveOutPrepareHeader "准备"采样数据，然后用 waveOutWrite 把"准备"好的数据发给设备开始回放，回放完毕后用 waveOutUnprepareHeader 取消采样数据的"准备"。

4. 关闭设备

用函数 waveOutClose 实现。

这些函数的详细用法请参见 Visual C++ 的联机帮助，下面是一段用底层函数回放的代码，假设采样数据已经从 WAV 文件中取出，放在指针 lpwavedata 指向的内存，大小为 wavesize，声音格式为 PCM、立体声、采样频率 44.1kHz、采样位 16：

```
//打开音频设备
HWAVEOUT hWave;
PCMWAVEFORMAT fmt ={{1,       //PCM
                    2,       //通道数
                    44100,   //采样频率
                    4*44100, //每秒数据量
                    4},      //采样帧大小
                    16};     //采样位
 HANDLE  hEvent =CreateEvent(NULL, FALSE, FALSE, NULL);
waveOutOpen(&hWave, WAVE_MAPPER, (LPWAVEFORMATEX)&fmt, (DWORD)hEvent, 0, CALLBACK_EVENT);
WaitForSingleObject(hEvent, INFINITE);
//"准备"数据
WAVEHDR whdr;
memset(&whdr, 0, sizeof(WAVEHDR));
whdr.lpData =lpwavedata;
whdr.dwBufferLength =wavesize;
waveOutPrepareHeader(hWave, &whdr, sizeof(WAVEHDR));
//输出
waveOutWrite(hWave, &whdr, sizeof(WAVEHDR));
//等待回放完毕
WaitForSingleObject(hEvent, INFINITE);
//取消"准备"
waveOutUnprepareHeader(hWave, &whdr, sizeof(WAVEHDR));
//关闭设备
waveOutClose(hWave);
WaitForSingleObject(hEvent, INFINITE);
CloseHandle(hEvent);
```

5. 双缓存 Double Buffer

在上面程序中，我们是先把采样数据全部取出，然后一次性发送给设备。如果数据量很大，比如说几十兆，这种方法就不合适了，必须分批传送数据。但是注意，不能想当然地采取这种方法：先读取一块数据发送给设备，然后等待回放完毕，再发送下一块数据。这样做的话，在两块数据之间有一个小小的间歇，会导致声音回放不连续。

可以采取一种被称之为"双缓存"的算法，具体步骤如下：

步骤 1 首先"准备"两块数据，连续发给设备，也就是说，连续执行两次 waveOutWrite；

步骤 2 等待一块数据回放完毕。因为设备缓冲区内有两块数据，一个回放完毕后，另一个自动接着回放，这样就避免了声音的停顿；

步骤 3 对已播放完的采样数据，取消其"准备"；

步骤 4 如果没有剩余数据，等待另一块数据回放完毕，对其取消准备后，跳至步骤 5；否则，读取下一块数据，"准备"后发送给设备，从而保证设备缓存区内总是有两块数据，然后回到步骤 2；

步骤 5 关闭设备。

三、例程

例程 PlayWav 是一个控制台命令行程序，演示了如何通过回调事件和双缓存来回放 WAV 文件。它由 3 个源文件组成：PCMWaveFile.h 和 PCMWaveFile.cpp 分别是类 CPCMWaveFile 的头文件和实现文件；PlayWav.cpp 是主函数 main 的实现文件。运行时通过命令行参数输入 WAV 文件名，比如

```
PlayWav c  hwindows hmedia htada.wav
```

为了避免代码过于冗长，底层接口函数的调用没有考虑出错情况下的处理，总是假定函数能够成功返回。

本文代码均在 Visual C++ 5.0 上调试通过。

附程序源代码：

```
/*************************************
 *                PCMWaveFile.h      *
 *************************************/
#include <stdio.h>
class CPCMWaveFile
{
protected:
    FILE*          m_fpWaveFile;      //文件指针
public:
    long           m_nWaveSize;       //波型数据的尺寸
    short          m_wformattag;      //声音格式, 1 为 PCM 格式
    unsigned short m_wchannels;       //通道数
    unsigned long  m_dwsamplespersec; //采样率
    unsigned long  m_dwavgbytespersec;//每秒波型的数据量
    unsigned short m_wblockalign;     //采样帧的字节数
    unsigned short m_wbitspersample;  //采样位数
public:
    CPCMWaveFile();
    ~CPCMWaveFile();
    //打开由 filename 指定的 WAV 文件, 并读取声音格式;
    //打开后, 把文件位置指针移动到波型数据起始处;
    //返回 0 调用成功; 1 打开文件失败; 2 不是合法的 WAV 文件; 3 不是 PCM 格式
    int Open(char* filename);
    //关闭 WAV 文件句柄
    void Close();
    //读取 WAV 文件的波型数据, 存放在缓存区 buffer, count 为要读取的字节数. 返回值为实际读取的字节数. 如果返回值为 0 或小于 count, 表示文件还没打开, 或遇到文件末尾.
    int Read(void* buffer, unsigned int count);
};
```

```
/*************************************
/*                PCMWaveFile.cpp                *
/*************************************
#include <stdio.h>
#include <string.h>
#include "PCMWaveFile.h"
CPCMWaveFile::CPCMWaveFile()
{
    m_fpWaveFile=NULL;
}
CPCMWaveFile::~CPCMWaveFile()
{
    Close();
}
int CPCMWaveFile::Open(char *filename)
{
    //数据块头的结构定义
    struct CHUNK_HDR {
        char chunkid[4];
        long chunksize;
    };
    //格式数据块的结构定义
    struct FORMATCHUNK {
        CHUNK_HDR hdr;    //格式数据头
        short wformattag;
        unsigned short wchannels;
        unsigned long dwsamplespersec;
        unsigned long dwavgbytespersec;
        unsigned short wblockalign;
        unsigned short wbitspersample;
    };
    //打开文件
    FILE *fp;
    if ((fp=fopen(filename,"rb"))==NULL)
        return 1;  //打开文件失败
    //读标志字 RIFF
    char buf[4];
    if (fread(buf,1,4,fp)<4) {
        fclose(fp);
        return 2;  //不是合法的 WAV 文件
    }
    if (memcmp(buf,"RIFF",4)!=0) {
        fclose(fp);
        return 2;  //不是合法的 WAV 文件
    }
    //跳过文件剩余字节数
    fseek(fp,sizeof(long),SEEK_CUR);
    //读标志字 WAVE
    if (fread(buf,1,4,fp)<4) {
        fclose(fp);
        return 2;  //不是合法的 WAV 文件
    }
    if (memcmp(buf,"WAVE",4)!=0) {
        fclose(fp);
        return 2;  //不是合法的 WAV 文件
    }
    //寻找格式数据块
    CHUNK_HDR header;
    long offset=0;  //数据块尺寸(不包括数据块头)
    do {
        //把文件指针定位到数据块起始部分
        fseek(fp,offset,SEEK_CUR);
        //读数据头
        if (fread(&header,1,sizeof(CHUNK_HDR),fp)<
sizeof(CHUNK_HDR)) {
            fclose(fp);
            return 2;  //不是合法的 WAV 文件
        }
        //计算 offset,如果 chunksize 为奇数,要凑成偶数
        offset=header.chunksize+header.chunksize%2;
    } while(memcmp(header.chunkid,"fmt",3)!=0);
//判断是否是格式数据块
    //已找到格式数据块,进行读取
    FORMATCHUNK fmtchunk;
    fmtchunk.hdr=header;
    int size=sizeof(FORMATCHUNK)-sizeof(CHUNK_HDR);
    if (fread((char *)&fmtchunk+sizeof(CHUNK_HDR),1,
size,fp)<(unsigned)size) {
        fclose(fp);
        return 2;  //不是合法的 WAV 文件
    }
    //判断是否是 PCM 格式
    if (fmtchunk.wformattag!=1) {
        fclose(fp);
        return 3;  //不是 PCM 格式
    }
    //寻找波型数据块
    offset=0;
    do {
        //把文件指针定位到数据块起始部分
        fseek(fp,offset,SEEK_CUR);
        //读数据头
        if (fread(&header,1,sizeof(CHUNK_HDR),fp)<
sizeof(CHUNK_HDR)) {
            fclose(fp);
            return 2;  //不是合法的 WAV 文件
        }
        //计算 offset,如果 chunksize 为奇数,要凑成偶数
        offset=header.chunksize+header.chunksize%2;
    } while(memcmp(header.chunkid,"data",4)!=0);
//判断是否是波型数据块
    //已找到波型数据块
    m_fpWaveFile=fp;
    m_nWaveSize=header.chunksize;
    m_wformattag=fmtchunk.wformattag;
    m_wchannels=fmtchunk.wchannels;
    m_dwsamplespersec=fmtchunk.dwsamplespersec;
    m_dwavgbytespersec=fmtchunk.dwavgbytespersec;
    m_wblockalign=fmtchunk.wblockalign;
    m_wbitspersample=fmtchunk.wbitspersample;
    return 0;
}
void CPCMWaveFile::Close()
{
```

```
    if (m_fpWaveFile) {
        fclose(m_fpWaveFile);
        m_fpWaveFile = NULL;
    }
}
int CPCMWaveFile: : Read(void * buffer, unsigned int count)
{
    if (m_fpWaveFile)
        return fread(buffer, 1, count, m_fpWaveFile);
    else
        return 0;
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
/*                    PlayWav. cpp                      *
/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
#include <windows. h>
#include <mmsystem. h>
#include <stdio. h>
#include "PCMWaveFile. h"
void main(int argc, char * argv[])
{
    if (argc < 2) {
        puts("Usgae: PlayWav WaveFileName");
        return;
    }
    CPCMWaveFile waveFile;
    int result;
    if (result = waveFile. Open(argv[1])) { //打开 WAV 文件
        printf("Can't open the file, error code %d\n", result);
        return;
    }
    HWAVEOUT hWave;
    PCMWAVEFORMAT fmt = {{waveFile. m_wformattag,
                          waveFile. m_wchannels,
                          waveFile. m_dwsamplespersec,
                          waveFile. m_dwavgbytespersec,
                          waveFile. m_wblockalign      },
                          waveFile. m_wbitspersample   };
HANDLE hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
 waveOutOpen (& hWave, WAVE_MAPPER, (LPWAVEFOR-
MATEX)& fmt, (DWORD) hEvent, 0, CALLBACK_EVENT);
    WaitForSingleObject(hEvent, INFINITE);
    int bufsize = 1 * fmt. wf. nAvgBytesPerSec;
    int leftsize = waveFile. m_nWaveSize; //记录剩余的字节数
    char * lpbuf1 = new char [bufsize];
    char * lpbuf2 = new char [bufsize];
    int bufID = 1;                    //记录当前播放的缓存区号
    int size;                         //记录实际读取的字节数
    WAVEHDR whdr1, whdr2;
    //准备好第一个缓存区数据并输出至音频设备
    size = min(bufsize, leftsize);
    leftsize - = size;
    waveFile. Read(lpbuf1, size);
    memset(& whdr1, 0, sizeof(WAVEHDR));
    whdr1. lpData = lpbuf1;
    whdr1. dwBufferLength = size;
waveOutPrepareHeader(hWave, & whdr1, sizeof(WAVEHDR));
    waveOutWrite(hWave, & whdr1, sizeof(WAVEHDR));
    if (leftsize = = 0) { //没有剩余数据, 则等待输出结束
        WaitForSingleObject(hEvent, INFINITE);
waveOutUnprepareHeader(hWave, & whdr1, sizeof(WAVEHDR));
    }
    else { //有剩余数据, 准备好第二个缓存区数据并输出至音频设备
        size = min(bufsize, leftsize);
        leftsize - = size;
        waveFile. Read(lpbuf2, size);
        memset(& whdr2, 0, sizeof(WAVEHDR));
        whdr2. lpData = lpbuf2;
        whdr2. dwBufferLength = size;
waveOutPrepareHeader(hWave, & whdr2, sizeof(WAVEHDR));
        waveOutWrite(hWave, & whdr2, sizeof(WAVEHDR));
        for (; ; ) {
//等待一个缓存区输出完毕, 对它执行 waveUnprepareHeader
            WaitForSingleObject(hEvent, INFINITE);
            if (bufID = = 1)
                //刚输出完毕的是缓存区 1
            waveOutUnprepareHeader(hWave, & whdr1, sizeof
(WAVEHDR));
            else
                //刚输出完毕的是缓存区 2
            waveOutUnprepareHeader(hWave, & whdr2, sizeof
(WAVEHDR));
//如果没有剩余数据, 等待另一个缓存区输出完毕, 跳出循环
            if (leftsize = = 0) {
                WaitForSingleObject(hEvent, INFINITE);
                if (bufID = = 1)
                    //如果上一个缓存区号是 1, 这个就是 2
            waveOutUnprepareHeader(hWave, & whdr2, sizeof
(WAVEHDR));
                else
                    //如果上一个缓存区号是 2, 这个就是 1
            waveOutUnprepareHeader(hWave, & whdr1, sizeof
(WAVEHDR));
                break;
            }
            //读取剩余数据, 并送给设备
            size = min(bufsize, leftsize);
            leftsize - = size;
            if (bufID = = 1) {
          //刚输出完毕的是缓存区 1, 则把新数据送入缓存区 1
                waveFile. Read(lpbuf1, size);
                memset(& whdr1, 0, sizeof(WAVEHDR));
                whdr1. lpData = lpbuf1;
                whdr1. dwBufferLength = size;
            waveOutPrepareHeader(hWave, & whdr1, sizeof
(WAVEHDR));
      waveOutWrite(hWave, & whdr1, sizeof(WAVEHDR));
                //现在正在播放的是缓存区 2
                bufID = 2;
            }
            else {
        //刚输出完毕的是缓存区 2, 则把新数据送入缓存区 2
```

(下转第 64 页)

使用 ASP 技术和 SQL 语句实现数据库的操作

张建安

随着企业级 Intranet 的广泛开发使用，在网页上访问数据库的需求越来越大。但在原始的设计方法下，为了访问数据库数据，人们使用表单和 HTML 语言所实现的静态网页，缺乏交互性，数据不能根据录入、修改、删除情况随时更新。为了维护更新网页内容，必须不断地重复修改 HTML 文档，工作量非常繁重。这就提出了如何实现数据库的动态访问的问题。

现今常用的实现动态网页的方法有 CGI、ISAPI（NSAPI）、PHP 或直接利用脚本语言等。但这些技术都不适合于快速应用开发和技术普及。如 CGI，编程实现过于复杂，而且采用页面对应进程的形式，会大量耗量系统资源。而 ISAPI、NSAPI 技术虽然采用 DLL 取代线程，提高了系统性能，但需要考虑进程同步且开发复杂，不易掌握。另外采用客户端脚本语言虽也可实现简单的动态交互式页面，但功能有限，并且不同的浏览器对脚本语言的支持不同。因此，在针对企业级 Intranet 网站的开发中，需要寻找一种开发相对简单、性能相对较高的技术。微软公司推出的 ASP 就是针对 Web 的应用程序开发技术，只需编写几行脚本语句，就可以产生并执行动态、交互式的站点服务器应用程序。

本文根据 ASP，结合结构化查询语言 SQL，实现了 Web 环境下对数据库的动态操作。主要包括数据记录的查询、修改、插入和删除等，并提供较为详细的源程序，只需稍作修改，即可应用到自己的网页中去。

一、 SQL 的简单介绍

SQL 语言是目前关系数据库管理系统的一种通用的结构查询语言，众多的数据库管理系统，如 ORACLE 、Sybase、 Informix 都支持 SQL 语言，甚至微软也推出了自己的 SQL。在进行网络应用程序开发中，我们知道，都要大量使用 SQL 语句来进行各种数据库的处理。现在 ISO 已为 SQL 语言制订出了国际标准。 但需要注意的是，不同的 DBMS 中的 SQL 语句语法有可能有细小区别，在开发中应尽可能使用标准格式。

SQL 的广泛使用正说明了它的优秀。它是一种非过程化的语言，允许用户在高层的数据结构上工作，即可操作记录集。所有的 SQL 语句接受集合作为输入，返回集合作为输出。SQL 不要求用户指定对数据的存放方法，使用户更易集中精力于要得到的结果。所有的 SQL 均使用查询优化器，由它决定对指定数据存取的最快速度的手段。用户不需要关心表是否有索引，有什么类型的索引。

在本程序中，我们主要完成的是数据记录的查询、修改、插入和删除，下面先简单介绍一下这四种操作的 SQL 语法。

(1) 查询—SELECT 语句

Select <字段名> from <数据表名> where <条件>

在 <字段名>中，允许指定一个字段或多个字段，也可用“*”符号代表所有字段；

where 子句指明查询的数据记录的范围，可通过关系操作符把多个条件联接在一起构成复杂条件查询，也可使用 in、like、between 等实现模糊查询。

如：Select * from 教务处人员情况表 where 职称 = ‘讲师’ and 姓名 like ‘张%’ 功能是从教务处人员情况表中，查找职称为讲师且姓张的教师的记录。这里职称、姓名为表中的字段名称。

(2) 修改—UPDATE 语句

Update <数据表名> set <修改内容> where <条件>

在 <修改内容>中，指明欲修改的字段名称和对此字段的新赋值。基本格式为：字段名 = ‘该字段新值’，新值要用引号引起来。多个字段之间要用逗号分隔。

如：Update 教务处人员情况表 set 职称 = ‘讲师’ 军衔 = “上尉” where 姓名 = ‘张建安’，功能是找到姓名为张建安的记录，把职称修改为讲师，军衔修改为上尉。

(3) 插入—INSERT 语句

Insert into <数据库表名>< 需赋值的字段名称 > value 对应字段的值

如有多个字段，字段之间及字段值之间需用逗号分隔，各字段值使用引号引起来。如包含全部字段，此时可省略字段名称部分。

如：insert into 教务处人员情况表 姓名 年龄 性别 value ‘张建安’ “28” ‘男’，功能是插入一条新记录，此记录姓名为张建安，年龄为 28，性别为男。

(4) 删除—DELETE 语句

Delete from <数据表名> where <条件>。

二、ASP 的简单介绍

ASP 是 WWW 服务器端的脚本运行环境，一个 ASP 文件实际就是嵌入可执行脚本 HTML 文档，以 .ASP 为扩展名，将 HTML 语言、脚本语言和 ActiveX 控件组合起来。ASP 属于

ActiveX 技术中服务器端的技术，服务器端根据 ASP 源程序，不需编译，直接在脚本语言引擎的解释下运行，并把动态生成的 WWW 页面传送到客户端浏览器。如果脚本中含有访问数据库的请求，则通过 ODBC 连结后台数据库，由数据库访问组件 ADO 执行访问数据库操作。最后，ASP 依据访库的结果集自动生成标准的 HTML 页面发送到客户器浏览。ASP 自身提供了 VBScript 和 Jscript 驱动，其中 VBScript 是缺省的脚本语言。在 ASP 文件中，脚本命令语句必须包含在 "<% %>" 之中，也可用 Include 命令在某处引入另一份文件的内容作为该文件的一部分，以省去在许多页面编写相同段落的重复工作。

ASP 和其他开发方法相比，主要有如下优点：

● 简易性：不需要编译或链接即可执行。可以使用普通的文本编辑器设计；

● 多语言性：由于 ASP 仅是一种开发环境而非一种语言，因此只要 WWW 服务器端中安装了相应的脚本引擎，就可使用任何一种脚本语言。且 ASP 中已安装了 VBScript 、JScript 引擎；

● 与浏览器无关性：用户端只要使用常规的可执行 HTML 代码的浏览器，即可浏览 ASP 所设计的网页内容，而脚本语言只在站点的服务器端执行，用户浏览器不需执行这些脚本语言；

● 面向对象性和可扩充性：ASP 采用了面向对象的设计方法，并且提供的多个内置对象供用户使用。ASP 脚本中可以方便地引用系统控件和 ASP 的内置控件，还能够通过插入 ActiveX 服务器控件来扩充其功能。而 ActiveX 控件可使用各种语言开发工具进行开发。

● 安全性：ASP 的源程序代码，不会传到用户的浏览器，用户端看到的仅仅是执行结果的 HTML 代码。因此可以避免自己设计的程序代码被剽窃，也提高了系统的安全性；

ASP 的内置对象和应用组件

ASP 提供了 5 个内置对象和 5 个内置服务器组件，不需要编写复杂程序段，就可以直接使用这些功能强大组件和对象。这五个内置对象是：

1. Request 对象：从客户端用户（Web 页面）读取信息，有两种方式。其一是读取附带在网址后面的参数；其二是读取 HTML 输入表单 FORM 里的关键字段内容，如语句 Request. Form "name" 可以取得用户填在文本框 name 里的内容。在本程序中，主要使用第二种方式接收用户界面信息。（在第二种方式情况下要与< Form> < /Form> 搭配，可参看有关 HTML 说明书。） 见后程序）

2. Response 对象：将信息送给客户端用户，Response 的 write 方法可以直接向 HTML 文档插入字符串，其格式为：Response. Write 数据，如语句 Response. Write Now，向浏览器输出了当前时间；Response 还提供了控制流程的方法： Response. Redirect 网址，将当前网页转到另一个网页；Response. End 则中止当前的 ASP 处理。在本程序中，使用 Redirect 来控制 WEB 之间的调用关系。 见后程序）

3. Server 对象：提供 Web 服务器工具。 Server 对象有 2 个重要方法： MapPath 和 CreateObject。MapPath 用来将 Web Server 的虚拟路径还原成实际路径。 CreateObject 用来产生服务器组件的对象实例，服务器组件可以连接服务器数据库、使用网络功能或访问服务器文件系统。要调用服务器组件，必须先产生组件对象实例，即 Server. CreateObject ObjectID ，其中 ObjectID 指定了组件标识，它可以是各种形式的可执行程序，组件产生后，就可以使用它的方法和属性进行工作。

4. Session 对象：存储在一次会话（Session）期间的用户信息，仅被该用户访问。 Session 对象最主要的用途是提供某一次连结所有网页之间的公用信息，可以实现多个应用程序间的公用变量的值的传递。在本程序中，使用了多个 Session 对象，如 Session "table" 在各个网页间传递了数据表名，而 Session "sql" 用来在各个网页间传递由上一步所得出的 SQL 语句，以便最后生成可执行 SQL 语句。

5. Application 对象：管理所有的会话信息，供所有用户共享。

ASP 提供了 5 个内置服务器组件：

● 数据库访问组件
● 旋转看板组件
● 浏览器兼容组件
● 文件访问组件
● 文件超链接组件

在本程序中，我们关心的是数据库访问组件 ADO DB 及其应用方法，在后续程序我们会看到实例。

三、ASP 与 SQL 结合操作数据库

ASP 通过 ActiveX 控件 ADO 与 ODBC 对话，实现同数据库的连接。对于任何一种 DBMS，只要安装了相应的 ODBC 驱动程序，均可与 ASP 相连。ADO 是系统提供的用于访问后台数据库的控件，它运行于服务器端，提供数据库信息的页面内容，通过执行 SQL 命令，可以在浏览器上动态地查询 Select 、修改 Update 、删除 Delete From 和插入 Insert into 数据库的记录信息。在 ASP 中，ADO 可以看作是一个服务器控件，应用其中的各种属性的对象，即可轻松完成对数据库复杂的操作。

实现 ASP 动态数据库操作主要有以下几步：

(1) 设置数据库源名（DSN）

在 Windows 9X 或 Windows NT 系统中，运行 "控制面板/ /ODBC 数据源管理器"，按提示添加 "系统 DSN" 即可；

(2) 创建数据库连接（Connection）

Set Conn = Server. CreateObject "ADODB. Connection"

⑶ 调用 Open 方法打开数据库

如：Conn. Open “数据源名称”

⑷ 创建数据对象 数据集）

如：Set rs = Conn. Execute “SQL 语句”

⑸ 数据库操作

⑹ 关闭数据对象和连接。

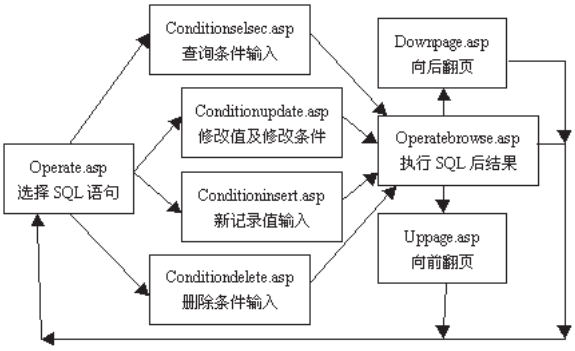
```
Rs. close
Conn. close
Set Conn = Nothing
```

四、程序实例

下面给出了一个在 Windows NT 服务器基础上构建的 Intranet 网络中，几个 ASP 程序的源文件，结合 SQL 语句，可以实现对教学信息数据库的动态操作，包括查询、录入、修改及删除。从此例程中，可看到 SQL 结合 ASP 后的强大功能，而实现却很简单。

为简单起见，数据库采用微软 ACCESS 数据库，文件名为 “mybase. mdb”，内含一个表，名称为 ‘教务处人员情况表”。

下面是各程序之间的调用关系



1. 选择 SQL 语句操作类型 operate. asp 节选

用 Request 接受用户选择信息，根据用户不同的选择，生成对应的 SQL 语句头，并保存在公共变量 Session “sql” 中，以便通过后续处理生成完整的 SQL 语句。另一个公共变量 Session “table”，存放表名，这样处理程序通用性要强一些。

```
<%session(″opt″) =request(″operatetype″)
if session(″opt″) <>″″ then
    if session(″opt″) =″select″ then
session(″sql″) =″select * from ″& session(″table″)& ″ where true ″
response. redirect ″conditionselect. asp″
    end if
    if session(″opt″) =″insert″ then
session(″sql″) =″insert into ″& session(″table″)& ″values(″
    response. redirect ″conditioninsert. asp″
    end if
    if session(″opt″) =″delete″ then
session(″sql″) =″delete from ″& session(″table″)& ″ where true ″
    response. redirect ″conditiondelete. asp″
    end if
    if session(″opt″) =″update″ then
    session(″sql″) =″update ″& session(″table″)& ″ ″
    response. redirect ″conditionupdate. asp″
    end if
end if%>
<%session(″table″) =″教务处人员情况表″%>
```

下面一段程序生成一个下拉式列表框和一个提交按钮，注意，第一个选项为空，这在后续程序的条件输入情况下特别重要，因为条件中可能不需此字段的值。

```
<form action =″operate. asp″ method =″GET″>
<table align =″center″><tr>
<td align =″right″>请选择操作类型 </td>
<td><select name =″Operatetype″ size =″1″>
<option value></option>
<option value =″select″>浏览数据库 </option>
<option value =″insert″>插入新记录 </option>
<option value =″delete″>删除指定记录 </option>
<option value =″update″>修改指定记录 </option>
</select></td></tr></table>
<div align =″center″><center><p><input type =″submit″ value =″确定″>
</p></center></div></form>
```

下面是运行页面图：

根据不同的选择项，第二步会进入不同的页面，或输入条件，或输入新值，或输入修改值。

2. 新记录值输入 conditioninsert. asp 节选）

在此页面下，输入新记录的值，生成完整的 SQL 语句并执行，执行后自动进入浏览页面。函数 equotchange 用于把双引号换为单引号。

```
<%Function equotchange(data)
    equotchange =″′″& replace(data, ″′″, ″′′″)& ″′″
  End Function%>
<%   name =request(″name″)
    if name =″″ then name =″?″
    sex =request(″sex″)
    if sex =″″ then sex =″?″
    age =request(″age″)
```

```
if age = "" then age = "?"
zw = request("zw")
if zw = "" then zw = "?"
zc = request("zc")
if zc = "" then zc = "?"
jx = request("jx")
if jx = "" then jx = "?"
zj = request("zj")
if zj = "" then zj = "?"
tele = request("tele")
if tele = "" then tele = "?"
email = request("email")
if email = "" then email = "?"
BP = request("BP")
if bp = "" then bp = "?"
sqll = session("sql")
```

姓名是主关键字，所以必须有值输入，否则不予处理；通过& 符号把从 operate. asp 传送过来的 SQL 语句头与此页面输入的值结合，生成完整的 SQL 语句。

```
if (name <>"")and(name <>"?") then
sqll = sqll& equotchange(name)& ", "
    sqll = sqll& equotchange(sex)& ", "
    sqll = sqll& equotchange(age)& ", "
    sqll = sqll& equotchange(zw)& ", "
    sqll = sqll& equotchange(zc)& ", "
    sqll = sqll& equotchange(jx)& ", "
    sqll = sqll& equotchange(zj)& ", "
    sqll = sqll& equotchange(tele)& ", "
    sqll = sqll& equotchange(email)& ", "
    sqll = sqll& equotchange(bp)& ")"
```

生成 SQL 语句后创建数据对象，连接后执行插入操作，执行结束应关闭数据集和数据对象，然后转入浏览程序，查看是否已插入。

```
set conn = Server. CreateObject("ADODB. Connection")
  conn. Open" Driver = {Microsoft Access Driver ( *
. mdb)}; dbq = " & Server. MapPath("mybase. mdb")
    conn. execute(sqll)
    conn. close
    set conn = nothing
    response. redirect "operatebrowse. asp"
end if%>
```

插入值输入页面图：

插入后的记录情况：

3. 修改指定记录值 conditionupdate. aap （节选）

有两个方面的问题：一方面是如何输入条件，以便找到符合条件的记录进行修改；另一方面是对应记录，应用什么值来更新。为此以表格的形式列为两栏进行处理，左栏用于选择条件，右栏用于指定新值。需要注意的是主关键字的问题，如果符合条件的记录不止一条，在右栏姓名 （主关键字）栏内又指定了新值，就会造成错误。另外，为了简化程序，左栏条件只采用了 "与 " 的形式生成总条件，在实际应用中，可根据实际情况，适当 Or、Not、Between、Like、 、 等来完成更复杂的条件设置。

部分程序如下：

```
<%set conn = Server. CreateObject("ADODB. Connection")
conn. Open" Driver = {Microsoft Access Driver ( * . mdb)};
dbq = " & Server. MapPath("mybase. mdb")
sqll = "select * from "& session("table")
set rs = conn. Execute(sqll)
```

上述一小段程序对数据库的操作，目的是取出数据表中各个字段的名称来生成 SQL 语句，这样做的优点是程序的可维护性强、通用性好。

```
    name = request("name1")
    sex = request("sex1")
    age = request("age1")
    zw = request("zw1")
    zc = request("zc1")
    jx = request("jx1")
    zj = request("zj1")
    tele = request("tele1")
    email = request("email1")
    BP = request("BP1")
```

从网页上接受用户输入，下面程序段生成 SQL 语句中的 SET 字段值部分，函数 left settext len settext - 1 用来把 SETTEXT 字符串中的最后一个逗号去掉。

```
settext = ""
if name <>"" then settext = settext& rs. fields(0). name& "
= "& epotchange(name)& ", "
if sex <>"" then settext = settext& rs. fields(1). name& "
```

```
=″& epotchange(sex)& ″, ″
if age <>″″ then settext =settext& rs. fields(2). name& ″ =″
& epotchange(age)& ″, ″
if zw <>″″ then settext =settext& rs. fields(3). name& ″ =″
& epotchange(zw)& ″, ″
if zc <>″″ then settext =settext& rs. fields(4). name& ″ =″
& epotchange(zc)& ″, ″
if jx <>″″ then settext =settext& rs. fields(5). name& ″ =″
& epotchange(jx)& ″, ″
if zj <>″″ then settext =settext& rs. fields(6). name& ″ =″
& epotchange(zj)& ″, ″
if tele <>″″ then settext =settext& rs. fields(7). name & ″
=″& epotchange(tele) & ″, ″
if email <>″″ then settext =settext& rs. fields(8). name& ″
=″ & epotchange(email) & ″, ″
if bp <>″″ then settext =settext& rs. fields(9). name& ″ =″
& epotchange(bp) & ″, ″
if settext <>″″ then
    settext =″ set ″& settext
    settext =left( settext, len(settext) -1)
end if
```

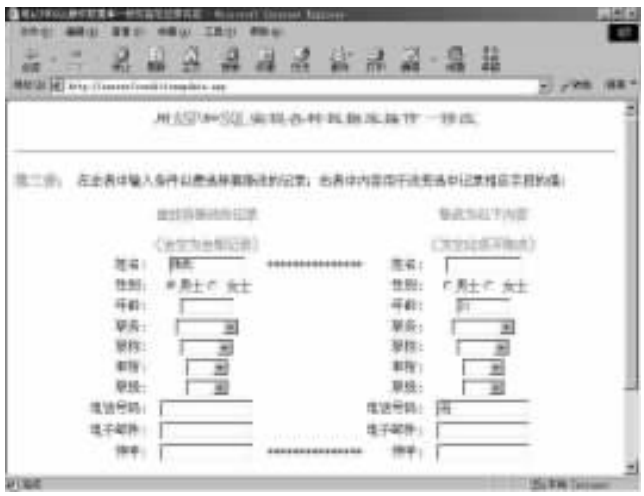
下面接受用户输入，生成 SQL 语句所需的条件，以指明对哪些记录进行修改。

```
    name =request(″name″)
    sex =request(″sex″)
    age =request(″age″)
    zw =request(″zw″)
    zc =request(″zc″)
    jx =request(″jx″)
    zj =request(″zj″)
    tele =request(″tele″)
    email =request(″email″)
    BP =request(″BP″)
where =″″
if name <>″″ then where =where& ″ and ″& rs. fields(0)
. name& ″ =″& epotchange(name)
if sex <>″″ then where =where& ″ and ″& rs. fields(1)
. name& ″ =″& epotchange(sex)
if age <>″″ then where =where& ″ and ″& rs. fields(2)
. name& ″ =″& epotchange(age)
if zw <>″″ then where =where& ″ and ″& rs. fields(3)
. name& ″ =″& epotchange(zw)
if zc <>″″ then where =where& ″ and ″& rs. fields(4)
. name& ″ =″& epotchange(zc)
if jx <>″″ then where =where& ″ and ″& rs. fields(5)
. name& ″ =″& epotchange(jx)
if zj <>″″ then where =where& ″ and ″& rs. fields(6)
. name& ″ =″& epotchange(zj)
if tele <>″″ then where =where& ″ and ″& rs. fields(7)
. name& ″ =″& epotchange(tele)
if email <>″″ then where =where& ″ and ″& rs. fields(8)
. name& ″ =″& epotchange(email)
if bp <>″″ then where =where& ″ and ″& rs. fields(9)
. name& ″ =″& epotchange(bp)
if where <>″″ then where =″ where true ″& where
```

根据条件和修改值生成完整 SQL 语句，执行后关闭，进入查看页面查看结果。

```
    rs. close
    if settext <>″″ then
        sqll =session(″sql″)& settext& ″ ″& where
        conn. execute(sqll)
        conn. close
        response. redirect ″operatebrowse. asp″
    end if%>
```

下面是修改页面和修改后的结果：



4. **分页显示的方法** operatebrowse. asp uppage. asp downpage. asp

结果显示进入 operatebrowse. asp **页面。为实现上一页和下一页的功能，采取了一种较简单的方法，即用** uppage. asp **处理向上翻页，用** downpage. asp **处理向下翻页。为保证翻页过程中记录处理的连续性，设置了两个公共变量，变量** session **“rs” 传递一个已打开的数据集，而变量** session **“line” 传递当前的记录号，以便确定翻页后的记录号。**

在程序 operatebrowse. asp **中，** <% set session **“rs” =** conn. Execute sqll % >**建立了公共变量。**

在程序 uppage. asp **中，过程** Tableprevious **用于向上翻页。**

```
<%Sub Tableprevious(rs)
dim j
```

```
Response. Write ″ <CENTER><TABLE BORDER =1>″
Response. Write ″ <TR BGCOLOR =#b0c4de>″
For i =0 to rs. Fields. Count -1
Response. Write ″ <TD>″& rs. Fields(i). Name & ″ </TD>″
        Next
        Response. Write ″ </TR>″
```

以上部分画出表头，注意表头项名称直接取自数据库字段名称，下面一小段程序的目的是调整记录指针位置，以便重新定位显示。

```
session(″line″) =session(″line″) -10
rs. movefirst
j =session(″line″)
while j <>0
    j =j -1
rs. movenext
wend
```

调整好记录指针后，顺序显示五条记录

```
j =5
While (Not rs. EOF) and (j <>0)
j =j -1
Response. Write ″ <TR>″
For i =0 to rs. Fields. Count -1
if rs. Fields(i). value <>″″ then
Response. Write ″ <TD>″ & rs. Fields(i). Value & ″ </TD>″
else
Response. Write ″ <TD>″ & ″?″ & ″ </TD>″
end if
Next
Response. Write ″ </TR>″
rs. MoveNext
Wend
Response. Write ″ </TABLE></CENTER>″
End Sub %>
```

直接根据传递来的两个公共变量，调用过程完成处理

```
<% if session(″line″)>5 then
tableprevious(session(″rs″))
session(″line″) =session(″line″) +5
end if%>
```

五、运行调试中应注意的问题

1. SQL 语句合格法的检查

调试过程中，对 SQL 语句可以采取先显示在屏幕上检查，之后再添加执行功能的方法，逐步进行。如 SQL 生成后加上一句 <% Response. Write session “sql ” % > 进行观察。

2. ASP 的运行环境有特殊的要求，

Windows NT Server 4. 0 **系统下安装** IIS 3. 0。

（PACK 3 **版本以上已内置，不须安装**）

Windows NT Workstation **系统下安装** PWS 3. 0 。

（PACK 3 **版本以上已内置**）

Windows 9X **系统下安装** PWS。

在安装光盘下，运行 hadd - ons hpws hsetup. exe）

3. ASP 程序的目录要求

程序必须存入 h Inetpub h wwwroot **下或其下的某个子目录下。**

4. ASP 程序的运行和调试

编好程序后，用 “预览 ” 方式无法查看结果，也不能用浏览器直接找到此文件查看结果，正确的方法是在浏览器地址栏内键入以下地址： “http / / servername / aspname. asp ”。其中，servername 是服务器名称，aspname. asp 是 ASP 页面文件名称。如 / / server / operate. asp。

收稿日期 ：2000 年 9 月 11 日）

上接第 47 页）

```
cmdplay. Left = Form1. Width - 1100
cmdpause. Left = Form1. Width - 1100
cmdstop. Left = Form1. Width - 1100
cmdend. Left = Form1. Width - 1100
Label1. Left = Form1. Width - 1200
Combo1. Left = Form1. Width - 1400
Slider1. Top = Form1. Height - 800
Slider1. Width = Form1. Width / 2
Slider1. Left = Form1. Width / 2 - Slider1. Width / 3 * 2
cmdopen. Height = Form1. Height / 13
cmdplay. Height = Form1. Height / 13
cmdpause. Height = Form1. Height / 13
cmdstop. Height = Form1. Height / 13
cmdend. Height = Form1. Height / 13
Label1. Height = Form1. Height / 13
cmdopen. Width = Form1. Width / 13
cmdplay. Width = Form1. Width / 13
cmdpause. Width = Form1. Width / 13
cmdstop. Width = Form1. Width / 13
cmdend. Width = Form1. Width / 13
End Sub
```

10. 当用鼠标单击或拖动进度条时，视频文件的位置要随之改变：

```
Private Sub Slider1_Click()
If ActiveMovie1. CurrentPosition <ActiveMovie1. SelectionEnd
Then
ActiveMovie1. CurrentPosition =Active-
Movie1. CurrentPosition + lider1. LargeChange
Else
ActiveMovie1. CurrentPosition =Active-
Movie1. CurrentPosition - tiveMovie1. SelectionEnd
End If
End Sub。
```

四、 编译并运行程序

按 F5 **键或单击运行按钮，编译并运行程序。**

注意 视频文件正在运行时不能选择全屏播放，要想选择全屏播放必须在暂停或停止状态下选择。要想从全屏回到原来状态，只要用鼠标单击屏幕即可。

收稿日期 ：2000 年 10 月 31 日）

用 ASP 实现对 SYBASE 数据库的万能查询

张建安

一、程序说明

客户端应用程序查询 SYBASE 数据库的数据一般只可以查询某些数据、根据某几个条件查询，常常使查询受到很大限制。而笔者编写的 ASP 的程序集中了 ASP 内置组件 ADO 的强大功能和 WEB 浏览器的安装维护简便、界面友好等优点，实现了对 SYBASE 数据库的所有表的所有数据的任意查询。

二、安装过程

1. 在服务器端和客户端分别安装 IIS 和 IE4.0 以上版本

2. 在服务器端和客户端分别安装 SYBASE 和 powerbuilder5.0 以上版本、PowerDesigner。

（1）在 NT 下安装 SYBASE 后，在 wisql32 下执行 sybtools 目录下的 pbsyc.sql 或 PB 安装盘自带的 pbsyc.sql。

（2）然后在 PB 下连接到目标库，这时便在对应的库中自动生成 PB 扩展属性表（pbcatcol、pbcatedt、pbcatfmt、pbcattbl、pbcatvld）。

（3）用 PowerDesigner 生成库表，因为它既是一种优秀的库表生成工具，又可指定样式生成扩展属性库。先连接库后，在 Dictionary / Extended Attributes / Default Attributes..... 指定 PowerDesigner 目录下的 powerbld.exa 文件，然后运行 Database / Generate Database.....，再运行 Client / PowerBuilder Attribute generation...。

注意：因为 SYBASE 数据库本身没有中文注释，而在客户端要用中文注释增强用户界面的可读性，所以要在客户端用 PB 自动生成 SYBASE 库中的扩展属性表（用于保存表和字段的中文注释等）和用 PowerDesigner 将表和注释信息一并生成到 SYBASE 库中。若您不想或应用 PB 和 PowerDesigner 有困难，可只安装 SYBASE，并修改下列含扩展属性表和记录集的语句，但那样界面将没有中文注释。

三、程序代码

在 NT 下定义 ODBC 系统数据源“sybase_newserver”，用户名和口令可设为“kf”、“ffffff”，把下列三个文件 query_11.asp、query_22.asp、query_33.asp 拷贝到 IIS 的 wwwroot 目录下，运行 query_11.asp。

（1）query_11.asp

```
<html>
<body>
<%
'建立连接和记录集
set cn=server.createobject("ADODB.Connection")
cn.open "sybase_newserver","kf","ffffff"
set rs=server.createobject("ADODB.Recordset")
rs.open "select * from sysobjects,pbcattbl where sysobjects.id=pbcattbl.pbt_tid and sysobjects.type=""U"" order by sysobjects.name",cn
%>
<form action=query_22.asp method=post>
<select name="biaoming">
<%do while not rs.eof
  response.write "<option value="""&rs("id")&"">"&rs("name")&"/"&mid(rs("pbt_cmnt"),1,40)&"</option>"
  rs.MoveNext
loop%>
</select>
<p><input type=submit name="send" value="选定表名">
</form>
<%
rs.close
set rs=nothing
cn.close
set cn=nothing
%>
</body>
</html>
```

query_11.asp 运行图示

（2）query_22.asp

```
<html>
<body>
<%
'从 query_11.asp 窗口得到传递的表 id 和表名
getstr=request.form("biaoming")
'建立连接和记录集
```

```
set cn = server. createobject( ″ADODB. Connection″)
cn. open ″sybase_newserver″, ″kf″, ″ffffff″
set rs2 = server. createobject( ″ADODB. Recordset″)
rs2. open ″select syscolumns. name, systypes. name, pbcatcol. pbc_hdr, systypes. type from systypes, syscolumns, pbcatcol where systypes. usertype = syscolumns. usertype and syscolumns. id = pbcatcol. pbc_tid and syscolumns. colid = pbcatcol. pbc_cid and syscolumns. id = ″ & left(getstr, instr(getstr, ″ ~ ″) - 1), cn
´ii 表示某表的字段列数
ii = 0
do while not rs2. eof
    ii = ii + 1
    rs2. MoveNext
loop
%>
<form action = query_33. asp method = post>
<input type = submit name = ″send″ value = ″ 查询 ″><p>
<%For i = 1 to ii%>
    <select name = ″lieming <% = i%>″>
    <option></option>
    <%rs2. MoveFirst%>
    <%do while not rs2. eof
response. write ″ <option value = ″″″ & rs2(0) & ″ ~ ″& rs2(3) & ″″″>″ & rs2(0) & ″/″& rs2(1) & ″/″& rs2(2) & ″ </option>″
rs2. MoveNext
    loop%>
    </select>
    <select name = ″suanfu <% = i%>″>
    <option></option>
    <option>等于 </option>
    <option>like </option>
    <option>大于 </option>
    <option>小于 </option>
    </select>
<input type = ″text″ size = ″20″ name = ″value <% = i%>″>
    <p>
<%Next%>
<input type = ″hidden″ name = ″getcount″ value = <% = i-i%>>
<input type = ″hidden″ name = ″tablename″ value = <% = mid(getstr, instr(getstr, ″ ~ ″) + 1)%>>
</form>
<%
rs2. close
set rs2 = nothing
cn. close
set cn = nothing
%>
</body>
</html>
```

（3）query_33. asp

```
<html>
```

query_22. asp 运行图示

```
<body>
<%
´从 query_22. asp 窗口得到传递的表列数, 并定义数组.
´数组 a 表示列名, 数组 b 表示得到的中文运算符, 数组 c 表示要查询列的值, 数组 d 表示列数据类型 id
´数组 s 表示 query_22 窗口的某一行的判断条件, 即 Where 子条件
Redim a(request. form( ″getcount″))))))
Redim b(request. form( ″getcount″))
Redim c(request. form( ″getcount″))
Redim d(request. form( ″getcount″))
Redim s(request. form( ″getcount″))
´对数组 a、b、c、d 赋初值, gg 表示得到的列名、列数据类型 id
For i = 1 to request. form( ″getcount″)
  gg = request. form( ″lieming″ & i)
  if gg <>empty then
      a(i) = left(gg, instr(gg, ″ ~ ″) - 1)
      d(i) = mid(gg, instr(gg, ″ ~ ″) + 1)
  else
      a(i) = empty
      d(i) = empty
  end if
  b(i) = request. form( ″suanfu″ & i)
  c(i) = request. form( ″value″ & i)
Next
´得到所有的 Where 子条件的集合—数组 s(i)
For i = 1 to request. form( ″getcount″)
  if a(i) = empty or b(i) = empty or c(i) = empty or d(i) = empty then
      s(i) = empty
  else
      ´处理 b(i), 把 b(i)由中文转换成运算符
      Select case b(i)
          case ″等于″
              b(i) = ″ = ″
          case ″like″
              b(i) = ″like″
          case ″大于″
              b(i) = ″>″
          case ″小于″
              b(i) = ″ <″
      End Select
  ´处理 c(i), 由列数据类型 id 值 d(i), (或 b(i))得到要查询列值 c(i)在语句中的相应格式
      Select case d(i)
          ´当类型是 ″datetime″, ″smalldatetime″
```

```
case "61", "58"
                c(i) = "'" + c(i) + "'"
        '当类型是 "char", "nchar", "nvarchar", "varchar"
            case "47", "47", "39", "39"
            Select case b(i)
                case " ="
                    c(i) = "'" + c(i) + "'"
                case "like"
                    c(i) = "'%" + c(i) + "%'"
            End Select
        '当类型是 "tinyint", "smallint", "int", "numeric",
"decimal", "float", "real"
        case "48", "52", "56", "63", "55", "62", "59"
                c(i) = c(i)
            case else
                '其它默认
                    c(i) = "'" + c(i) + "'"
        End Select
        '得到某一行的判断条件,即 Where 子条件
        s(i) = a(i) + " " + b(i) + " " + c(i)
    end if
Next
'由所有的 Where 子条件串联,得到相应的 sql 语句
For i = 1 to request. form("getcount")
    if s(i) <>empty then
        where_sub = where_sub + s(i) + " and "
    end if
Next
where_sub = left(where_sub, instrrev(where_sub, " and "))
if where_sub <>empty then
    sql = "select * from " + request. form("tablename") +
" where " + where_sub
else
    sql = "select * from " + request. form("tablename")
end if
'建立连接和记录集,rs 表示最终的 SQL 语句,rs3 表示表头
set cn = server. createobject("ADODB. Connection")
cn. open "sybase_newserver", "kf", "ffffff"
set rs = server. createobject("ADODB. Recordset")
rs. open sql, cn
set rs3 = server. createobject("ADODB. Recordset")
rs3. open "select * from pbcatcol where pbc_tid = (select
pbt_tid from pbcattbl where pbt_tnam = """ & request. form
("tablename") & """)", cn
'最终的结果输出
Response. write "<table border = 1>"
'表头
do while not rs3. eof
    response. write "<th>" & rs3("pbc_hdr") & "</th>"
    rs3. MoveNext
loop
response. write "</tr>"
'表内容
While not rs. eof
    For i = 0 to rs. fields. count - 1
        response. write "<td>" & rs(i) & "</td>"
    Next
    rs. MoveNext
    response. write "</tr>"
wend
Response. write "</table>"
%>
</body>
</html>。
```

query_33. asp 运行图示

收稿日期:2000 年 11 月 9 日)

(上接第 55 页)

```
            waveFile. Read(lpbuf2, size);
            memset(& whdr2, 0, sizeof(WAVEHDR));
            whdr2. lpData = lpbuf2;
            whdr2. dwBufferLength = size;
          waveOutPrepareHeader(hWave, & whdr2, sizeof
(WAVEHDR));
      waveOutWrite(hWave, & whdr2, sizeof(WAVEHDR));
            //现在正在播放的是缓存区 1
            bufID = 1;
        }
      }
    }
    waveOutClose(hWave);
    WaitForSingleObject(hEvent, INFINITE);
    delete [] lpbuf1;
    delete [] lpbuf2;
    CloseHandle(hEvent);
}
```

收稿日期:2000 年 11 月 28 日)

朗科发布支持苹果机 Mac OS 的优盘驱动程序

2001 年 2 月 6 日,朗科公司成功推出苹果 Macintosh 机操作系统 Mac OS 的优盘驱动程序 任何支持 Mac OS 操作系统和通用串行总线 (USB) 的苹果电脑 包括 imac、Power GX、ibook 等系列机型),都可以使用优盘自由进行数据存储及交换。尤其重要的是使用优盘可以方便实现苹果机、PC 及笔记本电脑三者之间跨平台的数据自由交换。

这一研发成果使今后从事各类广告、工程设计的人员再也不需要拿着设计效果图纸来回奔波于客户与公司之间了。只要一个优盘,就可在客户的 PC 机上按客户的意愿随意实现修改。

JSP 技术运用及其编程环境的建立

杜文峰 蔡自兴

JSP 是一种很容易学习和使用的在服务器端编译执行的 Web 设计语言，它是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准。在传统的网页 HTML 文件 ＊.htm ＊.html 中加入 Java 程序片段 Scriptlet 和 JSP 标记 tag，构成 JSP 网页 ＊.jsp。Web 服务器在遇到访问 JSP 网页的请求时，首先执行其中的程序片段，然后将执行结果以 HTML 格式返回给客户。程序片段可以操作数据库、重新定向网页以及发送 email 等等，这就是建立动态网站所需要的功能。所有程序操作都在服务器端执行，网络上传送给客户端的仅是得到的结果，对客户浏览器的要求很低。本文将以新建的中南大学信息工程学院智能控制研究所同学录来介绍一些 JSP 语法，以及如何利用 JSP 与数据库交互，从而实现动态网页。

由于 JSP 页面的内置脚本语言是基于 Java 编程语言的，而且所有的 JSP 页面都被编译成为 Java Servlet，JSP 页面就具有 Java 技术的所有好处，包括健壮的存储管理和安全性。

作为 Java 平台的一部分，JSP 拥有 Java 编程语言“一次编写，各处运行”的特点。随着越来越多的供应商将 JSP 支持添加到他们的产品中，您可以使用自己所选择的服务器和工具，更改工具或服务器并不影响当前的应用。当 JSP 页面第一次被调用时，如果它还不存在，就会被编译成为一个 Java Servlet 类，并且存储在服务器的内存中。这使得在接下来的对该页面的调用有非常快的响应。 这避免了 CGI - BIN 为每个 HTTP 请求生成一个新的进程的问题，或是在服务器端引用时所引起的运行时语法分析。）

另外，servlets 的字节代码只有在客户请求时才执行，所以尽管当首次调用 servlets 时会有几秒钟的加载时间，但后续的请求相应非常迅速，因为服务器已经缓存了运行的 servlets。当前的 JSP 服务器都带有 Java 即时编译器 (JIT)，因此，JSP 的执行比每次都要解释执行的 ASP 代码要快，尤其是在代码中存在循环操作时，JSP 的速度要快 1 到 2 个数量级。

1. 首先介绍一下 JSP 的开发环境和运行环境。 JDK1.2 + JSWDK

以 Windows 和 Windows NT 环境为例，首先安装 Java 语言的开发环境 JDK1.2 Java 2 SDK、Standard Edition、v 1.2.2 可在 http ／／java.sun.com／jdk／处下载 ，安装完 JDK 以后必需在 autoexec.bat 里加入以下语句，使 JDK 能正常运行。 以装在 C hjdk1.2 目录为例

SET PATH = c hjdk1.2 hbin

SET CLASSPATH = .；c hjdk1.2 hlib htools.jar

在安装完 JDK 以后，必须安装 JSWDK JavaServer Web Development Kit 1.0.1 可在 http ／／java.sun.com／products／jsp／处下载 JSWDK 的安装仅需将 jswdk1_0_1 - win.zip 带目录释放到硬盘根目录下 (c h、d h 等）就可以了，此时你将发现 jswdk - 1.0.1 这个目录，在 hjswdk - 1.0.1 h目录下执行 Startserver.bat，就可启动 JSWDK 中一个支持 JSP 网页技术的 Web 服务器。为了不与现有的 Web 服务器 例如 IIS、PWS 等）冲突，JSWDK 的 Web 服务器使用了 8080 端口。在浏览器的地址栏中键入 http ／／localhost 8080 或者 http ／／127.0.0.1 8080 后，如果能看到 JSWDK 的欢迎页就说明 JSP 实验环境已经建成，可进入下一步实验。 要关闭 Web 服务器则运行 Stopserver.bat 即可。但在有些情况下，安装的 JSWDK 不能运行，则需在 autoexec.bat 里加入以下语句：

SET JAVA_HOME = C hJDK1.2

如果在 Windows98 中，运行 JSWDK 目录中的 Startserver.bat 文件会出现 Out of Environment Space error message 的错误，这是因为你给环境变量设置的空间太小的原因。你可以再打开一个 Dos 窗口，然后点击窗口左上角的 MS - DOS 图标，此时将出现一个下拉菜单，选择属性项里的内存页，把常规内存里的初始环境值改为 2816，然后点击 OK 即可。此时运行 Startserver.bat 就不会出现那些错误了。在 Windows2000 里则不需要调整这个变量。安装好 JSWDK 以后，打开浏览器，在地址栏里键入 http ／／localhost 8080，即可以看到 JSWDK 的欢迎页面。如果使用 localhost 不行，可以将 localhost 改为你的主机名。 原因是你在 IE 里的设置不是在本地地址不使用代理服务器。）

JSWDK 缺省的文档目录是 hjswdk - 1.0.1 hwebpages，在此目录下可以建立子目录，例如 hjswdk - 1.0.1 hwebpages hmytongxuelu，就能在浏览器中用 http ／／localhost 8080／mytongxuelu 访问这个目录。为了使得这个子目录能执行 JSP 程序，还必须在 webserver.xml 中的＜Service＞ ＜／Service＞ 节加入：＜WebApplication id = “mytongxuelu” mapping = “／mytongxuelu” docBase = “webpages／mytongxuelu”／＞ (可参考原来的 webserver.cml 里的设置），并且还必须建立 h jswdk - 1.0.1 h webpages h mytongxuelu h WEB - INF 目录，并从 h jswdk - 1.0.1 h webpages h WEB - INF 目录中复制过来以下四个文件：mappings.properties、mime.properties、servlets.properties 以及 webapp.properties。完成

	功能描述	JSP 语法
HTML 注释	在客户端建立一个可见的注释	<! - - 注释内容 - - >
JSP 注释	JSP 注释,在客户端不可见	<% - - 注释 - - %>
声明变量	在 JSP 页面里声明 JAVA 变量	<%! TYPE 变量 %>
脚本语言	在脚本语言里加入 JAVA 语言脚本	<% 脚本代码 %>
表达式	显示 JSP 里的 JAVA 变量值	<% = JAVA 变量%>
包含指示	在 JSP 页面里加入动态或静态文件以及 JAVA 类	<%@ include file = "相关连接"%> <%@ page language = "java"%> <%@ page import = "java. sql. * "%>
使用 Bean	在 JSP 页面里使用具有特定名字的 JAVA Bean	<jsp useBean id = "本页里使用的 Bean 的标志"scope = "page ,request, Session 以及 application"class = "引入的 Bean 的包名及类名"> 使用 Bean 的代码…… </jsp useBean>
指向特定页		<jsp forward page = "相应的页面地址"/>

这些过程，才能通知 JSWDK 的 Web 服务器执行 http / /localhost / test 中的 JSP 程序。

2. 本同学录页面所使用的 JSP 语法

如果你学过 ASP，你将发现 JSP 的语法与 ASP 很相似。现将 JSP 的基本语法规则总结如下：

JSP 里有一些常用的隐含对象，使用它们可以与 HTML 页面很方便的实现交互。

(1) Request 它是 HttpServletRequest 的一个子类。该对象包含了所有有关当前浏览器请求的信息，包括 Cookies ，HTML 表单变量等等。通过使用 request 类的 getParameter 方法可以得到上一个 HTML 页面里的表单所提交的内容。如：上一个 HTML 页面里有一个名为 h_entryname 的单行文本框，则在那些想得到这个 HTML 页面里 entryname 值的紧跟着的 JSP 页面里使用 <% String name = request. getParameter ("h_entryname") %>来得到表单提交后的内容。然后你可以在 JSP 页面里使用 <% = name %>来显示 name 的值。

(2) Session 这个对象在第一个 JSP 页面被装载时自动创建，并被关联到 request 对象上。它与 ASP 中的会话对象很相似， JSP 中的 session 对象对于那些希望通过多个页面完成一个事务的应用是非常有用的。例如：在页面里使用 request 传递信息只能在相近的页面传递，如果传递的页面间隔大于 2 就不能得到数据，因此可以使用 Session 对象来传递数据，通过 request 和 Session 的配合，可以实现在间隔很大的页面中传递数据。

例如：在第三个页面里想得到第一个页面的数据，可以在第二个 JSP 页面里加入以下语句，把变量的值放入 Session 对象里。

```
<% entryname = request. getParameter( "h_entryname" );
        session. putValue( "h_entryname", entryname);
   %>
```

在第三个 JSP 页面里使用下列语句得到数据：

```
<% String entryname = (String) session. getValue( "
h_entryname" ); %>
```

(可以使用 <% = entryname %> 来显示 entryname 的值。)

3. JSP 与 JavaBean

JSP 网页吸引人的地方之一就是能结合 JavaBean 技术来扩充网页中程序的功能。JavaBean 是一种 Java 类 class ，通过封装属性和方法使其成为具有某种功能或者处理某个业务的对象。在 JavaBean 中可以用 Java 语言来编写一些功能强大的方法，通过这些方法来实现与数据库的交互，当然你也可以把一些要在 JSP 页面里实现的功能放在 JavaBean 里，从而简化 JSP 脚本的编写。而且，由于 JavaBean 的通用性很强，你可以在多个页面里使用同一个 JavaBean 来实现一些相同的功能。比起 ASP 里使用 COM 技术来说，JavaBean 编写简单，而且可以实现 COM 同样的功能。

与数据库的连接对于动态网站来说是最为重要的部分，Java 中连接数据库的技术是 JDBC Java Database Connectivity 。很多数据库系统带有 JDBC 驱动程序，Java 程序就通过 JDBC 驱动程序与数据库相连，执行查询、提取数据等等操作。Sun 公司还开发了 JDBC - ODBC bridge，用此技术 Java 程序就可以访问带有 ODBC 驱动程序的数据库，目前大多数数据库系统都带有 ODBC 驱动程序，所以 Java 程序能访问诸如 Oracle、Sybase、MS SQL Server 和 MS Access 等数据库。对于一个同学录来说，当有同学登录，或查询，修改同学录里的内容，与数据库的交互是不可避免的，在这里为了示例方便，我使用的数据库是 MS Access。

在数据库里建了一个名为 personinfo. mdb 的数据库，并在 Windows 里的 ODBC 数据源里把它设为系统 DSN，命名为 personinfo，登录名为 dwf，登录密码为 lmh，并让它指向 personinfo. mdb。在本同学录里，我的数据库所设的字段有 entryname (文字型，长度 50)、 password (文字型，长度 50)、 birth_year 数字型，长度 4 、 birth_month 数字型，长度 2 、 birth_day 数字型，长度 2 、 grdu_year 数字型，长度 4 、 grdu_month 数字型，长度 2 、 grdu_day 数字型，长度 2 、 sex (文字型，长度 10)、 telephone (文字型，长度 20)、 address (文字型，长度 200)、 workplace (文字型，长度 200)、 e_mail (文字型，长度 50)。让 entryname 为主键，然它来标志登录的用户。

在本同学录里，为了实现相应的功能，我制作了一个名为 beanfordenlu. java 的 JavaBean 文件。它使用了 JDBC - ODBC 桥与数据库连接。

下面是这个 Bean 的代码清单：

```
//导入必要的 package
import java. sql. * ;
public class beanfordenlu{     // 此处 beanfordenglu 为 JavaBean 类的名称
//声明变量
public Connection Conn;
public Statement Stmt;
public ResultSet Rs;
```

```
private String SQL;
//构造函数
public void beanfordenlu(){
}
//得到一个与数据库连接的 Statement
public void getStmt(){
String SEVER = "jdbc: odbc: personinfo";
String User = "dwf";
String Password = "lmh";
try{
Class. forName( "sun. jdbc. odbc. JdbcOdbcDriver"); //先得到一个 DriverManage
//得到一个与数据库连接的 Connection
Conn = DriverManager. getConnection(SEVER, User, Password); Conn. setAutoCommit(false);
//通过 Conn 的到一个 Statement
Stmt = Conn. createStatement();
}
catch(Exception e){
System. out. println("can't get a statement");
}
}
//设置 SQL 语句
public void setinsertSQL(String SQL){
this. SQL = SQL;
}
//由一个 name 产生一条 SQL 语句, 用于确定数据库里是否已经存在以 name 这个值为 entryname 的用户
public void setentrySQL(String name){
this. SQL = "select * from personinfo where entryname = '"
+name + "'";
}
//由 name 和 password 来产生一条 SQL 语句, 用于确定用户的身份
public void setchangeSQL(String name, String password){
this. SQL = "select * from personinfo where entryname = '"
+name + "' and password = '" +password + "'";
}
public void setQrySQL(){
this. SQL = "select * from personinfo";
}
//确定查询结果是否为空
public boolean isEmpty() throws SQLException{
  Rs = Stmt. executeQuery(SQL);
  if(Rs. next())
    return false;
  return true;
}
//返回一个 ResultSet
public ResultSet getRs(){
return Rs;
}
//执行 SQL 语句
public void doinsert(){
try{
  Stmt. executeUpdate(SQL);
  Conn. commit();
  };
catch(Exception e){
System. out. println("insert err")
}
}
//从数据库里得到数据
public String getdate(String datename) throws SQLException
{
String date = Rs. getString(datename). toString();
if(date = =null) date = "";
return date;
}
//关闭数据库
public void dbclose(){
try{
if(Rs! =null)
  Rs. close();
if(Conn! =null)
  Conn. close();
}
catch(SQLException e){
System. out. println("close err");
System. out. println(e. getMessage());
}
}
}
```

4. 在 JSP 中与数据库交互

在 JSP 页面里可以通过调用 JavaBean 实现与数据库的交互，通过 JavaBean 把从页面里得到的数据存到数据库里，同时从数据库里提取数据。在 JSP 里的使用 JavaBean 的脚本为：

<jsp: useBean id = "mytongxuelu" scope = "page" class = "beanfordenlu">

Java 代码：

</jsp: useBean>

通过在这里使用标记 <jsp useBean>把 JavaBean 引到 JSP 页面里，有效范围为本页。Beanfordenlu. class 在本页里的标志为 id，即 mytongxuelu，然后就可以在本页里通过 mytongxuelu. getStmt 等方法来调用 JavaBean 里的方法。

在同学录里有注册用户的功能。来访者通过表单向 JSP 引擎输入了数据,并保存在了 request 对象中。首先在 chooseentryname. html 里输入用户想得到的登录名，如果该登录名已经存在则显示错误信息，否则进入登录页面。 登录页面里用到上述的 Session 方法传送数据）。在第三个页面里我们使用 request 方法从第二页里得到数据，并用于产生 SQL 语句。对于在 SQL 语句里的 entryname 则用 Session 方法得到。

产生 SQL 语句内容页的代码清单如下： 对于一些重复的部分，笔者进行的适当的删减）

```
<html>
<head>
<meta http – equiv = " Content – Language" content = " zh – cn">
<meta http – equiv = "Content – Type" content = "text/html; charset = gb2312">
```

```
<title>智能所同学录 neirong </title>
</head>
<body>
<%@ page contentType = "text/html; charset = gb2312"%>
<%@ page language = "java" %>
<%@ page import = "beanfordenlu" %>
<! --调用 JavaBean -- >
<jsp: useBean id = " mydenglu" scope = " page" class = "
beanfordenlu">
<! --使用 Session 得到第一页里的变量 -- >
<% String entryname = (String) session. getValue( "
h_entryname");
//使用 request 得到第二页里用户所填入的数据
String password = request. getParameter("h_password1");
String name = request. getParameter("h_name");
String birth_year = request. getParameter("h_year1");
String birth_month = request. getParameter("h_month1");
String birth_day = request. getParameter("h_day1");
String e_mail = request. getParameter("h_e_mail");
String sqlforword = "insert into personinfo(";
String sqllast = "values(";
%>
<font color = "#0000FF" size = "5" face = "华文彩云> 您的
档案: </font>
<p> </p>
<table border = " 1" width = " 100% " bordercolorlight = " #
0000FF" bordercolor = "#0000FF">
<tr>
<td width = "50%">& nbsp; & nbsp; & nbsp; & nbsp; 项
& nbsp; 目& nbsp; 名& nbsp; 称 </td>
<td width = "50%" colspan = "4">& nbsp; & nbsp;
& nbsp; & nbsp; & nbsp; & nbsp; & nbsp; & nbsp; & nb-
sp; & nbsp; & nbsp; & nbsp;
内& nbsp; & nbsp; & nbsp; & nbsp; 容 </td>
</tr>
<! --产生 SQL 语句 -- >
<%
if(entryname! = "") {
entryname = new String(entryname. getBytes( "8859_1"));
//处理 JSP 页面里的中文问题
sqlforword = sqlforword + "entryname, ";
sqllast = sqllast + "'" + entryname + "', "; %>
<tr>
<td width = '50%'>& nbsp; & nbsp; & nbsp; & nbsp; 登录名
称: </td>
<td width = '50%' colspan = "4"><% = entryname %></
td>
</tr>
<% }
if(name! = "") {
name = new String(name. getBytes("8859_1")); %>
<tr>
<td width = '50%'>& nbsp; & nbsp; & nbsp; & nbsp; 您的姓
名: </td>
<td width = '50%' colspan = "4"><% = name%></td>
</tr>
<% }
if(password! = "") {
sqlforword = sqlforword + "password, ";
sqllast = sqllast + "'" + password + "', ";
}
if((birth_year! = "") || (birth_month! = "") || (birth_day!
= "")) {%>
<tr>
<td width = '50%'>& nbsp; & nbsp; & nbsp; & nbsp; 您的
出生日期: </td>
<% if(birth_year! = "") {
sqlforword = sqlforword + "birth_year, ";
sqllast = sqllast + birth_year + ", "; %>
<td width = '9%'><% = birth_year%>年 </td>
<% }
if(birth_month! = "") {
sqlforword = sqlforword + "birth_month, ";
sqllast = sqllast + birth_month + ", "; %>
<td width = '6%'><% = birth_month%>月 </td>
<% }
if(birth_day! = "") {
sqlforword = sqlforword + "birth_day, ";
sqllast = sqllast + birth_day + ", "; %>
<td width = '7%'><% = birth_day%>日 </td>
<% } %>
<td width = '28%'> </td>
</tr>
<% }
if(e_mail! = "") {
sqlforword = sqlforword + "e_mail, ";
sqllast = sqllast + "'" + e_mail + "', "; %>
<tr>
<td width = '50%'>& nbsp; & nbsp; & nbsp; & nbsp; 您的
e -mail: </td>
<td width = '50%' colspan = '4'><% = e_mail%></td>
</tr>
<% }
if(name! = "") {
name = new String(name. getBytes("8859_1"));
sqlforword = sqlforword + "name";
sqllast = sqllast + "'" + name + "'";
}
sqlforword = sqlforword + ") ";
sqllast = sqllast + ")";
String SQL = sqlforword + sqllast;
%>
</table>
<p>& nbsp; & nbsp; <font size = " 4">& nbsp; <font col-
or = "#0000FF" face = "华文行楷">祝贺您已经加入信息工程
学院智能控制研究所的同学录 </font></font>
</p> <p>
<font color = "#0000FF" face = "仿宋 _GB2312"><a href = "
AIPAGE1. htm">返回同学录首页 </a></font>
</p> <p>
</p>
<! --JavaBean 的方法调用 -- >
<%
mydenglu. getStmt();
```

```
    mydenglu. setinsertSQL(SQL);
    mydenglu. doinsert();
    mydenglu. dbclose();
%>
<! – – 停止 JavaBean 的调用 – – >
</jsp: useBean>
</body>
```

当执行了这段代码后，JSP 页面就把用户输入的数据通过 JDBC - ODBC 桥存入到 Access 数据库里。我们登录时就能使用此次登录的用户进行登录。对于登录的用户，我们可以使用 JavaBean 里的 SetchangeSQl String name String password 方法来产生查询 SQL 语句，并调 JavaBean 里的 isEmpty 方法来确定用户的身份，从而引导不同的操作。

5. JSP 的处理过程

一个 JSP 原文件的处理分为两个阶段：一个是 HTTP 的编译时候，一个是请求的处理时间。

1 HTTP 编译的时候，当用户第一次读 JSP 页面的时候，JSP 的原代码被编译成 CLASS，通常是 servlet。HTML 标签和 JSP 标签在这个时候同时被处理了，这之前用户还没有任何的请求被提交。

2 请求处理时间是当用户在 JSP 页面中提交了一个请求，这时请求由客户端被 request 对象传到了服务器端，接着 JSP 引擎把存放在 request 对象中的数据发到 JSP 页面指定的服务器端的组件（JavaBeans 组件 servlet 或者 enterprise bean），组件收到这些个数据以后，有可能再把这些数据存入到数据库或者其他的地方存放起来，同时，返回一个 response 对象给 JSP 引擎。JSP 引擎再把 response 对象传给 JSP 页面，这时的页面包含了定义好的格式和从服务器端得到的数据。这时 JSP 引擎和 Web 服务器再发送一个整理好的完整的页面给客户，也就是他们在浏览器上看到的结果。

6. JSP 在编写和调试中应注意的问题

在 JSP 中对写法非常敏感，不可以有一点错误，笔者所用的编写环境是 MicroSoft FrontPage。当出现错误时，在浏览器里的 JSP 页面就不能正常的显示出来，而且在 JSWDk 运行窗口会显示相应的错误提示，你可以根据窗口上的错误提示进行修改。如果实在找不到错误所在，不妨先检查一下你所使用的 JSP 语法，然后再检查你的逻辑错误。可以把 JSP 语法句逐条使用，直到可以完全使用为止。使用 MicroSoft FrontPage 可以很方便地对 JSP 页面进行修改，而在 IE 里快速地看到运行结果，这样可以很大地提高编写效率。

事实证明， Java Servlet 是一种开发 Web 应用的理想构架。 JSP 以 Servlet 技术为基础，又在许多方面作了改进。JSP 页面看起来象普通 HTML 页面，但它允许嵌入执行代码，在这一点上，它和 ASP 技术非常相似。利用跨平台运行的 JavaBean 组件， JSP 为分离处理逻辑与显示样式提供了卓越的解决方案。 JSP 必将成为 ASP 技术的有力竞争者。

收稿日期：2000 年 10 月 25 日）

关于 USB

彩虹天地的 GS - UMH 套装产品中提供了并口和 USB 接口的加密狗各一只，及一根 USB 接口的延长线。USB 接口的产品的推出紧跟技术发展的潮流，同时也解决了并口产品固有的某些问题。留心一下可以发现，在 PC 机上已经有了大量的 USB 外设可供使用，包括打印机、扫描仪、显示器、键盘、鼠标、硬盘、数码相机、音箱等等。当然彩虹天地的 GS - UMH 也是其中的一种。

USB （Universal Serial Bus），通用串行总线，是由 Intel 等公司为简化 PC 与外设之间的互连而共同研究开发的一种外部总线结构，它支持各种外设与 PC 之间的连接。USB 总线结构简单，使用的四根信号线中，两根是 5V 的电源线，另外的两根是数据线。目前普遍采用的 USB 总线标准 1. 1 主要应用在中低速外部设备上，提供的传输速率有低速 1. 5Mbps 和全速 12Mbps 两种。USB 总线标准 2. 0 的传输速率可达到 240Mbps，将有着更为广泛的应用。

在加密狗产品方面，USB 产品较并口产品的个头更小巧一些。并口产品有可能因串接的其他外设拉低并口工作电压或主机的节电方式导致加密狗不能正常工作。而 USB 接口中独立的电源线将提供稳定的电源，使得加密狗工作更稳定、可靠。尽管在加密狗的设计中考虑了与其他并口外设串联的问题，并且在安装了加密狗并口驱动程序后可以解决大多数的并口冲突问题（打印问题等），使加密狗可以和其他并口外设很好的共同工作。但是由于并口标准的局限性，并口加密狗不能保证和串联的所有并口外设都百分之百兼容。而 USB 本身支持连接多个外设，所以不存在此类问题，可以将 USB 加密狗插入任何一个 USB 端口中使用。并口产品需在关掉主机的情况下插拔外设，否则容易损毁并口或外设。USB 加密狗支持热插拔，客户可以直接插拔，使用更加安全、灵活。

对客户而言，加密工作只需进行一次即可支持并口和 USB 加密狗。原 MH 加密的软件产品不再改动，更换驱动程序后可直接使用 USB 加密狗。USB 加密狗的突出优点弥补了并口加密狗的不足，将满足更多客户的需求，保证客户发行的软件的高品质。

Windows NT 下的 Services 的实现

冯志林

摘 要 本文介绍了 Windows NT 的重要特性——服务 以及它的实现原理，并给出了一个服务的具体实现过程。

关键词 服务，服务控制管理器，线程

一、引言

几乎所有的操作系统都支持后台任务的运行，这些后台任务可以为系统或用户提供各种类别的服务。在 Windows NT 中，后台任务被称为服务 Services 。Windows NT 预先安装了许多服务来处理诸如网络消息通讯、命令调度等后台任务。Win32 服务是 Windows NT 的一个重要特性，通过定制服务便可以对操作系统的功能进行扩展。Win32 服务遵循一个特定的内部协议，该协议保证所有的服务能正确地与服务控制管理器 Service Control Manager ，简称 SCM 进行交互。所有的服务，无论复杂还是简单，都必须包含基本的 SCM 接口代码。NT 用户可以通过控制面板查看 NT 服务，当打开控制面板中的服务程序后，可以看到可供使用的 Win32 服务列表。在服务程序中，你除了可以对服务进行启动、停止、暂停或继续等操作外，还可以设置服务的启动参数。当创建完新服务后，必须执行单独的安装步骤把该服务插入到 Win32 服务列表中。通过安装可以把服务的名称、启动类型等信息插入到注册表中，这样 SCM 便可以获得服务的信息。当然，你也必须执行单独的卸载步骤将服务进行删除，从而恢复 Win32 服务列表和注册表中的信息。

二、创建新服务流程

在设计服务程序时必须满足特定函数调用的流程，否则新服务将不能正常工作。首先，调用标准 main 函数，然后调用 StartServiceCtrlDispatcher 函数。通过调用该函数，你可以把指向 ServiceMain 函数的指针传递给 SCM，当 SCM 准备启动服务时，SCM 便可以通过这个函数指针启动服务；在 ServiceMain 函数中调用 RegisterServiceCtrlHandle 函数，该函数产生服务状态句柄，同时注册与 SCM 进行交互的服务控制函数 Handler。Handler 函数中包含一个 Switch 语句，它用于分发由 SCM 发送的 5 个控制通知事件 SERVICE_CONTROL_STOP、SERVICE_CONTROL_PAUSE、SERVICE_CONTROL_CONTINUE、SERVICE_CONTROL_INTERROGATE、SERVICE_CONTROL_SHUTDOWN ，你可以分别对这些通知事件进行相应的处理，然后将处理后的新服务的最新状态消息发送给 SCM；最后，ServiceMain 函数还必须启动实现新服务特定功能的线程函数。因此，要创建一个服务程序，应在程序中包含 main、ServiceMain、Handler 函数以及体现服务特定功能的线程函数。当新服务实现后，还必须进行安装和卸载的操作，这可通过调用 OpenSCManager 函数，打开服务控制管理数据库，然后调用 CreateService 和 DeleteService 进行相应的安装和卸载操作。下面通过创建一个简单的新服务来阐述上述函数之间的交互关系，该服务的功能是每隔 3 秒进行一次蜂鸣。

三、服务制作的过程

制作过程由 3 部分组成，即新服务的实现、新服务的安装、新服务的卸载。

3. 1 新服务的实现 beep. cpp ：共有 6 个组成部分；

1. 定义服务使用的句柄：

```
#include <windows. h>
#include <stdio. h>
#include <iostream. h>
#include <stdlib. h>
#define DEFAULT_BEEP_DELAY 3000 // 定义蜂鸣的间隔缺省时间值
char *SERVICE_NAME ="BeepService"; //定义服务的名称
HANDLE terminateEvent = NULL; // 定义用于控制 ServiceMain 执行的事件句柄
SERVICE_STATUS_HANDLE serviceStatusHandle; // 定义与 SCM 通讯的服务状态句柄
int beepDelay = DEFAULT_BEEP_DELAY; // 定义蜂鸣的间隔时间
BOOL pauseService = FALSE; // 定义服务暂停的标志
BOOL runningService = FALSE; // 定义服务运行的标志
HANDLE threadHandle =0; //定义实现服务功能的线程句柄
```

2. SendStatusToSCM 函数：

用于填充包含服务状态信息的 SERVICE_STATUS 结构，并更新保存在 SCM 中的新服务的状态信息；

```
VOID SendStatusToSCM (DWORD dwCurrentState, DWORD dwWin32ExitCode,
DWORD dwServiceSpecificExitCode, DWORD dwCheckPoint, DWORD dwWaitHint)
{
  SERVICE_STATUS serviceStatus;
serviceStatus. dwServiceType =SERICE_WIN32_OWN_PROCESS;
  //设置服务运行在其自身的进程中
```

```
serviceStatus.dwCurrentState = dwCurrentState;    //设置服务的当前状态
// 如果服务正在启动,则不接收和处理控制通知事件,否则接收所有的控制通知事件
if (dwCurrentState == SERVICE_START_PENDING)
    serviceStatus.dwControlsAccepted = 0;
    else serviceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP |SERVICE_ACCEPT_PAUSE_CONTINUE |SERVICE_ACCEPT_SHUTDOWN;
 /* 如果服务具有特定的出错代码, 则设置 serviceStatus 的 dwWin32ExitCode 属性为 ERROR_SERVICE_SPECIFIC_ERROR,并设置 serviceStatus 的 dwServiceSpecificExitCode 属性为特定出错代码,以显示特定的错误信息*/
if (dwServiceSpecificExitCode == 0
    {
serviceStatus.dwWin32ExitCode = dwWin32ExitCode;
serviceStatus.dwServiceSpecificExitCode =0;
        }
    else
    {
serviceStatus.dwWin32ExitCode =ERROR_SERVICE_SPECIFIC_ERROR;
serviceStatus.dwServiceSpecificExitCode =dwServiceSpecificExitCode; //设置服务的特定出错代码
        }
 serviceStatus.dwCheckPoint = dwCheckPoint;
//设置服务在启动、关闭和运行操作中反映操作进度的值
serviceStatus.dwWaitHint = dwWaitHint;
//设置服务在执行启动、关闭和运行操作时将持续的时间值
 SetServiceStatus (serviceStatusHandle, &serviceStatus);
//更新 SCM 中服务的状态信息
}
```

3. ServiceThread 函数：用于定义体现服务功能的线程函数

```
DWORD ServiceThread(LPDWORD param)
{    //使用 Sleep 函数将 while 控制流程挂起,然后在指定的毫秒数后自动唤醒
        while (1)
        {
Beep(200,200);   //进行蜂鸣
Sleep(beepDelay); //将该线程挂起 beepDelay 中指定的毫秒数
        }
        return 0;
}
```

4. Handler 函数：用于分发从 SCM 获得的控制通知事件，对控制通知事件加以处理；

```
VOID Handler (DWORD controlCode)
{
        switch(controlCode)
        {    // 处理停止服务事件
        case SERVICE_CONTROL_STOP:
        //通知 SCM 服务即将停止
SendStatusToSCM(SERVICE_STOP_PENDING,
NO_ERROR, 0, 1, 5000);
    runningService =FALSE;    //设置服务运行的标志
    SetEvent(terminateEvent); //设置终止事件句柄为活动状态,从而使 ServiceMain 函数得以继续执行
    return;
  // 处理暂停服务事件
  case SERVICE_CONTROL_PAUSE:
  if (runningService && !pauseService)
                {
// 通知 SCM 服务即将暂停
SendStatusToSCM(SERVICE_PAUSE_PENDING, NO_ERROR, 0, 1, 1000);
  pauseService = TRUE;          //设置服务暂停的标志
  SuspendThread(threadHandle);  //挂起新服务的线程
SendStatusToSCM(SERVICE_PAUSED, NO_ERROR, 0, 0, 0);
//通知 SCM 服务暂停
        }
    break;
// 处理继续服务事件
    case SERVICE_CONTROL_CONTINUE:
        if (runningService && pauseService)
        {
        // 通知 SCM 服务即将继续
SendStatusToSCM(SERVICE_CONTINUE_PENDING, NO_ERROR, 0, 1, 1000);
  pauseService =FALSE;          //设置服务暂停的标志
  ResumeThread(threadHandle);   //继续新服务的线程
SendStatusToSCM(SERVICE_RUNNING, NO_ERROR, 0, 0, 0);   //通知 SCM 服务继续
        }
    break;
  // 处理更新服务当前状态事件
  case SERVICE_CONTROL_INTERROGATE:
    break;
  // 处理关闭系统事件
  case SERVICE_CONTROL_SHUTDOWN:
    return;
    }
}
```

5. ServiceMain 函数：

1 创建服务状态句柄 serviceStatusHandle 和服务控制函数 Handler；

2 向 SCM 发送服务启动状态消息，并创建服务所在的线程；

3 创建 terminateEvent 事件句柄，并结合 WaitForSingleObject 函数控制服务的停止；

```
VOID ServiceMain(DWORD argc, LPTSTR *argv)
{    //调用 RegisterServiceCtrlHandler 函数获得服务状态句柄,并注册生成服务控制处理函数
  serviceStatusHandle =RegisterServiceCtrlHandler
(SERVICE_NAME, (LPHANDLER_FUNCTION)Handler);
    //通知 SCM 服务即将启动,并设置启动操作的进度值为 1,以及估计完成的时间值为 5 秒
    SendStatusToSCM(SERVICE_START_PENDING,
NO_ERROR, 0, 1, 5000);
    // 创建 terminateEvent 事件句柄,以控制 ServiceMain 程序的执行
  terminateEvent = CreateEvent (0, TRUE, FALSE, 0);
    // 通知 SCM 服务即将启动,并设置启动操作的进度
```

值为 2, 以及估计完成的时间值为 1 秒
SendStatusToSCM(SERVICE_START_PENDING, NO_ERROR, 0, 2, 1000);
// 获取启动参数, 即蜂鸣的间隔时间值
if (argc == 2)
{
int temp = atoi(argv[1]);
//如果间隔值小于 1 秒, 则使用缺省的间隔值
if (temp < 1000) beepDelay = DEFAULT_BEEP_DELAY;
else beepDelay = temp;
}
// 创建服务线程
DWORD id;
//创建服务所在的线程, ServiceThread 是线程函数的起始地址, id 用于获得线程的 ID 号
threadHandle = CreateThread (0, 0, (LPTHREAD_START_ROUTINE) ServiceThread, 0, 0, & id);
runningService = TRUE; // 设置服务运行的标志
// 通知 SCM 服务开始运行
SendStatusToSCM(SERVICE_RUNNING, NO_ERROR, 0, 0, 0);
// 监视 terminateEvent 事件句柄的状态, 如果处于活动状态, 程序继续执行, 否则阻止程序继续执行
WaitForSingleObject (terminateEvent, INFINITE);
// 如果 terminateEvent 事件句柄处于活动状态, 则关闭相关句柄, 终止 ServiceMain 函数
CloseHandle(terminateEvent);
//关闭 terminateEvent 事件句柄
SendStatusToSCM(SERVICE_STOPPED, 0, 0, 0, 0); //显示出错信息 error, 并通知 SCM 服务停止
CloseHandle(threadHandle);
//关闭 threadHandle 线程句柄
}

6. main 函数：调用 StartServiceCtrlDispatcher 函数，并将指向 ServiceMain 函数的指针传递给 SCM；

```
VOID main(VOID)
{
//填充 SERVICE_TABLE_ENTRY 结构, 供 StartServiceCtrlDispatcher 函数调用
SERVICE_TABLE_ENTRY serviceTable[ ] ={{SERVICE_NAME, (LPSERVICE_MAIN_FUNCTION)
ServiceMain}, { NULL, NULL }};
BOOL success;
//通过将新服务的 ServiceMain 函数的指针传递给 SCM, 实现新服务在 SCM 中的注册
success = StartServiceCtrlDispatcher(serviceTable);
if (!success) ExitProcess(GetLastError());
//结束进程的执行
}
```

编译并生成 beep. exe，将其拷贝至 c h目录下，供安装程序调用。当安装完毕后，SCM 也将调用 beep. exe 来执行该服务。

3. 2 新服务的安装 install. cpp ：打开服务控制管理数据库，并创建新服务；

```
#include <windows. h>
#include <iostream. h>
void main(int argc, char * argv[])
{
SC_HANDLE scm; //定义服务控制管理数据库句柄
SC_HANDLE newService; //定义新服务句柄
if (argc ! = 4) return; //如果命令行参数小于 4 个, 则不执行安装程序
// 打开服务控制管理数据库, 并返回服务控制管理数据库的句柄
scm = OpenSCManager(0, 0, SC_MANAGER_CREATE_SERVICE);
if (!scm) ExitProcess(GetLastError()); //结束进程的执行
// 创建新服务, 并把它加入到服务控制管理数据库中
newService = CreateService(scm, argv[1], argv[2], SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS, SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL, argv[3], 0, 0, 0, 0, 0);
if (!newService) ExitProcess(GetLastError()); // 结束进程的执行
CloseServiceHandle(newService); // 关闭新服务句柄
CloseServiceHandle(scm); // 关闭服务控制管理数据库句柄
}
```

编译并生成 install. exe，将其拷贝至：c h目录中，然后在 Windows NT 中单击 “开始”，再单击 “运行”，使用 “运行”命令执行带 3 个参数的 installl. exe 程序。输入以下命令行进行安装：c h install. exe BeepService “Beeper” c h Beep. exe。其中 BeepService 是供 SCM 内部使用的服务名，Beeper 是显示在控制面板的服务程序中的名称，“c h Beep. exe”是服务所在的可执行文件的路径。安装完毕后，你可以在控制面板的服务程序中看到新增的服务 Beeper，点击右键选择 “属性”，你就可以设置服务的启动类型、服务的状态和启动参数。

3. 3 新服务的卸载 uninstall. cpp ：

打开服务控制管理数据库，查询服务是否处于停止状态，若不是则先停止服务，然后删除服务；

```
#include <windows. h>
#include <iostream. h>
void main(int argc, char * argv[])
{
SC_HANDLE service, scm; //定义服务句柄和服务控制管理数据库句柄
SERVICE_STATUS status; //定义服务状态结构
if (argc ! = 2) return; //如果命令行参数的个数小于 2, 则不执行卸载程序打开服务控制管理数据库, 并返回服务控制管理数据库的句柄
scm = OpenSCManager(0, 0, SC_MANAGER_CREATE_SERVICE);
// 获得服务句柄, 并添加服务访问中的删除选项
service = OpenService(scm, argv[1], SERVICE_ALL_ACCESS | DELETE);
// 获得服务的当前状态
QueryServiceStatus(service, & status);
// 如果服务不处于停止状态, 则将其状态设置为停止状态
if (status. dwCurrentState ! = SERVICE_STOPPED)
ControlService(service, SERVICE_CONTROL_STOP, & status);
```

WIN32 平台 IP 多播通信技术及其应用

蔡 倩

摘 要 本文介绍了多播通信的含义、特点及其意义，讨论了采用 Winsock2 接口实现 IP 多播的一般步骤，特别讲述了 C++Builder 环境下实现基于消息驱动的 Winsock I/O 模型的方法，最后给出了 IP 多播通信的完整实例。

关键词 Winsock2，TCP/IP，多播

随着 Internet 的发展，TCP/IP 协议在各类网络中得到了广泛的应用。在 Win32 平台上，多种编程语言（VB、VC、C++Builder、Delphi 等）都提供了实现 TCP/IP 协议的控件、组件或类，为广大编程人员带来了很大的方便。这些控件、组件或类都是基于 Winsock 编程接口的，并对其进行了封装。其实，Winsock 是一个用途很广泛的编程接口，它为各种不同的网络传输协议提供一致的、与协议无关的 API 接口，不仅用于 TCP/IP，也用于 SPX/IPX、NETBIOS、ATM、IrDA 等多种协议。当要实现高性能的网络应用或对网络进行控制时，我们不得不直接面对 Winsock，其中，基于 IP 的多播通信就是我们必须用 Winsock 实现的一种高级编程技术。

一、 多播通信

“多播”也称为“多点传送”（Multicasting），是一种让数据从一个成员送出，然后复制给其它多个成员的技术，它是与“广播”（Broadcasting）相对应的。在网络中，过度使用广播技术，极易造成网络带宽的大幅占用，影响整个网络的通讯效率。多播正好弥补了广播的不足，对一个网络内的各工作站而言，只有在上面运行的进程表示自己“有兴趣”，多播数据才会复制给它们，并且进程可以随时加入或撤出多播通讯过程。因此，多播可以有效地减轻网络通信的负担，避免资源的无谓浪费。

并非所有的协议都支持多播通信，对 Win32 平台而言，只有两种可以从 Winsock 内访问的协议（IP、ATM）才提供了对多播通信的支持。目前，Windows CE2.1、Windows95、Windows98、Windows NT4、Windows2000 都提供了对 IP 多播的支持，而只有 Windows98、Windows2000 才提供对 ATM 多播的支持。

1. 多播的特征

多播通信具有两个层面的重要特征：控制层面和数据层面。其中，“控制层面”（Control Plane）定义网络成员的组织方式；而“数据层面”（Data Plane）决定在不同的网络成员之间，数据如何传送。这两方面的特征既可以是“有根的”（Rooted），也可以是无根的（Nonrotted）。在“有根的”控制层面内，存在一个特殊的多播组成员，称为 c_root（根节点），其余的每个组成员叫做 c_leaf（叶节点）。对“无根的”控制层面而言，只存在叶节点。

“有根的”控制层面和数据层面的行为特征是：根节点负责多播组的建立，涉及同任意数量叶节点的连接。叶节点可以在某一时刻申请加入一个特定的多播组。数据传送只能在根节点和叶节点之间进行，根节点将数据一次性多播传送给每个叶节点，叶节点只能将数据传向根节点。ATM 多播具有上述特征。

“无根的”控制层面和数据层面的行为特征是：只存在叶节点，它们可以任意加入一个多播组。从一个叶节点发送的数据会蔓延到每一个叶节点（包括它自己），无论最开始是由谁发出的数据。IP 多播具有上述特征。

2. IP 多播

IP 多播通信需要依赖一个特殊的地址组，称为“多播地址”，该地址实现对一个指定组的命名。假如有五个节点想

```
DeleteService(service);                    // 删除服务
CloseServiceHandle(service);               // 关闭新服务句柄
CloseServiceHandle(scm); // 关闭服务控制管理数据库句柄
}
```

编译并生成 uninstall.exe，将其拷贝至：c:\目录中，然后在 Windows NT 中单击“开始”，再单击“运行”，使用“运行”命令执行带 1 个参数的 uninstall.exe 程序。输入以下命令行进行卸载：c:\uninstall.exe BeepService。其中 BeepService 就是在安装服务生成的供 SCM 内部使用的服务名。

参考文献

1. David Bennett 等著，徐军等译. Visual C++5 开发人员指南. 北京 机械工业出版社 1999

2. Beck Zaratian 著 詹津明、杨欣译. Microsoft Visual C++使用指南. 北京 清华大学出版社 1999

（收稿日期：2000 年 11 月 19 日）

通过 IP 多播实现彼此间的通信，它们可以加入同一个多播地址组，加入后，由一个节点发出的任何数据均会一模一样地复制一份，发给组内的每个成员，甚至包括发数据的那个节点。

多播 IP 地址是一个 D 类 IP 地址，范围在 224. 0. 0. 0 到 239. 255. 255. 255 之间，但其中有一些地址是为特殊用途而保留的，用户不要使用。一些保留的特殊地址如表 1 所示。

表 1 多播保留地址

特殊多播地址	用途
224. 0. 0. 0	基本地址
224. 0. 0. 1	这个子网上的所有系统
224. 0. 0. 2	这个子网上的所有路由器
224. 0. 1. 1	网络时间协议
224. 0. 0. 9	RIP 第 2 版本组地址
224. 0. 1. 24	WINS 服务器组地址

IP 多播一般需要网卡硬件的支持，通过网卡增加多播过滤器，从而提供最好的性能。目前，大多数网卡都提供了对 IP 多播的支持。Windows 2000、Windows NT4 可以在网卡硬件不支持的情况下提供驱动程序级的支持。

二、 Winsock 实现 IP 多播的一般步骤

IP 多播在 Winsock 1、和 Winsock2 中都可以实现，Winsock2 为 IP 多播提供了 Qos （常规服务质量）的能力。在此讲述 Winsock2 实现 IP 多播的一般步骤。Winsock2 在 Windows98、Windows2000、Windows NT 4 中得到支持。

1. WSASocket 创建套接字

WSASocket 的函数原型如下

```
SOCKET socket(
  int af,      //协议的地址家族, AF_INET 指定 IP 协议族
  int type,    //协议的套接字类型, 采用 SOCK_DGRAM 指定为数据报
  int protocol,   //设定为 0
  LPWSAPROTOCOL_INFO lpProtocolInfo, //一般设为 NULL
  GROUP g,    //设定为 0
  DWORD dwFlags //标志位
);
```

该套接字实际上是一个 UDP 协议的套接字，它是加入多播组的初始化套接字，并且以后数据的发送和接收都在该套接字上进行。这也说明 IP 多播是附加在 UDP 协议之上的，它具有 UDP 协议的一切特性。

针对 IP 多播通信，dwFlags 设置为 WSA_FLAG_MULTIPOINT_C_LEAF、 WSA_FLAG_MULTIPOINT_D_LEAF、 WSA_FLAG_OVERLAPPED 的位和，指明 IP 多播在控制层面和数据层面只具有叶节点的行为特征。

2. bind 绑定本地端口

bind 的函数原型如下：

```
int bind(
      SOCKET s,    //套接字接口
      sockaddr * FAR name, //网络定址
      int namelen //name 的数据长度
);
```

对 IP 网络的地址采用 sockaddr_in 数据结构，定义如下：

```
sockaddr_in{
short       sin_family;   //地址家族 AF_INET
u_short     sin_port;     //通信的端口号
  in_addr     sin_addr;     //IP 地址
  char        sin_zero[8];  //保留不用
};
```

完整的过程如下：

```
#define  MCASTPORT 0x4000
SOCKET Sock;
     Sockaddr_in Local;
    Sock = socket(AF_INET, SOCK_DGRAM, 0, NULL, 0,
WSA_FLAG_MULTIPOINT_C_LEAF|WSA_FLAG_MULTIPOINT_
D_LEAF|WSA_FLAG_OVERLAPPED);
     Local. sin_family = AF_INET;
     Local. sin_port = htons(MCASTPORT);
     Local. sin_addr. s_addr = htonl(INADDR_ANY);
     bind(Sock, (sockaddr * )& Local, sizeof(Local));
```

其中，MCASTPORT 为多播通讯端口，对多播通信来说，发送和接收数据采用同一个端口为好。INADDR_ANY 表明该套接口 Sock 可以接收任意一个 IP 主机发来的数据。

3. WSAJoinLeaf 加入一个多播组

WSAJoinLeaf 的函数原型如下：

```
SOCKET WSA JoinLeaf(
      SOCKET s,          //接受 socket 创建的套接字
      sockaddr FAR * name,    //多播组定址
      int  namelen,          //name 的长度
      LPWSABUF lpCallerData, //设为 NULL
      LPWSABUF lpCalleeData, //设为 NULL
      LPQOS    lpSQOS,    //指出应用程序要求的带宽
      LPQOS    lpGQOS,    //设定为 NULL
      DWORD  dwFlags //指定主机是发送数据、接收数据或收发兼并
);
```

dwFlag 的参数可选值分别为：

```
JL_SENDER_ONLY   //只发送数据
JL_RECEIVER_ONLY  //只接收数据
JL_BOTY          //收发兼并
```

函数调用成功会返回一个套接字，用 closesocket 关闭该套接字就离开了多播组，此时可以用 WSAJoinLeaf 再次加入多播组。注意，对多播组数据的接收和发送不能在该套接字上完成。

4. sendto 向多播组发送数据

sendto 的函数原型如下：

```
int sendto(
      SOCKET s,          //发送套接字
      char FAR * buf,       //发送数据区
      int len,           //发送数据的字节长度
      int flag,          //一般设为 0
      sockaddr FAR * to,     //指向接收方的 IP 定址
      int tolen          //to 定址的长度
```

);

向多播组发送数据只需要把 to 设定为多播组的 IP 定址就可以了。

5. recvfrom 从多播组接收数据

recvfrom 的函数原型如下：

```
int recvfrom(
    SOCKET s,          //接收套接字
    char FAR * buf,    //接收数据区
    int len,           //接收数据区的长度
    int flag,          //一般设为 0
    sockaddr FAR * from, //函数返回的数据发送方的 IP 定址
    int FAR * fromlen  //函数返回的 from 的长度
    );
```

多播通信中数据的发送与接收一般采用同一个端口，因此其发送套接字和接收套接字是一样的。

三、基于消息机制的 Winsock 套接口 I/O 模型

在 Win32 平台上，Winsock 要求对套接口的操作都是异步进行的，即套接口的操作不能阻塞，必须尽快地返回应用进程，这是 Windows 的多任务机制所要求的。总的来说，网络数据发送是程序进程的主动行为，一般不会造成阻塞；而数据接收（包括接收连接等）是典型的异步事件，我们无法确知数据到来的时机和数量，在没有数据到来的时候调用 recvfrom 就会阻塞其它程序的运行，这是不允许的。

为了满足异步 I/O 处理的要求，Winsock 提供了一系列的 I/O 管理模型供用户选择，这是 Winsock 编程的关键，不同的 I/O 模型决定了网络程序的性能。I/O 模型包括传统的 select 模型，基于消息的 WSAAsyncSelect 模型，基于事件的 WSAEventSelect 模型，重叠 I/O 模型和完成端口模型。

select 模型依靠软件轮询，效率较低，在 Windows 平台上一般不采用。重叠模型较复杂，但可以使应用程序达到更佳的系统性能，为高性能的网络程序开发所采用。完成端口模型最复杂，但可以达到最佳的系统性能，目前只在 Windows NT，Windows2000 中实现，用于网络服务器的开发。

WSAAsyncSelect 模型利用了 Windows 的消息驱动机制，应用程序可以在一个套接字上，接收以 Windows 消息为基础的网络事件通知，是最方便的一种模型。目前各类编程语言提供的网络控件、组件或类都采用了这一模型。WSAEventSelet 模型与 WSAAsyncSelet 相仿，不过它需要依靠线程来实现。

WSAAsyncSelect 模型对所有的 Winsock 应用程序的设计都是一样的，其实现方法如下所述。

1. 消息注册

WSAAsyncSelect 模型只能应用于有窗口和窗口例程的应用程序，它首先必须向应用程序窗口注册网络消息，该网络消息是一个用户自定义的消息。WSAAsyncSelect 的函数原型为：

```
int WSAAsyncSelect(
  SOCKET s,  //指定为希望产生网络事件的套接字
  HWND hWnd, //对应于网络事件发生后接收消息通知的窗口的句柄
  unsigned int wMsg,  //用户定义的网络消息名
  long lEvent //是一个位掩码，对应于一系列网络事件的组合
);
```

lEvent 的常用网络事件类型如表 2 所示。

表 2 常用网络事件类型

事件类型	含义
FD_READ	套接口可读的通知
FD_WRITE	套接口可写的通知
FD_ACCEPT	进入连接状态的通知
FD_CONNET	连接完成时的通知
FD_CLOSE	套接口关闭的通知

举例如下：

```
#define WM_SOCK_MSG  WM_USER +1000
WSAAsyncSelect(s, hwnd, WM_SOCK_MSG, FD_READ | FD_CLOSE);
```

这样一来一旦套接口上有数据可读或关闭套接口就会发消息 WM_SOCK_MSG 通知应用程序所在的窗口 hwnd。

2. 消息的响应

传统的 Windows 应用程序在窗口例程 WindowPro 中处理与窗口有关的各类消息，该例程在创建窗口类时向 Windows 系统注册。VC、C++Builder 等一般向用户隐藏程序的窗口例程，但都提供了映射用户消息的方法，通过用户消息的映射使程序结构更合理。在此只说明 C++Builder 消息映射的接口与方法。

映射 WM_SOCK_MSG 消息的 C++Builder 代码举例如下

```
void __fastcall OnSocketMsg(TMessage& Msg);
BEGIN_MESSAGE_MAP
  VCL_MESSAGE_HANDLER(WM_SOCK_MSG, TMessage, OnSocketMsg);
END_MESSAGE_MAP(TForm);
```

VCL_MESSAGE_HANDLER 向窗体类 TForm 注册 WM_SOCK_MSG 消息，并说明随着该消息一起发送的参数是 TMessage 类型，最后说明 WM_SOCK_MSG 消息的响应函数是 OnSocketMsg，从而完成了用户消息的映射。

我们在消息响应函数 OnSocketMsg 中完成对网络事件的处理，举例如下：

```
void __fastcall TMainForm::OnSocketMsg(TMessage& Msg)
{
    AnsiString RecvMsg;
    switch(WSAGETSELECTEVENT(Msg.LParam))
    {
      case FD_READ:
        int Length = recvfrom(Sock, RecvBuf, BUFSIZE, 0,
(sockaddr *)&From, &dwFrom);
        RecvBuf[Length] = '\0';
        RecvMsg.sprintf("RECV: '%s' from <%s>\n",
RecvBuf, inet_ntoa(From.sin_addr));
        ShowMessage(RecvMsg);
        break;
```

```
    }
}
```

Tmessage 结构中的 WParam 参数返回的是产生网络消息的套接字，LParam 参数的高字节包含了可能出现的错误代码，LParam 参数的低字节包含了网络消息触发的网络事件。

四、 应用实例

通过上述过程，我们完成了一个实用的 IP 多播应用程序的开发，它基于 Windows 的消息驱动机制，实现了异步化的网络 I/O 操作，可为各类普通 Winsock 应用程序的开发所采用。下面给出一个完整的 IP 多播的实例程序，IP 多播有一个有趣的特性，自己发出的数据自己可以收到，该特性只有在 Windows 2000 下可以关闭。通过该例子可以掌握 IP 多播应用程序的一般设计方法。

程序中通过一个命令按钮发送多播数据，一个编辑框显示接收的数据。这里有三个 IP 定址 Local、Remote 和 From。Local 代表本地地址用于数据接收；Remote 代表 IP 多播组地址用于数据发送；From 为接收函数返回的地址表示接收数据的来源地址。由于直接通过 Winsock 编程，因此打开和关闭 Winsock 动态库的工作需要由程序自己完成。

该程序用 C++Builder 5.0 开发，开发时要包含 <winsock2.h>的头文件，并在工程中加入 ws2_32.lib 的静态库，该静态库在在 C++Builder 的 hlib 目录下。程序在 Windows98、Windows200 下调试运行通过。

五、 结束语

IP 多播技术的应用十分广泛。对一个较大型的网络系统而言，往往需要共享一些公共网络信息，但可能每个站点对这些信息的内容和速度等要求是不一样的，这时采用多播技术是最合理的，我们可以把信息规划为不同要求的多播组，不同站点的用户加入符合自己要求的多播组，从而使网络资源得到合理的运用。多播技术同 QOS 一起也为网络视频会议等类似的应用提供了合理的网络带宽的保证。

程序清单

```
//头文件 - - - - - - - - - - - - - - - - - - - - - - - -
#ifndef UMCastH
#define UMCastH
//- - - - - - - - - - - - - - - - - - - - - - - - - - - -
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <winsock2.h>
#define MCASTADDR "234.5.6.7" //多播组地址
#define MCASTPORT 0x3000 //通信端口号
#define BUFSIZE 1024 //接收、发送缓冲区大小
#define WM_SOCK_MSG WM_USER+1000
//- - - - - - - - - - - - - - - - - - - - - - - - - - - -
class TMainForm : public TForm
{
__published: // IDE-managed Components
    TButton *Button1;
    TEdit *Edit1;
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall FormDestroy(TObject *Sender);
    void __fastcall Button1Click(TObject *Sender);
private:    // User declarations
    sockaddr_in Local;      //指向本地 IP 地址与端口
    sockaddr_in Remote;     //指向多播组 IP 地址与端口
    sockaddr_in From;       //指向数据来源的 IP 地址
    SOCKET Sock;            //UDP 套接口
    SOCKET SockM;           //多播套接口
    int dwFrom;
    int SendCount;
    char RecvBuf[BUFSIZE];
    char SendBuf[BUFSIZE];
    void __fastcall InitWinsock2(void);
    int __fastcall InitMSocket(void);
    void __fastcall OnSocketMsg(TMessage& Msg);
public:     // User declarations
    __fastcall TMainForm(TComponent* Owner);
    BEGIN_MESSAGE_MAP
      VCL_MESSAGE_HANDLER(WM_SOCK_MSG, TMessage, OnSocketMsg);
    END_MESSAGE_MAP(TForm);
};
//- - - - - - - - - - - - - - - - - - - - - - - - - - - -
extern PACKAGE TMainForm *MainForm;
//- - - - - - - - - - - - - - - - - - - - - - - - - - - -
#endif
//源文件 - - - - - - - - - - - - - - - - - - - -
#include <vcl.h>
#include <stdio.h>
#pragma hdrstop
#include "UMCast.h"
//- - - - - - - - - - - - - - - - - - - - - - - - - - - -
#pragma package(smart_init)
#pragma resource "*.dfm"
TMainForm *MainForm;
//- - - - - - - - - - - - - - - - - - - - - - - - - - - -
__fastcall TMainForm::TMainForm(TComponent* Owner)
    : TForm(Owner)
{
    SendCount=0;
    dwFrom=sizeof(From);
}
//- - - - - - - - - - - - - - - - - - - - - - - - - - - -
void __fastcall TMainForm::InitWinsock2(void)
{
    WSADATA WSADate;
    WSAStartup(MAKEWORD(2,2),&WSADate); //初始化 Winsock 动态连接库
}
//- - - - - - - - - - - - - - - - - - - - - - - - - - - -
void __fastcall TMainForm::FormCreate(TObject *Sender)
```

```
{
    InitWinsock2();
    InitMSocket();
}
//---------------------------
void __fastcall TMainForm::FormDestroy(TObject *Sender)
{
    closesocket(SockM);        //关闭套接口
    closesocket(Sock);         //关闭套接口
    WSACleanup();              //关闭 Winsock 动态连接库
}
//---------------------------
int __fastcall TMainForm::InitMSocket(void)
{
    AnsiString ErrStr;
        //创建套接口
 if((Sock=WSASocket(AF_INET, SOCK_DGRAM, 0, NULL, 0,
WSA_FLAG_MULTIPOINT_C_LEAF|WSA_FLAG_MULTIPOINT_
D_LEAF|WSA_FLAG_OVERLAPPED))==INVALID_SOCKET)
    {
     ErrStr.sprintf("socket failed with: %d\n", WSAGet-
LastError());
        ShowMessage(ErrStr);
        WSACleanup();
        return -1;
    }
        //邦定套接口
    Local.sin_family=AF_INET;
    Local.sin_port=htons(MCASTPORT);
    Local.sin_addr.s_addr=htonl(INADDR_ANY);
    if(bind(Sock, (sockaddr*)&Local, sizeof(Local))==
SOCKET_ERROR)
    {
        ErrStr.sprintf("bind failed with: %d\n", WSAGet-
LastError());
        ShowMessage(ErrStr);
        closesocket(Sock);
        WSACleanup();
        return -1;
    }
        //加入多播组
    Remote.sin_family=AF_INET;
    Remote.sin_port=htons(MCASTPORT);
    Remote.sin_addr.s_addr=inet_addr(MCASTADDR);
     if((SockM=WSAJoinLeaf(Sock, (sockaddr*)&Re-
mote, sizeof(Remote), NULL, NULL, NULL, NULL,
JL_BOTH))==INVALID_SOCKET)
    {
        ErrStr.sprintf("WSAJoinLeaf failed with: %d\n",
WSAGetLastError());
        ShowMessage(ErrStr);
        closesocket(Sock);
        WSACleanup();
        return -1;
    }
        //注册网络消息及其网络事件
        WSAAsyncSelect(Sock, Handle, WM_SOCK_MSG,
FD_READ);
    return 0;
}
//---------------------------
void __fastcall TMainForm::OnSocketMsg(TMessage& Msg)
{
    AnsiString ErrStr;
    AnsiString RecvMsg;
    int Length;
      //判断是否出现网络故障
    if(WSAGETSELECTERROR(Msg.LParam))
    {
        ErrStr.sprintf("socket failed with: %d\n",
WSAGetLastError());
        ShowMessage(ErrStr);
        closesocket(Sock);
        closesocket(SockM);
        WSACleanup();
        return;
    }
      //检索网络事件
    switch(WSAGETSELECTEVENT(Msg.LParam))
    {
        case FD_READ:
            Length=recvfrom(Sock, RecvBuf, BUFSIZE,
0, (sockaddr*)&From, &dwFrom);
            if(Length==SOCKET_ERROR)
            {
     ErrStr.sprintf("socket failed with: %d\n", WSAGet-
LastError());
              ShowMessage(ErrStr);
              closesocket(Sock);
              closesocket(SockM);
              WSACleanup();
              break;
            }
            RecvBuf[Length]='\0';
     RecvMsg.sprintf("RECV: '%s' from <%s>\n",
RecvBuf, inet_ntoa(From.sin_addr));
            //ShowMessage(RecvMsg);
            Edit1->Text=RecvMsg;
            break;
    }
}
//---------------------------
void __fastcall TMainForm::Button1Click(TObject *Sender)
{
    SendCount++;
sprintf(SendBuf, "This is multicasting test%d", SendCount);
     //向多播组发送数据
    sendto(Sock, (char*)SendBuf, strlen(SendBuf), 0,
(sockaddr*)&Remote, sizeof(Remote));
}
```

参考文献

Anthony Jones、Jim Ohlund. Windows 网络编程技术. 机械工业出版社，2000.3

收稿日期：2000 年 11 月 9 日）

应用 SOCKET 实现网络通信

任继平 王泗宏

摘 要 本文介绍了 SOCKET 编程的基本概念，详细说明了 SOCKET 编程的一些注意事项，对实际应用有较强的指导作用。

主题词 Windows，SOCKET

一、引言

1. SOCKET 编程基本概念

Windows NT 提供的一个最重要的通信程序设计机制——Windows Sockets WinSock 使我们在网络通信编程上有很大的发挥空间。

一个套接字（Socket）是一个通信端点。典型的通信发生于一个客户和一个服务器之间，就有两个端点，一个在客户端，一个在服务器端。对应的就有两个套接字，且这两个套接字在客户和服务器之间建立了双向数据传送的连接。

套接字基本上分为两类：流套接字、数据报套接字。流套接字（Stream Sockets）用于大流量数据的双向传输，数据流可分为记录流或字节流，这取决于协议。流通常用于无重复（UnDuplicated）的和顺序 Sequenced，保持包发送顺序 的传送和接收数据。流套接字保证数据发送。数据报套接字（Datagram Sockets）主要用于广播功能。数据报套接字支持双向数据流，不保证可靠、有序、无重复性，是面向无连接的传输机制。

你的套接字应用程序可以使用一个端口（Port）与其它套接字应用程序通信。端口的含义可以这样理解：它的作用是可以实现在具有一个 IP 地址的单台机器上可以同时有效地运行多个客户或演示软件，各个到达的 TCP 包或 UDP 包都被指定给某一特定的端口，例如，你可以在一个窗口中执行 FTP，同时在另一个窗口运行自己的套接字应用程序或其它通信程序，确保不同通信程序的数据不被混淆在一起的机制就是端口。公用通信功能使用保留端口，用户可指定未被保留且未被使用的端口，或传递 0 作为端口值由 Sockets 自动分配端口。

每个套接字还有一个套接字地址，通常是你的应用程序运行所在计算机的 IP 地址。

Socket 实际上代表了 IP 地址和端口号的组合，变成了通信中一种抽象化的终端节点。

套接字通信通常有三个阶段。

（1）执行安装功能。创建并绑定一个套接字，定位并与远程计算机建立一个套接字连接。

（2）发送和接收数据。若正在编写一个服务类型的套接字应用程序，则可创建一个套接字并监听从客户来的套接字连接输入。若有多个用户想同步的建立连接，则可请求积压连接请求。

（3）执行清除功能。断开和关闭套接字连接。

与其它的 Windows 程序设计领域一样，可用 API 或 MFC 库对 Windows Sockets 编程。WinSock 使用 TCP/IP 协议，但 TCP/IP 并不是 WinSock 支持的唯一协议，它还支持 Novell 的 IPX/SPX、Digital 的 DECNet 和除 TCP/IP 之外的其它协议。本文主要以 MFC 下 Windows Sockets 编程为例，介绍一般使用方法及注意事项。

2. 编程基本流程

MFC 下提供两种 SOCKET 类型供用户使用：SOCK_STREAM、SOCK_DGRAM，相应有两种类 CasyncSocket、Csocket 可供使用。

Csocket 封装的 socket 使用 TCP 协议，提供有序的、可靠的、双向的、连接的、无重复并且无记录边界的比特流传输机制。CasyncSocket 封装的 socket 使用 UDP 协议，支持双向数据流，但并不保证是可靠、有序、无重复的，为面向无连接的数据报传输机制。

MFC 下客户和服务器利用面向连接的 Csocket 进行通信的过程、具体编程细节参考 MFC 下 Chatter、Chatsrvr 程序示例。一般来说，客户方在 OnReceive 中处理数据，服务器方在监听 Socket 的 OnAccept 中处理客户连接，并在其中建立数据处理 Socket，数据处理 Socket 在 OnReceive 中处理数据。

客户和服务器利用 CasyncSocket 进行通信的过程，UDP 协议下的通信较简单，不需要事先建立连接，而是通过数据定向发送、接收实现服务器/客户通信。

二、编程点滴

1. SOCKET 创建

（1）一般过程

以 MFC 下异步 SOCKET 创建为例：

①首先声明 SOCKET 对象：

```
CAsyncSocket socket
```

②创建 SOCKET：

```
socket. Create( UINT nSocketPort = 0, int nSocketType = SOCK_DGRAM, long lEvent = FD_READ | FD_WRITE | FD_OOB | FD_ACCEPT | FD_CONNECT | FD_CLOSE, LPCTSTR lpszSocketAddress = NULL );
```

其中，nSocketPort 是此 socket 绑定的端口号，区分不同的应用程序；nSocketType 是 socket 类型，此处应用 UDP 协议，选用数据报类型；lEvent 指明此 socket 需要反应的消息，可重载相应的消息映射函数做出动作。

下面的讨论主要针对 nSocketPort、lpszSocketAddress 两个参数的指定，为叙述简便，标识 nSocketPort 为端口，标识 lpszSocketAddress 为 IP 地址。记住，使用该类进行 socket 创建后，不要再调用 bind（）函数，因为 MFC 的 CasyncSocket 类已经将 socket 的 create 和 bind 封装在一起，形成 Create。

（2）注意事项

①在同一计算机、同一应用程序中创建两个 CAsyncSocket：socket1、socket2

●socket1、socket2 不能指定同一端口、同一地址；

●socket1、socket2 指定相同端口、但地址必须不同；

●socket1、socket2 指定相同地址、但端口必须不同；

●若 socket1 调用 Create 创建，指明端口，但未指明 IP 地址

●socket1 的 IP 地址不能再改变，除非它是已连接的或 I/O 正在产生，否则得不到地址

●socket2 不能指定该端口、指定本机任一 IP 地址创建

●socket2 可以更换端口、指定本机任一 IP 地址创建

●socket2 可以更换端口、不指定 IP 地址创建

②创建一个 CAsyncSocket：socket

●不要试图尝试以任何方式再次创建同一个 socket

任何方式指端口、IP 地址的组合方式，即

●不能以相同端口、不同 IP 地址再次创建该 socket

●不能以相同端口、相同 IP 地址再次创建该 socket

●不能以不同端口、相同 IP 地址再次创建该 socket

●不能以不同端口、不同 IP 地址再次创建该 socket

2. 组广播 SOCKET 的建立

组广播通信可以实现一台或多台机器向网络中的多台机器发送数据信息，且这种方式对发送端来说编程简洁，通过向一个单一组地址发送来实现一点对多点通信。

（1）组广播通信原理

TCP/IP 协议地址分配中的 D 类地址即为多目地址（multicast address），范围是 224. 0. 0. 0 ~ 239. 255. 255. 255，组广播通信双方认知一个 D 类地址中的组广播地址，发送数据方向目的端口、该地址发送数据，接收数据方将本机 IP 地址加入该组地址中，并在该端口上建立 SOCKET 等待数据接收，数据以广播方式发送到各台联网计算机，由协议判断该由哪台计算机接收数据，判断的依据就是本机 IP 地址与组广播地址的关系。

（2）组广播实现

由于 VC + + 5. 0 的联机帮助文件上对组广播 SOCKET 编程信息不系统，初学者很难应用，本文特别对组广播 SOCKET 的建立给出示例。

组广播 SOCKET 的建立涉及到 SOCKET 属性的设置，即函数

```
int setsockopt（SOCKET s, int level, int optname, const char FAR * optval, int optlen）
```

设置了 SOCKET 的属性，创建组 SOCKET 时，level 设置为 IPPROTO_IP，optname 设置为 IP_ADD_MEMBERSHIP，optval 类型为 struct ip_mreq，组地址和本机地址就在该结构中指定，下面具体给出实例，函数和结构的具体信息可查 VC 帮助。

```
//接收方加入组地址实例
CAsyncSocket Socket; //声明异步 SOCKET
Socket. Create(5000, SOCK_DGRAM); //以端口 5000, 缺省地址创建 SOCKET
struct ip_mreq bindGroup; //声明结构
bindGroup. imr_multiaddr. s_addr = inet_addr( ″232. 20. 1. 1″);
  //填充组地址
bindGroup. imr_interface. s_addr = inet_addr( ″10. 3. 17. 13″);
  //填充本机地址
  //设置 SOCKET 属性
int status = setsockopt(Socket - >m_hSocket, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char * ) & bindGroup, sizeof( struct ip_mreq ) );
  if ( status = = SOCKET_ERROR ) //出错处理
  {
      int err = GetLastError();
      CString mes;
 mes. Format( ″Add Group Failed, Error Code: %d. ″, err );
      MessageBox( mes );
  }
```

3 注意事项

若本机有多个网卡，加入组时可以指明将某一个 IP 地址加入组中，发送方创建的 SOCKET 要绑定可以和该地址通信的本机 IP 地址；或者发送方缺省本机地址创建 SOCKET，接收方将本机全部 IP 地址加入组中（不能确认数据在哪条通路上传输），但必须保证一个网卡只配备了一个 IP 地址，否则可能重复接收。

下面给出在 Windows NT 下通过操作注册表获得本机网卡类型、IP 地址程序。

```
/ * 得到网卡注册表项 * /
/ * 打开注册表键 HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Services\\Tcpip\\Linkage hKey 为打开的键 * /
LONG status;
HKEY hKey;
status = RegOpenKey(HKEY_LOCAL_MACHINE, ″System\\CurrentControlSet\\Services\\Tcpip\\Linkage″, & hKey );
DWORD type, leng = 200;
UCHAR data0[200]; data0[0] = 0;
```

```
/* 从打开的键读取键值，放入 data0[200]中 */
status = RegQueryValueEx( hKey, "Bind", NULL, & type, data0, & leng);
/* 关闭键 */
RegCloseKey( hKey );
CString strNetcardName;
DWORD tmpLeng =0;
while ( tmpLeng +1 < leng )
{
strNetcardName = & (data0[tmpLeng]);
tmpLeng + = strNetcardName. GetLength() + 1;
int n = strNetcardName. ReverseFind( '\\' );
if ( n ! = -1 )
strNetcardName = strNetcardName. Mid( n + 1 );
else
break;
/* 得到 IP 地址和子网掩码 */
LONG status;
HKEY hKey;
status = RegOpenKey( HKEY_LOCAL_MACHINE, "System\\CurrentControlSet\\ Services\\" +strNetcardName +"\\ Parameters\\Tcpip", & hKey );
DWORD type, length = 200;
UCHAR data[200]; data[0] = 0;
status = RegQueryValueEx( hKey,
                          "IPAddress", //得到 IP 地址
                          NULL, & type, data, & length);
UCHAR data2[200]; data2[0] = 0;
DWORD length2 = 200;
status = RegQueryValueEx(hKey,
                          "SubnetMask", //得到子网掩码
                          NULL, & type, data2, & length2);
/* 关闭键 */
RegCloseKey( hKey );
/* 可能一个网卡有多个地址，需一一处理. */
DWORD tmpLength =0, tmpLength2 =0;
CString strAddr = data;
CString strMask = data2;
while ( tmpLength +1 < length ) // 还有数据
{
strAddr = & (data[tmpLength]); // 保留处理前数据
tmpLength + = strAddr. GetLength() + 1;
/* 处理子网屏蔽 */
if ( tmpLength2 +1 < length2 )
{
strMask = & (data2[tmpLength2]);
  tmpLength2 + = strMask. GetLength() + 1;
}
}
}
```

三、小结

本文介绍了 SOCKET 编程的基本概念，详细说明了 SOCKET 编程的一些注意事项，列举出 SOCKET 编程过程中的几点经验之谈，本文提供的程序在 Windows NT4. 0，Visual C + + 6. 0 下调试通过。

参考文献

1. [美] Douglas E. Comer . Internet Working with TCP / IP. 电子工业出版社，1998. 06

2. 蒋东兴等 . Windows Sockets 网络程序设计大全 . 清华大学出版社，1999. 11

3. [美] Kate Gregory. Visual C + + 5 开发使用手册 . 机械工业出版社，1998. 09

收稿日期：2000 年 10 月 26 日）

用 VC + +对 BMP 格式图像文件编程的技巧与实践

罗　瑜　袁志伟

摘　要　本文以实际中用 VC + +对 BMP 格式图像文件处理时，使用几何算法　旋转，缩放等）造成图像数据区数据排列错误，导致处理后的图像发生畸变的问题进行了总结，并给出了解决的方法。因为现在的图像算法大多是基于 256 色灰度图的，所以我们这里仅讨论 256 色灰度图，对于别的颜色的位图　如 24 位真彩色位图），也很容易类推得到。

关键词　Visual　C + +6.0，位图，BMP 格式文件，宽度，256 色位图，DibLook

一、前言

数字图像处理是一门新兴学科，在 Windows 环境中最重要的图像就是位图 Bitmap ，即位映象 bit map 。它在 1991 年与 Windows3.1 版本同时发表。

二、位图格式

BMP 位图文件的结构如图所示，包括位图文件头结构 BITMAPFILEHEADER、位图信息头结构 BITMAPINFOHEADER、位图颜色表 RGBQUAD 和位图像素数据四部分。此列出与宽度问题相关的部分。

位图文件头结构 BITMAPFILEHEADER
位图信息头结构 BITMAPINFOHEADER
位图颜色表 RGBQUAD
位图像素数据

图 1　BMP 位图文件的结构

与宽度相关的成员变量在位图信息头结构 BITMAPINFOHEADER 中，其原型为：

```
typedef struct tagBITMAPINFOHEADER {
    DWORD biSize          说明 BITMAPINFOHEADER 结构所需的字节数
    LONG biWidth          说明位图的宽度，以像素为单位
    LONG biHeight         说明位图的高度，以像素为单位
    WORD biplanes         说明目标设备的位平面数，必须为 1
    WORD biBitCount       说明每一像素的位数,必须为 1,4,8,24
    DWORD biCompression   说明一个压缩位图的压缩类型，为 BI_RGB BI_RLE8 BI_RLE4
    DWORD biSizeImage     说明图像的大小，以字节为单位，如果位图是 BI_RGB 格式，那么该成员置为零才有效
    LONG biXPelsPerMeter  说明位图的目标设备的水平分辨率
    LONG biYPelsPerMeter  说明位图的目标设备的垂直分辨率
    DWORD biClrUsed       说明实际使用的颜色表中的颜色变址数
    DWORD biClrImportant  重要颜色的索引数
}BITMAPINFOHEADER
```

三、问题的提出

BMP 存储像素数据的顺序是先行后列，由下而上。例如：一幅 4 × 15　高 × 宽）的白色　在 256 色灰度图中白色的像素值为 255）图像，其数据区如下：

```
00444230  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00
00444240  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00
00444250  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00
00444260  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00
```

可以看出，在数据区中每一行的末尾出现了与白色像素无关的 0，并且，内存中的宽度不是 15 而是 16。经过研究，发现造成这种现象的原因在于 BMP 图像的压缩方式与压缩结果遵循规则　表 1）：

表 1　压缩方式与压缩效果遵循规则

biBitCount	BiCompression	压缩方式	压缩结果遵循规则 每行的字节数符合
1　4　8　24	0	BI_RGB	long boundry
8	1	BI_RLE8	word boundry
4	2	BI_RLE4	word boundry

位图图像数据区每行压缩数据必须符合 long boundary 或 word boundary 的规定，long 代表 4 个字节，long boundary 代表 4 字节的倍数；同理 word boundary 代表 2 字节的倍数，当每行压缩结果不满 4 或 2 的倍数时，必须以 0 padding bytes 补足。由此可见，宽度为 15 的图像每行都要补上一个字节的零。这就说明了为什么宽度为 15 的图像其数据区每行的宽度是 16。我们再看一个现象，把一幅 32 × 32 的 256 色位图　图 1）放大到 32 × 39，如果不考虑宽度问题，将会得出图 2 的错误结

图 1 原图 宽度为 32

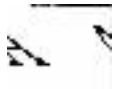


图 2 宽度变为 39

果。

如果能够在处理的时候能够使图像的每行宽度自动调节到 4 字节的倍数，这个问题就迎刃而解。下面就以图像缩放图 1 为例详细的解释怎样解决这个问题。

四、解决问题

经过研究，发现 BITMAPINFOHEADERZ 结构中 biWidth 成员的值是图像的实际宽度值，那么我们就可以编程将放大后的宽度 39 变为 40，即把 32 × 32 的图像放大为 32 × 40 的图像来处理，最后将目标图像的 biWidth 值设置为 39，这个问题就解决了。下面我们就进行实际的编程。程序框架是 Microsoft 公司的 DibLook，该程序是 Microsoft 公司对于 BMP 位图文件的演示程序，它包含了位图的打开、存储、显示等基本操作，DibLook 位于 Microsoft Visual Studio6. 0 光盘中的 MSDN 第一张盘的 hSAMPLES hVC98 hMFC hGENERAL 目录中，现把缩放的程序写成 API 函数以供方便调用。

//在 dibapi. h 文件中加入以下代码:

```
HDIB WINAPI ImageStretch(HDIB hDIB , int nWidthNew, int nHeightNew);
```

//在 dibapi. cpp 文件中加入以下代码:

```
#include "math. h"
HDIB WINAPI ImageStretch(HDIB hDIB, int nWidthNew, int nHeightNew)
{
  LPBITMAPINFOHEADER pOldBih, pNewBih;
  LPSTR lpOldbi = (LPSTR): : GlobalLock(hDIB);
  LPSTR pOldBits = (LPSTR): : FindDIBBits(lpOldbi);
  pOldBih = (LPBITMAPINFOHEADER)lpOldbi;
  int nWidthOld = (int): : DIBWidth(lpOldbi); //原图像的宽度
 int nHeightOld = (int): : DIBHeight(lpOldbi); //原图像的高度
  int nWidth = nWidthNew; //缩放后的图像宽度
  nWidth = (int)ceil((double)nWidthNew/4. 0) * 4; //将缩放后的图像宽度转换为 4 字节的倍数
  DWORD dwNewMemSize; //新图像的大小
dwNewMemSize = (DWORD) 256 * sizeof(RGBQUAD) + sizeof(BIT MAPINFOHEADER) + nWidth * nHeightNew;
HDIB hDIBNew;  //目标图像的句柄
hDIBNew = (HDIB): : GlobalAlloc(GMEM_MOVEABLE|GMEM_ZEROINIT, dwNewMemSize); //为目标图像分配内存
 LPSTR lpNewbi = (LPSTR): : GlobalLock((HDIB)hDIBNew);
  pNewBih = (LPBITMAPINFOHEADER)lpNewbi;
  pOldBih = (LPBITMAPINFOHEADER)lpOldbi;
  memcpy(lpNewbi, lpOldbi, sizeof(BITMAPINFOHEADER) + 256 * sizeof(RGBQUAD));
  pNewBih - >biSizeImage = nWidth * nHeightNew;
  pNewBih - >biWidth = nWidthNew;
  pNewBih - >biHeight = nHeightNew;
  LPSTR pNewBits = (LPSTR): : FindDIBBits(lpNewbi);
  int nWidth2 = (int)ceil((double)nWidthOld/4. 0) * 4; //将原图像宽度转换为 4 字节的倍数
  double X, Y;
  int Xold, Yold, Xnew, Ynew;
  double ScaleV = (double)nHeightNew/(double)nHeightOld;
  double ScaleH = (double)nWidthNew/(double)nWidthOld;
  for(Ynew = 0; Ynew < nHeightNew; Ynew + + )
  {
    Y = Ynew/ScaleV;
    Yold = (unsigned)Y;
    for(Xnew = 0; Xnew < nWidthNew; Xnew + + )
    {
      X = Xnew/ScaleH;
      Xold = (unsigned)X;
      * (pNewBits + Ynew * nWidth + Xnew) = * (pOldBits + Yold * nWidth2 + Xold);
    }
  }
  : : GlobalUnlock(hDIB);
  : : GlobalUnlock(hDIBNew);
  : : GlobalFree(hDIB);
  return hDIBNew;
}
```

然后在需要缩放的地方加入以下代码 例如要放大到 32 × 39）：

```
m_hDIB = : : ImageStretch(m_hDIB, 39, 32);
UpdateAllViews(NULL);
```

结果如下 程序在 Windows 98 SE，Visual C + + 6. 0 环境下通过 ：

图 3 结果

五、进一步的思考

令人感到疑惑的是，为什么诸如 ACDSee 之类的图像软件在显示和处理宽度非 4 字节倍数的图像时不会发生错误，经过对 DibLook 程序的研究，发现 DibLook 在显示图像时采用的是 API 函数 PaintDIB ，由此受到启发，发现 Microsoft 将显示宽度为非 4 字节倍数图像问题的解决方法隐藏在 PaintDIB 函数中。这也可以解释为什么 ACDSee 软件能够正确处理 比如缩放，旋转等）和显示非 4 字节倍数的图像了。

六、结束语

采用本文介绍的方法，可以避免宽度非 4 字节倍数的图像在进行图像缩放、旋转时出现空间扭曲的问题。本文是针对 256 色灰度图来讲述的，很容易就能拓展到 24 位真彩色或是 BI_RLE4 等格式图像。

收稿日期：2000 年 11 月 10 日）

Visual C + +6.0 下 JPEG 图像的显示

亓　波

摘　要　本文论述了 JPEG 图像格式的结构以及压缩/解压缩的过程，并通过具体的实例实现了图像解压缩以及图像的显示。

关键词　JPEG，DCT 变换，数据编码

一、引言

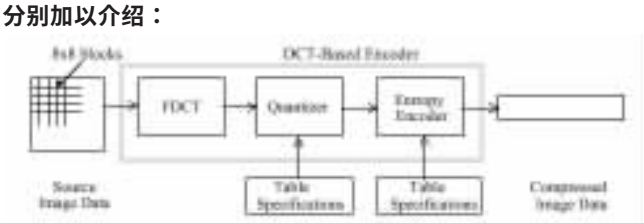
JPEG 是 Joint Photographic Experts Group 联合摄影专家小组 的首字母缩写。该小组是 ISO 下属的一个组织，它由很多国家和地区的标准图像组织联合组成。JPEG 的主要作用是用于数字化图像的标准编码技术，通过该标准编码技术，可以很好地对图形图像进行处理。JPEG 图像具有很高的压缩比，因此广泛应用于多媒体和网络程序中。

二、JPEG 文件结构

JPEG 文件大体上可以分为一下两个部分：标记码（tag）、压缩数据。标记码部分给出了 JPEG 图像的所有信息，如图像的高、宽、Huffman 表、量化等。标记码很多，但绝大多数的 JPEG 文件只包含几种。因此读 JPEG 文件的关键在于压缩数据部分，为此我们首先来看一下 JPEG 图像的压缩过程。

三、JPEG 图像的压缩过程

JPEG 图像文件是一种像素格式文件格式，它比诸如像 GIF、BMP 等图像文件要复杂的多，无论是图像文件格式的内容，还是文件格式的编码方式，它都是十分复杂的。JPEG 有几种模式，其中最常用的是基于离散余弦变换（DCT）的顺序型模式。数码相机也大都采用这种模式，以下就针对这种格式进行讨论。JPEG 基本压缩法一般分为颜色转换和采样、DCT 变换、数据量化、数据编码四个过程（如下图）。下面分别加以介绍：



DCT - Based　Encoder　Processing　Steps

（一）颜色转换和采样

因为 JPEG 压缩法只支持 YCbCr 颜色格式的数据结构，而不支持 RGB 颜色模式的图像数据结构，所以在将全色彩图像进行数据压缩之前，必须对颜色模式进行数据转换，将 RGB 模式转换成 YcbCr 模式，以适应 JPEG 压缩法的需要。它们之间的具体转换公式为：

Y = 0.299R + 0.587G + 0.114B

Cb = - 0.169R - 0.3313G + 0.5B

Cr = 0.5R - 0.4187G - 0.0813B

其中 Y 代表亮度基色，Cb 和 Cr 代表颜色基色，Cb 是代表像素的蓝色度，而 Cr 是代表像素的红色度。

（二）DCT 变换

在进行变换之前，必须先将图像分为 8×8 矩阵。这样就可以把数据代入下面的 DCT 公式中，进行 DCT 变换。由于离散余弦变换公式所能接受的值的范围为 - 128 到 127 之间，所以在变换之前，必须先将图像数据值减去 128，以达到公式所能接受的值的范围。具体变换公式如下表示：

$$F(u,v)=\frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7}f(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y-1)v\pi}{16}$$

上式中　x　y　代表在图像数据矩阵中的某个数据值的坐标位置。

f x y　是指图像数据矩阵中在该坐标点的数据值。

u，v　是指经过 DCT 变换后在矩阵的某个数值点的坐标位置。

F u v　是指经过 DCT 变换后的在该坐标点的数据值。

当 u = 0，v = 0 时：C u C v = $1/\sqrt{2}$

当 u>0，v>0 时：C u C v = 1

经过变换后的数据值 F u v 称为频率系数。当 u v = 0 时，F u v 的值为最大，我们称之为直流 DC 系数，而其它的 63 个频率系数，我们称之为交流 AC 系数。

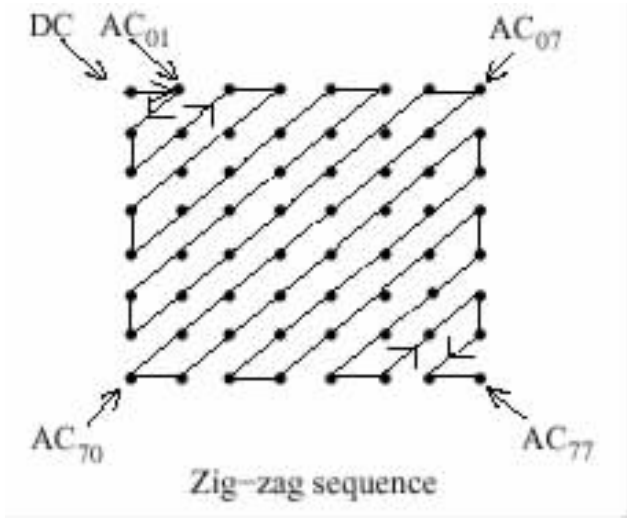
（三）数据量化

由于频率系数是一些浮点数，而编码表只是针对整数而言，所以在进行编码之前，需将矩阵中的浮点数转换成整数，这一过程即为数据量化过程。在对频率系数进行量化时，需用到两个 88 的矩阵。这两个矩阵中，一个是用来处理亮度频率系数的，另一个是用来处理色度频率系数的。进行数据量化的过程就是将频率系数中的频率系数值除以对应量

化矩阵中对应的值，然后取与得到的结果最接近的整数。这样所得结果与原始数据值之间就有了差异，这一差异就是造成图像在压缩后失真的主要原因。

（四） 数据编码

由于两个相邻的 88 子块的 DC 系数相差很小，所以对它们采用差分编码 DPCM，也就是说对相邻子块 DC 系数的差值进行编码。88 子块的其它 63 个 AC 系数，采用行程编码。为了保证低频分量先出现，高频分量后出现，以增加行程中连续“0”的个数，这 63 个元素采用了“之”字型（zig - zag）的排列方法，如下所示：



四、JPEG 解码的程序实现

JPEG 的整个解压缩过程是其压缩过程的逆过程，但比其压缩过程还要复杂，好在利用 IJG V. 6a code 可以较容易的实现 JPEG 的解压缩过程。IJG V. 6a code 是专门操作 JPEG 的代码。

（一）利用 IJG code 生成操作 JPEG 的 LIB

1. 首先在 Visual C + + 6. 0 下 NEW 一个 WIN32 STATIC LIBRARY，并取名为 JPEGLIB2。

2. 将 IJG 的源代码（CPP . H 文件加入到该项目中），IJG 的源代码可以从网上下载。

3. 编译生成 JPEGLIB2. lib 文件。

（二）创建操作 JPEG 的类

虽然利用 JPEGLIB2. lib 就可以实现 JPEG 的解压缩了，但利用烦琐的子程序来真正实现仍是一件头疼的事情，如果有一个类专门来操作 JPEG 的压缩就好了。Chris Losinger 便创建了一个 JPEGFILE 类来操作 JPEG 图像，下面是经作者改进后的代码：

```
//  JpegFile 类的头文件 JpegFile. h
#define WIDTHBYTES(bits)  (((bits) + 31) / 32 * 4)
class JpegFile
{
public:
// read a JPEG file to an RGB buffer  – 3 bytes per pixel
// returns a ptr to a buffer .
// caller is responsible for cleanup!!!
// BYTE * buf = JpegFile: : JpegFileToRGB( . . . . );
// delete [ ] buf;
static BYTE * JpegFileToRGB(CString fileName, //path to image
                  UINT * width, // image width in pixels
                  UINT * height);    // image height
  // allocates a DWORD – aligned buffer, copies data buffer
  // caller is responsible for delete [ ]´ing the buffer
static BYTE * MakeDwordAlignedBuf(
                  BYTE * dataBuf,    //input buf
                  UINT widthPix,     // input pixels
                  UINT height,       // lines
                UINT * uiOutWidthBytes); //new width bytes
  // vertically flip a buffer – for BMPs
  // in – place
  // note, this routine works on a buffer of width widthBytes: not a buffer of widthPixels.
  static BOOL VertFlipBuf(BYTE * inbuf,   // input buf
            UINT widthBytes,        // input width bytes
            UINT height);           // height
  // swap Red and Blue bytes
  // in – place
static BOOL BGRFromRGB(BYTE * buf,    // input buf
                  UINT widthPix,   // width in pixels
                  UINT height);    // lines
  JpegFile();          // creates an empty object
  ~JpegFile();     // destroys nothing
};
```

//JpegFile 类的实现文件 JpegFile. cpp 内容较多，具体详见源程序

（三） 程序实现

（1）在 VC6. 0 下 NEW 一个 MFC APPWINZARD（EXE），取名为 Mfcapp。

（2）将 JpegFile. h 和 JpegFile. cpp 以及 JPEGLIB2. lib 添加到程序中。

（3） 在 CmfcappView 类中添加菜单 OPEN 项所对应的函数为 LoadJPG CString fileName 。

```
void CMfcappView: : LoadJPG(CString fileName)
{
 // m_buf is the global buffer
if (m_buf! = NULL) {
     delete [ ] m_buf;
     m_buf = NULL;
}
 // read to buffer tmp
m_buf = JpegFile: :  JpegFileToRGB(fileName,  &  m_width, & m_height);
JpegFile: : BGRFromRGB(m_buf,  m_width,  m_height);
JpegFile: : VertFlipBuf(m_buf,  m_width * 3,  m_height);
}
```

4）图像的显示部分由 CMfcappView DrawBMP 实现。

```
void CMfcappView::DrawBMP()
{
    // if we don't have an image, get out of here
    if (m_buf==NULL) return;
    CDC *theDC = GetDC();
    if (theDC!=NULL) {
        CRect clientRect;
        GetClientRect(clientRect);
        BYTE *tmp;
        mm_widthDW=0;
        // DWORD-align for display
  tmp=JpegFile::MakeDwordAlignedBuf(m_buf, m_width,
m_height, &mm_widthDW);
        // set up a DIB
        BITMAPINFOHEADER bmiHeader;
    bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
        bmiHeader.biWidth = m_width;
        bmiHeader.biHeight = m_height;
        bmiHeader.biPlanes = 1;
        bmiHeader.biBitCount = 24;
        bmiHeader.biCompression = BI_RGB;
        bmiHeader.biSizeImage = 0;
        bmiHeader.biXPelsPerMeter = 0;
        bmiHeader.biYPelsPerMeter = 0;
        bmiHeader.biClrUsed = 0;
        bmiHeader.biClrImportant = 0;
        // now blast it to the CDC passed in.
    // lines returns the number of lines actually displayed
 int lines = StretchDIBits(theDC->m_hDC, 0, 0, bmiHeader.biWidth, bmiHeader.biHeight, 0, 0, bmiHeader.biWidth, bmiHeader.biHeight, tmp, (LPBITMAPINFO)&bmiHeader, DIB_RGB_COLORS, SRCCOPY);
      delete tmp;
        ReleaseDC(theDC);
    }
}
```

6）编译链接，以上程序在 VC6.0，Win98 下调试通过。运行时界面如下：

收稿日期：2000 年 11 月 13 日）

国际化时代的多语种字处理软件——文杰

2001 年 2 月 15 日 韩软集团的中国独立法人公司——韩软 北京 信息技术有限公司隆重推出其多语种字处理软件——文杰。

文杰将中，英，日，韩多语种资源内置其中使您不仅摆脱操作系统的束缚 而且还可利用多样化的输入法进行自由的输入与编辑。文杰界面亲切友好 提供了丰富的图形模板库 强大的表格制作与图表功能 有了文杰，您可方便快捷的制作精美的文档。文杰的推出是韩软集团在海外事业企划中 非常重要的一步。考虑到海外市场地域不同 用户群不同，对产品功能进行了相应的补充和变更。如在文杰中 针对中国用户追加了中英 英中词典 五笔字型输入法，北大方正字体等。

——为进一步扩大市场，韩软将与 8848，天极网，连邦，万众合力，晶合时代，东方 IT 等众多公司合作 进行网上和渠道销售。

——上市之际，韩软还将为用户提供各种优惠。如向用户提供韩软子公司与中国公司合资经营的网吧——光明信息港的免费上网优惠券及文杰纪念品等。活动期间，韩软将举行各种活动，如您在使用过程中，把对文杰的意见，建议，以及对文杰的指正告之韩软，将会得到精美的礼品。另外 韩软还将举行精美文档制作竞赛，如果您使用文杰果然不同凡响 可将制作的文档通过电子邮件发送给韩软 您不仅可以得到精美的礼物 韩软还将把您制作的文档 放入文杰的文档模板库中。说不定，有一天，文杰程序中会出现您的大作。

——2001 年 2 月 13 日 韩软集团与韩国 SM Entertainment 公司合作。在不久的将来，凡购买文杰的用户可得到韩国 HOT，神话等歌手组合的音像制品优惠券，购买 HOT，神话音像制品的用户也可得到文杰产品的优惠券等。

——另外，如果您是 SMEntertainment 公司签约歌手歌迷会的会员 也可得到光明信息港提供的上网优惠券。

如何用 VC + + 实现 ACDSee 风格的双界面

李安东

提起 ACDSee32 可能大家都不陌生，其最大的特点就是界面友好、使用方便，并因此而倍受广大计算机用户的青睐。ACDSee 在浏览和显示图像时分别采用不同的用户界面（browser 和 viewer），是它的特点之一。但打开其文件夹却发现只有一个可执行程序，表明这两种界面是由同一个程序所实现的。本文准备探讨如何用 VC + + 6.0 实现这种 ACDSee 风格的双界面程序。

一、建立工程

首先我们在 VC + + 6.0 中，建立一个名为 Test 的单文档界面（SDI）工程，在建立过程中，选择工程类型为“MFC AppWizard［exe］”，风格为“Windows 资源管理器”风格。

随后即会自动产生两个视类 CTestView 和 CLeftView，它们分别派生自 CListView 和 CTreeView，组成浏览窗口中的两个分隔窗口。相当于建立了 ACDSee 中的 browser 窗口。

二、新建一个视类和一个主框架类

接下来，我们再新建一个视类，类名为 CMyView，派生自 CView。这样就手工建立一个 viewer 窗口。

还必须专门为它建立一个主框架类，可按如下步骤进行：

1. 新建一个单文档界面的工程 Test1，其风格为“MFC 标准”；

2. 将此工程中的 MainFrm.h 和 MainFrm.cpp 分别改名为 ViewMainFrm.h 和 ViewMainFrm.cpp，并复制到 Test 工程目录中；

3. 将这两个文件中的所有 CMainFrame 类名替换为 CviewMainFrame；

4. 将这两个文件 Import 加到 Test 工程中。

三、在 InitInstance 函数中添加代码

下一步的任务就是让程序在不同的情况下显示不同的窗口，按如下步骤进行：

1. 打开文件“Test.cpp”，并将插入点移动到 InitInstance 函数的开头，并插入如下代码：

```
//注册自己的消息
CLOSEMYSELF = RegisterWindowMessage( " CLOSEMYSELF");
if(!CLOSEMYSELF)return FALSE;
//通知其他实例关闭
: : PostMessage(HWND_BROADCAST, CLOSEMYSELF, 0, 0);
```

利用 RegisterWindowMessage 函数注册自定义消息，达到不同实例间相互通讯的目的。

2. 在 InitInstance 中删去如下行：

```
pDocTemplate = new CSingleDocTemplate(IDR_MAINFRAME,
RUNTIME_CLASS(CTestDoc), RUNTIME_CLASS(CMainFrame),
RUNTIME_CLASS(CLeftView));
AddDocTemplate(pDocTemplate);
```

3. 在 ParseCommandLine cmdInfo 行之后插入如下行：

```
//根据不同命令行参数,显示不同界面
if(cmdInfo.m_nShellCommand = = CCommandLineInfo: :
FileOpen)
{
//如果命令行参数中含有要打开的文件名,则使用 viewer 界面
    pDocTemplate = new CSingleDocTemplate(IDR_MAIN-
FRAME, RUNTIME_CLASS(CTestDoc), RUNTIME_CLASS
(CViewMainFrame), RUNTIME_CLASS(CMyView));
    AddDocTemplate(pDocTemplate);
}
else
{//否则使用 browser 界面
    pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CTestDoc),
RUNTIME_CLASS(CMainFrame), //custom MDI child frame
    RUNTIME_CLASS(CLeftView));
    AddDocTemplate(pDocTemplate);
}
```

其中 CLOSEMYSELF 是个全局变量，因此在 InitInstance 的前面添加：

```
UINT CLOSEMYSELF
```

CcommandLineInfo 是 MFC 中的一个类，其作用就是对命令行参数进行分析和处理，本例就是利用它对命令行参数进行判断。需要深入了解的朋友，可查阅 Microsoft 公司的 MSDN 帮助文档。

四、处理自定义消息

1. 在 CMainFrame 内建立窗口函数 WindowProc，以便响应 CLOSEMYSELF 消息：

```
extern UINT CLOSEMYSELF;
LRESULT CMainFrame: : WindowProc(UINT message,
WPARAM wParam, LPARAM lParam)
{
  // TODO: Add your specialized code here and/or call the
base class
  if(message = = CLOSEMYSELF)
      PostQuitMessage(1);
  return CFrameWnd: : WindowProc(message, wParam,
lParam);
}
```

接到自定义消息后，用 PostQuitMessage 1 退出消息循环，关闭本实例。

2. 同样在 CViewMainFrame 内建立窗口函数 WindowProc，

以便响应 CLOSEMYSELF 消息：

```
extern UINT CLOSEMYSELF;
LRESULT CViewMainFrame::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
    // TODO: Add your specialized code here and/or call the base class
    if(message==CLOSEMYSELF)
        PostQuitMessage(1);
    return CFrameWnd::WindowProc(message, wParam, lParam);
}
```

五、启动新的实例

1. 为了让用户从 browser 中切换到 viewer 窗口，在你认为合适的地方，比如在 CLeftView 类的 OnFileOpen 内插入如下代码：

```
void CLeftView::OnFileOpen()
{
    CFileDialog dlg(TRUE);
    char filter[]={
    'A', 'l', 'l', ' ', 'F', 'i', 'l', 'e', 's', ':', '\t', '*', '.', '*', 0, '*', '.', '*', 0, 0
        };
    CHAR name[256]={0};
    dlg.m_ofn.lpstrFilter=filter;
    dlg.m_ofn.lpstrFile=name;
    dlg.m_ofn.nMaxFile=256;
    dlg.m_ofn.lStructSize=sizeof(dlg.m_ofn);
    if(dlg.DoModal()==IDOK)
    {
        char path[201];
        GetModuleFileName(//取模块路径
        NULL, // handle to module to find filename for
        path, // pointer to buffer to receive module path
        200 // size of buffer, in characters
        );
        CString s=CString(path)+" \""+name+"\"";
                //命令行参数含有要打开的文件名
        STARTUPINFO info;
        PROCESS_INFORMATION newinfo;
        GetStartupInfo(&info);
        //启动新实例,打开 viewer 窗口
        CreateProcess(NULL, (LPTSTR)(LPCTSTR)s, NULL, NULL, FALSE, CREATE_DEFAULT_ERROR_MODE, NULL, NULL, &info, &newinfo);
    }
}
```

利用 CreateProcess 函数启动新的实例，这里须注意将命令行参数中的文件名放到双引号内（如本例中的 h），以免遇到中间含有空格的长文件名。

2. 为了让用户从 viewer 中切换到 browser 窗口，在你认为合适的地方，比如在 CMyView 类的 OnFileNew 内插入如下代码：

```
void CMyView::OnFileNew()
{
  char pt[201];
  GetModuleFileName(//取模块路径
        NULL, // handle to module to find filename for
        pt, // pointer to buffer to receive module path
        200 // size of buffer, in characters
        );
  STARTUPINFO info;
  PROCESS_INFORMATION newinfo;
  GetStartupInfo(&info);
  //启动新实例,切换到 browser 窗口
  CreateProcess(NULL, pt, NULL, NULL, FALSE, CREATE_DEFAULT_ERROR_MODE, NULL, NULL, &info, &newinfo);
}
```

完成上述工作后，编译并运行该程序，即会发现它具有了浏览和察看两个界面。

五、结束语

本文初步探讨了用VC++实现 ACDSee 风格双界面的方法，即让同一个程序的两个实例分别实现不同的界面。其关键是根据启动实例时，是否有要被打开的文件名作为命令行参数，若有则选择 viewer 界面，否则选择 browser 界面。以上代码均由 VC++6.0 编译，并在 Windows 98 环境下调试通过。

本文仅介绍了实现该种风格的一个途径，但不一定是最好的，比如将两种界面在同一个实例之内轮流实现，可能速度会更快一些。但本文介绍的方法比较简单和容易实现，是一种比较实用的方法。

收稿日期：2000 年 11 月 14 日）

中国万网推出不中断服务的新技术

最近，国内最大的网站服务提供商中国万网推出并应用了一种基于四层交换技术的负载均衡技术方案。该新技术方案的推出，不仅保证了其网站服务的高可靠性、可用性和可维护性，而且还提高了旗下的网站的访问速度，实现了其网站访问服务的不中断。

随着 Internet 网络和应用的发展，用户对网站系统性能的要求越来越高，网站系统必须有一套智能的解决方案来保证其服务器的高可靠性，保证每个访问用户都以最快的速度读取其想要的信息。此时，负载均衡机制就发挥了其无可替代的重要作用。负载均衡是指采用一组服务器来共同提供一项或多项应用服务。

中国万网采用服务器负载均衡技术将满足客户提高服务器性能的需求，极大地降低减低服务器链路的故障率，提供强有力的容错性能。为广大企业客户提供了一种成本效益型的解决方案。它的推出，保证了其旗下众多的网站群的不中断服务，极大提高了其接受用户的访问速度，有效地维护了广大用户的利益。

用 C + + builder 做一个系统锁

张云潮

一、设计目标

1. 该程序能够禁止非法用户的输入，包括鼠标操作，系统热键动作。

2. 能够设置密码。

3. 能够隐藏为托盘图标，以便随时响应用户呼叫。

4. 自启动运行。

二、关键问题的解决

1. 锁住系统热键，这里系统热键包括：Alt + Tab，WIN，Ctrl + Alt + Delete Ctrl + Esc。要锁住这些系统热键一般有两种方法：第一种方法是写一个键盘钩子，接管系统热键。第二方法是让系统认为当前处于屏保状态。这里我们讨论第二种方法。让系统认为当前处于屏保状态的方法是：

SystemParametersInfo SPI_SETSCREENSAVERRUNNING true & pOld SPIF_UPDATEINIFILE

SPI_SETSCREENSAVERRUNNING 该参数在微软的文档中并不建议使用，是因为在该状态下，系统无法弹出任务列表，一旦某一进程被挂起，只能重新起动，有可能造成数据丢失，但可以达到所需效果，我们仍然使用它。当程要取消该状态：

SystemParametersInfo SPI_SETSCREENSAVERRUNNING false & pOld SPIF_UPDATEINIFILE

pOld 为我们定义的布尔变量。

2. 控制鼠标动作。当然可以用鼠标钩子鼠标动作，但我们这里把鼠标限制在一个区域里以达到控制鼠标的目的。

```
TRect rt;
GetWindowRect(Handle, & rt);
ClipCursor(& rt);
```

ClipCursor 把鼠标动作限制在一个矩形区域里，如果当前鼠标位置不在这个区域里，系统将会自动调整鼠标位置到该区域里。

3. 注册表操作。把密码写到注册表里，让程序在系统启动时自动运行都要用到注册表操作。在这里我们用的是 C + + builder 的 TRegistry 类。TRegistry 是我见到的对注册表操作封装最好的类库。下面的代码是让程序在系统启动时自动运行。

```
char * tempchar = new char[255];
GetModuleFileName(HInstance, tempchar, 255);
TRegistry * myreg = new TRegistry;
myreg ->RootKey = HKEY_LOCAL_MACHINE;
myreg ->OpenKey("SOFTWARE\\MICROSOFT\\WINDOWS\\CURRENTVERSION\\RUN", true);
myreg ->WriteString("syslock", StrPas(tempchar));
myreg ->CloseKey();
delete tempchar;
delete myreg;
```

4. 窗体隐藏时，系统状态栏上标题条的隐去。普通的窗体隐藏或最小化时都会在系统状态栏上形成一个标题条。如何隐去它呢？有一类窗口不会形成系统状态栏上的标题条，这类窗体带有 WS_EX_TOOLWINDOW 的风格。下面代码让窗体加上了 WS_EX_TOOLWINDOW 风格。

```
DWORD dwExStyle = GetWindowLong(Application ->Handle, GWL_EXSTYLE);
dwExStyle | = WS_EX_TOOLWINDOW;
SetWindowLong(Application ->Handle, GWL_EXSTYLE, dwExStyle);
```

三、主窗体设计

按照图 1 设计主窗体 “MainForm”。

图 1

主窗体的属性设置为：BorderStyle：bsNone

Name MainForm

编辑框的属性设置为：PasswordChar *

Name PassEdit

图 2

双击 PopupMenu1 按图 2 所示增加菜单项。三项菜单项的 ImageIndex 依次为 0，1，2。三项菜单项的 Name 依次为 LockSys PassItem Exit

双击 ImageList1 增加三个图标。

BitBtn 按钮的属性设置为：Kind bkOK

Name OkBtn

四、密码设置窗体设计

按照图 3 设计密码设置窗体。

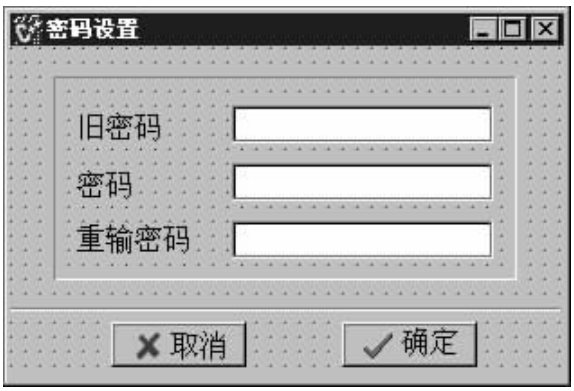


图 3

窗体的属性设置为： BorderStyle bsToolWindow

Name PassSet

Caption 密码设置

三个编辑框属性设置为：

Name 依次为：MaskEdit1 MaskEdit2 MaskEdit3.

PasswordChar *

BitBtn 按钮 “取消”属性设置为 Name BitBtn1

Caption 取消

Kind bkCancel

BitBtn 按钮 “确定”属性设置为 Name BitBtn2

Caption 确定

Kind bkOK

五、代码设计

主工程文件 Locksys. cpp 的代码为：

```
//---------------------------
#include <vcl.h>
#pragma hdrstop
USERES("locksys.res");
USEFORM("main.cpp", MainForm);
USEFORM("Pass.cpp", PassSet);
//---------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    DWORD dwExStyle = GetWindowLong(Application->
Handle, GWL_EXSTYLE);
    dwExStyle |= WS_EX_TOOLWINDOW;
    SetWindowLong(Application->Handle, GWL_EXSTYLE,
dwExStyle);
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TMainForm),
& MainForm);
Application->CreateForm(__classid(TPassSet), & PassSet);
        Application->MainForm->Visible = false;
        Application->Run();
    }
    catch (Exception & exception)
    {
        Application->ShowException(& exception);
    }
    return 0;
}
//---------------------------
```

主窗体 MainForm 的代码 Main. cpp 为：

```
//---------------------------
#ifndef mainH
#define mainH
//---------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ImgList.hpp>
#include <Menus.hpp>
#define MYWM_NOTIFY (WM_APP + 100)
//---------------------------
class TMainForm : public TForm
{
__published: // IDE-managed Components
    TLabel * Label1;
    TEdit * PassEdit;
    TBitBtn * OkBtn;
    TPopupMenu * PopupMenu1;
    TMenuItem * LockSys;
    TMenuItem * PassItem;
    TMenuItem * Exit;
    TImageList * ImageList1;
    void __fastcall OkBtnClick(TObject * Sender);
    void __fastcall FormCreate(TObject * Sender);
    void __fastcall LockSysClick(TObject * Sender);
    void __fastcall FormActivate(TObject * Sender);
    void __fastcall ExitClick(TObject * Sender);
    void __fastcall PassItemClick(TObject * Sender);
    void __fastcall FormCloseQuery(TObject * Sender, bool
& CanClose);
private:    // User declarations
    void __fastcall MyNotify(TMessage& Msg);
bool __fastcall TrayMessage(DWORD dwMessage, AnsiString
pszTip);
    bool pOld;
    bool FirstRun;
    bool bCanClose;
```

```
public:                 // User declarations
        __fastcall TMainForm(TComponent* Owner);
        __fastcall ~TMainForm();
      BEGIN_MESSAGE_MAP
MESSAGE_HANDLER(MYWM_NOTIFY, TMessage, MyNotify)
      END_MESSAGE_MAP(TForm)
};
//---------------------------
extern PACKAGE TMainForm *MainForm;
//---------------------------
#endif
```

主窗体 MainForm 的代码 Main.cpp 为：

```
//---------------------------
#include <vcl.h>
#pragma hdrstop
#include <shellapi.h>
#include "main.h"
#include "Pass.h"
//---------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMainForm *MainForm;
//---------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
        : TForm(Owner)
{
    FirstRun=true;
    bCanClose=true;
}
//---------------------------
__fastcall TMainForm:: ~TMainForm()
{
}
void __fastcall TMainForm::OkBtnClick(TObject *Sender)
{
       if(PassEdit->Text==PassSet->NewPass)
       {;
    SystemParametersInfo(SPI_SETSCREENSAVERRUNNING,
false, &pOld, SPIF_UPDATEINIFILE)
              ReleaseCapture();
              ::ShowCursor(true);
              ClipCursor(NULL);
              ShowWindow(Handle, SW_HIDE);
              bCanClose=true;
       }
       else
       {
             bCanClose=false;
             ShowMessage("无效的密码,请重新输入!");
       }
}
//---------------------------
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    TrayMessage(NIM_ADD,"系统锁");
}
//---------------------------
bool __fastcall TMainForm::TrayMessage(DWORD dwMes-
sage, AnsiString pszTip)
{
  NOTIFYICONDATA tnd;
  memset(&tnd,0,sizeof(tnd));
  tnd.cbSize          =sizeof(NOTIFYICONDATA);
  tnd.hWnd            =Handle;
  tnd.uID             =0;
  tnd.uFlags          =NIF_MESSAGE|NIF_ICON|NIF_TIP;
  tnd.hIcon           =Application->Icon->Handle;
  lstrcpyn(tnd.szTip, pszTip.c_str(), sizeof(tnd.szTip));
  tnd.uCallbackMessage      =MYWM_NOTIFY;
  if(dwMessage==NIM_DELETE)
   {
      tnd.hIcon=NULL;
      tnd.szTip[0]='\0';
   }
  return(Shell_NotifyIcon(dwMessage,&tnd));
}
void __fastcall TMainForm::MyNotify(TMessage& Msg)
{
   POINT MousePos;
   switch(Msg.LParam)
   {
      case WM_RBUTTONDOWN:
         if(GetCursorPos(&MousePos))
         {
            PopupMenu1->PopupComponent=MainForm;
            SetForegroundWindow(Handle);
        PopupMenu1->Popup(MousePos.x, MousePos.y);
         }
         break;
      case WM_LBUTTONDBLCLK:
         TrayMessage(NIM_DELETE,NULL);
         Close();
         break;
      default:
         break;
   }
   TForm::Dispatch(&Msg);
}
void __fastcall TMainForm::LockSysClick(TObject *Sender)
{
       TRect rt;
       ShowWindow(Handle,SW_SHOWNORMAL);
       ::SetForegroundWindow(Handle);
  GetWindowRect(Handle,&rt);
  ClipCursor(&rt);
//      ::SetCursor(LoadCursor(NULL,IDC_NO));
  SystemParametersInfo(SPI_SETSCREENSAVERRUNNING,
true, &pOld, SPIF_UPDATEINIFILE);
}
//---------------------------
void __fastcall TMainForm::FormActivate(TObject *Sender)
{
      if(FirstRun)
```

```
    {
     ShowWindow(Handle, SW_HIDE);
     FirstRun = false;
     }
    if(PassSet->FirstUse)
     if(PassSet->ShowModal() == mrOk)
       PassSet->FirstUse = false;
}
//---------------------------
void __fastcall TMainForm::ExitClick(TObject *Sender)
{
    TrayMessage(NIM_DELETE, "");
    Close();
}
//---------------------------
void __fastcall TMainForm::PassItemClick(TObject *Sender)
{
    PassSet->ShowModal();
}
//---------------------------
void __fastcall TMainForm::FormCloseQuery(TObject *
Sender, bool & CanClose)
{
  CanClose = bCanClose;
}
//---------------------------
```

密码设置窗体代码 Pass.h

```
//---------------------------
#ifndef PassH
#define PassH
//---------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
#include <Mask.hpp>
#include <registry.hpp>
//---------------------------
class TPassSet : public TForm
{
__published:    // IDE-managed Components
    TMaskEdit *MaskEdit1;
    TMaskEdit *MaskEdit2;
    TMaskEdit *MaskEdit3;
    TLabel *Label1;
    TLabel *Label2;
    TLabel *Label3;
    TBevel *Bevel1;
    TBevel *Bevel2;
    TBitBtn *BitBtn1;
    TBitBtn *OkBtn;
    void __fastcall OkBtnClick(TObject *Sender);
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall FormCloseQuery(TObject *Sender, bool
& CanClose);
private:    // User declarations
    TRegistry *reg;
    bool canclose;
public:    // User declarations
    bool FirstUse;
    AnsiString OldPass;
    AnsiString NewPass;
    __fastcall TPassSet(TComponent* Owner);
    __fastcall ~TPassSet();
};
//---------------------------
extern PACKAGE TPassSet *PassSet;
//---------------------------
#endif
```

密码设置窗体代码 Pass.cpp

```
//---------------------------
#include <vcl.h>
#pragma hdrstop
#include "Pass.h"
//---------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TPassSet *PassSet;
//---------------------------
__fastcall TPassSet::TPassSet(TComponent* Owner)
    : TForm(Owner)
{
    reg = new TRegistry();
    FirstUse = false;
    canclose = true;
}
//---------------------------
__fastcall TPassSet::~TPassSet()
{
    delete reg;
}
//---------------------------
void __fastcall TPassSet::OkBtnClick(TObject *Sender)
{
    AnsiString str1, str2, str3;
    str1 = MaskEdit1->Text;
    str2 = MaskEdit2->Text;
    str3 = MaskEdit3->Text;
    canclose = true;
    if(!(str1 == OldPass))
    {
      ShowMessage("旧密码不正确");
      canclose = false;
    }
    if(str2.AnsiCompare(str3))
    {
      ShowMessage("密码输入不正确");
```

```
        canclose = false;
      }
      if(str2. IsEmpty())
      {
        ShowMessage("密码不能为空");
        canclose = false;
      }
      if(canclose)
      {
        NewPass = str2;
        OldPass = NewPass;
        reg ->OpenKey("My System Locker", false);
        reg ->WriteString("PassWord", str3);
        reg ->CloseKey();
      }
}
//---------------------------
void __fastcall TPassSet::FormCreate(TObject *Sender)
{
      reg ->RootKey = HKEY_CURRENT_USER;
      if(!(reg ->KeyExists("My System Locker")))
      {
        FirstUse = true;
        reg ->OpenKey("My System Locker", true);
        OldPass = "";
        reg ->WriteString("PassWord", OldPass);
      }
      else
      {
          reg ->OpenKey("My System Locker", false);
          OldPass = reg ->ReadString("PassWord");
      }
      NewPass = OldPass;
      reg ->CloseKey();
}
//---------------------------
void __fastcall TPassSet::FormCloseQuery(TObject *Sender,
bool & CanClose)
{
      CanClose = canclose;
}
//---------------------------
```

收稿日期 2000 年 11 月 6 日

杀毒软件的生死时速

众所周知，反病毒软件厂商能否在第一时间对突发的新病毒作出有效反应，是检验厂商技术与产品能力的硬性指标。在今天，网络传播已赋予了计算机病毒无与比拟的能量，生出翅膀的病毒能够在几小时甚至几分钟内席卷到世界上每一个角落。据统计，目前有 87% 以上的病毒来源于电子邮件，通过互联网进行大面积的感染，这已成为全球反病毒各厂商面临的最大挑战。

在年前国家公安部组织进行的病毒防治产品统一标准评测中，以瑞星为代表的一批国内著名防病毒软件在产品性能、病毒检测与查杀能力及本地化服务方面优于众多国外同业厂商而名列前茅。业内专家指出，能否彻底查杀流行于国内外、且破坏性较强的病毒，以及能否以最快速度对新病毒做出反应，并提出解决方案，才是真正体现厂商实力的重要标准。对于北京瑞星公司，拥有目前世界一流的反病毒技术，以及对中国用户需求的深刻认识和强大的市场运作能力，最终的成绩终于证明：他们的产品是最适用于用户的。究其根本，当国内的反病毒产业仍处于个人作坊时代，瑞星已经是一个具有 150 多人的公司，其中反病毒研发队伍 50 多人、技术服务队伍 30 多人，并建立了国际化的企业制度与管理机制。正是由于高水平的反病毒研发队伍和严密的病毒监测网，在国内历次剿灭最新恶性病毒的战斗中，瑞星公司无不夺得先机，成为反病毒市场上的“最快、最大”赢家。

2000 年 12 月，瑞星公司推出其新一代单机版产品瑞星杀毒软件 2001 版，在国内首次具备了一项重要功能——实时监控并自动清除电子邮件病毒，由此，反病毒技术与恶性病毒间的“生死时速”战也上升到一个全新的境界。与此同时，这场生死时速间的对抗，已绝不仅仅对产品研发能力提出了快速应变的要求，而且对定期地升级最新技术产品以供用户及时防范也提出了重要要求。目前，杀毒软件升级多采用下载升级包进行升级的方式，用户必须先登录相关厂商的网站，使用甚不方便。那么，能否直接在用户界面上点击一下相关按钮就可以直接下载升级包并进行升级呢？瑞星杀毒软件 2001 版已实现了这种智能升级功能。用户只须在产品使用界面中点击升级按钮，软件便可以通过 FTP 的方式直接从瑞星网站上下载最新数据，并自动完成升级过程，极大地提升了反病毒软件更新的速度与易用性。同时，其独有的断点续传技术，极大地提高了智能升级的效率。业内人士认为，真正实现智能升级的功能，也将成为杀毒软件产品的一个行业标准。

在经历了最公平也最具说服力的时速的较量，相信用户已有了足够的信心与能力为自己选购一套真正有效且最为适用的病毒防治产品。因为，有瑞星如此强大的技术实力与完备的服务品质做后盾，使用这样优秀的反病毒软件将是一次最明智的选择。

博士信箱 QUESTION&ANSWER

电脑硬盘系统使用与维护常见问题解答

黄建龙

如何唤醒“睡眠”的硬盘

有时 PC 机经常会冻结 10～15 秒，甚至鼠标都不起作用，这可能是 PC 机的电源管理设置，在一段时间不活动关闭了的缘故。这对于依靠电池提供能量的笔记本电脑是有意义的，但对于桌面电脑来说，为了节省一点能量而损失工作能力是不值得的。

通常，能调整系统 BIOS 中的硬盘驱动的电源管理设置。在启动时按 Del、F1 或者其他键（可查阅系统文档）进入 CMOS 设置程序，寻找一个“电源管理”区。里面应该有一个硬盘区（标为 HDD 或 HDC），把“关电特征”设为无效。另外一个可选办法是，可以指定必须要一段更长非活动时间后，PC 才可以停止驱动。同时不要忘了检查一下控制面板的电源或电源管理图标。所有的设置因 Windows 版本的不同而异。但是总的一点，这样做了之后，都能够防止硬盘“睡眠”。

怎样利用磁盘拷贝工具恢复软磁盘中损坏的重要文件

磁盘拷贝工具 HD - copy 是修复软盘文件的重要工具软件之一。用工具软件 HD - copy 进行读盘时，能够强行读出磁盘受损的数据，并将其放在内存中。读盘结束后，将整盘数据放在内存“映像文件”的相应扇区中。再按照这样的格式写到另外一张软盘上，达到修复软盘文件的目的。具体操作步骤如下：

（1）进入 HD - copy，插入受损软盘，选择“auto verity”和“expert mode”选项，再选“read”，就会进行读盘。当读到损坏区域时，HD 会发出报警，并出现红色“E”字小框，源盘读完时，会在“source”菜单中陆续用绿色“V”（校验）字符覆盖掉原来的“E”字小块。

（2）插一张相同规格的完好空盘，选择“write”，进行写盘。写盘时“W”与“V”字符交替出现在“destination”中。写盘结束，就把原坏软盘上的数据备份至新软盘上，达到恢复数据的目的。这种方法对有些格式特殊的文档，不一定能完全恢复。

怎样利用 Norton 工具恢复 0 磁道上的数据

软盘 0 磁道物理损坏后，操作系统、应用软件均不能对其进行读写，如常用的磁盘修理工具（Scandisk、NDD、Disktools）也无法对其进行修复时，可以试着用 Norton 8.0 中的 Diskedit 把磁盘作为物理盘进行操作，将盘中数据拷出。具体操作步骤如下：

（1）启动 Norton 8.0 的 Diskedit，选择“Tools”菜单中的“Advanced recovery mode”，再根据屏幕提示，依次选“drive”、“physical disk”、“floopy drive A”、“1.44M”（以 1.44MB 软盘为例）、“virtual”，这样就将物理 A 盘变为虚拟逻辑 A 盘，设成虚拟逻辑盘后，将根据所选容量建立根目录和文件分配表，即使引导区完全损坏，也能对软盘进行目录、文件操作。在读虚拟逻辑盘时，当遇到损坏的磁盘介质不能正确读出数据时，则用已读出的相邻扇区对应地址数据来代替当前扇区无法读出数据的部分，进行数据恢复。

（2）如果存放欲恢复文件目录项和簇链的扇区没有损坏，则用鼠标双击文件名选定并打开，然后选择“Tools”菜单中的“write to”命令将它们一一拷出，达到恢复文件的目的。

（3）如果文件打开后长度只有一个扇区，而在目录项中字节数超过 512 字节，说明簇链已损坏。可选“Tools”菜单中的“use 2ND FAT table”，即改用第二个文件分配表，再拷出文件。

（4）如果文件在两个 FAT 中的簇链均已损坏，而该文件是文本文件，其中数据仍可恢复。方法是使用“Tools”菜单下的“find”功能，搜索特征字符串或按 PgUp、PgDn 键对数据区各扇区翻页浏览，然后用“write to”功能，将存储有用数据的扇区以文件格式拷出。在有较多数据的情况下，由于读盘速度慢，不能用文件拷贝的方法恢复，可将整个数据区以文件格式拷贝到硬盘，改为对硬盘文件的操作。注意：这些文件不能直接用字处理软件处理，因为文件中有许多文件结束符，而字处理软件一般以读到第一个文件结束符作为文件结尾，这样会丢失大量数据。应先用 Larry Michaels 编辑器的 hedit. exe 进行初加工，删掉大片空白和无用数据，用 16 进制中的替换功能将所有文件结束符 1AH 改成 20H，然后再用字处理软件进行处理。

怎样修复硬盘的坏道

对硬盘的坏道作修复的思路和操作方法如下：

（1）利用 Windows 9x 磁盘工具

在 Windows98 的资源管理器中选择硬盘盘符，右击鼠标，在快捷菜单中选择“属性”，在“工具”项中对硬盘盘面作完全扫描处理，并且对可能出现的坏簇作自动修正。对于不能进入 Windows98 的现象，则可以用 Windows98 的启动盘引导机器，然后在“A：>”提示符后键入“scandisk X：”来扫描硬盘，其中“X”是具体的硬盘盘符。对于坏簇，程序会以黑底红字的“B”（Bad）标出。

如果上述方法不能奏效，因为 Windows98 对“坏道”的自动修复很大程度上是对逻辑坏道而言，而不能自动修复物理坏道。对坏道处理一般采用硬盘高级格式化的方法，将有坏道的区域划成一个区，以后就不要在这个区上存取文件了。要说明的是，不要为节约硬盘空间而把这个区划得过分“经济”，而应留有适当的余地，因为读取坏道周围的“好道”是不明智的，坏道具有蔓延性，如果动用与坏道靠得过分近的“好道”，那么过不了多久，硬盘上新的坏道又将出

现。

(2) 对于硬盘 0 扇区损坏的情况，可把报废的 0 扇区屏蔽，而用 1 扇区取而代之就行了，完成这项工作的理想软件是 Pctools 9.0。使用 Pctools9.0 中的 DE 工具，但要注意的是，修改扇区完成后，只有对硬盘作格式化后才会把分区表的信息写入 1 扇区（现在作为 0 扇区了）。

(3) 对硬盘作低格。不到万不得已，这一招最好不要用，因为对硬盘作低格至少有两点害处：一是磨损盘片，二是对有坏道的硬盘来说，低格还会加速坏道的扩散。

(4) 主板 BIOS 的相关内容要设置得当，特别是对于一些 TX 芯片组级别以前的主板，由于没有自动识别硬盘规格的能力，往往会因设置不当而影响硬盘的使用，轻则硬盘不能物尽其用，重则损伤硬盘。

以上介绍的是硬盘有物理损伤时的解决方法。但是，这些方法大多数是消极的，是以牺牲硬盘容量为代价的。硬盘有了坏道，如果不是因为老化问题，则说明平时在使用上有不妥之处，比如对硬盘过分频繁地整理碎片、内存太少以致应用软件对硬盘频频访问等。而忽略对硬盘的防尘处理也会导致硬盘磁头因为定位困难引发机械故障。另外，对 CPU 超频引起外频增高，迫使硬盘长时间在过高的电压下工作，也会引发故障，平时对硬盘的使用还应以谨慎操作为上策，注意做好平时硬盘的保养维护工作。

如何设置超大容量硬盘？

BIOS 的柱面限制有四种：1994 年 7 月份以前的 BIOS：它们的界限为 504MB（1024 柱面）。这时的 BIOS 不具有 LBA（逻辑块寻址）规范。尽管有的 BIOS 的 setup 中有 LBA 选项，但不能正常使用。

1994 年 7 月份以后的 BIOS：最初它们的界限为 2.048GB。一般来说，这时的 BIOS 提供了对大于 504MB 硬盘的支持。然而，对于不同厂家，不同版本的 BIOS 来说，支持的上限（即最大容量）不同，通常在 4093 到 4096 柱面之间。

随着采用 LBA 的技术，BIOS 对硬盘的限制逐渐放松，到了 4.2GB 时，有的 BIOS 就到了极限（4.2GB 硬盘的物理参数为柱面 8190，磁头 16，扇区 63）。当配置这样 BIOS 的机器启动时（无论从硬盘还软盘），系统死机。这是因为 BIOS 向操作系统报告的磁头数为 256（00h），而 DOS / Windows 95 保存磁头数的变量只有两位（最大是 255，二进制是 FFh）。这样的 BIOS 的代表是 Phoenix Core BIOS 和 Compaq BIOS。

为了支持容量日益增大的硬盘，新型的 BIOS 又超越 8.4GB。8.4GB 硬盘的物理参数为柱面 16383，磁头 16，扇区 63。为了支持它，一个新的扩展 INT 13 被引入了 BIOS。值得注意的是，尽管有的 BIOS 版本和日期正确，它也可能不支持扩展 INT 13，因为厂家对各自 BIOS 内核设计的标准不同。

怎样处理 CIH 病毒导致硬盘损坏的问题

处理步骤如下：

(1) 先使用 KV300 的硬盘修复功能（F10）修好硬盘的引导记录，此时最好用 KV300 / B 保存分区表的数据，以免下一步有错可再一次挽救（注：用最新推出的 KV300 的 Z + 特版修复被 CIH 破坏的硬盘更有效）；用其他能修好引导记录的软件如 Kill 98 等也行。

(2) 用 NDD 可以把非主分区其他逻辑盘全部恢复（用 NDD[盘符] / Rebuild 可选定一个盘，如 D：等），会自动搜索非主分区，一旦按“Yes”即可恢复全部非主分区，大部分数据就能得到恢复，如果是 FAT32 就要用 32 位 NDD 对硬盘进行处理。（注：第一个逻辑盘，通常是 C：盘，通常是不可恢复的）也可在网址 www.vrv.com.cn 去下载 Vrvfix.exe 软件，按照说明可找回除 C 外的其他逻辑盘。

(3) 在网址 www.sdxf.com.cn 下载 Fixmmbr 软件运行，可恢复 C 盘的 60% - 95% 的资料（视破坏程度而定），也可手工恢复，但太麻烦。

怎样处理使用 Fdisk 对硬盘分区时显示的容量与实际容量不相符的问题

尽管硬盘实际容量在 2GB 以上，但利用 Fdisk 分区时显示的硬盘容量却只有几十或几百兆，最多不超过 528MB。这是由于早期硬盘读写中断 INT 13H 在设计上存在着限制。当硬盘容量大于 528MB 后就会出现卷绕现象，较新的 BIOS 对此进行了改进，支持扩展的 INT 13H 接口，支持 LBA 工作方式。如果发生问题的系统的 BIOS 支持 LBA，那么进入 BIOS 设置并打开 LBA 方式后，Fdisk 就会显示正常的容量值，如果系统的 BIOS 较老，那么可以安装 DM Disk Manager 软件并用它进行分区，此后 DM 会在系统启动时装载内存驻留程序，以完成对大容量硬盘的访问。

最近，8GB 以上 IDE 硬盘逐渐成为市场主流，在某些情况下，Fdisk 只能处理 8GB 以内的硬盘空间，此时可以利用最新版的 Partition Magic 来进行分区，它能支持 8GB 以上的 IDE 硬盘。

怎样处理硬盘无法工作在 Ultra DMA / 33 模式下的问题

较新的硬盘都支持 Ultra DMA / 33 工作模式，在这种模式下，硬盘进行读写操作时 CPU 占用率很低。为了让硬盘能够工作在 Ultra DMA / 33 模式下，硬盘、主板和操作系统都必须支持 Ultra DMA / 33。另外，还应在 BIOS 设置中把硬盘所在的 IDE 接口的 Ultra DMA 方式打开或者设置成自动检测。PWin 95 和 PWin 97 本身不支持 Ultra DMA / 33，这是因为它们所带的标准 IDE 驱动程序不支持的缘故。为了在 PWin 95 和 PWin 97 下使用 Ultra DMA / 33，需要安装最新版的 Bus Master 驱动程序，主板所带的光盘或软盘中都有这些驱动程序，主板生产厂商的网站上也有。PWin 98 直接支持 Ultra DMA / 33，因此原则上不需要再安装其他驱动程序，除非主板用的是比较新的兼容芯片组，而不是 Intel 的芯片组。上述条件满足之后，可以在系统属性中的‘磁盘驱动器’中选中硬盘，在其属性的设置一页中选中 DMA，然后重新启动机器即可。