电脑编程技巧与维护月刊

2001 年第 8 期　　总第 86 期　1994 年 7 月创刊）　　每月 3 日出版

名誉社长	张　琪
社　　长	孙茹萍
副 社 长	毕研元
总　　编	王路敬
执行主编	杨林涛
副 主 编	张　涛
编　　辑	程　芳　管逸群 叶　永
公关部主任	黄德琏
副 主 任	苏加友
出版发行部	毕　波
编辑出版	《电脑编程技巧与维护》 杂志社
法律顾问	佟秋平 商安律师事务所
主管部门	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
国外发行代号	M6232
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

2002 年《电脑编程技巧与维护》杂志订单

<table>
<tr><td colspan="2">订购单位</td><td colspan="3"></td><td>收件人</td><td></td><td>经办人</td><td></td></tr>
<tr><td colspan="2">通讯地址</td><td colspan="5"></td><td>邮　编</td><td></td></tr>
<tr><td colspan="2">订户银行及账号</td><td colspan="5"></td><td rowspan="3">收款单位</td><td colspan="2" rowspan="3">盖章
年　月　日</td></tr>
<tr><td>单价</td><td>7.80 元/期</td><td>份数</td><td></td><td>合计</td><td></td></tr>
<tr><td colspan="2">大写金额</td><td colspan="4"></td></tr>
</table>

合 订 本　1994 年：20 元；1995 年：35 元；1996、1997、1999 年合订本各 80 元。
2000、2001 年合订本各 98 元（包括挂号邮寄费）

（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　2001 年单本：7.80 元/期
2002 年单本：7.80 元/期

（凡订阅 2002 年全年的用户，随 2002 年第 12 期刊物赠送 2002 年全年源代码光盘。）

同期软盘　1998 年：20 元/季；1999 - 2002 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. 2002 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. 本刊原代码光盘所有源程序均经调试通过。
4. 凡在杂志社订阅全年杂志者可享受 10%的优惠。
5. 网上征订请查询：http://www.my8848.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　　**联系人**：田小姐

AOL 推出 AOL 7. 0 beta 版

参与美国在线 AOL 7. 0 开发计划的测试人员刚刚得到了这一 beta 版本。该最新的客户端软件产品代码为：Taz。AOL 7. 0 beta 版的程序界面与 AOL 6. 0 非常相似，大多数的新功能都放置在不起眼的地方，AOL 似乎希望在新版本中保持原来 AOL6. 0 的风格和使用感觉。

EMC 公司为 Oracle9i 提供信息存储基础架构

近日 Oracle 公司发布最新旗舰产品数据库和应用服务器系统 Oracle9i。Oracle9i 的开发和测试采用了世界信息存储领袖美国 EMC 公司的信息存储基础架构，保证了信息基础架构的可扩展性和实用性，最大地优化利用了 IT 资源。另外，全球超过 9000 台系统通过综合采用 Oracle 的服务器系统与美国 EMC 公司的存储系统和软件，验证了 Oracle9i 的整体性能和兼容性。EMC 公司通过与世界著名 IT 厂商合作，通过优势互补，保证用户在使用其存储系统时与其他设备的兼容性，同时

发挥出最大的功效。

Extreme 基于 MPLS 的光城域以太网设备兼容性测试成功

日前，Extreme Networks 美国极进网络公司 宣布已测试成功了在光城域网上实现透明局域网服务 TLS，Transparent LAN Services ，并兼容马蒂尼 IETF 工作草案 Draft - Martini Internet Engineering Task Force working document ——由 Level3 通讯组织的 Luca Martini 起草。MPLS 技术由于能够在 IP 网上达到 ATM 交换的高效性、QoS 保证和流量控制等性能，被公认为是在以太网上提供运营商级服务质量的最佳方案。

FireWall - 1 及 VPN - 1 存在安全漏洞

以色列 Check Point Software Technologies 以下称 "Check Point" 于近日宣布，该公司的 4.1 版本防火墙产品 "FireWall - 1" 及 "VPN - 1" 存在着安全漏洞。恶意用户有可能绕过防火墙将非法数据包发送到系统里去。解决的办法是使用该公司公布的补丁程序。此外，美国 CERT / CC 等也对该安全漏洞发出了警告。CERT / CC 等要求人们在使用补丁程序之前将路由器上的特定端口 UDP 259 号 关闭掉。

NEC 正在研发 0. 10 微米芯片

日本 NEC 公司在周一向外界透露下一代 0. 10 微米芯片制造技术，该技术将应用于超级到移动电话的广泛领域，计划在 2003 年初投入使用。NEC 称此大规模集成 LSI 芯片的的设计完成将是与台湾联华半导体公司 TSMC 的共同合作成果，该公司是全球最大的合约制造商。此次合作也是两家公司到目前为止最密切的一次芯片技术合作。

Oracle 修复数据库软件中的安全漏洞

Network Associates 公司的研究人员在 Oracle 的 8i 数据库软件中发现了一个可以让黑客控制整个运行 Windows 操作系统的计算机的安全漏洞，并将其危险级别定为 "高"。Oracle 已经承认存在这一漏洞，并对 9i 数据库软件进行了修改，为此前的数据库软件发布了补丁程序。Network Associates 公司的研究人员吉姆表示，这一问题发生在 Oracle 的数据库软件调用 listener 期间， listener 是用来处理数据库软件的用户和数据库软件本身之间的通讯 黑客可以通过发送大量超出软件所需的数据 缓存溢出 来实施攻击。

PC 硬盘容量将突破 137GB

20 年前，XT 个人电脑刚问世时，电脑硬盘容量最大只有 10GB，而最近个人电脑硬盘最大容量已达到 137GB，而美国电脑硬盘大厂麦斯特公司等即将开发出的新硬盘的容量可达 144GB。

Red Hat Linux 支持安腾

Linux 系统销售商 Red Hat 公司发表了 Red Hat Linux 7. 1 操作系统，该系统将可以利用 Intel Itanium 芯片 安腾处理器的最新先进特性。Red Hat 公司推出的最新操作系统标价为 499 美元，提供为期 30 天的技术支持和 6 个月的软件更新。该操作系统支持 64GB 内存，其最显著的性能提升是可以支持 64 位的安腾处理器，而以往只支持 32 位的 Intel 处理器。Red Hat 公司称，Red Hat 7. 1 为配置 8 个处理器的服务器做最优化处理。

VeriSign 为微软 . NET 计划安全上锁

日前，微软和 VeriSign 公司联合宣布，VeriSign 将为微软即将推出的一系列被称作 . NET 的互联网计划提供安全服务。自宣布推出 . NET 和 Hailstorm 计划后，微软在安全性和隐私方面受到了各方面的密切关注。微软的这两种服务可以让客户通过其 "通行证" 系统在互联网上存储各种资料。根据双方的协议，VeriSign 将为需要特别安全服务的客户在 "通行证" 系统的基础上提供附加的 "数字证书" 安全服务。两家公司指出，由于附加的数字证书不需要另外的口令，因此整个系统的易用性非常高。但 VeriSign 公司的 CEO 斯特拉顿承认，在无需额外的口令的情况下提供数字证书安全服务可能会损害系统的安全性，因为黑客只要知道一个口令就可以得到所有的信息。

W3C 发表 SOAP1. 2 规范

近日，W3C World Wide Web Consortium 发表了第一个 SOAP1. 2 Simple Object Access Protocol，简单对象访问协议 规范的公用设计。SOAP 规范最初是有微软公司领头发起的，该协议基于 XML，用于互联网上应用程序之间信息的相互交换。该规范的确立提供了通过 HTTP 传递 XML 定义数据的方法，使得运行在不同操作系统和位于防火墙后的应用程序更容易的进行集成。

阿尔卡特推出集成管理软件

法国电讯设备制造商阿尔卡特公司将于近日推出一种可在单一管理服务器上实现公司数据设备与 IP PBX 产品统一管理的软件产品。随着 VoIP 技术在大型企业领域的应用，不少用户期望能够实现语音与数据的集中管理，阿尔卡特公司的这一 OmniVista 软件正是基于企业用户的这种需求而研发的产品。OmniVista 可通过单一接口实现基于阿尔卡特公司产品构建的整个网络内所有产品的管理，这意味着用户能够借助这一软件产品简便地实现从骨干路由器、交换机到公司的 OmniPCX IP PBX 平台及 IP 电话在内的所有网络设备的管理。

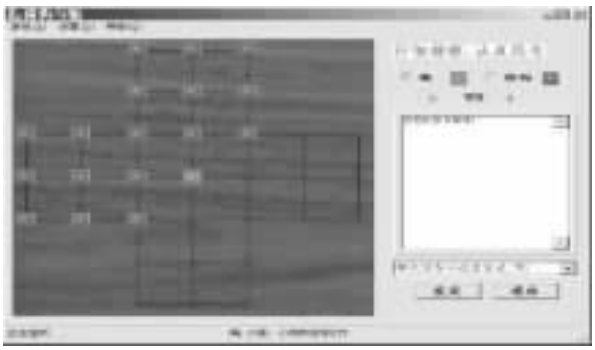
C + + Builder 讲座

夏祖柱

第三讲 网络编程

本讲结合狼与猎狗游戏连机对弈功能的实现，介绍了 WinSock 编程的机理，详细讲解了 WinSock 编程的方法与技巧，同时还介绍了一些高级的 WinSock 编程知识，对其他的网络编程组件也作了各有侧重的介绍。力图通过此讲，让读者了解有关的网络编程知识，并能进行 WinSock 编程，编写简单的网络应用程序。

网络技术堪称为当今计算机领域、最活跃、最为看好的技术了，越来越多的应用程序也要求具有网络通信、协同处理的功能。多媒体技术也只有真正和网络技术结合起来，才能发出更迷人的光彩。Internet 中不同数据的传输是基于不同的协议的，TCP / IP 是 Internet 的基础，实现基本的网络连接与通信。另外还有 HTTP 超文本传输协议 、FTP 文件传输协议）、SMTP 简单邮件传输协议 和 POP3 邮局协议）等。C + + Builder 中对于不同协议提供了相应封装它们的实现组件，我们可以根据需要，方便地使用。WinSock 是基于 TCP / IP 的网络编程，Internet 又是基于 TCP / IP 协议实现通信的，所以 WinSock 网络编程应用最为广泛，它也是我们要讲解的重点。本讲将通过 ‘狼与猎狗 连机对弈功能的实现来详细讲解如何用 C + + Builder 进行 WinSock 编程。游戏界面如图：



连机对战功能包括：1. 当一方移动棋子时，除了自己的屏幕上显示移动外，对方机器屏幕上也要显示相应移动。2. 当游戏结束时，能正确判断谁胜谁负并且在不同机器的屏幕上显示相应的输赢信息。3. 可以通过右边的文本框发送聊天信息并显示在双方屏幕上的空白区域中。

一、WinSock 编程

1. C + + Builder 中的 WinSock 编程机理

Windows Socket 又被称为 Windows 套接字，可以被看作是一组用 C 语言编写的 API，它可以在不同的计算机之间建立起一个通信渠道，实现基于 TCP / IP 协议下的数据通信。进行 WinSock 编程意味着直接调用 WinSockAPI 函数，直接与 API 打交道是很不方便的，因而在 C + + Builder 中封装了低级的 WinSockAPI 函数，从而极大地方便了用户的 WinSock 编程。C + + Builder 提供了 TClientSocket 组件和 TServerSocket 组件给用户进行 WinSock 编程，一个负责维护 Socket 的客户端，一个负责服务器端的通信连接。具体通信过程是：服务器首先启动，处于监听状态，等待客户端的请求。客户端必须首先描绘它要连接的服务器，主要是 IP 地址和端口号，再连接到所要连接的服务器端 Socket，找到后就向服务器端 Socket 请求连接。当服务器向客户端发出 ‘允许连接 ” 的信号后，双方的连接就建立起来了。本程序中设定选择狗的一方为服务器方，选择狼的一方为客户方，并且狼先走棋。

2. 建立服务器端 Socket

将一个 TServerSocket 组件加到应用程序窗体上，该应用程序就变为了 TCP / IP 服务器。要使服务器能监听客户请求还必须对服务器进行设置。这里主要是设定 TServerSocket 组件的 Port 属性，如我们可以将其设为 1024。TServerSocket 组件还有一个 Service 属性也可以用来确定端口号。Service 表示服务器提供何种类型的服务如 FTP、FINGER 或 TIME，这些端口号都是服务器默认的，从而也就确定了端口号。注意如果同时设定 Port 和 Service 属性，则将忽略 Port 属性。

3. TServerSocket 的重要属性、方法和事件

1 Active 属性

前面介绍了它的 Port 属性，它还有一个重要的属性是 Active 属性，该属性用于指明和其他机器进行通信的 Socket 连接是否处于打开状态。在狼和猎狗的程序中，初始化时我们没有指明谁是客户方，谁是服务器方，因而两边都处于监听状态。在程序的 InItialize 函数的开始加入如下代码即可：

```
this -> RadioButton1 -> Checked = true;
  StatusBar1 -> Panels -> Items[1] -> Text = "嗨, 小狼、小狗欢迎你们!";
  StatusBar1 -> Panels -> Items[0] -> Text = " 正在监听.....";
  ServerSocket1 -> Active = true;
```

说明：这里 RadiaoButton1 单选按钮选中表示自己现在是狼，窗体上加入了一个状态条，显示当前状态，如正在监

听。

2 connections 属性

该属性数组的每一个元素代表一个当前活动的连接，通过该属性可以方便地获取每一个客户的详细信息。如我们可以再有一个客户方，而此时的 TServerSocket 的连接就有两个，分别用 connections[0]和 connections[1]表示。它的具体用法在后面讲发送数据时可以看到。

3 Close、Open 方法

Close 方法关闭一个 Socket 连接，Open 打开一个 Socket 连接。

4 OnAccept 事件

当接受了一个客户端的 Socket 连接后，该事件被触发。本程序中当服务器方接受了来自对方的连接请求后，触发 OnAccept 事件。代码如下：

```
void __fastcall Tmain: : ServerSocket1Accept(TObject * Sender, CustomWinSocket * Socket)
{
  Isserver = true;
   StatusBar1 -> Panels -> Items[0] -> Text = ″连接到″ + Socket -> RemoteAddress;
    this -> RadioButton1 -> Checked = false;
    this -> RadioButton2 -> Checked = true; //第二个单选按钮选中表示服务器方为狗
Iswolf = false;
    canput = true; //此刻狗先走, 狼不允许走
}
```

5 OnClientRead 事件

当客户机有数据传输过来时，将触发此事件通知服务器 Socket 读取有关信息。函数原型如下：

```
    void __fastcall Tmain: : ServerSocket1ClientRead(TObject * Sender, TCustomWinSocket * Socket)
```

这里传送了一个 Socket 连接对象过来，从中读取数据。如程序中当客户端传数据过来时，服务端 Socket 的 OnClientRead 事件如下：

```
    void __fastcall Tmain: : ServerSocket1ClientRead(TObject * Sender, TCustomWinSocket * Socket)
{     int wolf;
    strcpy(Receive, Socket -> ReceiveText(). c_str()); / * 这里调用了 ReceiveText 方法和 strcpy 函数, 将 Socket 传送的字符串复制到 Receive 字符串中去, 这里传输的数据都被格式化成了字符串 * /
    switch(Receive[0]){
     case ′A′ : //发送的是输赢消息, 这里要显示输赢的消息
        ShowMessage(″嘿, 你输了!″);
        break;
    case ′C′: //发送的是聊天信息, 这里除了显示聊天的消息, 同时还要显示对方的名字
          if (Iswolf)
          Memo1 -> Lines -> Add(″[猎狗]″);
          else Memo1 -> Lines -> Add(″[老狼]″);
          Memo1 -> Lines -> Add(AnsiString(& Receive[1]));
          break;
    case ′P′:   / * 发送的是移动棋子的消息, Receive[1], Receive[2], Receive[3], Receive[4]分别代表移动棋子的新位置和原来位置. * /
          prexpos = Receive[3];
          preypos = Receive[4];
          int xxx = Receive[1];
           int yyy = Receive[2];
          if(Iswolf = = true)
        {  checkroute(xxx, yyy, 0); //检查路径是否正确, 同时消除原位置棋子
          dispchess(xxx, yyy, 0); //在新位置显示棋子
        wolf = 0;
          }
        else {  checkroute(xxx, yyy, 1);
        dispchess(xxx, yyy, 1);
          wolf = 1;
        }
        if(check(wolf))
{  MessageBox(this, ″真对不起, 你输了!″, ″哈哈!″, MB_OK);
MessageBox(this, ″别泄气, 重新来过!″, ″重来!″, MB_OK);
          }
         canput = true;
         break;
   }
     }
```

说明：本程序中对弈双方传送的信息有三种：1. 移动棋子的消息；2. 聊天的信息；3. 显示输赢的消息。为了区别这三种不同的消息，我们用传送的信息头进行标识，“A”表示输赢信息，“C”表示聊天信息，“P”表示移动棋子消息，针对三种不同的消息服务器端分别进行处理。

4. 建立客户端 Socket

与服务器端 Socket 建立的方式相似，将一个 TClientSocket 组件加到应用程序窗体上，这个应用程序就变成了一个 TCP / IP 客户机。同样，我们要设置它的端口号 Port，它的 Service 属性与 TServerSocket 组件的 Service 属性相似，这里不再赘述。

5. TClientSocket 的重要属性、方法和事件

1 Host、Address、Active 属性

Host 或 Address 用来指定要连接的服务器，Host 支持通过服务器名来指定服务器，Address 支持通过 IP 地址来指定服务器。Active 设为 true，客户端就向服务器端发送连接请求，如果服务器处于监听状态就会接受请求，并建立连接。本程序中的连接代码如下：

```
void __fastcall Tmain: : startClick(TObject Sender)  // 按下菜单命令 start 时触发该事件表示开始连接
{
    if(ClientSocket1 -> Active)
      ClientSocket1 -> Active = false;
    if (ServerSocket1 -> Active)
       ServerSocket1 -> Active = false;
if(InputQuery(″我的计算机连接到″, ″计算机地址″, server));
     {    ClientSocket1 -> Host = server;
ClientSocket1 -> Active = true;
```

```
StatusBar1 -->Panels -->Items[0] -->Text = "正在连接.....";
        }
    }
```

注意：同一应用程序中，当设置 TClientSocket 的 Host 属性，并将 Active 设置为 true 进行连接时，TServerSocket 组件的 Active 必须设为 False。

2 Connect 事件

在客户端和服务器的连接建立后将触发该事件。本程序中的 Connect 事件触发后在状态条上显示成功连接到相应主机的信息。代码如下：

```
void __fastcall Tmain::ClientSocket1Connect(TObject *Sender, CustomWinSocket *Socket)
{
 StatusBar1->Panels->Items[0]->Text = "成功连接到计算机" + Socket->RemoteHost;
 canput = false;
}
```

3 OnRead 事件

当服务器端 Socket 向客户端发送信息，客户端的 Socket 将从 Socket 连接中读取信息时，触发该事件。该事件与 TServerSocket 的 OnClientRead 事件颇有相似之处，只不过各自实现不同方的接受数据功能罢了。本程序中，ClientSocket 的 OnRead 事件代码与 TServerSocket 的 OnClientSocket 事件的代码基本一样。

以上介绍了 TServerSocket 组件和 TClientSocket 组件的重要属性、方法和事件。下面我们介绍如何发送数据。

对于具体的数据传送要操作实际的 Socket 对象。前面介绍的两个组件中的部分属性和方法实质上就操纵了 Socket 对象。TServerSocket 对象为 TServerWinSocket 类的实例，TClientSocket 对象为 TClinetWinSocket 类的实例 而 TServerWinSocket 类和 TClientWinSocket 类都继承于共同父类 TCustomWinSocket，该类包含了重要的通信和数据传送方法。SendText 就是该类的重要方法，用来向另一端 Socket 发送一个字符串。本程序中当按下聊天区域中的发送按钮时，聊天信息将被发送出去。代码如下：

```
void __fastcall Tmain::Button1Click(TObject *Sender)
{
  Send[0] = 'C';
  strcpy(&Send[1], ComboBox1->Text.c_str());
  if(Isserver)//判断是服务器方还是客户方，以便用不同组件的方法发送
      ServerSocket1->Socket->Connections[0]->SendText(AnsiString(Send));
  else
   ClientSocket1->Socket->SendText(AnsiString(Send));
  if (Iswolf)
   Memo1->Lines->Add("[猎狗]");
  else
   Memo1->Lines->Add("[老狼]");
  Memo1->Lines->Add(ComboBox1->Text);
  ComboBox1->Text = EmptyStr;
  }
```

注意：这里的 TServerSocket 发送数据时就用 connections[0]表示了当前的一个活动连接。

要完全实现棋子在连机状态下的移动，我们还必须写一个重要的事件代码，那就是鼠标按下时的响应事件。代码如下：

```
void __fastcall Tmain::Image1MouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{  String *temp;
  int yy = X/63;   //yy 为列
  int xx = Y/63;   //xx 为行
   if (canput&& Button == mbLeft)
    {  if (putchess[xx][yy] == true)
{ if (state[xx][yy]! = sNone)  //如果当前按下的地方不为空，即选中了某一棋子
if (state[xx][yy] == swolf) //判断是否选中了狼
{
if(!Iswolf)//如果自己不是狼则棋子选错，给出错信息
ShowMessage("不许走对方棋子");
else saveprechess(xx, yy, 1);  //保存 xx 行，yy 列的位置
}
else
{
 if(Iswolf) //如果自己是狼而选中了狗，则给出错信息
ShowMessage("不许走对方棋子");
else  saveprechess(xx, yy, 0);  //将选中棋子位置保存；
 }
else if( checkroute(xx, yy, flag))/*如果当前按下的地方为空则检测路径是否正确，检测时如正确将删除原位置的猎狗*/
 {//如果路径正确允许行走，则发送移动棋子消息
 Send[0] = 'P';
 Send[1] = xx;
 Send[2] = yy;
 Send[3] = prexpos;
 Send[4] = preypos;
 Send[5] = '\0';    //'\0'表示信息结束
 dispchess(xx, yy, flag);
 if (Isserver)
     ServerSocket1->Socket->Connections[0]->SendText(AnsiString(Send));
   else
  ClientSocket1->Socket->SendText(AnsiString(Send));
 stepcount++;
   canput = false;
   if (check(flag))
{MessageBox(Handle, "好样的，你赢了！", "提示框", MB_OK);
MessageBox(Handle, "请再接再励，重新来过！", "提示框", MB_OK);
if(flag == 1)
Send[0] = 'A'; // 发送自己赢了的消息
 Send[1] = '\0';
 if (Isserver)
ServerSocket1->Socket->Connections[0]->SendText(AnsiString(Send));
```

```
else
ClientSocket1 -->Socket -->SendText(AnsiString(Send));
}
  }
   else MessageBox(Handle, "路径错误", "提示框", MB_OK);
   }
 }
```

至此，本游戏的连机对弈功能已基本实现，其中的路径检测和输赢检测算法较复杂，限于篇幅这里不列出，但其他的双方通信机制已完全实现。

二、高级 WinSock 编程

1. 流式数据传输

上面的例子中，我们实现了服务器和客户机之间的数据传输，但双方之间传输的仅仅是文本。使用 Socket 对象的 SendText 方法传送的字符串大小是有限制的，使用另外一种简单的方式即使用 SendBuf 和 Receive 方法传输数据的大小也是有限制的。因而它们都不能满足大量数据的传输。一般地，传输大量数据时我们使用流式传输方法。使用 Socket 对象的 SendStream 方法可传输一个流类对象，这样就可以将文件、声音、图片等数据一步传输到位。但接受方并没有相应的 ReceiveStream 方法将发送过来的流全部接受，这样我们就必须采用灵活的编程方法来解决这一问题。如我们可以在接受方使用 ReceiveBuf 方法分段接受数据。程序示例如下：

```
char Buffer[1000]
Tmemorystream  * mystream;
While(true){
  n = Socket -->ReceiveBuf(Buffer,  sizeof(BufFer)); //n 是 ReceiveBuf 方法返回的实际读取的字节数
       if (n < =0) break;
       else mystream -->write(Buffer, n);
       sleep(200); //延迟 200 毫秒以便发受双方保持同步.
      }
```

2. Socket 的连接方式

细心的读者可能发现，当我们将一个 TServerSocket 组件或一个 TClientSocket 组件放在窗体上时，它的属性栏中分别有一个 ServerType 属性和 ClientType 属性且缺省值都是 ctNonBlocking。这里设置的就是 Socket 的连接方式。Socket 的连接方式有两种，非阻塞式和阻塞式。在非阻塞方式下，Socket 连接上的读写操作异步发生，读写操作不影响程序中其他代码的执行。上面的流式数据传输的解决办法就是在非阻塞方式下采取的。非阻塞方式的连接在传送大量数据时容易造成数据收发双方的不同步。阻塞方式下，Socket 以同步方式读写数据，在读写操作完成之前，应用程序处于等待状态。阻塞方式因为应用程序经常处于等待状态，所以会影响整个应用程序的性能。但在需要传输大量数据的情况下，使用这种连接方式，并用 TWinSocketstream 类进行实际的流类读写操作，却是很好的办法。

三、其他的网络编程组件

在组件面板中可以看到有 FastNet 一栏，这里放有 NetMasters 公司给我们提供的一组称为 FastNet Tools 的组件。我们下面所介绍的组件都是被 C++Builder 所集成进来的 FastNet Tools 组件。

1. TPowerSocket 组件类

许多网络的组件如 NMHTTP、NMFTP，NMPOP3 等都是从 PowerSocket 组件继承下来的。Powersock 组件是 C++Builder 本身的 TServerSocket 组件和 TClientSocket 组件的非常有力的替代品，实际上，通过 TPoerSock 组件和 TNMGeneralserver 组件完全可以替代 C++Builder 自己的 Socket 组件。TPower 组件提供了完整的套接字支持，因而一般将 TPower 组件作为创建新协议的基类，包括用户定制协议。TNMGeneralserver 组件提供了多线程服务所需的支持。另外 TPower 和 TNMGeneralserver 组件也可以直接使用，且功能更强大。

2. HTML 组件、NMHTTP 组件

Internet 最为常用的协议莫过于 HTTP（超文本传输协议）了，我们就是通过此协议来请求 HTML 文档、下载 HTML 文档，然后在客户端解释并转换为页面显示出来。C++Builder5.0 中提供了一个 NMHTTP 组件，该组件在 C++Builder4.0 中是不可视的，在 C++Builder5.0 中不仅可视，而且集成了原 C++Builder4.0 中 HTML 组件的功能，HTML 组件也由此而被取消。NMHTTP 组件用来通过 WWW，实现基于 HTTP 的数据传输，它提供了对 HTTP 协议的完整支持，它不但可以获取数据，而且能发送数据，并且支持 Cookie 和代理服务器。

3. NMFTP、NMSMTP、NMPOP3 组件

文件传输协议 FTP、简单邮件传输协议 SMTP 和邮局协议 3 POP3，也是 Internet 上常用的标准协议。C++Builder 也包含了为这些协议编程所提供的相应组件。NMFTP 组件继承于 PowerSock 类，功能十分强大，它对代理服务器和断点续传提供直接支持，它的许多属性和方法与 NMHTTP 组件相同。

NMSMTP 组件用于实现简单的邮件协议，该协议用于在 Internet 上发送电子邮件。使用该组件只需设置其主机和端口号，并将发送的内容存储在 PostMessage 属性中，然后调用 SendMail 对象方法就可以完成电子邮件的发送。

NMPOP3 组件实现邮局协议 3，该协议用于在 Internet 上检索电子邮件。NMPOP3 组件提供了对 POP3 的完整支持，包括身份认证登录、信箱邮件检索等。

本讲结合狼和猎狗游戏连机对弈功能的实现，介绍了 WinSock 编程的机理，详细讲解了 C++Builder 中的 WinSocket 编程方法与技巧，另外介绍了一些高级的 WinSock 编程知识，最后还对其他网络编程组件也作了各有侧重的介绍。

（收稿日期：2001 年 2 月 5 日）

Windows 环境下开机日记程序的实现

苏成翔 王 峰

摘 要 本文通过编写 Windows 环境下自动开机日记程序 讨论了以下编程技术 ①系统关机时应用程序的自动关闭 ②不使用部件库的 Windows 编程方法 ③禁止运行应用程序的第二个实例 ④任务条状态区图标的编程。

关键词 开机日记，Windows 编程，Delphi 编程

编写一个开机日记程序 使之自动记录计算机每次开机的用户名字及启动、关机时间等信息 可以帮助我们了解机器的使用情况 无论对个人的机器还是单位的公用机器 都是有益的。对于那些保密性强、只允许特定用户使用的机器，开机日记程序更有必要。

在 Windows 环境下实现自动开机日记 需要解决几个技术问题 分别是 ①如何实现开机时自动运行和关机时自动写下要记录的信息并自动关闭 ②不使用部件库的 Windows 编程方法 ③禁止运行应用程序的第二个实例 ④任务条状态区图标的编程 ⑤记录信息观察器 提供一种手段让用户能观察到开机日记程序所记录的计算机使用信息

本文中的程序是用 Object Pascal 语言编写并在 Delphi4. 0 环境下编译通过的。

一、自动运行与退出

1. 自动运行

有两种方法能使一个程序在 Windows 启动时自动运行 可以创建程序的 ‘快捷方式 ’并把它加入到 ‘开始 ’菜单的 ‘程序 h启动 ’栏内 也可以在注册表内加入相应的信息 即在 “HKEY_LOCAL_MACHINE h Software h Microsoft h Windows h CurrentVersion h Run ”或 “HKEY_LOCAL_MACHINE h Software h Microsoft hWindows hCurrentVersion hRunServices ”键中建立一字符串键值 使之指向所要自动运行的程序。

2. 自动退出

一般情况下若要使 Windows 应用程序退出运行 单击 ‘文件 ’菜单上的 ‘退出 ’ 或单击标题栏最右边的按钮。但是要让程序在 Windows 系统关机时完成一定的任务后自动退出 则要在编写程序时建立特定的消息响应机制。

Windows 系统是消息驱动的操作系统 Windows 的应用程序通过对接收到的各种消息进行处理来完成任务。Windows 9x 操作系统在关机前向所有的窗口发送 WM_QUERYENDSESSION 消息 应用程序处理完这个消息后如果可以结束运行 则返回 True 如果所有的应用程序都返回 True 则操作系统向所有的窗口发送 WM_ENDSESSION 消息来指示关闭系统。

编写开机日记程序时 让程序在响应 WM_QUERYENDSESSION 消息的过程中完成信息的记录 从而达到预期目的。

二、程序框架设计

使用可视化编程工具 如 Delphi、VC 等 编写程序比较直观、快捷 但是得到的可执行代码长度较大。开机日记程序是在计算机的运行期间一直驻留在内存中 希望它占用的内存越少越好。虽然现在计算机的内存一般都有几十 MB 甚至一两百 MB，不在乎一两百 KB 的额外开销 但是作为一种编程手段 讨论一下如何以最小的可执行代码实现预期的功能 仍然是有意义的。

为减少开机日记程序的尺寸 必须不使用可视化编程工具提供的可视化部件库 而要从头做起 用传统的方法设计 Windows 的应用程序。虽然选择这样的编程方法实质上是一种 ‘倒退 ’ 但是通过编写这样的程序可以加深我们对可视化编程工具的部件库的理解 因为这些部件库也是从底层一步步编起的 而设计一个高质量的程序必须不仅要 ‘知其然 ’也要 ‘知其所以然 ’ 同时我们可以得到最小尺寸的应用程序。实际上这也是可以做到的。因为 Delphi、VC 等高级语言已经提供了 Windows API Win32 的接口 我们可以在应用程序中调用 API 中的任何一个函数。本文中的开机日记程序编译后得到的代码长度仅为 75KB。

1. 主函数

主函数是 Windows 应用程序的入口点 在 C 语言中其名字一般为 WinMain。

Windows 在载入运行一个 C 应用程序时调用 WinMain 函数并传递给它四个参数 1 本应用程序实例的句柄 hInstance 2 先前正在运行的本应用程序的另一个实例的句柄 hPrevInstance 3 命令行字符 lpCmdLine 4 确定本应用程序的显示状态的一个整数参数 nCmdShow。其中第二个参数 hPrevInstance 在 Win32 应用程序中永远为 NULL。

若用 Pascal 语言编写一个应用程序 我们不可能定义一个

使 Windows 在程序开始运行时就会调用的 WinMain 函数 但是我们可以采取其他办法。因为系统单元 System 已经提供了四个预定义的变量 hInstance、 hPrevInst、 CmdLine、 CmdShow 这四个变量的初始值就是 Windows 系统传递过来的那四个参数的值 所以我们可以定义一个过程 WinMain 在程序一开始运行就调用它 并使得整个应用程序的可执行程序段只调用此过程。如下所示

```
procedure WinMain(hInstCurrent, hInstPrev: HINST; lp-
szCmdLine: LPSTR; nCmdShow: integer);
begin
  // . . .
  Halt(n);
end;
begin   // Main program
  WinMain(hInstance, hPrevInst, CmdLine, CmdShow);
end.
```

过程 WinMain 将运行到程序结束 我们可在 WinMain 中用 Halt n 来结束程序 其中 n 为某个程序指定的整数 这个整数将被存放于 System 单元的 ExitCode 变量中 并由 System 单元返回给 Windows。

2. 窗口的建立

每个 Windows 应用程序建立至少一个窗口 称作主窗口 作为用户和应用程序之间的接口界面。要建立一个窗口 必须先登记窗口类 方法是填写一个 WNDCLASS 的数据结构 然后将它作为参数调用 Win32 API 函数 RegisterClass。若操作成功 返回 True 否则返回 False。如果窗口类登记成功 则应用程序必须通过调用 CreateWindow 建立自己的主窗口 若成功地建立了窗口 则返回所建立窗口的句柄 否则返回零。在后一种情况下 应用程序可以调用 Halt 0 结束程序 对于前一种情况 应用程序必须调用 ShowWindow 来显示窗口 否则窗口仍然是不可见的 再调用 UpdataWindow 使 Windows 发送消息 WM_PAINT 给主窗口 从而刷新主窗口。

3. 消息循环

应用程序成功建立并显示了主窗口之后 就进入了检测消息队列的消息循环 在消息循环中可以使用函数 GetMessage 或 PeekMessage 检查有无消息 然后应用程序可以对检出的消息进行处理。如下所示：

```
if PeekMessage(AMsg, 0, 0, 0, PM_REMOVE) then
  begin
   Result: = True;
   case AMsg. Message of
     WM_CLOSE:
       语句 1;
     WM_QUIT:
       语句 2;
     else
       语句 3;
   end;   //of case
  end;    //of if
```

4. 窗口函数

Windows 操作系统中每个窗口类都有一个窗口函数 应用程序在登记窗口类时必须指明窗口函数的地址。窗口函数必须带有四个参数 依此是窗口句柄、 消息、 消息字参数、 消息长参数。窗口函数是被 Windows 调用而不是被应用程序调用 它在操作系统和应用程序间起着桥梁的作用。代码如下

```
function MainWndProc(AhWnd: HWND; AMessage: UINT;
wParam: WPARAM; lParam: LPARAM): LRESULT; Std-
Call; export;
begin
  Result: = 0;
  case AMessage of
    WM_消息 1:
      语句 1;
WM_消息 2:
语句 2;
    WM_消息 3,
      语句 3
    WM_消息 4:
      语句 4
   else
    Result: = DefWindowProc(AhWnd, AMessage, wParam,
lParam);
  end;
end;
```

三、 禁止第二实例的运行

开机日记程序只能允许有一个实例在运行 若有更多的实例在运行的话 必定会造成记录信息的错乱 因此必须禁止第二实例运行。在 Windows3. x 的 16 位操作系统中 系统传递给应用程序的第二个参数 hPrevInstance 是先前正在运行的本应用程序的另一个实例的句柄 如果此参数值为零表示没有先前的实例在运行 从而本实例为第一实例 此参数值不为零表示本实例不是第一实例。

但是对 Win32 系统的应用程序此参数恒为零 无法据此判断是否为第一实例。我们可以用以下方法限定应用程序只能有一个实例运行。用 API 函数 CreateMutex 创建一个名字的互斥体 Mutex 接着调用 GetLastError 如果返回值是 ERROR_ALREADY_EXISTS 说明本应用程序的另一个实例已在运行。程序编码如下：

```
function TTinyApp. NotFirst: Bool;
var
  ASecurityAttr: SECURITY_ATTRIBUTES;
begin
  with ASecurityAttr do
   begin
    nLength: = Sizeof(SECURITY_ATTRIBUTES);
    lpSecurityDescriptor: = nil;
    bInheritHandle: = True;
   end;
  CreateMutex(@ ASe-
curityAttr,          False,
```

下转第 22 页）

用 Visual C + + 制作 Office2000 风格的字体组合框

元凯宁

Office2000 系列软件的工具栏上都有一个字体选择组合框，它有三大特色，一是具有扁平和鼠标热点的效果；二是字体的样式在下拉列表框中直接可见，即用相应的字体来绘制字体名称；三是将最近使用过的字体列在下拉列表框的顶端，方便下次使用。

那么，我们如何在程序中实现这种功能呢？

Windows 的基本控件，如列表框、组合框和按钮等，可以由系统来绘制，也可以由用户程序自己负责绘制，需要自己绘制时，必须为控件指定 OWNERDRAW 风格，通过这个手段，我们可以绘制出组合框的下拉列表中的字体效果；然后，通过接管 WM_PAINT 消息，可以为组合框绘制出扁平和鼠标热点效果来。

1. 使用字体本身来绘制字体名称
2. 具有扁平的效果
3. 保存了 4 种刚刚用过的字体

要自画组合框，首先要为组合框指定 CBS_OWNERDRAWVARIABLE 风格或 CBS_OWNERDRAWFIXED 风格。区别是前者可以为不同的列表项指定不同的高度，而后者则用于所有的列表项目都具有相同的高度的情况。使用 MFC 实现自画组合框需要从 CComboBox 类派生出一个新类，再为它重载三个虚函数：CompareItem、DrawItem 和 MeasureItem。其中，CompareItem 用于在排序时比较两个列表项目的先后顺序，DrawItem 负责每一个列表项的绘制工作，MeasureItem 则为每个列表项目指定不同的高度。只要是自画的组合框，就一定要重载这三个函数（用不到的函数，函数体可以为空），否则会出现 ASSERT 对话框，因为在基类 CComboxBox 的虚函数实现代码中包含了一行 ASSERT FALSE 。这三个函数要用到三个结构，分别介绍如下。

COMPAREITEMSTRUCT 结构的定义在 winuser. h 的 2317 行

```
typedef struct tagCOMPAREITEMSTRUCT { //cis
    UINT        CtlType; //控件的类型
    UINT        CtlID; //控件的 ID
    HWND        hwndItem; //窗口句柄
    UINT        itemID1; //要比较的第一个项目的 ID
    DWORD       itemData1; //要比较的第一个项目的 32 位用户指定的关联值
    UINT        itemID2; //要比较的第二个项目的 ID
    DWORD       itemData2; //要比较的第二个项目的 32 位用户指定的关联值
    DWORD       dwLocaleId; //
} COMPAREITEMSTRUCT, NEAR * PCOMPAREITEMSTRUCT, FAR * LPCOMPAREITEMSTRUCT;
```

在 CompareItem 函数中，一般是通过所给出的两个要比较项目的 ID 或是关联值，查找自己的数据结构，再返回不同的值来表明两个项目的先后顺序。返回 - 1 表明第一个项目排列在第二个项目之前，返回 0 表明两个项目无法分出先后顺序，返回 1 表明第一个项目排在第二个项目之后。

DRAWITEMSTRUCT 结构的定义如下 （winuser. h 的第 2291 行）

```
typedef struct tagDRAWITEMSTRUCT { //dis
    UINT        CtlType; //控件的类型
    UINT        CtlID; //控件的 ID
    UINT        itemID; //项目的 ID, 在组合框中是序号
    UINT        itemAction; //需要的动作, 可为全体的、有焦点的或是已选择的
    UINT        itemState; //项目的状态, 可为选择的、禁止的、有焦点的等
    HWND        hwndItem; //窗口句柄(菜单句柄)
    HDC         hDC; //设备上下文
    RECT        rcItem; //项目的矩形区
    DWORD       itemData; //项目的一个 32 位的用户指定的关联值
} DRAWITEMSTRUCT, NEAR * PDRAWITEMSTRUCT, FAR * LPDRAWITEMSTRUCT;
```

一般在程序中要根据 itemState 判断项目所处的状态，然后再使用 hDC 来绘图。

itemState 的可用值有 ODS_SELECTED、ODS_GRAYED、ODS_DISABLED、ODS_CHECKED、ODS_FOCUS、ODS_DEFAULT、ODS_COMBOBOXEDIT、ODS_HOTLIGHT、ODS_INACTIVE，后两个值仅用于 Windows98 和 Windows2000。

MEASUREITEMSTRUCT 结构则是在 winuser. h 的 2278 行定义的。

```
typedef struct tagMEASUREITEMSTRUCT { //mis
    UINT       CtlType; //控件的类型
    UINT       CtlID; //控件的 ID
    UINT       itemID; //项目的 ID, 在组合框中是序号
    UINT       itemWidth; //项目的宽度
    UINT       itemHeight; //项目的高度
    DWORD   itemData; //项目的一个 32 位的用户指定的关联值
}MEASUREITEMSTRUCT, NEAR * PMEASUREITEMSTRUCT, FAR * LPMEASUREITEMSTRUCT;
```

在程序中一般要通过 itemID 或是 itemData 来在自定义的数据结构中查找数据，然后计算并返回宽度与高度值，在函数中,可以为不同的项目指定相同或不同的高度。

为了实现使用相应的字体来绘制字体名称的效果，我们可以在 DrawItem 函数中创建所需要的字体对象，将它选入设备上下文，并且用它来绘图，这样，每种字体的名称就可以用本身的字体来绘制了。

为组合框实现扁平和鼠标热点效果并不难，有多种办法可以实现。最通用的办法是编制一个类，通过安装钩子函数来跟踪每一个控件的创建，然后替换掉并保存下它的窗口过程，这样，就可以在新的窗口过程中对不同的消息分别进行处理 当然也就可以处理 WM_PAINT 等消息并绘图），而且因为原先的窗口过程已经保存下来，所以并不会损失任何原先控件的功能，只不过这样做需要编制大量的代码。在本例中，为了简单起见，直接在 CComboBox 的派生类中处理 WM_PAINT 等消息，根据不同的情况绘制出扁平效果，而鼠标热点则依赖于窗口定时器的使用。对于如何绘制扁平效果的控件已经讨论得很多了，不再详述。下面将如何取得系统中所有已安装的字体的信息的方法作一介绍。

要枚举系统中所有可用的字体，可以通过 EnumFontFamiliesEx 函数来实现。另一个可以实现相同功能的 EnumFontFamilies 函数是为了与 16 位的程序兼容才保留的，不再推荐使用。EnumFontFamiliesEx 函数的原型如下：

```
int EnumFontFamiliesEx(
  HDC hdc,                  // 设备上下文, 由它来决定是屏幕字体或是打印机字体
  LPLOGFONT lpLogfont,  // 一个用于提供信息的 LOGFONT 结构
  FONTENUMPROC lpEnumFontFamExProc,  // 回调函数
  LPARAM lParam,          // 送往回调函数的用户指定的指针
  DWORD dwFlags           // 保留参数, 必须为 0
);
```

其实，与其它的 Windows API 一样，如果去查它们的原始定义，是有 - A 和 - W 的区别的， - A 用于 ANSI 和 MBCS 字符集， - W 用于 Unicode 字符集。但是一般情况下可以不严格区分二者，UNICODE 标志已经很好地处理了这个问题。

lpLogFont 参数只有三个成员对于 EnumFontFamiliesEx 函数是有效的，它们是 lfCharset：如果设为 DEFAULT_CHARSET 会枚举所有的字体；如果指定为 GB2312_CHARSET、ANSI_CHARSET 等值，则只会枚举指定的字符集下的字体。

LfFaceName：枚举不同字符集下的同名字体。

LfPitchAndFamily：除去希伯来语和阿拉伯语的操作系统都设置为 0，当 EnumFontFamiliesEx 函数被用之后，它将会对每一种字体调用回调函数一次，在回调函数中要保持这个动作需要返回 1，否则返回 0 值，退出回调过程。回调函数的原型是：

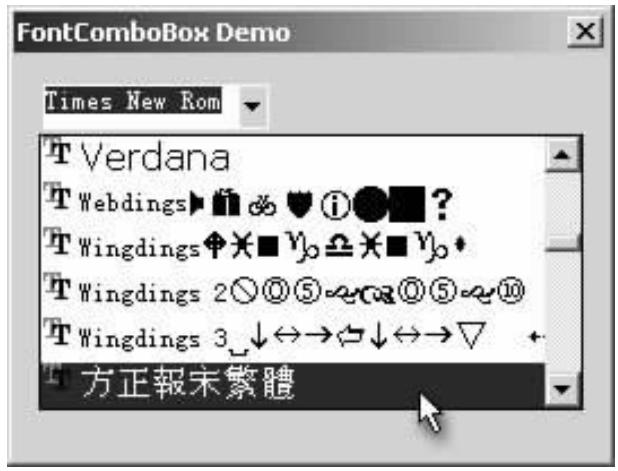
```
int CALLBACK EnumFontFamExProc(
  ENUMLOGFONTEX * lpelfe, // 包含了一个 LOGFONT 结构和一些可读性的信息
  NEWTEXTMETRICEX * lpntme, //
  int FontType, // 字体的类型
  LPARAM lParam// 由 EnumFontFamilies 函数传来的参数
);
```

lpntme 成员对于 true type 字体，是一个 NEWTEXTMETRICEX 结构，对于其它字体，则是一个 NEWTEXTMETRICEX 结构。

字体类型的可用值为 DEVICE_FONTTYPE、RASTER_FONTTYPE 或者 TRUETYPE_FONTTYPE，分别对应设备字体、光栅字体和 true type 字体。

在回调函数中使用 lpelfe - >elfLogFont. lfFaceName 就可以得到字体的名称。

下面让我们来实现这种组合框。



1. 下拉列表的宽度与最宽的列表项相同
2. 对于符号字体,先有默认字体绘制名称

创建一个对话框工程，添加一个从 CComboBox 类派生出的新类，名为 CFontComboBox，重载 CompareItem、DrawItem 和 MeasureItem 等函数，最后得到如下的文件。

对话框图资源中的组合框要具有 CBS_OWNERDRAWVARIABLE、CBS_SORT 和 CBS_HASSTRINGS 风格。

为了速度上的考虑，使用映射 （Hash 表）来保存所有的字体名称信息。共有两个映射，一个用于所有的字体名称，一个用于最近使用过的字体名称，最近使用过的字体名称位于组合框的下拉列表的最上端。

所有的字体在下拉列表中均有一幅位图位于其左侧。在最近使用过的字体与普通的字体列表之间有一道双线间隔开。对于符号字体，因为使用字体本身来绘制字体名称会导致无法辨认出字体名，所以使用默认的字体 父对话框的字体）先绘制一遍该字体的名字，再用该字体绘制字体名字。

下拉列表的宽度与列表项目的最大宽度相同。

下面是 CFontComboBox 类的文件列表。

头文件

```
#if !defined(_FNTCOMBO_H__INCLUDED_)
#define _FNTCOMBO_H__INCLUDED_
#include <afxtempl.h>
#pragma once
#define FCB_INMRU          0x00000001
#define FCB_STARTPOS       0x00000002
#define FCB_DRAWNORMAL     0x00000001
#define FCB_DRAWRAISED     0x00000002
#define FCB_DRAWPRESSD     0x00000004
////////////////////////////////////////
// CFontComboBox window
class CFontComboBox : public CComboBox
{
    typedef struct tagFCBDATA{
        LOGFONT lf;
        UINT flag;
    }FCBDATA, * PFCBDATA;
public:
    void Initialize(int nMRUCount = 10);
    //{{AFX_VIRTUAL(CFontComboBox)
    virtual void PreSubclassWindow();
protected:
    virtual int CompareItem(LPCOMPAREITEMSTRUCT lpCompareItemStruct);
    virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);
    virtual void MeasureItem(LPMEASUREITEMSTRUCT lpMeasureItemStruct);
    //}}AFX_VIRTUAL
protected:
    // font - drawing members
    int m_nMRUCount;
    CImageList m_imagelist;
    CTypedPtrMap<CMapStringToPtr, CString, PFCBDATA> m_mapFont;
    CTypedPtrMap<CMapStringToPtr, CString, PFCBDATA> m_mapFontMRU;
    // font - drawing functions
    static BOOL CALLBACK AFX_EXPORT EnumScreenFontCallbackFn(ENUMLOGFONTEX * pelf,
NEWTEXTMETRICEX * /*lpntm*/, int FontType, LPVOID pThis);
    BOOL EnumFont();
    void SetFontMRU(const CString& strFont);
    // flat - drawing members
    BOOL m_bLBtnDown;
    // flat - drawing functions
    void DrawCombo(DWORD dwStyle, COLORREF clrToLeft, COLORREF clrBottomRight);
    int Offset()
    {
    return ::GetSystemMetrics(SM_CXHTHUMB);
    };
//{{AFX_MSG(CFontComboBox)
afx_msg void OnCloseup();
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnNcDestroy();
    afx_msg void OnPaint();
    afx_msg void OnTimer(UINT nIDEvent);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
#endif // !defined(_FNTCOMBO_H__INCLUDED_)
```

实现文件

```
#include "stdafx.h"
#include "FntCombo.h"
#include "resource.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define FCB_IMAGEWIDTH 15
////////////////////////////////////////
// CFontComboBox
BEGIN_MESSAGE_MAP(CFontComboBox, CComboBox)
    //{{AFX_MSG_MAP(CFontComboBox)
    ON_CONTROL_REFLECT(CBN_CLOSEUP, OnCloseup)
    ON_WM_NCDESTROY()
    ON_WM_MOUSEMOVE()
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONUP()
    ON_WM_TIMER()
    ON_WM_PAINT()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////
// CFontComboBox message handlers
int CFontComboBox::CompareItem(LPCOMPAREITEMSTRUCT lpCompareItemStruct)
{2
    // return -1 = item 1 sorts before item
    // return 0 = item 1 and item 2 sort the same
    // return 1 = item 1 sorts after item 2
/*      CString str1, str2;
    GetLBText(lpCompareItemStruct->itemID1, str1);
    GetLBText(lpCompareItemStruct->itemID2, str2);
    if ( str1 < str2 )
    {
    return -1;
    }
    else if ( str1 == str2 )
    {
    return 0;
```

```
    }
    else
    {
    return 1;
    }
*/
    return 0;
}
void CFontComboBox::DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct)
{
ASSERT(lpDrawItemStruct->CtlType==ODT_COMBOBOX);
    // for easy use
CDC *pDC=CDC::FromHandle(lpDrawItemStruct->hDC);
    ASSERT(pDC); // attaching failed
    CRect rect(lpDrawItemStruct->rcItem);
    // draw focus rectangle
    if (lpDrawItemStruct->itemState & ODS_FOCUS)
        pDC->DrawFocusRect(rect);
    // save the context attributes
    int nIndexDC = pDC->SaveDC();
    CBrush brush;
    // draw the selection state
    if (lpDrawItemStruct->itemState & ODS_SELECTED)
    {
brush.CreateSolidBrush(::GetSysColor(COLOR_HIGHLIGHT));
pDC->SetTextColor (::GetSysColor (COLOR_HIGHLIGHTTEXT));
    }
    else
    {
    brush.CreateSolidBrush(pDC->GetBkColor());
    }
    pDC->SetBkMode(TRANSPARENT);
    pDC->FillRect(rect, &brush);
    CString strCurFont;
    GetLBText(lpDrawItemStruct->itemID, strCurFont);
    m_imagelist.Draw(pDC, 0, CPoint(rect.left, rect.top),
ILD_TRANSPARENT);
    int nX = rect.left; // x pos for MRU separater lines
    rect.left += FCB_IMAGEWIDTH + 2;
    // create font object
    LOGFONT lf;
    ::ZeroMemory(&lf, sizeof(lf));
    strcpy(lf.lfFaceName, strCurFont);
    lf.lfCharSet = DEFAULT_CHARSET;
    CFont font;
    font.CreateFontIndirect(&lf);
    // if it is a symbol font,
    // draw the font's facename using default font
    FCBDATA *pFCBData = NULL;
    if ( m_mapFont.Lookup(strCurFont, pFCBData)
&& pFCBData->lf.lfCharSet == SYMBOL_CHARSET )
    {
    pDC->TextOut(rect.left, rect.top + 3, strCurFont);
    SIZE size = pDC->GetTextExtent(strCurFont);
    rect.left += size.cx;
    }
    // now draw the text using created font,
    CFont *pOldFont = pDC->SelectObject(&font);
    pDC->TextOut(rect.left, rect.top, strCurFont);
    // and clean up
    pDC->SelectObject(pOldFont);
    font.DeleteObject();
    // draw font MRU separator, a line like ——
  if (lpDrawItemStruct->itemID==(UINT)m_mapFontMRU.
GetCount() - 1 )
    {
        pDC->MoveTo(nX, rect.bottom - 3);
        pDC->LineTo(rect.right, rect.bottom - 3);
        pDC->MoveTo(nX, rect.bottom - 1);
        pDC->LineTo(rect.right, rect.bottom -1);
    }
    // restore state of DC
    pDC->RestoreDC(nIndexDC);
}
BOOL CFontComboBox::EnumFont()
{
    LOGFONT lf;
HDC hDC=::GetWindowDC(NULL); //DC for entire screen
    ZeroMemory(&lf, sizeof(lf));
    lf.lfCharSet = DEFAULT_CHARSET;
    if (!EnumFontFamiliesEx(
            hDC,      // handle to device context
            &lf, // pointer to logical font information
    (FONTENUMPROC)EnumScreenFontCallbackFn, // pointer to callback function
        (LPARAM) this,      // application-supplied data
        (DWORD) 0))
    return FALSE;
    ::ReleaseDC(NULL, hDC);
    return TRUE;
}
BOOL CALLBACK AFX_EXPORT CFontComboBox::EnumScreenFontCallbackFn(ENUMLOGFONTEX* pelf, NEWTEXTMETRICEX* /*lpntm*/, int FontType, LPVOID pThis)
{
    if (FontType & RASTER_FONTTYPE)
    {
        return 1;
    }
    LPCTSTR lpszFace = pelf->elfLogFont.lfFaceName;
    FCBDATA * pFCBData = NULL;
    if ( ! strchr(lpszFace, '@') && ! ((CFontComboBox*)
pThis)->m_mapFont.Lookup(lpszFace, pFCBData))
    {
        pFCBData = new FCBDATA;
        pFCBData->flag = 0;
        pFCBData->lf = pelf->elfLogFont;
        ((CFontComboBox * ) pThis)->m_mapFont.SetAt
(lpszFace, pFCBData);
    }
    return 1; // call me back next time
}
void CFontComboBox::Initialize(int nMRUCount /* =10 */)
```

```
{
    FCBDATA * pFCBData = NULL;
    CString strKey, strComp;
// enumerate all fonts in system, excluding raster fonts
    EnumFont();
    ResetContent();
    int cxMax = 0;
    CDC * pDC = GetDC();
    CFont font;
    POSITION pos = m_mapFont.GetStartPosition();
    while (pos)
    {
    m_mapFont.GetNextAssoc(pos, strKey, pFCBData);
        // calculate width to set
    if ( pFCBData->lf.lfCharSet == SYMBOL_CHARSET)
        {
        pFCBData->lf.lfHeight = 0;
        pFCBData->lf.lfWidth = 0;
        font.CreateFontIndirect(&pFCBData->lf);
    CFont * pOldFont = pDC->SelectObject(&font);
    SIZE size = pDC->GetTextExtent(pFCBData->
lf.lfFaceName, strlen(pFCBData->lf.lfFaceName));
        pDC->SelectObject(pOldFont);
        font.DeleteObject();
        SIZE size2 = pDC->GetTextExtent(pFCBData->
lf.lfFaceName, strlen(pFCBData->lf.lfFaceName));
        if ( cxMax < size.cx + size2.cx)
        {
        cxMax = size.cx + size2.cx;
        }
        }
        else
        {
   SIZE size=pDC->GetTextExtent(pFCBData->lf.lfFace-
Name, strlen(pFCBData->lf.lfFaceName));
    cxMax = size.cx > cxMax ? size.cx : cxMax;
        }
    AddString(strKey);
    }
    // set dropped down width to show all the characters
    SetDroppedWidth(cxMax);
    m_nMRUCount = nMRUCount;
m_imagelist.Create(IDB_FONTTYPE, 15, 1, RGB(255, 0,
255));
}
void CFontComboBox::MeasureItem ( LPMEASUREITEM-
STRUCT lpMeasureItemStruct)
{
lpMeasureItemStruct->itemHeight = 20;
}
void CFontComboBox::OnCloseup()
{
    int nSel = GetCurSel();
    if (nSel != CB_ERR)
    {
    CString strFont;
    GetLBText(nSel, strFont);
    SetFontMRU(strFont);
    }
})
void CFontComboBox::OnNcDestroy(
{
CComboBox::OnNcDestroy();
m_imagelist.DeleteImageList();
// clean up work to do,
// avoiding memory leaks
FCBDATA * pFCBData;
CString strKey;
POSITION pos = m_mapFont.GetStartPosition();
  while (pos)
  {
m_mapFont.GetNextAssoc(pos, strKey, pFCBData);
  delete pFCBData;
}
}
void CFontComboBox::PreSubclassWindow()
{
CComboBox::PreSubclassWindow();
UINT style = GetWindowLong(m_hWnd, GWL_STYLE);
style |= CBS_SORT|CBS_OWNERDRAWVARIABLE;
SetWindowLong(m_hWnd, GWL_STYLE, style);
//    SetWindowPos(NULL, 0, 0, 0, 0, SWP_NOSIZE |
SWP_NOMOVE|SWP_NOZORDER);
//    ModifyStyle(0, CBS_SORT|CBS_OWNERDRAWVAR-
IABLE);
}
void CFontComboBox::SetFontMRU(const CString & str-
Font)
{
  FCBDATA * pFCBData = NULL;
  CString strFontDesc;
      // try to find the font, if not, exit function
  if ( m_mapFont.Lookup(strFont, pFCBData) )
  {
  if ( m_mapFontMRU.Lookup(strFont, pFCBData) )
  {
  int nSel = FindStringExact(-1, strFont );
  ASSERT(nSel != CB_ERR);
  if ( nSel != 0 )
  {
      DeleteString(nSel);
      int index = InsertString(0, strFont);
      ASSERT( index != CB_ERR );
      SetCurSel(index);
      }
      }
      else // not in MRU(s)
      {
  int nSel = FindStringExact(-1, strFont);
  if ( m_mapFontMRU.GetCount() >= m_nMRUCount )
      {
CString str;
GetLBText(m_nMRUCount - 1, str);
DeleteString(m_nMRUCount - 1);
m_mapFontMRU.RemoveKey(str);
  }
```

```
    if ( nSel ! = CB_ERR )
        {
        int index = InsertString(0, strFont);
        ASSERT(index ! = CB_ERR);
        SetCurSel(index);
        pFCBData – >flag | = FCB_INMRU;
        m_mapFontMRU. SetAt(strFont, pFCBData);
        }
        }
    } // lookup in all fonts
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * NOTE that the following funcitons is for the purpose of flat – drawing
 * it could be removed if the flat effect is not desired
\* * * * * * * * * * * * * * * * * * * * * * * * * * * *
void CFontComboBox: : DrawCombo(DWORD dwStyle, COLORREF clrTopLeft, COLORREF clrBottomRight)
{
    CRect rcItem;
    GetClientRect(& rcItem);
    CDC * pDC = GetDC();
        // Cover up dark 3D shadow.
pDC – >Draw3dRect(rcItem, clrTopLeft, clrBottomRight);
    rcItem. DeflateRect(1, 1);
        if (! IsWindowEnabled())
    {
pDC – >Draw3dRect ( rcItem, : : GetSysColor ( COLOR_BTNHIGHLIGHT), : : GetSysColor(COLOR_BTNHIGHLIGHT));
    }
    else
    {
pDC – >Draw3dRect( rcItem, : : GetSysColor ( COLOR_BTNFACE), : : GetSysColor(COLOR_BTNFACE));
    }
    // Cover up dark 3D shadow on drop arrow.
    rcItem. DeflateRect(1, 1);
    rcItem. left = rcItem. right – Offset();
  pDC – >Draw3dRect(rcItem, : : GetSysColor(COLOR_BTNFACE), : : GetSysColor(COLOR_BTNFACE));
        // Cover up normal 3D shadow on drop arrow.
    rcItem. DeflateRect(1, 1);
  pDC – >Draw3dRect(rcItem, : : GetSysColor(COLOR_BTNFACE), : : GetSysColor(COLOR_BTNFACE));
        if (! IsWindowEnabled())
    {
        return;
    }
    switch (dwStyle)
    {
    case FCB_DRAWNORMAL:
    rcItem. top – = 1;
    rcItem. bottom + = 1;
pDC – >Draw3dRect( rcItem, : : GetSysColor ( COLOR_BTNHIGHLIGHT), : : GetSysColor(COLOR_BTNHIGHLIGHT));
        rcItem. left – = 1;
    pDC – >Draw3dRect(rcItem, : : GetSysColor (COLOR_BTN-
HIGHLIGHT), : : GetSysColor(COLOR_BTNHIGHLIGHT));
        break;
  case FCB_DRAWRAISED:
        rcItem. top – = 1;
        rcItem. bottom + = 1;
  pDC – >Draw3dRect(rcItem, : : GetSysColor(COLOR_BTN-
HIGHLIGHT), : : GetSysColor(COLOR_BTNSHADOW));
        break;
  case FCB_DRAWPRESSD:
        rcItem. top – = 1;
        rcItem. bottom + = 1;
        rcItem. OffsetRect(1, 1);
  pDC – >Draw3dRect(rcItem, : : GetSysColor(COLOR_BTN-
SHADOW), : : GetSysColor(COLOR_BTNHIGHLIGHT));
  break;
  }
  ReleaseDC(pDC);
}
void CFontComboBox: : OnLButtonDown(UINT nFlags, CPoint point)
{
  m_bLBtnDown = TRUE;
  CComboBox: : OnLButtonDown(nFlags, point);
}
void CFontComboBox: : OnLButtonUp(UINT nFlags, CPoint point)
{
  m_bLBtnDown = FALSE;
  Invalidate();
  CComboBox: : OnLButtonUp(nFlags, point);
}
void CFontComboBox: : OnMouseMove(UINT nFlags, CPoint point)
{
  SetTimer(1, 10, NULL);
  CComboBox: : OnMouseMove(nFlags, point);
}
void CFontComboBox: : OnPaint()
{
  Default();
DrawCombo(FCB_DRAWNORMAL, : : GetSysColor(COLOR_BTNFACE), : : GetSysColor(COLOR_BTNFACE));
}
void CFontComboBox: : OnTimer(UINT nIDEvent)
{
  POINT pt;
  GetCursorPos(& pt);
  CRect rcItem;
  GetWindowRect(& rcItem);
  static bool bPainted = FALSE;
  // OnLButtonDown, show pressed.
  if (m_bLBtnDown)
  {
        KillTimer (1);
        if (bPainted)
        {
 DrawCombo(FCB_DRAWPRESSD, : : GetSysColor(COLOR_BTNSHADOW), : : GetSysColor(COLOR_BTNHIGHLIGHT));
```

```
            bPainted = FALSE;
            }
            return;
        }
        // If mouse leaves, show flat.
        if (!rcItem.PtInRect(pt))
        {
        KillTimer (1);
            if (bPainted)
            {
DrawCombo(FCB_DRAWNORMAL, ::GetSysColor(COLOR_
BTNFACE), ::GetSysColor(COLOR_BTNFACE));
        bPainted = FALSE;
        }
        return;
        }
        // On mouse over, show raised.
        else
        {
if (bPainted)
        return;
        else
        {
        bPainted = TRUE;
DrawCombo ( FCB_DRAWRAISED, ::GetSysColor(COLOR_
BTNSHADOW), ::GetSysColor(COLOR_BTNHIGHLIGHT));
        }
}
        CComboBox::OnTimer(nIDEvent);
}
```

要使用这个类，首先要包含它的头文件，并为准备一幅位图，ID 为 IDB_FONTTYPE，大小为 15 点宽，12 点高，使用 magenta 为透明色，这幅位图将画在所有字体的左侧。可以使用 ClassWizard 为组合框指定 CFontComboBox 类型的成员变量，例如名为 m_fontCombo，并在 OnInitDialog 函数中加入一行 m_fontCombo.Initialize 4

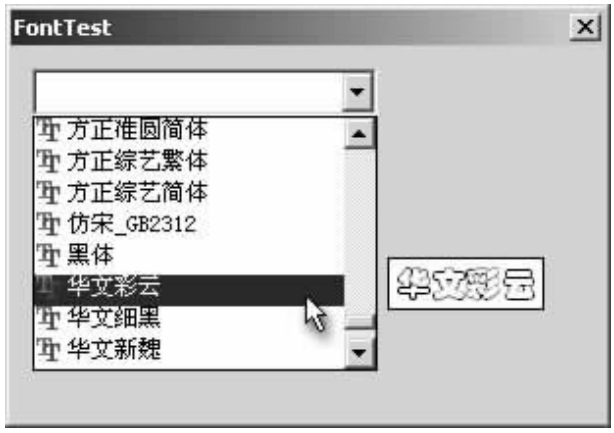
其中，参数 4 表示最近使用的字体列表最多有 4 项。

下面是一个使用 CComboBox 的 MFC 工程的例子，这个例子是手式编写的。这个文件与前面的两个文件构成了一个完整的工程的所有代码文件。

```
#include "stdafx.h"
#include "resource.h"
#include "fntcombo.h"
struct CDemoApp : public CWinApp
{
        virtual BOOL InitInstance();
} theApp;
struct CDemoDlg : public CDialog
{
        CDemoDlg(CWnd* pParent = NULL);
        enum { IDD = IDD_DIALOG };
        CFontComboBox       m_fontCombo;
protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        virtual BOOL OnInitDialog();
};
CDemoDlg::CDemoDlg(CWnd* pParent /* =NULL */)
    : CDialog(CDemoDlg::IDD, pParent)
{
}
void CDemoDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_FONT, m_fontCombo);
}
BOOL CDemoDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_fontCombo.Initialize(4);
    m_fontCombo.SetCurSel(0);
    return TRUE;
}
BOOL CDemoApp::InitInstance()
{
    CDemoDlg dlg;
    dlg.DoModal();
    return FALSE;
}
```

实现的效果可参见文首的插图。最后要说的是，这样的列表框在第一次下拉时的速度有一些慢，因为要将每个字体都创建一次。在我的 PIII733、256MB 内存和 SCSI RAID5 的机器上，仍然有一定的滞留感。但是，熊掌和鱼不可得兼，Office 2000 的字体组合框在下拉时同样也很慢，也许这就是为什么在 Office 2000 中才将此功能引入的原因。

如果既要得到所见即所得的效果，又要求很高的速度，可以在组合框的旁边显示一个窗口，在这个窗口上用选中的字体绘制出当前的字体名称。如图所示：



另一种方法，显示提示窗口 （无滞留感）

测试环境：Windows2000 Professional Service Pack 1 Visual C++ 6.0 Service Pack 4。

收稿日期：2001 年 2 月 19 日）

建立一个完善的 SystemTray 类

楚广明

在参看了前几期的《计算机维护与编程》时 发现了几篇关于系统任务栏编程的例程 但在实际工作时 我发现了这几篇文章所提供的类库有着很多的问题

1. 类库设计不合理，不利于类库的重用性。

2. 功能单一，有许多功能并没有实现。如动态图标功能、修改提示等。

总结以上两点，我参照在新闻组上的有关讨论与我实际工作中的实践，重写了有关的类库 在重写过程中得到了许多不知名的朋友的帮助，如果没有他们的帮助，我想大家也得不到这么棒的类库 ，实现了有关动静态图标转换等功能，并较好地考虑了其代码的重用性与扩充性。当然本人设计的类库也许也会存在一些问题，希望大家看到后能提出，由于在上几期都有关于任务栏编程的基本原理，我在这里就不再花很多的篇幅去介绍它了。

```
    SystemTray.h : header file
//////////////////////////////////
// SystemTray.h : header file
//
// Written by ChangGuangMing (chu888@cmmail.com)
//////////////////////////////////
#ifndef _INCLUDED_SYSTEMTRAY_H_
#define _INCLUDED_SYSTEMTRAY_H_
#include <afxtempl.h>//由于使用了 CArray 类,所以加入 afxtempl.h 模板库的支持
//////////////////////////////////////
// CSystemTray window
class CSystemTray : public CWnd
{
// 构造函数与构析函数
public:
    CSystemTray();
    CSystemTray(CWnd * pWnd, UINT uCallbackMessage,
LPCTSTR szTip, HICON icon, UINT uID);
      //UINT uCallbackMessage 为回调消息
      //LPCTSTR szTip 为提示字符串
      //HICON icon 为静态图标
      //UINT nID 为菜单 ID 号
    virtual ~CSystemTray();
    DECLARE_DYNAMIC(CSystemTray)
// Operations
public:
    BOOL Enabled() { return m_bEnabled; } }
    BOOL Visible() { return !m_bHidden;
    // 建立任务栏图标
      //其中大部分参数同上
     Create(CWnd * pParent, UINT uCallbackMessage,
LPCTSTR szTip, HICON icon, UINT uID);
    // 重新设置或提取提示字符串
    BOOL    SetTooltipText(LPCTSTR pszTooltipText);
    BOOL    SetTooltipText(UINT nID);
    CString GetTooltipText() const;
    // 重新设置或图标
    BOOL  SetIcon(HICON hIcon);
    BOOL  SetIcon(LPCTSTR lpszIconName);
    BOOL  SetIcon(UINT nIDResource);
    BOOL  SetStandardIcon(LPCTSTR lpIconName);
    BOOL  SetStandardIcon(UINT nIDResource);
    HICON GetIcon() const;
    void  HideIcon();
    void  ShowIcon();
    void  RemoveIcon();
    void  MoveToRight();
    // 用于动态图标
BOOL  SetIconList(UINT uFirstIconID, UINT uLastIconID);
BOOL  SetIconList(HICON * pHIconList, UINT nNumIcons);
BOOL  Animate(UINT nDelayMilliSeconds, int nNumSeconds
 = -1);
    BOOL  StepAnimation();
    BOOL  StopAnimation();
    // 设置或提取菜单的默认索引
void GetMenuDefaultItem(UINT& uItem, BOOL& bByPos);
BOOL SetMenuDefaultItem(UINT uItem, BOOL bByPos);
    // 设置或提取任务栏自定义消息
    BOOL  SetNotificationWnd(CWnd * pNotifyWnd);
    CWnd * GetNotificationWnd() const;
    // 默认的任务栏响应消息
virtual LRESULT OnTrayNotification(WPARAM uID, LPARAM
lEvent);
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSystemTray)
  protected:
  virtual LRESULT WindowProc(UINT message, WPARAM
wParam, LPARAM lParam);
  //}}AFX_VIRTUAL
// Implementation
protected:
    void Initialise();
    BOOL          m_bEnabled;   // 是否支持任务栏图标
    BOOL          m_bHidden;    // 是否隐藏任务栏图标
    NOTIFYICONDATA  m_tnd;
    CArray<HICON, HICON> m_IconList;
    static UINT  m_nIDEvent;
    UINT         m_uIDTimer;
    int          m_nCurrentIcon;
```

```
    COleDateTime m_StartTime;
    int          m_nAnimationPeriod;
    HICON        m_hSavedIcon;
    UINT         m_DefaultMenuItemID;
    BOOL         m_DefaultMenuItemByPos;
// Generated message map functions
protected:
      //{{AFX_MSG(CSystemTray)
      afx_msg void OnTimer(UINT nIDEvent);
      //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
#endif
////////////////////////////////////
// SystemTray.cpp : implementation file
////////////////////////////////////
  #include "stdafx.h"
#include "SystemTray.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
IMPLEMENT_DYNAMIC(CSystemTray, CWnd)
UINT CSystemTray::m_nIDEvent = 4567;
////////////////////////////////////
// CSystemTray 构造与构析函数
CSystemTray::CSystemTray()
{
    Initialise();
}
CSystemTray::CSystemTray(CWnd* pParent, UINT uCallbackMessage, LPCTSTR szToolTip, HICON icon, UINT uID)
{
    Initialise(); //初始化函数
Create(pParent, uCallbackMessage, szToolTip, icon, uID);
}
void CSystemTray::Initialise()
{
    memset(&m_tnd, 0, sizeof(m_tnd)); // 采用 ANSI 标准函数 memset 清空
        //NOTIFYICONDATA 数据结构.
      /*
      typedef struct _NOTIFYICONDATA {
    DWORD cbSize;
    HWND hWnd;
    UINT uID;
    UINT uFlags;
    UINT uCallbackMessage;
    HICON hIcon;
    char szTip[64];
    } NOTIFYICONDATA, *PNOTIFYICONDATA;
      */
    m_bEnabled    = FALSE;
    m_bHidden     = FALSE;
    m_uIDTimer    = 0;
    m_hSavedIcon  = NULL;
    m_DefaultMenuItemID  = 0;
    m_DefaultMenuItemByPos  = TRUE;
}
BOOL CSystemTray::Create(CWnd* pParent, UINT uCallbackMessage, LPCTSTR szToolTip, HICON icon, UINT uID)
{
    // 调用 WINDOWS SDK 函数 GetVersion()来检查 Windows 的版本
   /*
   Platform     High order bit Low order byte (主版本号)
    Windows NT   zero    3 or 4
    Windows 95 and Windows 98  1  4
    Win32s with Windows 3.1     1       3
    */
    VERIFY(m_bEnabled = ( GetVersion() & 0xff ) >= 4);
    if (!m_bEnabled) return FALSE;
  // 避免和其他消息冲突, WM_USER 以前的消息为 windows 内部使用
  //WM_USER 在 WINUSER.H 中定义
   //#define WM_USER                 0x0400
   ASSERT(uCallbackMessage >= WM_USER);
    // 任务栏提示仅支持 64 个字符之内. 任何大于 64 个字符的定义都是非法的.
    ASSERT(_tcslen(szToolTip) <= 64);
    // 建立一个不可见窗体
     CWnd::CreateEx(0, AfxRegisterWndClass(0), _T(""), WS_POPUP, 0, 0, 10, 10, NULL, 0);
    // 设置 NOTIFYICONDATA 结构
    m_tnd.cbSize = sizeof(NOTIFYICONDATA);
    m_tnd.hWnd   = pParent->GetSafeHwnd()? pParent->GetSafeHwnd() : m_hWnd;
    m_tnd.uID    = uID;
    m_tnd.hIcon  = icon;
    m_tnd.uFlags = NIF_MESSAGE | NIF_ICON | NIF_TIP;
    m_tnd.uCallbackMessage = uCallbackMessage;
    _tcscpy(m_tnd.szTip, szToolTip);
    // 设置任务栏图标
VERIFY(m_bEnabled=Shell_NotifyIcon(NIM_ADD, &m_tnd));
    return m_bEnabled;
}
CSystemTray::~CSystemTray()
{
    RemoveIcon();
    m_IconList.RemoveAll();
    DestroyWindow();
}
////////////////////////////////////
// CSystemTray 的任务栏图标操作处理
void CSystemTray::MoveToRight()
{
    HideIcon();
    ShowIcon();
}
void CSystemTray::RemoveIcon()
{
    if (!m_bEnabled) return; //如果可见标以为假那么返回
    m_tnd.uFlags = 0; //清空 uFlags 结构
```

```
    Shell_NotifyIcon(NIM_DELETE, & m_tnd);
      //删除任务栏图标
      /*
      NIM_ADD  加入图标
    NIM_DELETE  删除图标
    NIM_MODIFY  修改图标
    以下二个操作也用到了 Shell_Notify()这个 SDK 函数,详细见 MSDN
       */
    m_bEnabled = FALSE; //设置可见标为假
}
void CSystemTray::HideIcon()
{
    if (m_bEnabled && !m_bHidden) {
        m_tnd.uFlags = NIF_ICON;
        Shell_NotifyIcon (NIM_DELETE, & m_tnd);
        m_bHidden = TRUE;
    }
}
void CSystemTray::ShowIcon()
{
    if (m_bEnabled && m_bHidden) {
        m_tnd.uFlags = NIF_MESSAGE | NIF_ICON | NIF_TIP;
        Shell_NotifyIcon(NIM_ADD, & m_tnd);
        m_bHidden = FALSE;
    }
}
//充分利用 C++语言的优势,实现从资源文件或本地原文件载入图标
BOOL CSystemTray::SetIcon(HICON hIcon)
{
    if (!m_bEnabled) return FALSE;
    m_tnd.uFlags = NIF_ICON;
    m_tnd.hIcon = hIcon;
    return Shell_NotifyIcon(NIM_MODIFY, & m_tnd);
}
BOOL CSystemTray::SetIcon(LPCTSTR lpszIconName)
{
HICON hIcon = AfxGetApp()->LoadIcon(lpszIconName);
return SetIcon(hIcon);
}
BOOL CSystemTray::SetIcon(UINT nIDResource)
{
    HICON hIcon = AfxGetApp()->LoadIcon(nIDResource);
    return SetIcon(hIcon);
}
BOOL CSystemTray::SetStandardIcon(LPCTSTR lpIcon-
Name)
{
    HICON hIcon = LoadIcon(NULL, lpIconName);
    return SetIcon(hIcon);
}
BOOL CSystemTray::SetStandardIcon(UINT nIDResource)
{
     HICON hIcon = LoadIcon(NULL, MAKEINTRESOURCE
(nIDResource));
    return SetIcon(hIcon);
}t
HICON CSystemTray::GetIcon() cons
{
    return (m_bEnabled)? m_tnd.hIcon : NULL;
}
//在任务栏的动画图标是通过任意个静态的图标组成的
//在这里只需要设置出第一与最后一个就可以了.
BOOL CSystemTray::SetIconList(UINT uFirstIconID, UINT
uLastIconID)
{
   if (uFirstIconID > uLastIconID)
        return FALSE;
   UINT uIconArraySize = uLastIconID - uFirstIconID + 1;
   const CWinApp * pApp = AfxGetApp();
    ASSERT(pApp != 0);
    m_IconList.RemoveAll(); // 清空 m_IconList 变量,这是
CArray 变量
       //在这里有不少关于 CArray 类的知识
          //在这里我就不一一解答了,详细见 MSDN
    try {
      for (UINT i = uFirstIconID; i <= uLastIconID; i++)
//通过一个循环把所有静态图标,读入到 m_IconList 中.
      m_IconList.Add(pApp->LoadIcon(i));
    }
    catch (CMemoryException * e)
    {
        e->ReportError();
        e->Delete();
        m_IconList.RemoveAll();
        return FALSE;
    }
    return TRUE;
}
BOOL CSystemTray::SetIconList(HICON * pHIconList, UINT
nNumIcons)
{
    m_IconList.RemoveAll();
    try {
      for (UINT i = 0; i <= nNumIcons; i++)
            m_IconList.Add(pHIconList[i]);
    }
    catch (CMemoryException * e)
    {
        e->ReportError();
        e->Delete();
        m_IconList.RemoveAll();
        return FALSE;
    }
    return TRUE;
}
BOOL CSystemTray::Animate(UINT nDelayMilliSeconds, int
nNumSeconds /* = -1 */)
{
    StopAnimation();
     m_nCurrentIcon = 0;
    m_StartTime = COleDateTime::GetCurrentTime(); //返
回当前的时间
```

```
    m_nAnimationPeriod = nNumSeconds;
    m_hSavedIcon = GetIcon();
      // 设置动画时间处理
m_uIDTimer = SetTimer(m_nIDEvent, nDelayMilliSeconds,
NULL);
    return (m_uIDTimer ! = 0);
})
BOOL CSystemTray: : StepAnimation(
{
    if (!m_IconList. GetSize())
        return FALSE;
    m_nCurrentIcon + +;
    if (m_nCurrentIcon >= m_IconList. GetSize())
        m_nCurrentIcon = 0;
    return SetIcon(m_IconList[m_nCurrentIcon]);
})
BOOL CSystemTray: : StopAnimation(
{
    BOOL bResult = FALSE;
    if (m_uIDTimer)
          bResult = KillTimer(m_uIDTimer);
    m_uIDTimer = 0;
    if (m_hSavedIcon)
        SetIcon(m_hSavedIcon);
    m_hSavedIcon = NULL;
    return bResult;
}
////////////////////////////////
// CSystemTray 提示符处理
BOOL CSystemTray: : SetTooltipText(LPCTSTR pszTip)
{
    if (!m_bEnabled) return FALSE;
    m_tnd. uFlags = NIF_TIP;
    _tcscpy(m_tnd. szTip, pszTip); //_tcscpy 其实就是 strcpy,
由于使用了 UniCode 所以用了它 .
    return Shell_NotifyIcon(NIM_MODIFY, & m_tnd);
}
BOOL CSystemTray: : SetTooltipText(UINT nID)
{;
    CString strText;
    VERIFY(strText. LoadString(nID))
    return SetTooltipText(strText);
}
CString CSystemTray: : GetTooltipText() const
{
    CString strText;
    if (m_bEnabled)
        strText = m_tnd. szTip;

    return strText;
}
////////////////////////////////
// CSystemTray notification window stuff
BOOL CSystemTray: : SetNotificationWnd(CWnd * pWnd)
{
    if (!m_bEnabled) return FALSE;
ASSERT(pWnd& & : : IsWindow(pWnd - >GetSafeHwnd()));
    m_tnd. hWnd = pWnd - >GetSafeHwnd();
    m_tnd. uFlags = 0;
    return Shell_NotifyIcon(NIM_MODIFY, & m_tnd);
}
CWnd * CSystemTray: : GetNotificationWnd() const
{
    return CWnd: : FromHandle(m_tnd. hWnd);
}
////////////////////////////////
// CSystemTray 菜单处理
BOOL CSystemTray: : SetMenuDefaultItem(UINT uItem,
BOOL bByPos)
{
     if ((m_DefaultMenuItemID = = uItem) & & (m_Default-
MenuItemByPos = = bByPos))
        return TRUE;
    m_DefaultMenuItemID = uItem;
    m_DefaultMenuItemByPos = bByPos;
    CMenu menu, * pSubMenu;
    if (!menu. LoadMenu(m_tnd. uID)) return FALSE;
if (!(pSubMenu = menu. GetSubMenu(0))) return FALSE;
 : : SetMenuDefaultItem(pSubMenu - >m_hMenu, m_Default-
MenuItemID, m_DefaultMenuItemByPos);
    return TRUE;
}
void CSystemTray: : GetMenuDefaultItem(UINT & uItem,
BOOL& bByPos)
{
    uItem = m_DefaultMenuItemID;
    bByPos = m_DefaultMenuItemByPos;
}
////////////////////////////////
// CSystemTray 消息句柄处理
BEGIN_MESSAGE_MAP(CSystemTray, CWnd)
  //{{AFX_MSG_MAP(CSystemTray)
  ON_WM_TIMER()
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
void CSystemTray: : OnTimer(UINT nIDEvent)
{
    ASSERT(nIDEvent = = m_nIDEvent);
     COleDateTime CurrentTime = COleDateTime: : GetCur-
rentTime();
COleDateTimeSpan period = CurrentTime - m_StartTime;
    if (m_nAnimationPeriod > 0 & & m_nAnimationPeriod <
period. GetTotalSeconds())
    {
        StopAnimation();
        return;
    }
    StepAnimation();
}
LRESULT CSystemTray: : OnTrayNotification(UINT wParam,
LONG lParam)
{
    //如果不是针对此图标操作, 返回 0L;
    if (wParam ! = m_tnd. uID)
```

```
    return 0L;
  CMenu menu, *pSubMenu;
  CWnd* pTarget = AfxGetMainWnd();
//如果发现产生消息为 WM_RBUTTONUP, 那么载入上下文菜单
  if (LOWORD(lParam) == WM_RBUTTONUP)
  {
    if (!menu.LoadMenu(m_tnd.uID)) return 0;
    if (!(pSubMenu = menu.GetSubMenu(0))) return 0;
    // 设置默认菜单
      ::  SetMenuDefaultItem(pSubMenu->m_hMenu,
m_DefaultMenuItemID, m_DefaultMenuItemByPos);
    // 显示 POP 菜单
    CPoint pos;
    GetCursorPos(&pos);
    pTarget->SetForegroundWindow();
   ::TrackPopupMenu(pSubMenu->m_hMenu, 0, pos.x,
pos.y, 0, pTarget->GetSafeHwnd(), NULL);
    pTarget->PostMessage(WM_NULL, 0, 0);
    menu.DestroyMenu();
  }
  else if (LOWORD(lParam) == WM_LBUTTONDBLCLK)
  {
    pTarget->SetForegroundWindow();
   UINT uItem;
    if (m_DefaultMenuItemByPos)
    {
      if (!menu.LoadMenu(m_tnd.uID)) return 0;
    if (!(pSubMenu = menu.GetSubMenu(0))) return 0;
uItem =pSubMenu->GetMenutemIDm_DefaultMenuItemID);
    }
    else
      uItem = m_DefaultMenuItemID;
   pTarget->SendMessage(WM_COMMAND, uItem, 0);
    menu.DestroyMenu();
  }
  return 1;
}
LRESULT CSystemTray::WindowProc(UINT message, WPA-
RAM wParam, LPARAM lParam)
{
  if (message == m_tnd.uCallbackMessage)
    return OnTrayNotification(wParam, lParam);
 return CWnd::WindowProc(message, wParam, lParam);
}
```

结束语

对任务栏进行编程是现在比较流行的一种设计方式 这种方法在大量的软件中得以使用 如 Oicq 等 通进这种手法可以使程序简洁而独特 减少对桌面空间的占用 你也可以在这个类库的基础上做些改进，希望它对您能有一点用处。

收稿日期：2001 年 2 月 21 日）

（上接第 10 页）

```
FAppName);
 Result:= (GetLastError = ERROR_ALREADY_EXISTS);
end;
```

四、 任务条状态区图标的编程

开机日记程序在任务条状态区设立一个图标 给用户以必要的提示信息 并且定义一个回调消息 Callback Message 。当把鼠标箭头停留在这个图标上时 系统会显示提示信息 比如 “开机 8 12” 点击鼠标 系统会向应用程序发送用户定义的回调消息 由此消息的长参数 lParam 可以确定按键状态 字参数 wParam 确定是哪个图标 如果有多个图标的话 。任务条状态区的编程属于 Shell 编程的一部分 主要包括图标、提示和消息三个方面 通过调用 Shell_NotifyIcon 函数完成。这个函数的定义为

```
WINSHELLAPI BOOL WINAPI Shell_NotifyIcon(
 DWORD dwMessage,          // 信息标志
 PNOTIFYICONDATA pnid);  // 结构指针
```

其中 dwMessage 参数有三种取值 这些取值及含义是

NIM_ADD	往状态区添加一个图标
NIM_DELETE	删除状态区的一个图标
NIM_MODIFY	更改状态区的一个图标

pnid 是一个指向 NOTIFYICONDATA 结构的指针 其内容因 dwMessage 而异。结构定义为

```
typedef struct _NOTIFYICONDATA {
  DWORD cbSize;  //结构所占的字节数
  HWND hWnd;     //窗口句柄
  UINT uID;      //图标识别标志
  UINT uFlags;   //
  UINT uCallbackMessage; //自定义的消息
  HICON hIcon;   //图标句柄
  char szTip[64]; //提示字符
} NOTIFYICONDATA, *PNOTIFYICONDATA;
```

其中 uFlags 可以以下三种取值的组合 or

NIF_ICON	表示 hIcon 有效。
NIF_MESSAGE	表示 uCallbackMessage 有效。
NIF_TIP	表示 szTip 有效。

为了实现回调消息的处理 由 uCallbackMessage 定义所要处理的消息 该消息的长参数 lParam 包含 Win32 定义的鼠标消息 用以确定按键状态 左键、右键 单击、双击等 字参数 wParam 表示图标识别标志。

五、 记录阅读器

为了使用户能够看到开机日记程序所记录的信息 必须提供一个阅读工具。这里记录信息是以文本文件的形式保存的因而用普通的文本编辑器即可 但为了防止对记录文件无意的修改 最好另编写一个文本阅读程序 这里就不详细讨论了。

附程序源代码 TinyApp. pas

收稿日期：2001 年 1 月 8 日）

Powerbuilder 中数据管道的应用

刘　辉　叶念渝

摘　要　本文以一个使用数据管道转移数据的程序为例子，介绍如何使用 Powerbuilder 的数据管道。

关键字　Powerbuilder，数据管道

一、引言

在设计数据库系统时经常需要先建立一个功能不是太完善的原型系统让用户试用或者从一个大型的数据库中提取一部分数据建立一个小型的测试数据库进行测试工作。这时就需要在不同的数据库间转移数据。庆幸的是 Powerbuilder 提供了数据管道工具，使得数据的转移可以自动的完成。

数据管道通常在以下情况下是极为有用的：

① 在不同数据库间转移数据；

② 远程用户从数据库服务器下载数据；

③ 将各自事务表集中起来提交给一个主事务表；

④ 从一个大型数据库中抽取数据，建立一个小型的测试数据库。

本文以一个完全转移一个数据库表的例子　当然也可以只转移部分数据）来说明如何在应用程序中使用数据管道。这个例子将数据库中 rc_baseinfo 表中的内容完全转移到 rc_baseinfo_copy　该表原来并不存在）中，数据转移的过程中显示从源表读出、向目的表写入、出错数据的行数等信息，转移结束后显示新表中的所有数据，若有出错数据则可以修正它们。另外数据转移过程中可以终止转移，但终止前已转移的数据将存在于新表中。

二、程序中所需的对象

在使用数据管道的程序中应创建 3 个对象：一个管道对象、一个窗口、一个管道用户对象。管道对象负责数据的转移，窗口负责与用户的交互，管道用户对象负责管道的初始化、启动、状态显示、修正错误、关闭管道等操作。其中管道对象和窗口的定义如图 1、图 2 所示。

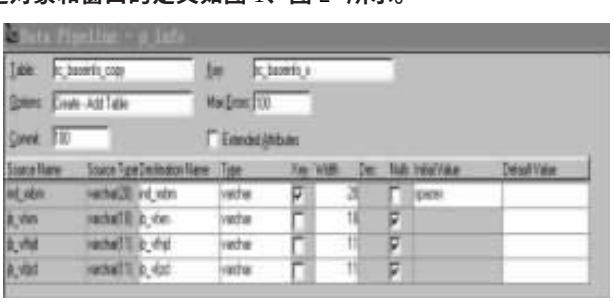


图 1　管道对象 p_info 的定义

图 2　窗口 w_piptest 的定义

其中窗口中的主要控件如下表所示：

表 1　窗口中的主要控件

控　件	功　能
Cb_1	开始转移
Cb_2	取消转移
Cb_3	纠错
Dw_1	错误行信息
Dw_2	已转移信息
St _read	已读行数
St_write	已写行数
St_error	出错行数

管道用户对象是完成数据转移的核心。管道的启动、修正错误、终止管道等操作是通过管道用户对象的 Start、Repair、Cancel 等函数实现；数据转移的过程中显示的从源表读出、向目的表写入、出错数据的行数等信息是通过管道用户对象的 RowsRead、RowsWritten、RowsInError 等特性来获得的。管道用户对象可以从标准的 Pipeline 对象继承，即在 User Object 画板中选择 New 在 New User Object 对话框选择 Class / Standard 在 Select Standard Class 中选择 Pipeline 即可建立一个管道用户对象，取名为 uo_ptest。

三、程序的实现

1. 定义实例变量

定义管道用户对象 uo_ptest 的实例变量：

```
transaction itr_s, itr_d        //定义两个事务变量
```

```
datawindow idw_error      //定义错误显示数据窗口的实例
statictext ist_read, ist_write, ist_error  //定义显示读、写、错误行数信息的文本实例
```

定义窗口 w_piptest 的实例变量

```
transaction itr_source, itr_dest
uo_ptest iu_pipline
```

注意，必须同时定义两个事务变量，分别连接源数据库和目的数据库　即使源数据库和目的数据库为同一个库也必须这样做）。

2. 连接数据库

窗口 w_piptest 的 Open 事件：

```
setpointer(hourglass!)
iu_pipline = create uo_ptest          //创建管道用户对象的实例
itr_source = create transaction
itr_dest = create transaction
if wf_connect() <>0 then
   close(this)
end if
this.postevent(´_ue_init´)            //触发初始化
```

其中窗口函数 wf_connect　实现与源数据库和目的数据库的连接，代码如下：

```
//连接源数据库
itr_source.DBMS = "ODBC"
itr_source.AutoCommit = False
itr_source.DBParm = "Connectstring = ´DSN = piptest; uid = dba; pwd = sql´"& +", CommitOnDisconnect = ´No´"
connect using itr_source;
if itr_source.sqlcode < 0 then
messagebox("连接源数据库失败", itr_source.sqlerrtext, stopsign!)
        return -1
end if
//连接目的数据库
itr_dest.DBMS = "ODBC"
itr_dest.AutoCommit = False
itr_dest.DBParm = "Connectstring = ´DSN = piptest;
uid = dba; pwd = sql´"& +", CommitOnDisconnect = ´No´"
connect using itr_dest;
if itr_dest.sqlcode < 0 then
messagebox("连接目的数据库失败", itr_dest.sqlerrtext, stopsign!)
  return -1
end if
return 0
```

3. 在窗口 w_piptest 的用户定义事件 _ue_init 中初始化管道用户对象

```
iu_pipline.uf_init(itr_source, itr_dest, dw_1)
iu_pipline.ist_read = st_read
iu_pipline.ist_write = st_write
iu_pipline.ist_error = st_error
```

而初始化管道用户对象所调用的初始化函数 uf_init　在管道用户对象 uo_ptest 中定义如下：

```
itr_s = arg_s
itr_d = arg_d
idw_error = arg_error
```

其中 arg_s、 arg_d、 arg_error 为 uf_init　的形参。

4. 管道操作

在窗口 w_piptest 的开始按钮的 Clicked 事件添加如下代码：

```
int li_rtn
iu_pipline.dataobject = ´p_info´
cb_2.enabled = true                   //使能取消按钮
li_rtn = iu_pipline.uf_start()
if li_rtn <>1 then
iu_pipline.uf_error(li_rtn)
cb_3.enabled = true                   //使能纠错按钮
end if
```

其中管道用户对象的 uf_start　定义如下：

```
return this.start(itr_s, itr_d, idw_error)
```

管道用户对象的 uf_error　分别定义如下：

```
string ls_msg
choose case code                      //code 为 uf_error()的形参
  case -1
       ls_msg = ´管道打开错误´
  case -2
       ls_msg = ´列太多´
  case -3
       ls_msg = ´表已存在´
  case -4
       ls_msg = ´表不存在´
         .
         .
  end choose                          //其他错误代码略
if code <>1 then
  messagebox(´管道错误´, ls_msg, stopsign!)
else
messagebox(´管道成功´, ´操作完成´)
end if
```

在数据转移过程中按取消按钮可终止该过程，取消按钮的 Clicked 事件代码如下：

```
if iu_pipline.uf_cancel() <0 then
    messagebox(´取消´, ´取消失败´)
end if
```

而其中的 uf_cancel　用户事件定义如下：

```
return this.cancel
```

5．数据转移过程中的状态显示

在管道用户对象 uo_ptest 的 Pipemeter 事件中添加如下代码：

```
ist_error.text = string(this.rowsinerror)      //错误行
ist_read.text = string(this.rowsread)          //已读行
ist_write.text = string(this.rowswritten)      //已写行
```

6. 纠错

按钮纠错的 Clicked 事件代码：

```
  int li_rtn
li_rtn = iu_pipline.uf_fix()                   //修正错误行
if li_rtn <>1 then
   iu_pipline.uf_error(li_rtn)
end if
```

其中管道用户对象 uo_ptest 的 uf_fix()定义如下:

```
return this.repair(itr_d)
```

7．显示刚转移过来的数据

在管道用户对象 uo_ptest 的 Pipeend 事件中添加如下代码：

```
//动态建立数据窗口以显示新建表的数据
string ls_sqlstr, ls_style, ls_errmsg, ls_syntax, ls_errbuf
ls_sqlstr = ´ select ind_xxbm, jb_vfxm, jb_vfhjd, jb_vfjzd from rc_baseinfo_copy´
ls_style = ″style(type = grid)″
ls_syntax = itr_d. syntaxfromsql(ls_sqlstr, ls_style, ls_errmsg)
if len(ls_errmsg)>0 then
messagebox( ´ syntaxfromsql 函数错误´, ls_errmsg, stopsign! )
else
   w_piptest. dw_2. create(ls_syntax, ls_errbuf)
      if len(ls_errbuf)>0 then
messagebox(´create 函数错误´, ls_errbuf, stopsign! )
      end if
end if
w_piptest. dw_2. settransobject(itr_d)
w_piptest. dw_2. retrieve( )
```

8. 关闭窗口并断开与数据源的连接

在窗口 w_piptest 的 Closequery 事件的代码：

```
if wf_disconnect( ) = -1 then
        message. returnvalue = 1
end if
```

其中窗口函数 wf_disconnect　代码如下：

```
//断开与目的数据库的连接
disconnect using itr_dest;
if itr_dest. sqlcode <>0 then
messagebox(´数据库错误´, ´无法断开和目的数据库的连接´, stopsign! )
                return  -1
end if
//断开与源数据库的连接
disconnect using itr_source;
if itr_source. sqlcode <>0 then
messagebox(´数据库错误´, ´无法断开和源数据库的连接´, stopsign! )
       return  -1
end if
destroy itr_dest
destroy itr_source
destroy iu_pipline
return 0
```

程序运行成功后的窗口如图 3 所示。

以上就是利用数据管道转移数据的示例的全部代码。以上程序在 Powerbuilder6. 5、Windows98 环境下调试通过。以上程序仅供参考。

参考文献

1. 王蓉等编．Powerbuilder 应用开发技术详解．电子工业出版社，1999

2. 吴洁明．Powerbuilder6. 0 应用与开发．清华大学出版社，1999

收稿日期：2001 年 3 月 7 日）

图 3　程序运行结果

浅析桌面精灵的实现

杨 华

一、软件的开发目的

想必大家对桌面精灵很熟悉吧，笔者为了取得美眉的喜欢，引出了开发本软件的目的。如果读者有我同样的需求，那么请继续看下去，我将和你共同探讨这个问题。注意以下示例代码均用 DELPHI 描述。

二、实现原理

其实桌面精灵的原理很简单，主要分以下几步：

1. 获取桌面窗口的 HDC。

API 定义如下：

GetDC 函数用于获取指定窗口的图形设备描述表

```
HDC GetDC(
HWND hWnd   // 窗口句柄
);
```

例如

```
DeskTopDC: HDC; //定义桌面窗口的图形设备描述表句柄
DeskTopDC: =GetDC(0);
```

或者 DeskTopDC =GetDC GetDesktopWindow

2. 创建一个内存位图，把桌面中将要绘图的区域，保存到内存位图中去，以便绘图完成时恢复桌面。为此我定义了一个函数：

```
procedure savebackground(BKCanvas :TCanvas; //内存位图的画布对象
                         sp_w integer  //要保存区域的宽度
                         sp_h integer  //要保存区域的高度
                         nx integer  //要保存区域的 X 坐标
                         ny integer  //要保存区域的 Y 坐标
```

3. 将动画对象透明地拷贝到桌面的绘图区域，笔者用了一个 GDIAPI 函数方便地实现了此功能。

定义如下：

```
BOOL TransparentBlt(HDC hdcDest, //目标图形设备描述表句柄
 int nXOriginDest, //绘图矩形的 X 坐标
 int nYOriginDest, //绘图矩形的 Y 坐标
 int nWidthDest, //绘图矩形的宽度
 int hHeightDest, //绘图矩形的高度
 HDC hdcSrc, //源图形设备描述表句柄
 int nXOriginSrc, //源绘图矩形的 X 坐标
 int nYOriginSrc, //源绘图矩形的 Y 坐标
 int nWidthSrc, //源绘图矩形的宽度
 int nHeightSrc, //源绘图矩形的高度
UINT crTransparent//设置透明色 RGB(r, g, b)
);
```

注意：

Windows NT 需要 5. 0 或以上版本。

Windows 需要 Windows 98 或 以上版本。

其它低版本不支持。

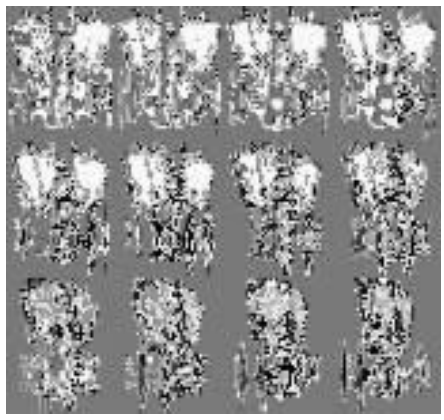
此函数包含在 msimg32. dll 中。

笔者定义了一个 tranbit 函数来动态调用 TransparentBlt 函数，具体定义见第三节。

4. 将第二步生成的内存位图拷贝到桌面。这样一帧动画就显示完成。不断循环 1 - 4 步，你就能看到连续的动画场景了。

三、具体代码

以下是一个演示程序，在 Delphi5. 0 + Windows2000P 中调试通过。创建一个窗体 Form1 放上两个 Image 控件 命名为 Image1、Image2，再放上一个 Timer 控件，命名为 Timer1。准备两张位图，一张放入 Image1 另一张放入 Image2。笔者用了如下样式的位图 截取了一部分），你可以自己画动画对象，也可以借用别人的，笔者就是用微软画的图片。



从图片可以看出，图片中包括了许多连续的动画帧，一张图片完成一个动作，如旋转一周等，每帧动画大小完全一样，除了动画对象其它像素用一种透明色填充。代码如下：

```
unit Unitmain;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls, StdCtrls, mmsystem;
type
  TForm1 = class(TForm)
    Timer1: TTimer; //爆炸定时器
```

```
    Image1: TImage; //储存爆炸的图片
    Image2: TImage; //储存飞行器的图片
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
     procedure FormClose(Sender: TObject; var Action:
TCloseAction);
  private
    { Private declarations }
    DeskTopDC: HDC; //桌面窗口的图形设备描述表句柄
    stop: boolean; //控制循环的变量
    expnum: integer; //爆炸的当前次数
    procedure Explode(X: integer; Y: integer); //爆炸函数
    procedure shipmove(X: integer; Y: integer); //飞行器函数
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
//保存桌面背景
procedure savebackground(BKCanvas : TCanvas;
                         sp_w: integer;
                         sp_h : integer ;
                         nx: integer;
                         ny: integer);
var sc: TCanvas;
begin
  sc: =TCanvas.Create;
  try
    sc.Handle: =GetDC(0);
    bkcanvas.CopyRect( rect(0, 0, sp_w, sp_h), sc, rect(nx,
ny, nx+sp_w, ny+sp_h));
    ReleaseDC(0, sc.handle);
  finally
    sc.free;
  end;
end;
//透明拷贝图像函数
//静态调用 API 函数 TransparentBlt
procedure tranbit(hdcDest: HDC;
                  nXOriginDest: integer;
                  nYOriginDest: integer;
                  nWidthDest: integer;
                  hHeightDest: integer;
                  hdcSrc: HDC;
                  nXOriginSrc: integer;
                  nYOriginSrc: integer;
                  nWidthSrc: integer;
                  nHeightSrc: integer;
                  crTransparent: UINT) ;
Var
    LibHandle: HWND; //动态连接库句柄
      //函数原型定义
    DllName: Procedure(hdcDest: HDC;
                  nXOriginDest: integer;
                  nYOriginDest: integer;
                  nWidthDest: integer;
                  hHeightDest: integer;
                  hdcSrc: HDC;
                  nXOriginSrc: integer;
                  nYOriginSrc: integer;
                  nWidthSrc: integer;
                  nHeightSrc: integer;
                  crTransparent: UINT); Stdcall;
begin
   //以下是静态调用 dll 中函数的例行公事
    LibHandle: =LoadLibrary(´msimg32.dll´);
    if LibHandle <32 then
    begin
MessageBox(Form1.Handle, ´Not Found msimg32.dll´, ´Er-
ror´, 0);
       Exit;
    end;
@DllName: =GetProcAddress(LibHandle, ´TransparentBlt´);
    if @DllName =nil then
    begin
       MessageBox(Form1.Handle, ´Not Found Transparent-
Blt in msimg32.dll´, ´Error´, 0);
       FreeLibrary(LibHandle);
       Exit;
    end;
    try
       TransparentBlt(hdcDest, nXOriginDest, nYOriginDest,
nWidthDest, hHeightDest, hdcSrc, nXOriginSrc, nYOriginSrc,
nWidthSrc, nHeightSrc, crTransparent);
    finally
       FreeLibrary(LibHandle);
    end;
end;
//爆炸函数
//在桌面的 X, Y 坐标处发生爆炸
procedure TForm1.Explode(X: integer; Y: integer);
var
   BitMapB : TBitMap; //保存桌面指定区域的内存位图
    w: integer; //一帧动画的宽度
    h: integer; //一帧动画的高度
    i: integer;
    j: integer;
begin
   BitMapB: =TBitMap.Create;
    try
       //动画位图为 4*5=20 帧
      w: =Image1.Width div 4; //计算每帧的宽度
      h: =image1.Height div 5; //计算每帧的高度
       //初始化内存为图的大小
      BitMapB.Height : =h;
      BitMapB.Width : =w;
       //保存桌面上指定区域的位图
       //注意, 由于爆炸是在同一位置完成的, 所以只要保存
爆炸区域一次就行了
      savebackground(BitMapB.canvas, w, h, X, Y);
       for i: =0 to 4 do
       begin
       for j: =0 to 3 do
       begin
```

```
        //把相应帧画到桌面上
tranbit(DeskTopDC , x, y, w, h, image1. Canvas. Handle, j * w,
i * h, w, h, RGB(208, 2, 178));
Sleep(20); //显示速度太快, 停顿 20 毫秒
      //恢复桌面
bitblt(DeskTopDC, X, Y, w, h, BitMapB. Canvas. handle, 0, 0,
srccopy);
            end;
        end;
    finally
        BitMapB. Free;
    end;
end;
//飞行器的飞行函数
//参数 x, y 指定飞行器飞行的目的地
procedure TForm1. shipmove(X: integer; Y: integer);
var
    w: integer;
    h: integer;
    i: integer;
    j: integer;
    k: integer;
    l: integer;
      BitMapB : TBitMap;
begin
    Randomize();
      BitMapB: =TBitMap. Create;
    try
        //动画位图为 4 * 16 -3 帧空帧 =61 帧
            w: =Image2. Width div 4;
        h: =image2. Height div 16;
            BitMapB. Height : =h;
        BitMapB. Width : =w;
        k: =0;
        l: =0;
        while not stop do
            for i: =0 to 15 do
                for j: =0 to 3 do
                begin
      If (i =15) and (i>0) then break; //如果是空帧就不画了
      //保存桌面上指定区域的位图
//注意, 由于飞行是在不同位置完成的, 所以要保存即将被绘
图的桌面区域
        savebackground(BitMapB. canvas, w, h, k, l);
       tranbit(DeskTopDC , k, l, w, h, image2. Canvas. Handle,
j * w, i * h, w, h, RGB(208, 2, 178));
        sleep(10);
bitblt(DeskTopDC, k, l, w, h, BitMapB. Canvas. handle, 0, 0, sr-
ccopy);
if(k <x)then k: =k +1;
if(l <y)then l: =l +1;
if timer1. Enabled =false then
if(k>x -10)then//到达目的地就停止飞行, 并引爆炸弹
                begin
                    stop: =true;
                    timer1. Enabled : =true; //炸弹引爆器
                end;
        end;
    finally
        BitMapB. Free;
    end;
end;
procedure TForm1. Timer1Timer(Sender: TObject);
var
    x, y: integer;
begin
  if(expnum = 0) then
    begin
Explode(screen. width div 2 -20, screen. Height div 2 -20);
        sndPlaySound(´explosion. wav´, SND_NOSTOP);
        expnum: =expnum +1;
    end
    else if expnum <10 then//爆炸最多 10 次
    begin
          //产生随机位置
        x: =Random(screen. Width -100);
        y: =Random(Screen. Height -100);
        Explode(x, y); //爆炸
sndPlaySound(´explosion. wav´, SND_NOSTOP); //播放爆炸
声音
        expnum: =expnum +1;
    end
    else
    begin
        stop: =true;
        timer1. Enabled : =false;
        close();
    end;
end;
procedure TForm1. FormCreate(Sender: TObject);
begin
    DeskTopDC: =GetDC(0);
    chdir(ExtractFilePath(application. ExeName));
    stop: =false;
    expnum: =0;
   //飞行器开始飞行, 目的地为屏幕中央
    self. shipmove(screen. width div 2, screen. Height div 2);
end;
procedure TForm1. FormClose(Sender: TObject; var Action:
TCloseAction);
begin
    stop: =true;
    Timer1. Enabled : =false;
    ReleaseDC(0, DeskTopDC);
end;
end.
```

四、结束语

正如我们所希望的那样：一个飞行器飞入桌面，慢慢向屏幕中央靠近，当它到达目的地时就爆炸了，并引出一连串的爆炸。程序顺利地完成了我们的希望，但是程序还有许多不足，最好用 directx 来完成动画，这样效果可能会更好。 （E_MAIL CODEHUNTER@SOHU. COM）

收稿日期：2001 年 3 月 29 日）

怎样屏蔽系统热键

张云潮

摘 要 本文讨论了在 Win9x，Win NT Win NT + sp3 和 Win2000 等不同的操作系统下屏蔽系统热键的不同方法。详细介绍了底层系统钩子的原理应用方法。另外还介绍了注册热键的方法。

关键词 底层系统钩子 系统热键

在程序开发过程中，为了达到一些特殊的要求，有时需要屏蔽系统热键。这些系统热键包括 Alt + Tab、Ctrl + Alt + Del、Alt + Esc、Ctrl + Esc、Win 键等。由于屏蔽系统热键涉及到系统的安全，所以微软并不推荐这样做，我们只能在一些未公开的文档中去探究它。下面我们分三种情况来讨论屏蔽系统热键的方法。

一、 在 Win95 / Win98 操作系统下

在 Win95 / Win98 操作系统下屏蔽系统热键可以用一种非常简单的方法做到，就是让系统认为当前处于屏保状态。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，系统无法弹出任务列表，一旦某一进程被挂起，只能重新启动，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。

当程序要取消该状态：

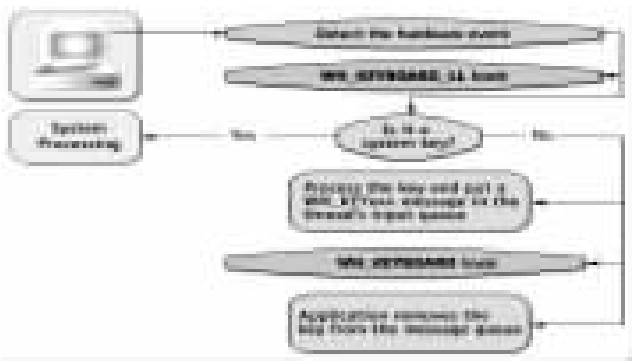
SystemParametersInfo SPI_SETSCREENSAVERRUNNING false & pOld SPIF_UPDATEINIFILE

pOld 为我们定义的布尔变量。

二、 在 WinNT 4. 0 Service Pack 3 或 Win2000 下

也许你认为使用一个键盘钩子 WH_KEYBOARD 不就解决问题了吗？问题并不是那么简单。这是因为键盘钩子 WH_KEYBOARD 并不能截取到系统键的输入！幸好在 Winnt4. 0 Service Pack3 或 Win2000 下系统给我们提供了一个底层系统钩子 （Low Level Hook ） WH_KEYBOARD_LL。WH_KEYBOARD_LL 的工作流程如右图所示。

从右图我们可以看到，底层键盘钩子存在于用户敲击键盘和系统处理之间，而普通键盘钩子则存在于系统产生 WM_KEYxxx 之后。很清楚，普通键盘钩子只能截获 WM_KEYxxx 消息，而不能对系统键进行操作。但是底层键盘钩子有一个致命的弱点，就是如果调用它的进程或线程出现死循环，系统将不能处理任何键盘操作。为了解决这个问



题，微软在注册表中给出了一个底层键盘钩子处理的限制时间，如果超出了这个时间，系统将正常处理。这个时间的键值在注册表的 HKEY_CURRENT_USER hControl Panel hDesktop h LowLevelHooksTimeout 下给出。

下面我们来讨论底层键盘钩子的用法。首先需要安装钩子，这需要 API 函数：

```
HHOOK SetWindowsHookEx(
  int iHookCode,
  HOOKPROC lpfn,
  HINSTANCE hModule,
  DWORD dwThreadId
);
```

其中，第一个参数是钩子的类型；第二个参数是钩子函数的地址；第三个参数是包含钩子函数的模块句柄；第四个参数指定监视的线程。如果指定确定的线程，即为线程专用钩子；如果指定为空，即为全局钩子。其中，全局钩子函数必须包含在 DLL 动态链接库）中，而线程专用钩子还可以包含在可执行文件中。得到控制权的钩子函数在完成对消息的处理后，如果想要该消息继续传递，那么它必须调用另外一个 SDK 中的 API 函数 CallNextHookEx 来传递它。钩子函数也可以通过直接返回 TRUE 来丢弃该消息，并阻止该消息的传递。

下面是实现底层键盘钩子的完整源代码，在 VC5. 0，VC6. 0 下调试通过。

```
#include <Windows. h>
LRESULT CALLBACK LowLevelKeyboardProc(int nCode,
WPARAM wParam, LPARAM lParam) {
```

```
    BOOL fEatKeystroke = FALSE;
  I  f (nCode = = HC_ACTION) {
    switch (wParam) {
    case WM_KEYDOWN:  case WM_SYSKEYDOWN:
    case WM_KEYUP:    case WM_SYSKEYUP:
PKBDLLHOOKSTRUCT p =(PKBDLLHOOKSTRUCT)lParam;
      fEatKeystroke = ((p->vkCode = = VK_TAB) && ((p->flags & LLKHF_ALTDOWN) ! = 0)) ||((p->vkCode = = VK_ESCAPE) && ((p->flags & LLKHF_ALTDOWN) ! = 0)) || ((p->vkCode = = VK_ESCAPE) && ((GetKeyState(VK_CONTROL) & 0x8000) ! = 0));
      break;
    }
  }
  return(fEatKeystroke ? 1 : CallNextHookEx(NULL, nCode, wParam, lParam));
}
////////////////////////////////////
int WINAPI WinMain(HINSTANCE hinstExe, HINSTANCE, PTSTR pszCmdLine, int) {
  //按装底层键盘钩子
    HHOOK hhkLowLevelKybd = SetWindowsHookEx(WH_KEYBOARD_LL, LowLevelKeyboardProc, hinstExe, 0);
   MessageBox(NULL, TEXT(" Alt + Esc, Ctrl + Esc, and Alt + Tab are now disabled. \n")
TEXT("Click \"Ok\" to terminate this application and re - enable these keys. "),
    TEXT("Disable Low - Level Keys"), MB_OK);
  UnhookWindowsHookEx(hhkLowLevelKybd);
  return(0);
}
```

三、 在低于 WinNT 4.0 Service Pack 2 或更低版本的 NT 操作系统下

很遗憾，在低于 WinNT 4.0 Service Pack 2 或更低版本的 NT 操作系统下我们所能做的并不能屏蔽掉所有的系统热键，而仅仅可以屏蔽 Alt + Tab Alt + Esc。

完成这一功能的方法是使用 RegisterHotKey 函数。在调用该函数后你的进程会在 Alt + Tab 按下时比系统先得到通知。你需要处理的消息是 WM_HOTKEY，下面是相关代码讲解。

RegisterHotKey 函数原型及说明：

```
BOOL RegisterHotKey(
  HWND hWnd,   // window to receive hot - key notification
  int id,      // identifier of hot key
  UINT fsModifiers,   // key - modifier flags
  UINT vk      // virtual - key code);
```

参数 id 为你自己定义的一个 ID 值，对一个线程来讲其值必需在 0x0000 - 0xBFFF 范围之内，对 DLL 来讲其值必需在 0xC000 - 0xFFFF 范围之内，在同一进程内该值必须唯一。

参数 fsModifiers 指明与热键联合使用按键，可取值为：MOD_ALT MOD_CONTROL MOD_WIN MOD_SHIFT。

参数 vk 指明热键的虚拟键码 。

```
// 初始化
CMainFrame: : CMainFrame()
{
 m_nHotKeyID = 100;
 BOOL m_isKeyRegistered = RegisterHotKey(GetSafeHwnd(), m_nHotKeyID, MOD_ALT, VK_TAB);
 ASSERT(m_isKeyRegistered ! = FALSE);
}
//取消
CMainFrame: : ~CMainFrame()
{
  BOOL m_iskeyUnregistered = UnregisterHotKey(GetSafeHwnd(), m_nHotKeyID);
 ASSERT(m_isKeyUnregistered ! = FALSE);
}
```

WM_HOTKEY 消息含义:

```
idHotKey = (int) wParam;          // identifier of hot key
fuModifiers = (UINT) LOWORD(lParam);  //key - modifier flags
uVirtKey = (UINT) HIWORD(lParam); // virtual - key code
```

这三个值分别和调用 RegisterHotKey 时的 id fsModifiers vk 对应。

最后需要在文件中定义 ON_MESSAGE 消息映射。

在头文件中：

```
class CMainFrame : public XXXX
{
  afx_msg LONG OnHotKey(WPARAM wP, LPARAM lP);
}
```

在 CPP 文件中 MESSAGE_MAP 处添加：

```
ON_MESSAGE  WM_HOTKEY  OnHotKey
```

该热键在你的进程运行时一直有效，在进程结束后其状态会被系统恢复。

通过对以上三种情况的分析，我们基本上涵盖了屏蔽系统热键的所有方法，有什么问题请于 zhangyunchao@citiz.net 联系。

收稿日期：2001 年 4 月 9 日）

金仕达多媒体成为 Lisa 会员

总部位于瑞士的 Lisa 组织是专门服务于本地化、国际软件项目交流、信息化服务的国际认证机构，在信息领域内享有很高的声誉。Lisa 组织对会员资格有严格的评判机制，通过认证的公司在技术、质量体系、品牌形象、服务保障方面有良好的基础。

金仕达多媒体日前顺利通过了 Lisa 认证，正式成为 Lisa 会员，这标志着金仕达多媒体的公司运作水平又登上了一个新台阶。并且，作为 Lisa 组织成员，金仕达多媒体将有机会获得更多的国际业务，为业绩的成长提供强有力的支持。

也谈开发硬件中断虚拟驱动程序

张雄飞 刘 平

摘 要 本文介绍了使用 VtoolsD 工具开发硬件中断虚拟驱动程序中的一些常见的问题，如动态虚拟中断号、VxD 与应用程序通讯、调试 VxD、实时控制等，并提出了解决办法。

关键词 虚拟驱动程序 VxD 硬件中断

一、 引言

中断，是 CPU 与计算机外部设备进行通讯的有效手段。例如数据采集卡，ADC 转换结束，就发一个“转换结束”的硬件中断信号，要求 CPU 读取数据。硬件中断编程，在 DOS 操作系统下并不复杂。目前，微机普遍由 DOS 系统转向了 Win98 系统，由于操作系统接管了硬件资源，使得一般的应用程序不能直接操作硬件资源，因此，开发 Win98 环境下的虚拟设备驱动程序 Virtual Device Driver：VxD 成了许多外部设备（如数据采集卡）开发中的主要工作之一，具有重要意义。

使用 VToolsD 软件包，结合 Visual C++，用 C/C++语言可以高效快速地开发 VxD，编程者根据 VtoolsD 提供的范例，能够很快掌握实现硬件中断 VxD 的一般方法。所以本文不准备讨论 Win98 中断处理机制和编写硬件中断 VxD 的一般方法，需要的同志请看 VtoolsD 自带的帮助或者参考文献 1。笔者关注于 VxD 编程工作中遇到的一些实际问题。

二、 VxD 与应用程序间通讯

由于开发硬件中断 VxD 的目的在于控制硬件和提供硬件和应用程序的接口，所以实现 VxD 与应用程序之间的通讯成了 VxD 开发中的关键。

1. 从 Win32 程序与 VxD 通讯

一般的 Win32 应用程序工作在 ring3 级，而 VxD 作为操作系统的扩展工作于 ring0 级，要实现 ring3 级到 ring0 级的通讯，首先必须获得所要控制的 VxD 句柄，这可以使用一个 Win32 API 函数 CreateFile。CreateFile 用于获取一个静态或动态 VxD 的句柄。对于动态 VxD 而言，CreateFile 首先会装载它（如果没有被装载的话），如果装载成功，操作系统将会发出 SYS_DYNAMIC_DEVICE_INT 消息，通知 VxD 进行动态初始化，而后获得它的句柄。

Win32 API 函数 DeviceIOControl 是 Win32 程序与 VxD 通讯的关键。任何时候，在 Win32 程序中调用 DeviceIOControl，相应的 VxD 就会收到系统发来的 W32DEVICEIOCONTROL 消息，VtoolsD 在生成的框架中用 OnW32DeviceControl 函数响应这个消息：OnW32DeviceControl 函数只有一个 IOCTRPARAMS 结构的参数，该结构中的 dioc_IOCtrlCode 成员对应于 DeviceIOControl 函数中的 dwIoControlCode 参数（Win32 程序向 VxD 传递的消息号）；编程者通过重载这个函数，使 VxD 根据不同的消息号作出相应的响应，就可以实现 Win32 App 到 VxD 的通讯。

下面是一个例子，Win32 App 通过 VxD 获得 ADC 转换的数据：

在 Win32 App 中：

```
#define PASS_DATA (0x8000 + 1) //定义数据传输的消息号
……
hVxD = CreateFile( "\\\\. \\adc. vxd", 0, 0, 0, CREATE_NEW,
FILE_FLAG_DELETE_ON_CLOSE, 0); //动态加载 VxD 并获取 VxD 句柄
DeviceIOControl( hVxD, PASS_DATA, (LPVOID) ibuf, sizeof (ibuf), (LPVOID) obuf, sizeof(obuf), & cbReturned, NULL);
//与 adc. vxd 的事件过程 OnW32DeviceIoControl 交换数据
//ibuf 是 Win32 程序传递给 VxD 的数据缓冲地址，如果不需要可以给 NULL
//obuf 是 VxD 返回数据所存放的缓冲地址
//VxD 传回的实际数据总量放在 cbReturned 中
CloseHandle(hVxD); //卸载 VxD
```

在 VxD 中：

```
#define PASS_DATA (0x8000 + 1)
DWORD ADCDevice : : OnW32DeviceIoControl ( PIOCTLPARAMS pDIOCParams)
{
  switch(pDIOCParams -> dioc_IOCtlCode)
  {
……
        case PASS_DATA:
memcpy(pDIOCParams -> dioc_OutBuf, adc_data, 4096);
//每次从 VxD 向 App 传输 4096 个(4k)字节
return 0;
        ……
}
return 0;
}
```

值得注意的是，为了避免和 VC 编译系统中一些已定义的变量发生冲突，VtoolsD 建议自定义的消息号大于 32768 0x8000 。

2. 从 VxD 与 Win32 程序通讯

从 VxD 与 Win32 程序通讯最简单的办法就是使用 SHELL_PostMessage 服务. SHELL_PostMessage 服务可以用来从 VxD 中向 Win32 应用程序发送消息，其定义为：

```
BOOL SHELL_PostMessage(HANDLE hWnd, DWORD uMsg, WORD wParam, DWORD lParam, PPostMessage_HANDLER pCallback, PVOID refData)
```

其中 hWnd 是处理该消息的 Win32 程序窗口句柄，在使用 SHELL 服务前必须将此消息处理窗口句柄传给 VxD 可以使用从 Win32 App 到 VxD 通讯的方法，即 DeviceIoControl 机制来传递 。uMsg 是自定义的消息号，pCallback refData 用于回调函数，如果不需要，全部设置为 NULL。

例如上例中 VxD 与 Win32 App 数据传输可以采取这样一种思路：每采集 4K 数据，在 VxD 里用 SHELL_postMessage 传递消息给 Win32 App，例如：

```
SHELL_PostMessage(appWnd, WM_BUFFER_FULL, 0, 0, 0, NULL);
```

//WM_BUFFER_FULL 是自定义消息, appWnd 是消息接受窗口句柄, Win32 App 相应的窗口接到消息后, 就可以用 DeviceIOControl 来取数

值得注意的是，SHELL_PostMessage 服务不能用于 Win32 控制台程序 Win32 Console Application ，也不能在中断处理时调用；另外，由于 SHELL_PostMessage 发送给应用程序的消息可能因处于线程的消息循环中造成一定的延时，所以建议创建一个专门的线程等待该消息。

三、 动态虚拟化中断号

VtoolsD 提供 VhardwareInt 硬件中断类处理硬件中断，该类构造函数的第一个参数就是要虚拟化的中断号，在许多示例中都是采用一个事先确定的中断号，但也可以将这个参数改为一个变量，由应用程序决定要虚拟哪一个未用的中断号。具体做法： 注意不在 OnSysDynamicDeviceInit 里建立中断类实例，应该使用 DeviceIoControl 机制发个自定义消息给 VxD，通知要虚拟的中断号，并建立中断类实例。

```
DWORD ADCDevice::OnW32DeviceIoControl ( PIOCTLPARAMS pDIOCParams)
{
    switch(pDIOCParams->dioc_IOCtlCode)
    {
case INITIRQ:  //动态虚拟中断号消息
    IRQNumbear = *(int *) pDIOCParams->dioc_InBuf;
//获得要虚拟的中断号
pMyIRQ = new MyHwInt(); //建立硬件中断类实例
if(pMyIRQ && pMyIRQ->hook()) //挂钩中断号
{
    ……
}
……
```

四、 使用 SoftICE 调试 VxD

VxD 编好以后，一般都要经过调试。SoftICE 是一个核心调试器，是调试 VxD 的利器。SoftICE 甚至和 VtoolsD 集成到 NuMega DriverStudio 中一起发行，成为开发驱动程序的首选工具。一般资料上甚少介绍用 SoftICE 调试 VxD 的具体方法，本文把我们的经验介绍给大家。其实,使用 SoftICE 调试 VxD 只要遵循一定的顺序,一步一步地进行就可以了：

1. 用 VC 创建 VxD

2. 使用 SoftICE 的 Loader 加载 VxD 文件。分为两步 :第一步用 Open 打开一个 VxD 文件，注意，如果用 VC 生成 VxD，会同时生成一个 . PDB 文件，该文件包涵了 VxD 的源文件和调试信息，所以要确保 PDB 文件和 VxD 文件在同一目录下 或者在 Loader 可以搜寻到的目录下）；第二步点击 Load 按键，Loader 将自动把 . PDB 转换成 . NMS 调试符号文件，并加载调试符号文件和源文件。

3. 加载完成后，就可以开始调试 VxD 了。首先用 Ctrl + D 呼出 SoftICE，用 “file * ”看已经加载的源文件，用 “file + 文件名 ”选择要调试的文件，就可以看到到 VxD 的源代码。用 SRC 命令使显示在源代码和汇编代码间切换。

4. 使用 BPX 命令设置断点，对应代码高亮表示断点设置成功。VxD 运行到断点处会自动呼出 SoftICE，停到断点处。

5. 启动调用 VxD 的 Win32 应用程序。SoftICE 其它几个常用命令：BC 清除断点；O：往端口写一个字节；I：从端口读入一个字节；GENINT：产生一个中断；T：Trace 跟踪执行指令；G：执行程序；EXIT：退出。

五、 关于实时控制的问题

众所周知，Windows95 /98 系统并不是一个实时操作系统，它通过 VPICD Virtual Programmable Interrupt Controller Device 管理所有硬件中断事件。我们编写的硬件中断 VxD 从 VPICD 手中获得中断的处理权。整个 Windows 硬件中断响应过程比 DOS 复杂得多，Windows 的 ISR Interrtupt Service Routine 时间可以是 DOS 的 ISR 的 20 倍。不过通过编写合适的 VxD，我们还是可以获得比较满意的准实时效果的。有资料表明，一个中断 VxD 在 486 /66 计算机上能处理频率为 10KHz 的中断信息，不会漏掉一个中断。另外在处理硬件中断时，最好不要运行其它软件。

六、 结束语

以上技术，已在笔者编写的 ISA 多道脉冲幅度分析器虚拟驱动程序中实现。编译和调试工具：VtoolsD3. 0、 VC6. 0、SoftICE4. 05 for Win9X。

在 VxD 的编写和文章写作过程中参考了互联网上的许多资料，在此表示感谢！如有错误之处，请广大读者批评指正。

联系 Email vrbear@263. net 或者 zxf903@octopus. cdit. edu. cn

收稿日期：2001 年 3 月 13 日）

用 ATL 编写 COM 应用程序

李正平

摘　要　随着 Windows2000 的闪亮登场　COM + 已经和广大的 Windows 程序设计人员和广大的 Windows 用户见面。可以说,新世纪的 Windows 程序员,不懂得 COM 就很难跟上时代。COM + 为我们带来了新的开发 COM 应用程序的便利，但是要真正掌握基于 COM + 的 Windows 程序开发，就必须深入理解COM + 的技术基础——COM 及其开发。本文介绍了怎样用活动模板库（Active Template Library ——ATL）来开发低层的 COM 应用。

关键词　COM / COM + ,ATL 组件，接口

一、 COM 和 ATL 概述

COM （Component Object Model）是 Windows 操作系统的技术和核心，是搭建 Windows 应用程序的基础。COM 提供了二进制级别的代码复用机制。它提供了一种基于接口查询的机制来完成对象之间的通讯，以客户/服务器程序的体系结构协同完成特定的功能。COM 提供了以下的特性：

1. 同语言的无关性：二进制的设计标准

这使得基于 Windows 平台的开发工具可以混用 COM 组件，比如你可以用 C + + 编写组件，而在 Visual Basic 中使用该组件。

2. 软件版本升级的健壮性

COM 组件通过提供多接口机制支持软件版本升级的健壮性，COM 组件的不同功能用不同的接口向外展示。要为旧的组件添加新的功能时，为组件新加一个接口就行了。在这种情况下，允许旧的应用程序在不进行更改的情况下运行，同时新的应用程序可以通过查询新接口利用组件的新功能。

3. 面向对象的特性

COM 允许软件模块以面向对象的方式传递其功能。对于 C + + 程序员而言，面向对象的特征是最为常见的。COM 提供三种基本的面向对象的特征，即封装、继承和多态，并且 COM 是以一种语言无关的方式对这三种特征提供了支持。

4. 位置透明性

该特性意味着组件的用户并不需要明确的了解组件所处的位置，这个组件可能在一个 DLL 中（进程内服务）、可能在一个 EXE 中（进程外服务）也可能在一个远程计算机上运行（分布式组件对象）。

为什么要使用 ATL 呢？很简单，当 Microsoft 的开发者们最早提出 ATL 的时候，它们是为了实现这样一个目标：开发一种框架结构，使得通过它可以更加方便地建立小型的、快速的、基于 COM 的组件。ATL 的使用是为了给软件开发人员在实现组件的时候带来更大的灵活性，而不必依赖于任何的 DLL 文件（其中包括标准的 C 运行时 DLL）。现在，ATL 逐渐成为未来的 Windows 程序开发中 C + + 的框架。ATL 使用了 C + + 的新的基于模板的特征，并且 Microsoft 提供了 ATL 的一些源代码，使它成为 Visual C + + 开发环境的一部分。Visual C + + 的 Developer Studio IDE 也包含了一些 Visual C + + 的向导程序，它们使你在实现 ALT 时变的更加容易。

二、 用 ATL 开发一个 COM 组件

1. 工程的建立

1　打开 Visual C + + 6. 0 的开发环境，新建一工程，工程类型为 ATL COM AppWizard，工程名为 FileManager，单击 OK 进入下一步。

2　选择 Dynamic Link Library DLL 服务类型，其它保持缺省设置。单击 Finish 建立工程。

2. 第一版的组件设计及实现——FileExpert

我们这儿有一个需求，希望开发一个能够复用的 COM 组件，用于实现文件的操作，包括文件的创建，删除、拷贝、移动等。下面我们来设计这个组件：

1　接口设计：

设计一个接口 IFileExpert，它包括如表 1 的一些方法：

表 1

名称	功能	参数
Create	完成创建文件的功能	LPCSTR strFileName
Delete	完成删除文件的功能	LPCSTR strFileName
Copy	完成拷贝文件的功能	LPCSTR strSource　LPCSTR strTarget
Move	完成移动文件的功能	LPCSTR strSource　LPCSTR strTarget

2　对象框架建立

在 WorkSpace 框中选择 ClassView 选卡，右键点击 FileManager classes 在弹出菜单中选择 New ATL Object. . . 项，进入 ATL Object Wizard 向导，在向导第一步中的类别框中选择 Objects Objects 框中的 Simple Object。单击下一步，在接下来的一步中我们需要对新建的 COM 对象的属性进行一些设置。在 Names 页的 C + + 组的 Short Name 框中输入组件的短名字，这里我们输入 FileExpert，其他项目由向导自动填充。进入 At-

tributes 页。这一页中我们需要对 COM 对象的线程模型、接口类型、是否支持聚合、是否支持错误处理接口、是否支持连接点接口、是否需要自由线程调度等等设置。这里我们选择 Support ConnectionPoints 项，其它项保持缺省。如图 1：

图 1

单击确定生成 FileExpert 对象的框架，后面的工作将从这一框架基础上进行。Object Wizard 为工程添加了如下的组件实现文件：

FileExpert. h：CFileExpert 的头文件

FileExpert. cpp：CFileExpert 的实现文件

FileExpert. rgs：FileExpert 组件的 Registrar 脚本文件

FileManager. idl：FileExpert 组件的接口描述语言文件

3 为接口增加方法

在 ClassView 里面右键单击 IFileExpert 项目，从弹出菜单中选择 Add Method. . . ，进入添加方法对话框。在方法名（Method Name）框中填入 Copy，参数 Parameters 框中填入 "[in]LPCSTR strSource [in]LPCSTR strTarget" 其中 "[in]" 表示该参数为输入参数 （dl 语法规定[in]修饰符表示输入参数，[out]修饰符表示输出参数，[retval]修饰符表示返回值，[out retval]为输出和返回值的组合），LPCSTR 为参数的类型，strSource 为参数名。以相同的方法添加表一中列出的其他方法。为每一方法添加处理，如下 黑体部分为自己加入的）。

```
STDMETHODIMP CFileExpert: : Copy(LPCSTR strSource, LPCSTR strTarget)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())
    if(!: : CopyFile(strSource, strTarget, FALSE))
//简单调用 SDK 函数 CopyFile 完成文件拷贝
    {
return S_FALSE; //在添加错误事件处理以前，简单的返回 S_FALSE
    }
    return S_OK;
}
STDMETHODIMP CFileExpert: : Move(LPCSTR strSource, LPCSTR strTarget)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())
    if(!: : MoveFile( strSource, strTarget)) //简单调用 SDK 函数 MoveFile 完成文件移动
    {
return S_FALSE; //在添加错误事件处理以前，简单的返回 S_FALSE
    }
    return S_OK;
}
STDMETHODIMP CFileExpert: : Delete(LPCSTR strFileName)
{/
    AFX_MANAGE_STATE(AfxGetStaticModuleState())
    if( !: : DeleteFile(strFileName)) / 简单调用 SDK 函数 DeleteFile 完成文件删除
    {
        return S_FALSE; //在添加错误事件处理以前，简单的返回 S_FALSE
    }
    return S_OK;
}
STDMETHODIMP CFileExpert: : Create(LPCSTR strFileName)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())
if(!: : CreateFile( strFileName, 0, 2, NULL, 1, 1, NULL))
//简单调用 SDK 函数 CreateFile 完成文件创建
//为了实例，我们武断地为 CreateFile 传递其他参数
    {
    return S_FALSE; //在添加错误事件处理以前，简单的返回 S_FALSE
    }
    return S_OK;
}
```

4）建立工程

按 F7 键 或 Build 菜单下的 Build FileMananger. dll）建立工程。至此，我们的文件管理组件已现雏形。我们用 OLE / COM Object Viewer （Visual Studio 自带的一个工具）来测试一下我们的组件，在 Visual C + + IDE 的 Tools 菜单下选择 OLE / COM Object Viewer 进入如图 2：

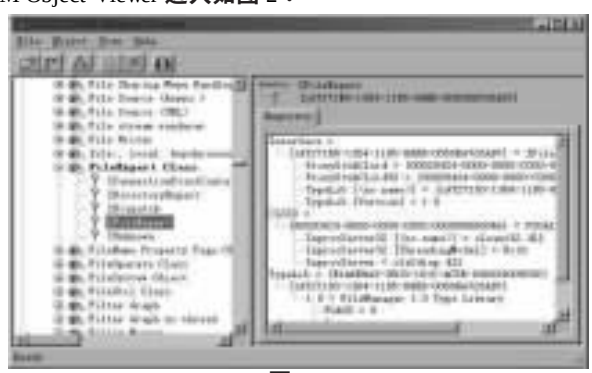


图 2

在这里我们可以创建组件的实例，查看注册信息，查看类型库等。通过测试，如果组件工作正常，可以进入下一步的工作。

5）添加事件处理接口

还记得我们在运行 ATL 对象向导时选择了 Support ConnectionPoints 选项吗？它自动为我们加入了事件处理接口 _IFileExpertEvents，我们现在的任务是要为其添加两个事件：

一个用于处理出错报告 OnErrorOccur，一个用于任务完成报告 OnMissionComplete，如下：

表 2

事件名称	功能	参数
OnErrorOccur	出错报告	[in] BSTR ErrorCode
OnMissionComplete	任务完成报告	[in]BOOL bCompleted

同为 IFileExpert 接口添加方法一样，为 _IFileExpertEvents 接口添加以上两个方法。在进行下一步工作之前，我们需要编译工程，以保证生成事件接口的类型库。

运行 Implement Connection Point Wizard 连接点向导）为事件创建代理类 （proxy）和包装类 它是从 IConnection-PontImpl 类派生的）。在 ClassView 页的 CFileExpert 点击右键，弹出右键菜单，选择 Implement Connection Point . . . ，进入连接点向导。如图 3 所示。在向导对话框的 Interfaces 项目中，包含了类型库中所有标记为 source 属性的接口。这儿我们只看到一个接口即 _IFileExpertEvents 接口，选择该接口并单击 OK 按钮。向导自动为我们在 FileManagerCP. h 中实现了 proxy 类。同时为我们添加了相应的接口映射条目。

图 3

6）修改 IFileExpert 的方法处理，如下粗体字所示：

```
STDMETHODIMP CFileExpert: : Copy(LPCSTR strSource, LPCSTR strTarget)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())
    if(! : : CopyFile(strSource, strTarget, FALSE))
    {
Fire_OnErrorOccur(L"拷贝文件时发生错误! "); //激发事件返回错误码
    //为了示例我们仅仅返回一个自定义的错误码
    return S_FALSE;
    }
Fire_OnMissionComplete(TRUE); //激发事件报告任务完成
    return S_OK;
}
```

其他的几个方法同上处理，这儿不在重复。

三、 第二版的 COM 组件——加入第二个接口 IDirectoryExpert

随着时间的推移或任务需要，我们的文件管理组件已不能胜任业务的需要，急需增加新的功能，如目录操作。这时我们可以修改 IFileExpert 接口来增加目录操作的功能，但这是万万不可取的。因为这样会导致已经建立的客户程序的崩溃，必须重新修改客户，再编译建立。

第二种方法就是为组件增加一个新的接口来完成目录操作。这样旧的应用程序不需要修改可以继续服务，而新的应用可以通过查询组件新的接口来使用组件的目录操作功能。下面我们为 FileExpert 加入一个目录操作接口 IDirectoryExpert。由于 ATL 没有提供向导来添加更多的接口，我们不得不手工加入此接口。

首先在接口描述 （DL）中加入对 IDirectoryExpert 的描述。如下，其中黑体为手工加入的内容。有删节。

```
import "oaidl. idl";
import "ocidl. idl";
        .
        .
        .
  [
        object,      uuid(C9B24EA1 – 13D7 – 11d5 – 89EB – 0050BA728AD7), //此值必须由 GuidGen 工具生成不要引用此值.
        dual,
        helpstring("IDirectoryExpert Interface"),
        pointer_default(unique)
  ]
  interface IDirectoryExpert: IDispatch
  {
  [id(1), helpstring("method MakeDirectory")] HRESULT MakeDirectory([in]LPCSTR strDirectory);
  };
[
uuid(1A7271D3 – 13D4 – 11D5 – 89EB – 0050BA728AD7),
  version(1. 0),
  helpstring("FileManager 1. 0 Type Library")
]
library FILEMANAGERLib
{
  importlib("stdole32. tlb");
  importlib("stdole2. tlb");
  .
  .
  .
  coclass FileExpert
  {
        [default] interface IFileExpert;
        [default, source] dispinterface _IFileExpertEvents;
        interface IDirectoryExpert;
  };
};
```

第二，在包装类中加上继承层次与接口映射，如下，其中黑体为手工加入或改动。

```
class ATL_NO_VTABLE CFileExpert :
public CComObjectRootEx <CComSingleThreadModel>,
```

（下转第 39 页）

用 Detours 拦截 Win 32 API 函数

冉林仓

摘 要 本文主要介绍 Detours 的功能、原理、使用方法，并以一个具体的实例说明如何使用 detours 拦截 Win32 API 函数。

关键词 Detours，Trampoline，Target，Payloads，Patch

一、引言

一个创新系统的研究关键在于调试和扩充现有操作系统和应有程序功能，而无论这些代码位于应用程序、库中或者是操作系统的动态链接库。拦截函数的主要目的是增加功能、修改返回值，或者是插入调试和评估性能指令。如果有适当的源程序的话，通过插入新的函数指令，然后重建操作系统和应用程序，可以说是小菜一碟、不值一提。然而在这个商业软件充斥的世界，研究人员接触到的只是二进制的机器码 根本无从接触相关的源程序。

为此微软公司开发了一个 Detours 库，用于 x86 机器在运行态动态地拦截任意的 Win 32 二进制函数。Detours 把 Target（要拦截的目标函数）开始的几条指令替换成一个无条件跳转到 Detour 函数（用户提供的替换函数）的跳转指令。Target 函数（目标函数）的指令被保存到一个名叫 Trampoline 函数（跳板函数）之中。Trampoline 函数由 Target 函数开始几条指令和一条跳转到 Target 函数的残余位置的指令组成。Detours 函数既可以完全替换 Target 函数，也可以通过调用 Trampoline 函数，变相地调用原来 Target 函数，来实现对原来 Target 函数功能的扩充。

Detours 函数插入是在运行态进行的。Target 函数的修改完全是在内存中进行的，而不是在磁盘上，因此是在很小的时间间隔内就轻松地完成了函数的替换工作。利用 Detours 可以实现动态链接库的一个过程在一个应用程序的进程之中被挂接（API hook），而在另一个同时运行的进程中不被挂接。与动态链接库再连接和静态重定位不同的是，Detours 库使用的拦截技术可以保证无论应用程序和系统代码采用何种方法定位 Target 函数，拦截都会起作用。

Detours 作为一个通用性的开发工具包，它把原始的 target 函数作为一个可调用函数过程（Trampoline 函数）保存起来。大大方便了对现有二进制 Win32 软件的扩充。

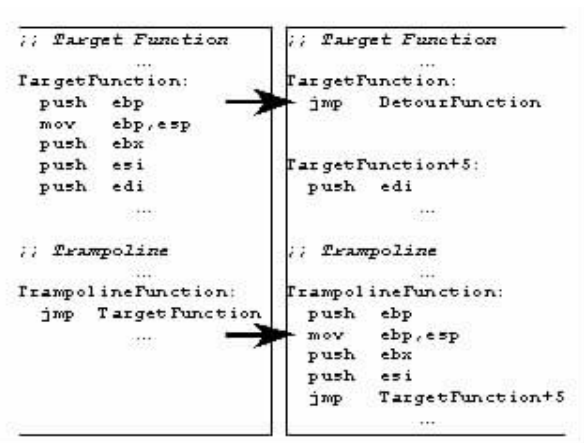
除了基本的 Detours（API HOOK）功能之外，Detours 还包括了几个函数，以实现 DLL 二进制输入表的编辑、把任意的数据段插入到存在的二进制代码中、把 DLL 插入到一个新的或者已经存在的进程之中。一旦实现 DLL 注入到一个新进程，DLL 库中的代码就会生效，它可以完成任何二进制的挂接，不管这些 Target 函数位于应用程序或者是系统库中。

二、工作原理

1. 二进制函数的拦截

拦截是在运行态动态进行的。Detours 首先把 Target 函数的开始几条指令替换成一个无条件跳转指令（JMP） 这个指令跳转到用户提供的 Detour 函数。而这些将要被覆盖的指令被保存到一个 Trampoline 函数中（我们可以称它为跳板函数） Trampoline 函数由这些指令和一个跳转指令（JMP）组成，跳转指令跳转到修改后 Target 函数的第二条指令，即开始的 JMP Detour 指令后的一条指令。

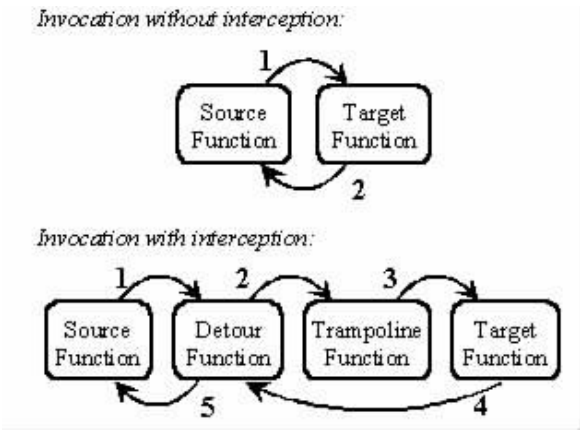
图式如下：



当程序运行到目标函数时，控制直接跳转到用户提供的 Detour 函数，Detour 函数执行一些拦截预处理，然后可以直接将控制权返回函数的调用处 也可以调用 Trampoline 函数，实现原来 Target 函数的功能。Trampoline 函数执行完后控制权又回到 Detour 函数，再由 Detour 函数执行一些事后处理工作，然后将控制权返回 Target 函数的调用处。

图式如页。

Detours 库通过改写进程内二进制映象实现对 Target 函数的拦截。针对每一个 Target 函数，Detours 实际上修改了两个函数，Target 函数和 Trampoline 函数。Trampoline 函数既可以动态分配也可以静态分配。静态分配 Trampoline 函数总是直接跳转到 Target 函数执行。在 Detour 函数插入之前，静态分配

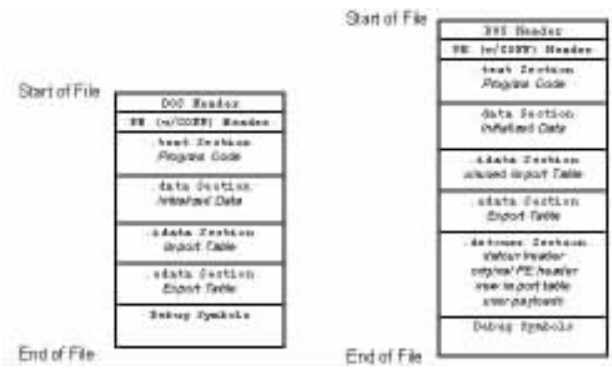


Trampoline 函数只包含一条跳转指令。Detour 函数插入后，Target 函数包含了 Target 函数的最初几条初始指令和一个跳转到 Target 函数残余部分的指令。对于动态分配的 Trampoline 函数，Detours 首先要为动态的 Trampoline 函数分配内存 然后修改内存读写权限，以便对 Target 函数和 Trampoline 函数进行读写存取。Detours 首先把 Target 函数的几个起始指令复制到 Trampoline 函数地址，至少 5 个字节，以便容纳 5 字节的无条件长跳转指令。如果 Target 函数长度少于 5 个字节 Detours 就会放弃操作返回一个错误代码。为了复制这些指令代码，Detours 采用了简单的表控反汇编，在 Trampoline 函数的最后添加了一条跳转指令，跳转到 Target 函数的第一个没有被复制的字节处。执行完毕后，Detours 恢复两个函数的所在内存页原有权限。然后针对多处理器机器调用 FlushInstructionCache 函数 刷新 cpu 指令高速缓存。

2. Payloads 负载 和 DLL 输入表编辑

Detours 提供了一组函数实现 DLL 的输入表编辑和任意数据段的注入 即把一个任意的 DLL 库注入到一个进程之中，这个过程既可以在内存中进行，也可以在磁盘上进行 把数据段的信息写到一个磁盘文件中，而且这个过程是可逆的。

在下面的左图中，是一个 Win 32 PE Portable Executable 二进制文件的基本结构，Win 32 二进制 PE 格式是一个 COFF 扩展 （Common Object File Format）。一个 Win32 二进制文件是由下面几部分组成的：一个 DOS 兼容头、一个 PE 头、 一个包含程序代码的文本段、一个包含初始化数据的数据段、一个列举引用 DLL 和函数名的引入表、一个列举代码和输出



符号的输出表，除了两个头结构之外 其余部分都是可选的 在某些 Win32 文件中可能某些部分不存在。

为了修改 Win 32 二进制代码，Detours 又在输出表和调试符号部分中间创建了一个新的 . Detours 段 （如上图所示）

往：调试符号部分必须位于 Win32 二进值文件的最后），这个新的 . Detours 段包含了一个 Detours 头记录和一个原来 Pe 头的副本。如果要修改输入表的话，Detours 就会创建一个新的输入表，并把它添加到复制的 Pe 头中，然后修改原始的 Pe 头结构，使其指向新的输入表。最后，Detours 把用户的 Payloads 负载 追加到 . Detours 段的尾部，然后在文件尾部添加调试符号信息结束文件。由于 Detours 备份了原始的 Pe 头，所以这个过程完全可逆，以便去除 Detours 段 恢复 exe 文件的原始面目。

创建一个新的输入表有两个目的，一个目的是它能够保存原来的输入表，以便需要时恢复对原来文件的修改 另一个目的是，新的输入表可以包含重命名的动态链接库和函数 还可以包含新的动态链接库和函数。

Detours 既提供了一组编辑输入表，添加、枚举、去除 Payloads 负载 再绑定二进制的函数 也提供了一组枚举映射地址空间的二进制文件、定位映射文件 Payloads 负载 的函数。每一个 Payload 负载 均由一个 GUID a 128 - bit globally unque identifier 全球唯一标识符 标识。

如果只需要把指令插入到进程的地址空间，无须修改二进制 Win32 磁盘文件 Detours 提供了一组函数能够把 DLL 库插入到一个新的或者是一个已经存在的进程。为了注入 DLL Detours 通过 VirtualAllocEx 和 WriteProcessMemory API 函数把 LoadLibrary 调用写进了目标进程，然后用 CreateRemoteThread 函数触发这个调用。

上述的工作原理阐述可能不尽人意，感兴趣的读者不妨分析 Detours 提供的源程序代码。

三、使用 Detours 库函数

使用 Detours 非常简单，只需要在程序中包含 Detours. h 头文件，连接 Detours. lib 库即可 程序包提供了大量的例子程序供参考，这里仅给出一个简单的例子，以便说明。

```
#include <stdio. h>
#include <windows. h>
#include <detours. h>
int (WINAPI * s_pShellAboutA ) (HWND hWnd, LPCTSTR szApp, LPCTSTR szOtherStuff, HICON hIcon) = NULL;
int (WINAPI * s_ pReal_ShellAboutA ) (HWND hWnd, LPCTSTR szApp, LPCTSTR szOtherStuff, HICON hIcon) = NULL;
DETOUR_TRAMPOLINE( VOID WINAPI Trampoline_Sleep(DWORD dwMilliseconds), Sleep);
static int WINAPI Catch_ShellAboutA (HWND hWnd, LPCTSTR szApp, LPCTSTR szOtherStuff, HICON hIcon)
{ s_pReal_ShellAboutA(hWnd, szApp, "进行拦截前的预处理", hIcon);
```

```
int nReturn = s_pReal_ShellAboutA(hWnd, szApp, szOther-
Stuff, hIcon); s_pReal_ShellAboutA(hWnd, szApp, "进行拦截
后的善后处理处理", hIcon);
return nReturn;
}
static VOID WINAPI Detour_Sleep(DWORD dwMilliseconds)
{ printf("Starting to sleep for %d milliseconds. \n", dwMil-
liseconds);
      Trampoline_Sleep(dwMilliseconds);
      printf(" Done sleeping. \n");
}
static VOID WINAPI Test(VOID)
{    printf( "Test\n");
}
DETOUR_TRAMPOLINE(VOID WINAPI Trampoline_Test(VOID),
Test);
static VOID WINAPI Detour_Test(VOID)
{  printf( "Starting Test. \n");
   Trampoline_Test();
    printf( "End test. \n");
}
DETOUR_TRAMPOLINE_EMPTY ( VOID WINAPI Trampoline_
Test1(VOID));
DETOUR_TRAMPOLINE_EMPTY ( VOID WINAPI Trampoline_
Test2(VOID));
DETOUR_TRAMPOLINE_EMPTY ( VOID WINAPI Trampoline_
Test3(VOID));
static VOID WINAPI Detour_Test1(VOID)
{    printf( "Starting Test1. \n");
   Trampoline_Test1();
   printf( "End Test1. \n");
}
static VOID WINAPI Detour_Test2(VOID)
{    printf( "Starting Test2. \n");
   Trampoline_Test2();
   printf( "End Test2. \n");
}
static VOID WINAPI Detour_Test3(VOID)
{    printf( "Starting Test3. \n");
   Trampoline_Test3();
   printf( "End Test3. \n");
}
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hprev,
LPSTR lpszCmdLine, int nCmdShow)
{ DetourFunctionWithTrampoline((PBYTE) Trampoline_Sleep,
(PBYTE)Detour_Sleep);
      printf("Calling %d\n", GetLastError());
    DetourFunctionWithTrampoline((PBYTE) Trampoline_Test,
(PBYTE)Detour_Test);
   s_pShellAboutA = ((int (WINAPI *)( HWND, LPCTSTR,
LPCTSTR, HICON))
DetourFindFunction("Shell32. dll", " ShellAboutA"));
s_pReal_ShellAboutA = ((int (WINAPI * )( HWND, LPCT-
STR, LPCTSTR, HICON))
DetourFunction (( PBYTE) s_pShellAboutA, (PBYTE) Catch_
ShellAboutA));
          Sleep(2);
    ShellAboutA(NULL, "OK", "OK", NULL);
  printf("Calling Test. \n");
  Test();
DetourFunctionWithEmptyTrampoline((PBYTE)  Trampo-
line_Test1, (PBYTE)Test, (PBYTE)Detour_Test1);
DetourFunctionWithEmptyTrampoline (( PBYTE) Trampoline_
Test2, (PBYTE)Test, (PBYTE)Detour_Test2);
  DetourFunctionWithEmptyTrampoline((PBYTE)Trampoline_
Test3, (PBYTE)Test, (PBYTE)Detour_Test3);
   printf("Calling test again with recursive detours. \n");
  Test();
       printf("Done. \n");
  DetourRemove((PBYTE) s_pShellAboutA, (PBYTE) Catch_
ShellAboutA));
  return 0;
}
```

上面的例子程序可以看出，Trampolines 函数既可以静态创建也可以动态创建。使用静态的 Trampolines 函数拦截 Target 函数时 应用程序需要使用 DETOUR_TRAMPOLINE 宏创建 Trampolines 函数。DETOUR_TRAMPOLINE 宏带有两个参数，静态的 Trampolines 函数原形和 target 函数名。

值得注意的是，target 函数拦截时，target、 trampoline、detour 函数必须遵循完全一致的参数约定，即出口和入口参数在个数和对应类型上要求完全匹配。detour 函数可以根据需要把它的入口参数传递到 trampoline 函数以此调用 Target 函数。遵循相同的调用约定可以保证有关的寄存器能够被正确保存 detour 和 Target 函数的堆栈能够保持平衡。

Target 函数的拦截可以通过 DetourFunctionWithTrampoline 函数实现，这个函数有两个参数，trampoline 和一个指向 detour 函数的指针。目标函数 Target 之所以没有作为一个参数是因为它已经编码到 trampoline 函数之中。

动态的 trampoline 可以通过调用 DetourFunction 函数实现。这个函数有两个参数 分别是两个指向 target 和 detour 函数的指针 DetourFunction 可以分配一个新的 trampoline 函数指针变量 然后在 target 函数中插入适当的拦截代码。

当 Target 函数可以作为一个连接符号时 静态 Trampoline 函数使用非常容易。然而当 Target 函数不能作为一个连接符号 你可以使用一个动态的 trampoline 函数。通常情况下 可以通过另外的辅助函数获得指向 target 函数指针。当指向 Target 函数的指针不易得到时，你可以使用 DetourFindFunction 获得该指针 无论这些函数来自一个已知的 DLL 库，或者是 target 函数二进制代码的调试符号，DetourFindFunction 函数接受两个参数 二进制 Win32 文件名和函数名。如果函数执行后找到了 Target 的符号 它就会返回一个有效的函数指针，否则返回 NULL。DetourFindFunction 首先尝试利用 LoadLibrary 和 GetProcAddress Win32 函数定位函数 如果在 DLL 的函数输出表中找不到 Target，DetourFindFunction 就会使用 ImageHlp 库查找可用的调试符号信息。返回的函数指针可以作为一个参数传递到 DetourFunction 函数用以创建一个动态的 trampoline 函数。

拦截的 target 函数可以通过调用 DetourRemoveTrampoline 函数恢复到拦截前的状态。

值得注意的是，由于 Detours 库函数修改代码是在进程地址空间进行的 程序员必须在 detour 插入和移去时没有别的线程在运行。显然保证单线程运行最简单的办法是在 DLL 的 DllMain 例程中调用 Detours 库函数。

动态链接库注入到一个存在进程可以通过下列代码实现

```
    HANDLE hProcess = OpenProcess ( PROCESS_ALL_
ACCESS, FALSE, nProcessId);
// nProcessId 运行进程的进程标识
    if (hProcess = = NULL) {
    printf(″OpenProcess(%d) failed: %d\n″, nProcessId,
GetLastError());
        return 2;
    }//下面 szDllPath 为指定的 dll 文件路径
if (!DetourContinueProcessWithDllA(hProcess, szDllPath)) {
printf( ″ DetourContinueProcessWithDll(%s) failed: %d″,
szDllPath, GetLastError());
        return 3;
    }
```

动态链接库注入到一个新进程可以通过下列代码实现

```
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CHAR szCommand[2048];
    PCHAR pszDLlPath =NULL;
    CHAR szExe[1024];
    Strcpy(szExe, ″.....″); //可执行文件名
 Strcpy(szCommand, ″.....″); //命令行
 Strcpy(pszDllPath, ″...″); //动态连接库文件名字
    ZeroMemory(& si, sizeof(si));
    ZeroMemory(& pi, sizeof(pi));
    si. cb = sizeof(si);
  DWORD dwCreationFlags = (CREATE_DEFAULT_ERROR_
MODE);
if ( ! DetourCreateProcessWithDll(szFullExe, szCommand,
NULL, NULL, TRUE, dwCreationFlags, NULL, NULL, & si,
& pi, pszDllPath, NULL)) {
    printf( ″ DetourCreateProcessWithDll failed: %d\ n″,
GetLastError());
        ExitProcess(2);
        }
```

把一个动态链接库附加到一个 Win32 可执行文件，以便可执行文件运行时 DLL 自动加载，限于篇幅，感兴趣的读者不妨参考一下 Detours 包提供的 SetDll. cpp 文件。

四、兼容性问题

所有的 Detours 函数都可以在 Windows NT 的所有版本和 Windows 2000 下正常运行。然而，在 Windows 95 和 Windows 98 环境下 DetourFunction 只能运行在使用 DEBUG_PROCESS 标志用 CreateProcess 创建的调试进程中，这时候你会发现一个有趣的现象，在 Visual Studio 环境下，按 F5 键 (Go) 和按 Ctrl + F5 键 Execute Program 运行程序会得到不同的运行结果，这明显是上述原因造成的。Windows NT 和 Windows 2000 总是使用写后引用的技术实现 DLL 到进程的映射，这正是 Detours 为 Win32 文件映象打补丁所采用的技术，而 Win 9x 仅当进程创建采用 DEBUG_PROCESS 标志用 CreateProcess 创建时才使用这种写后引用的 dll 映象方法。不过对于象 ShellAboutA 这样的 Shell 函数 两种方式运行并没有什么两样。为保险起见，在 Win 9x 环境下你最好使用其它 API HOOK 技术。

由于 Win 9x 不支持 CreateRemoteThread 函数，DLL 插入有关 detours 函数无法在两种环境下运行。不过你可以采用钩子函数实现 DLL 的注入。

为添加 Payloads 和修改输入表设计的二进制的复写函数在所有的 Win 32 环境都得到了全面支持，你可以大胆使用 以便实现任何一个 Win32 程序运行后能够自动加载你设计的 DLL 文件，而不需要另外的加载程序。

五、补充说明

Detours 在 Borland C + + builder 环境也可以使用 不过你要注意在涉及字符串的时候，要留意一下拦截函数的 ANSI 和 UNICODE 两个版本。 有关 Detours 的信息请访问

http / / www. research. microsoft. com / sn / detours

http / / research. microsoft. com / sn / detours

http / / toolbox MS Internal

参考文献

Galen Hunt and Doug Brubacher. Detours Binary Interception of Win32 Functions

Microsoft Research. One Microsoft Way. Redmond WA 98052

收稿日期：2001 年 4 月 2 日）

（续接第 35 页）

```
public CComCoClass <CFileExpert, & CLSID_FileExpert>,
      public IConnectionPointContainerImpl <CFileExpert>,
  public IDispatchImpl <IFileExpert, & IID_IFileExpert, & LI-
BID_FILEMANAGERLib>,
   public CProxy_IFileExpertEvents < CFileExpert >,
public IDispatchImpl < IDirectoryExpert , & II D_IDirectory-
Expert, & LIBID_FILEMANAGERLib>,
    {
public:
    CFileExpert()
    {
    })
DECLARE_REGISTRY_RESOURCEID(IDR_FILEEXPERT)
DECLARE_PROTECT_FINAL_CONSTRUCT()
BEGIN_COM_MAP(CFileExpert)
    COM_INTERFACE_ENTRY(IFileExpert
```

收稿日期 2001 年 3 月 14 日

Windows CE 的红外通信及其应用

丁胜昔 范慧琴

摘 要 本文描述了基于 Windows CE 设备的 IrDA 通信功能，介绍了几种进行红外通信的方法，并给出了用 IrSock 方法实现 IrDA 数据传输的程序代码。

关键词 红外通信，IrDA

Windows CE 设备几乎都有 IrDA 兼容的红外端口，实际上，所有 H/PC 和掌上 PC 系统都有这样一个端口，IrDA 标准详细说明物理实现的任何信息，如使用的光波频率、两设备间的同步以及远程系统如何查找对方并问话。

IR 端口有多种使用方法，如原始 IR、IrComm 和 IrSock。

一、原始 IR

在最基本的应用中，IR 端口可以作为附加的 IR 发射器和接收器的串行端口进行访问，这种方法就是原始 IR。当使用原始 IR 时，端口可以不是 IrDA 兼容的，它不需要 IrDA 标准的同步交换。但当红外光束被切断或者受到严重干扰时，则可能丢失数据，应用程序必须检测这些错误并予以纠正。

要使用原始 IR，首先必须找到与 IR 收发器相连的串行端口，在有些 Wiindows CE 设备中，串行端口和 IR 端口使用同一个串行硬件，在使用 IR 端口时就不能使用串行端口了。要查找原始 IR 所用的 COM 端口，可以查看注册表中的 HKEY_LOCAL_MACHINE hComm hIrDA 键，Port 值就是 COM 端口号。下面的代码可以返回 IR 端口的设备名称。

```
int GetRawIrdaName(LPTSTR pIrdaName) {
    DWORD dwSize, dwType, dwData;
    HKEY  hKey;
    *pIrdaName = TEXT('\0');
// 打开 IrDA 键
    if ( RegOpenKeyEx ( HKEY_LOCAL_MACHINE, TEXT
("Comm\IrDA"), 0, 0, & hKey) = =ERROR_SUCCESS) {
// 查询 IR 端口号
    dwSize = sizeof(dwData);
if( RegQueryValueEx(hKey, TEXT( " Port"), 0, & dwType,
(PBYTE)& dwData, & dwSize) = =ERROR_SUCCESS )
    wsprintf(pDevName, TEXT("COM%d:"), dwData);
    RegCloseKey(hKey);
    }
    return lstrlen(pDevName);
}
```

有了 IR 端口名称后，如果该端口硬件由串行端口和 IR 端口共享，就必须调用 EscapeCommFunction 函数并设置 dwFunc 参数为 SETIR 来告诉驱动程序是通过 IR 收发器控制串行数据流。其他的就可以和串行通信一样进行了，其中的一个重要不同点是，因为发射器和接收器使用的是同一个光波空间，一个设备不能在接收数据的同时还要传送数据，这种通信是半双工的。

二、IrCOMM

IrCOMM 端口是一个模拟端口而不是真实设备，因此在许多方面与原始 IR 不同，例如，应用程序不能配置 IrCOMM，Windows CE 透明的使用 IrSock 配置 IrCOMM 端口一支持 IrDA 协议。同样，在使用 IrCOMM 之前必须先要找到 IrCOMM 端口，下面的代码可以返回 IrCOMM 端口的设备名称。

```
int GetIrCommDeviceName(LPTSTR pDevName)
{
  DWORD dwSize, dwType, dwData;
  HKEY hKey;
  *pDevName = TEXT('\0');
// 打开 IrCOMM 键
if(RegOpenKeyEx(HKEY_LOCAL_MACHINE, TEXT("Drivers\\
BuiltIn\\IrCOMM"), 0, 0, & hKey) = =ERROR_SUCCESS ) {
// 查询 IrCOMM 端口号
      dwSize = sizeof(dwData);
  if( RegQueryValueEx(hKey, TEXT("Index"), 0, & dwType,
(PBYTE)& dwData, & dwSize) = =ERROR_SUCCESS )
  wsprintf(pDevName, TEXT("COM%d:"), dwData);
      RegCloseKey(hKey);
  }
  return lstrlen(pDevName);
}
```

有了设备名称后，可以使用普通的 Comm API 对 IrCOMM 进行编程了，需要注意的是 IrCOMM 是点对点协议，只能两台设备之间连接。

三、IrSock

使用 IR 端口最可靠也是最复杂的方法是 IrSock，IrSock 是 WinSock 的一种扩展，这种 API 建立在用于红外通信的 IrDA 堆栈顶端。IrSock 与 WinSock 之间的主要区别在于 IrSock 并不支持数据报，也不支持安全性。并且它用来寻址的方法与 WinSock 中使用的方法完全不同，IrDA 被设计用于处理浏览一定范围内的资源。

从程序设计者的观点来看，IrSock 与 WinSock 程序设计的主要不用在于客户机端需要一种方法来检测哪些红外设备在范围内，并已经准备好接收套接子连接，这可以通过调用 get-

sockopt 函数来完成。

使用 IrSock 的基本过程与使用 WinSock 类似，服务器应用程序和客户应用程序具有不同的过程，下面分别列出了创建 IrSock 应用程序的步骤：

1. 创建 IrSock 服务器应用程序

(1) 用 socket 函数打开流接口，其中 af 参数设置为 AF_IRDA，type 参数设置为 SOCK_STREAM，protocol 参数设置为 NULL；

(2) 用 bind 函数将套接字绑定到服务器地址，其中 addr 参数为 SOCKADDR_IRDA 结构；

(3) 用 listen 函数收听客户连接；

(4) 用 accept 接收客户连接

(5) 用 send 和 recv 与客户通信

(6) 用 closesocket 关闭套接字

2. 创建 IrSock 客户应用程序

(1) 用 socket 函数打开流接口，其中 af 参数设置为 AF_IRDA，type 参数设置为 SOCK_STREAM，protocol 参数设置为 NULL；

(2) 用 getsockopt 查找服务器并获取服务器的标识；

(3) 用 connect 将套接字连接到服务器，其中 addr 参数为 SOCKADDR_IRDA 结构；

(4) 用 send 和 recv 与服务器通信；

(5) 用 closesocket 关闭套接字。

下面分别是 IrSock 服务器应用程序和 IrSock 客户应用程序的实例。

1) IrSock 服务器应用程序实例

```
#include <windows.h>
#include <af_irda.h>
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow) {
  SOCKET              ServerSock, ClientSock;
SOCKADDR_IRDA address = {AF_IRDA, 0, 0, 0, 0, "IRServer"};  // 服务器 IrSock 地址
  int                 index = 0, iReturn;
  char                szServerA[100];     // 接收数据的 ASCII 码
  TCHAR               szServerW[100], szError[100];
  // 创建服务器套接字
  if ((ServerSock = socket (AF_IRDA, SOCK_STREAM, 0)) == INVALID_SOCKET) {
    wsprintf (szError, TEXT("Allocating socket failed. Error: %d"), WSAGetLastError ());
    MessageBox (NULL, szError, TEXT("Error"), MB_OK);
    return FALSE;
  }
  // 将套接字绑定到服务器地址
  if (bind (ServerSock, (struct sockaddr *)& address, sizeof (address)) == SOCKET_ERROR) {
    wsprintf (szError, TEXT("Binding socket failed. Error: %d"), WSAGetLastError ());
    MessageBox (NULL, szError, TEXT("Error"), MB_OK);
    closesocket (ServerSock);
    return FALSE;
  }
  // 监听客户连接
  if (listen (ServerSock, 5) == SOCKET_ERROR) {
    wsprintf (szError, TEXT("Listening to the client failed. Error: %d"), WSAGetLastError ());
    MessageBox (NULL, szError, TEXT("Error"), MB_OK);
    closesocket (ServerSock);
    return FALSE;
  }
  // 接收客户连接
  if ((ClientSock = accept (ServerSock, 0, 0)) == INVALID_SOCKET) {
    wsprintf (szError, TEXT("Accepting connection with client failed. Error: %d"), WSAGetLastError ());
    MessageBox (NULL, szError, TEXT("Error"), MB_OK);
    closesocket (ServerSock);
    return FALSE;
  }
  // 关闭服务器套接字, 停止监听客户连接
  closesocket (ServerSock);
  // 从服务器 IrSock 发送一个字符串到客户 IrSock
  if (send (ClientSock, "To Client!", strlen ("To Client!") + 1, 0) == SOCKET_ERROR) {
    wsprintf (szError, TEXT("Sending data to the client failed. Error: %d"), WSAGetLastError ());
    MessageBox (NULL, szError, TEXT("Error"), MB_OK);
  }
  // 接收来自客户的数据
iReturn = recv(ClientSock, szServer, sizeof (szServerA), 0);
  // 检查接收到的数据, 如果收到了, 就显示它
  if (iReturn == SOCKET_ERROR) {
    wsprintf (szError, TEXT("No data is received, recv failed. Error: %d"), WSAGetLastError ());
   MessageBox (NULL, szError, TEXT("Server"), MB_OK);
  }
  else if (iReturn == 0)
    MessageBox (NULL, TEXT("Finished receiving data"), TEXT("Server"), MB_OK);
  else {
    // 将 ASCII 字符变成 UNICODE 字符
for (index = 0; index <= sizeof (szServerA); index++)
    szServerW[index] = szServerA[index];
    MessageBox (NULL, szServerW, TEXT("Received From Client"), MB_OK);
  }
  // 关闭客户套接字
  closesocket (ClientSock);
  return 0;
}
```

2) IrSock 客户应用程序实例

```
#include <windows.h>
#include <af_irda.h>
#define NUMRETYR 5
int WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow) {
  SOCKET sock;
  DEVICELIST devList;
SOCKADDR_IRDA address = { AF_IRDA, 0, 0, 0, 0, "IRServer"};  // 服务器 IrSock 地址
```

```
int iCount = 0, index = 0, iReturn, iDevListLen = sizeof (devList);
 char szClientA[100];          // 接收数据的 ASCII 码
 TCHAR szClientW[100];         // 接收数据的 Unicode 码
 TCHAR szError[100];
 // 创建绑定到服务器地址的 IrSock
 if ((sock = socket (AF_IRDA, SOCK_STREAM, 0)) = = INVALID_SOCKET) {
   wsprintf (szError, TEXT("Allocating socket failed. Error: %d"), WSAGetLastError ());
   MessageBox (NULL, szError, TEXT("Error"), MB_OK);
   return FALSE;
 }
 devList. numDevice = 0;
 while ( (devList. numDevice = = 0) & & (iCount < = NUMRETYR)) {
   // 获取 socket 选项
 If (getsockopt (sock, SOL_IRLMP, IRLMP_ENUMDEVICES, (char * ) & devList, & iDevListLen) = = SOCKET_ERROR) {wsprintf (szError, TEXT("Server can't located, getsockopt. Error: %d"), WSAGetLastError ());
    MessageBox (NULL, szError, TEXT("Error"), MB_OK);
    closesocket (sock);
    return FALSE;
   }
   iCount + +;
   // 等待 1 秒钟
   Sleep (1000);
 }
 if (iCount > NUMRETYR) {
   MessageBox (NULL, TEXT ("Server could not be located!"), TEXT ("Error"), MB_OK);
   closesocket (sock);
   return FALSE;
 }
// 得到服务器 socket 地址
 for (index = 0; index < = 3; index + +) {
address. irdaDeviceID[index] =devList. Device[0]. irdaDeviceID[index];
 }
 // 将套接字连接到服务器
 if (connect (sock, (struct sockaddr * ) & address, sizeof (SOCKADDR_IRDA)) = = SOCKET_ERROR) {
   wsprintf (szError, TEXT("Connecting to the server failed. Error: %d"), WSAGetLastError ());
   MessageBox (NULL, szError, TEXT("Error"), MB_OK);
   closesocket (sock);
   return FALSE;
 }
 // 从客户 socket 向服务器 socket 发送字符串
 if (send (sock, "To Server.", strlen ("To Server.") + 1, 0) = = SOCKET_ERROR) {
   wsprintf (szError, TEXT("Send data to the server failed. Error: %d"), WSAGetLastError ());
   MessageBox (NULL, szError, TEXT("Error"), MB_OK);
 }
 // 接收来自服务器的数据
 iReturn = recv (sock, szClientA, sizeof (szClientA), 0);
 // 检查是否收到数据, 并显示它
 if (iReturn = = SOCKET_ERROR) {
   wsprintf (szError, TEXT("No data is received, recv failed. Error: %d"), WSAGetLastError ());
   MessageBox (NULL, szError, TEXT("Client"), MB_OK);
 }
 else if (iReturn = = 0) {
   MessageBox (NULL, TEXT("Finished receiving data"), TEXT("Client"), MB_OK);
 }
 else {
   // 将 ASCII 字符转换为 UNICODE 字符
  for (index = 0; index < = sizeof (szClientA); index + +)
    szClientW[index] = szClientA[index];
   MessageBox (NULL, szClientW, TEXT("Received From Server"), MB_OK);
 }
 // 关闭套接字
 closesocket (sock);
 return 0;
}
```

参考文献

1. 微软公司 . Microsoft Windows CE 通信指南 1999
2. Micorsoft Corporation. Platform Builder Library 2000
3. Douglas Boling. Windows CE 程序设计 1999

收稿日期：2001 年 4 月 11 日）

可视化专栏 VISUAL COLUMN

用 Visual C++ 创建基于 HTML 的可交互对话框

司瑞红　元凯宁

大多数的应用程序都有一个“关于”对话框，用来显示程序的版本信息、技术支持信息和一些超级链接。相信任何软件的作者都想把这个对话框做得漂亮一些，但这并不十分容易。有没有非常简单的方法达到这个目的呢？如果您打开过 IE4 或 IE5 的“关于”对话框并留心观察过，也许您会发现，整个的“关于”对话框就是一个网页。如果我们可以将网页的内容制成对话框，那么，所有要做的就是设计一个漂亮的网页了。我们可以在网页中加入超级链接（当然可以了），文字、声音、图片和 GIF 动画（或 Flash 动画），还可以加入 HTML 表单，并用 JavaScript 控制网页中任何元素的行为，甚至可以用 JavaScript 来实现复活节彩蛋。比起用程序控制达到相同的效果来，所作的工作减少了许多。那么，能够实现这样的功能吗？

在 IE 的核心库 mshtml.dll 中，第 13 号导出函数名叫 ShowHTMLDialog，它可以用来实现将网页以对话框形式显示的功能。所有的 HTML 语法分析等等工作都由它来完成了，我们要做的就是调用它。而且，还可以向生成的 HTML 对话框传递参数并接受从 HTML 对话框来的返回值，从而达到简单交互的目的。能够调用这个函数的必要条件是安装了 IE4.0 或以上版本。相信这不是一个十分苛刻的要求。

mshtml.dll 在整个 IE 浏览器体系中占有重要地位。IE 浏览器的可执行文件 iexplore.exe 仅有不到 60K 的大小。主要的工作全都是由 shdocvw.dll 和 mshtml.dll 完成的。iexplore.exe 是一个 COM 客户，两个主要的 COM 服务器就是 shdocvw.dll 和 mshtml.dll。整个 IE 浏览器体系结构见图 1。这张图是由 MSDN 提供的。

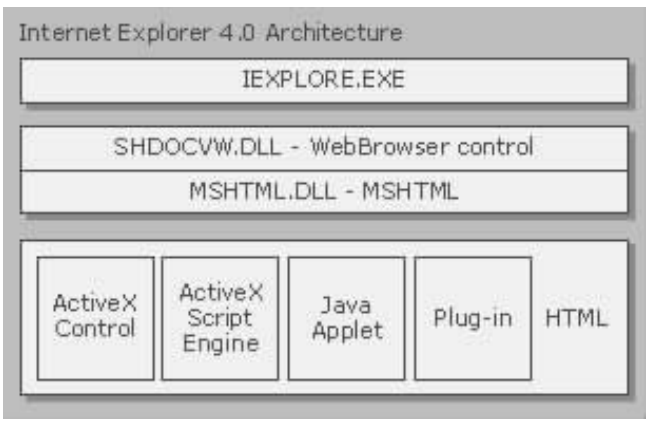


图 1　IE 的体系结构

在 IE 众多的动态链接库中，还有一个可能是我们比较感兴趣的，它就是 shdoclc.dll，它同样位于%SystemRoot%之下。在 IE 的安装程序中，它位于 IE_S2.CAB 之中。前面提到的两个 dll 文件和 iexplore.exe 均位于 IE_S1.CAB 中，由此也可见其重要性。shdoclc.dll 中包含了我们常见的一些网页，比如常见的取消浏览页、Web 页不可脱机使用页、DNS 错误页、无法打印页、发生 HTTP400、403、404、406、410、500、501 等错误页。shdoclc.dll 中还包含了一些我们常见的对话框，比如 IE 的关于对话框、在网页中查找对话框、脚本发生错误对话框、网页无法正常显示等对话框。除此之外，shdoc.dll 还包含我们常见的一些图片。我们可以用 RES 协议来直接查看这些页，在 IE 的地址栏中输入“res://shdoclc.dll/about.dlg”（无引号，下同），就可以在浏览器窗口中直接打开 IE 的“关于”对话框（不是产生一个对话框，而是在 IE 的窗口中显示“关于”对话框的内容），如图 2 所示。如果想要查看 sldoclc.dll 中所有的资源，请用 Developer Studio 按资源类型打开 shdoclc.dll，并按以上 RES 协议格式依次在 IE 的地址栏中输入在“2110”和“HTML”资源类型下的名字，比如“res://shdoclc.dll/navcancl.htm”、“res://shdoclc.dll/offcancl.htm”或“res://shdoclc.dll/ie5.gif”等（网页已过期页则是在 mshtml.dll 中，地址为“res://mshtml.dll/repost.htm”）。

图 2　使用 RES 协议查看 dll 内的资源

在我们自己的程序中，也可以导入 HTML 类型的资源，只要直接使用 IMPORT 功能就可以。相关的图片、声音文件也要一并导入，但会出现指定导入类型对话框，填入 2110 即可。然后，将导入的资源命名为原来的文件名 在 Develop Studio 的属性对话框中的 ID 栏里输入文件名，并用引号括起来，强制用字符串命名）。这样，就可以在 IE 浏览器中使用 RES 协议查看自己的资源了 要输入全路径）。

最后，让我们来看一看究竟如何从 HTML 资源创建一个对话框。当然，从 HTML 文件创建也可以，但从资源创建可以保证程序界面不被人随意修改。我们要调用的 ShowHTMLDialog 函数并没有在任何头文件中定义，所以要用 Win32 函数 LoadLibary 将 mshtml. dll 动态地装入，再用 GetProcAddress 找到 ShowHTMLDialog 函数的指针并调用它，指针的类型说明可用 SHOWHTMLDIALOGFN 预定义类型，这个类型在头文件 mshtmhst. h 中说明。最后，用 FreeLibary 函数将 mshtml. dll 库卸出。

ShowHTMLDialog 函数的原型如下：

```
HRESULT STDAPICALLTYPE ShowHTMLDialog(
HWND hwndParent, // 父窗口的句柄,在对话框出现期间它将不允许输入键鼠消息
IMoniker * pmk, // IMoniker 接口的指针
VARIANT * pvarArgIn, // 输入参数,在 HTML 中可以通过 window. dialogArguments 访问
TCHAR * pchOptions, // 附加的参数,含义同 window. showModalDialog 的参数
VARIANT * pvArgOut // 输出参数,在 HTML 中通过 window. returnValue 指定
);
```

如果函数成功，则返回 0。

要使用这个函数，必需要有一个 URL 的 IMoniker 指针（其余的参数都可以填 NULL）。这个指针可以通过 CreateURLMoniker 函数来得到，CreateURLMoniker 函数的原型如下：

```
HRESULT CreateURLMoniker(
[in] IMoniker * pmkContext, // 输入的 IMoniker 接口指针
[in] LPWSTR szURL, // 以 Unicode 形式出现的 URL
[out] IMoniker * * ppmk // 输出的 IMoniker 接口指针
);
```

如果函数成功，返回 S_OK。

输出的 IMoniker 接口指针是我们要得到的，输入的 IMoniker 指针则是在 URL 以相对路径形式出现时，基准 URL 的 IMoniker 接口指针，可以为 NULL，这时 URL 字串必须是绝对路径。Unicode 形式的 URL 字串可以通过 MultiByteToWideChar 函数来得到。关于 MultiByteToWideChar 函数不再详述，使用方法请参见下面的程序代码。

现在我们就来创建 HTML 对话框。在这个对话框中，有一个编辑框，它最初显示由程序传到 HTML 文本的参数，这在 HTML 文本中是通过 window. dialogArguments 来得到的，当对话框显示期间，可以向它键入字符，这些字符会在窗口关闭前会被赋给 window. returnValue，从而以 Unicode 的形式传回到程序中，然后程序将通过一个消息框来显示它。程序的所有代码如下：

```
#include "stdafx. h"
#include "resource. h"
#include <mshtmhst. h>
class CHtmlDlgApp : public CWinApp
{
public:
  virtual BOOL InitInstance();
} theApp;
BOOL CHtmlDlgApp: : InitInstance()
{
  HINSTANCE hMSHTML;
  // 找到 mshtml. dll 的句柄
VERIFY(hMSHTML = LoadLibrary(_T("MSHTML. DLL")));
  // 得到 ShowHTMLDialog 函数的指针
  SHOWHTMLDIALOGFN * pfnShowHTMLDialog;
VERIFY(pfnShowHTMLDialog = (SHOWHTMLDIALOGFN * )
GetProcAddress(hMSHTML, TEXT("ShowHTMLDialog")));
  // 得到本进程的句柄
  HINSTANCE hInstance;
  VERIFY(hInstance = AfxGetInstanceHandle());
  TCHAR szPath[_MAX_PATH]; // _MAX_PATH eqs 260
  // 得到程序的名字(全路径)
  GetModuleFileName(hInstance, szPath, _MAX_PATH);
  // 128 这个数字在实际应用时需要修改,它应是资源最长的名字的值 +1
  TCHAR szURL[_MAX_PATH + 128];
// 用 RES 协议来指定地址, IDH_DIALOG 是 HTML 资源的名字
 wsprintf( szURL, _T( "res: //%s/%d"), szPath, IDH_DIALOG); // format the URL string
  int nLen = strlen(szURL);
  LPWSTR lpUnicodeURL = new WCHAR[nLen +1];
  // 转变到 Unicode
  VERIFY(MultiByteToWideChar(CP_ACP,
MB_PRECOMPOSED, szURL, nLen +1, lpUnicodeURL, sizeof(WCHAR ) * (nLen)));
  // 尝试去得到 URL Moniker 的接口指针
  IMoniker * pIMoniker;
 VERIFY(SUCCEEDED(CreateURLMoniker(NULL, lpUnicodeURL, & pIMoniker)));
#define VAL_TO_HTML 6 / * 这个值将补传递到 HTML * /
  VARIANT var; // 用于输入和输出的变量
  var. vt = VT_I4; // 4 字节整型
  var. lVal = VAL_TO_HTML ; // 值
  // 现在,显示对话框
HRESULT h = pfnShowHTMLDialog(NULL, // 没有父窗口
  pIMoniker, // moniker 接口指针
& var, // 传到 HTML 的 window. dialogArguments 的参数
NULL, //附加参数,本例中为 NULL,详见 window. showModalDialog
& var // 得到 HTML 中的 window. returnValue
  );
```

```
    // 现在,对话框已经关闭,检查返回值
    switch(var.vt)
    {
    case VT_BSTR:  // Unicode 字串
        {
    TCHAR szText[512];  // 512 并不能保证足够大
wsprintf(szText, _T("从网页中返回的 Unicode 字符串是:"%ls"\n"), var.bstrVal);
::MessageBox(NULL, szText, _T("HTML 对话框"), MB_OK);
        }
        break;
    default:
::MessageBox(NULL, _T("未处理的返回类型,可能是因为您关闭窗口用的不是"确定"按钮"), _T("HTML 对话框"),
MB_OK);
    break;
    }
    delete lpUnicodeURL;
    FreeLibrary(hMSHTML);

    // 返回 FALSE 使程序终止
    return FALSE;
}
```

要使用这个程序，必需导入 HTML 资源。将 HTML 文件导入，并命名为 IDH_DIALOG，再将它所需要的图形文件以 2110 类型导入，并以文件的原名来命名。或者可以修改指定 RES 地址的程序段，将它指定为一个固定的 HTML 文件名，可以是 file //协议或是 http //协议等。

本例程序的运行结果如图 3 和图 4 所示。

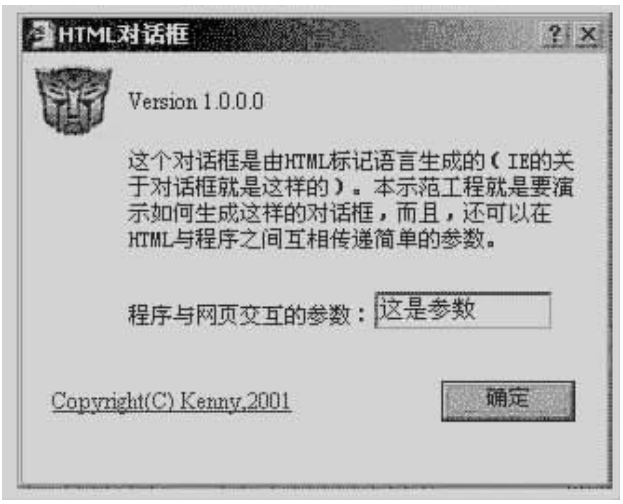


图 3　程序运行示例

图 4　程序显示了 HTML 返回的参数

本例中使用的 HTML 文件列出如下，请阅读其中的两个 JavaScript 函数并参阅前面的程序代码，就可以了解参数是怎样在程序和 HTML 文本中传递的了。

```
<html style="width: 20em; height: 16em">
<head>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=gb2312">
<title>HTML 对话框</title>
<script lang="jscript">
<!--
function returnParameter()
{
// 在这里,我们只是简单地返回文本输入框的内容
// 实际应用时可能需要将许多输入框的内容连接成一个字串
  window.returnValue = document.all.disp.value;
  window.close();
}
function processArguments()
{
  // 处理从程序传递来的参数,本例中固定为 6
  // 将它显示在文本输入框中
  document.all.disp.value = window.dialogArguments;
}
-->
</script>
</head>
<body onLoad="jscript:processArguments();">
<TABLE border=0>
  <TR>
    <TD><IMG alt="Autobots: transform and roll out!" src="autobots.gif"></TD>
    <TD><p align=bottom style="FONT: 9pt Times New Roman">Version 1.0.0.0</p></TD>
    <TD></TD>
  </TR>
  <TR>
    <TD></TD>
      <TD colspan=2><p style="FONT 9pt 宋体 WIDTH 20em">这个对话框是由 HTML 标记语言生成的 (IE 的关于对话框就是这样的)。本示范工程就是要演示如何生成这样的对话框,而且,还可以在 HTML 与程序之间互相传递简单的参数。</p><p style="FONT 9pt 宋体 WIDTH 20em">程序与网页交互的参数:<INPUT type="text" id=disp name=disp style="width 8em FONT 9pt Times New Roman 宋体" onFocus="document.all.disp.select()"></p></TD>
  </TR>
  <TR>
  <TD colspan=3><P> </p></TD>
  </TR>
  <TR>
    <TD colspan=2><p align=center style="FONT: 9pt Times New Roman"><A href="mailto:kenny_yuan@chi-
```

(下接第 51 页)

在 MFC DLLs 中导出资源及其相关类的实现方法

何珍文

摘 要 本文以一具有一定实用性的调色板对话框类的导出为例,详细介绍了在扩展 MFC 动态库导出对话框类的注意事项、方法和技巧。实例验证具有实用性。

关键词 MFC DLLs 资源,动态库

一、引言

在进行软件开发过程中，整个系统要分解成若干个小模块，各个模块的实现大多以动态库的形式存在在动态库中完整的导出资源（对话框、字符串、工具条等）和相应的资源响应函数或类是必要的，尤其是在面向对象编程方法（OOP）中，类的完整导出将使动态库的使用更加方便。在一些参考文献中对于动态库的原理和 Win32 动态库介绍得比较详细，而对于 MFC DLLs 中的资源及资源类的导出介绍得很少，大多一笔代过，或根本没讲。本文将结合一个调色板对话框类 CRisPalDlg 的导出示例对此加以说明。

二、在扩展 MFC DLLs 中导出对话框类

从扩展 MFC DLLs 中导出一个不和任何资源相关的类是很容易的。首先生成扩展 MFC 用 DLLs 程序框架，再用 Class Wizard 生成一个要导出的类（当然也可以自己定义类），加入要在该类中实现的功能函数，最后在类定义处插入 AFX_EXT_CLASS 宏，编译即可。剩下的工作便是如何在应用程序中调用的问题了。关于动态库的调用问题可以参见参考文献。

要在扩展 MFC DLLs 中导出对话框类，可在扩展 MFC 用 DLLs 程序框架基础上用 Class Wizard 插入从 CDialog 公有继承的 CRisPalDlg 类,再在 CRisPalDlg 定义处插入 AFX_EXT_CLASS 宏。代码如下：

```
/* RisPalDlg.h */
#if !defined(AFX_RISPALDLG_H)
#define AFX_RISPALDLG_H
class AFX_EXT_CLASS CRisPalDlg : public CDialog
{
public:
    CRisPalDlg(CWnd* pParent = NULL);
    //{{AFX_DATA(CRisPalDlg)
    enum { IDD = IDD_DIALOG_PALETTE };
    //}}AFX_DATA
    //{{AFX_VIRTUAL(CRisPalDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    //}}AFX_VIRTUAL
// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CRisPalDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
#endif
```

这样编译是通不过的，因为在编译时对话框资源没有包含在对话框类的文件中。在生成的 palettedialog.cpp 中有这样一行代码#include “h add additional includes here”，在此加入 resource.h 头文件，这样即可以顺利编译通过。另建工程 Test 测试动态库。在 Test 工程中加入头文件 RisPalDlg.h，并设置好连接项。编译工程会发现，应用程序不能通过编译，并发出如下编译错误信息：

```
error C2065: ´IDD_DIALOG_PALETTE´ : undeclared identifier
error C2057: expected constant expression
```

出错的原因是，在应用程序 Test 中无法对资源 IDD_DIALOG_PALETTE 进行识别。在应用程序调用动态库时，其资源搜索顺序是，首先在 EXE 中搜索，然后在 DLLs 中搜索，最后在 MFC 动态库中搜索。为了使程序能正确地获取资源，对话框的源文件作如下改动：

1. 在含有 DllMain 函数的 PaletteDLL.cpp 文件中实现获取本模块句柄的函数。

```
HMODULE GetDllModule();
static AFX_EXTENSION_MODULE PaletteDLL = { NULL, NULL };
HMODULE GetDllModule()
{
        return PaletteDLL.hModule;
}
```

2. 对话框头文件 RisPalDlg.h 中，要把含有资源号的无名枚举注销，用同名 UINT 型变量 IDD 代替，并显示出声明对话框的析构函数和保存资源句柄用的句柄变量，源代码如下：

```
#if !defined(AFX_RISPALDLG_H)
#define AFX_RISPALDLG_H
class AFX_EXT_CLASS CRisPalDlg : public CDialog
{
protected:
HINSTANCE m_hInst; //用于保存资源句柄
public:
  CRisPalDlg(CWnd* pParent = NULL);
  virtual ~CRisPalDlg(); //析构函数
//enum { IDD = IDD_DIALOG_PALETTE }; //注释掉无名枚举
```

```
    UINT IDD; //声明对话框资源 ID
protected:
virtual void DoDataExchange(CDataExchange * pDX);
protected:
    DECLARE_MESSAGE_MAP()
};
#endif
```

3. 在对话框实现文件 RisPalDlg. cpp 中，首先声明外部函数 GeDllModule，在构造函数中把 CRisPalDlg IDD 直接用 IDD_DIALOG_PALETTE 代替，在其中对成员变量 IDD 进行赋值，并在构造函数中改变资源搜索顺序，使搜索顺序变为 DLLs - >EXE - >MFC DLLs；最后在析构函数中恢复正常的搜索顺序，源代码如下：

```
#include "stdafx. h"
#include "resource. h"
#include "RisPalDlg. h"
extern HMODULE GetDllModule(); //动态库工程中的获取本模块句柄的函数
//////////////////////////////////
CRisPalDlg: : CRisPalDlg(CWnd * pParent /* =NULL */)
: CDialog(/* CRisPalDlg: : IDD */IDD_DIALOG_PALETTE, pParent)
{
IDD = IDD_DIALOG_PALETTE; //赋值 IDD 使 CRisPalDlg: : IDD 仍然可用
//改变资源搜索顺序
m_hInst =AfxGetResourceHandle(); //获取执文件的资源句柄
AfxSetResourceHandle(GetDllModule()); //获 DLL 的资源句柄，改变资源搜索顺序
    }
CRisPalDlg: : ~CRisPalDlg()//析构函数
{/
  AfxSetResourceHandle(m_hInst);/ 恢复原有资源搜索顺序
}
void CRisPalDlg: : DoDataExchange(CDataExchange * pDX)
{
    CDialog: : DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CRisPalDlg, CDialog)
END_MESSAGE_MAP()
```

上面的修改主要是避免在头文件中出现资源号，并在对话框类的构造函数和析构函数中实现对资源搜索顺序的变换。

经过上面三步修改，程序便能顺利通过编译，并且有效地防止了主程序资源与动态库中资源重名带来运行错误。

三、方法的改进

上面的方法是对单个的对话框导出而作的具体修改，在一个动态库中往往要导出多个对话框类。因此可以从 CDialog 类继承生成一抽象基类 CDllDialog，再让要导出的对话框类从此类继承。CDllDialog 类的实现方法于 CRisPalDlg 相似，在此不加详述。

此外，对二中的方法还可以这样改进：

1. 实现一简单的类 CDllStatus

```
class CDllStatus
{
  private:
      HINSTANCE m_hInst;
  public:
CDllStatus(HMODULE hModule)
      {
m_hInst =AfxGetResourceHandle(); //获取执文件的资源句柄
  AfxSetResourceHandle(hModule); //改变资源搜索顺序
}
 ~CDllStatus()
{/
  AfxSetResourceHandle(m_hInst);/ 改变资源搜索顺序
}
}
```

2. 设置导出类

如果导出类为模态对话框，则可以在导出类中重载 DoModal 函数，在 DoModal 函数中声明有 CDllStatus 变量即可，也可以在导出类中声明 CDllStatus 的成员变量。对于非模态对话框则只能在导出类中声明成员变量。

四、应用与结论

作者在资源信息系统 RIS 开发中的很多含有资源的类大都采用了上述方法导出，有效地防止了资源的错漏。下面是调用自制调色板对话框动态库运行时的界面。

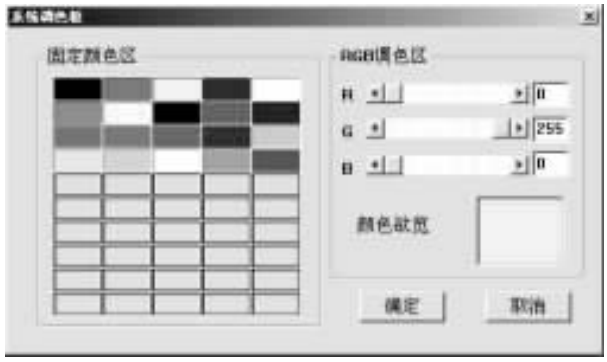


图 1 用动态库导出实现的调色板

本例对话框资源中含有 53 个 BUTTON 资源和三个滚动条资源，不失一般性，本例的成功证明了这种方法的可行性。

参考文献

1. Eugene Olafsen、Kenn Scribner、K. David White 等著 王建华、陈一飞、张焕生等译. MFC Visual C + + 6 编程技术内幕. 机械工业出版社 2000 年 2 月

2. David J. Kruglinski Scot Wingo 等著 希望图书创作室译. Programing Visual C + + 6. 0 技术内幕. 北京希望电子出版社 1999 年 5 月

3. 徐炉清、顿敦. 动态链接库 DLL 的应用. 机械电子出版社，1999 年 4 月

收稿日期：2001 年 2 月 26 日）

增强 MFC 的 CListCtrl 控件

秦 胜

关键词 控件，列表框，ListCtrl，Report

一、引言

在编写用 MFC 编写 Windows 程序的过程中，经常需要用到列表框控件 ListCtrl ，尤其是报表 Report 形式的列表框控件，而且往往需要对其中的表项进行编辑。

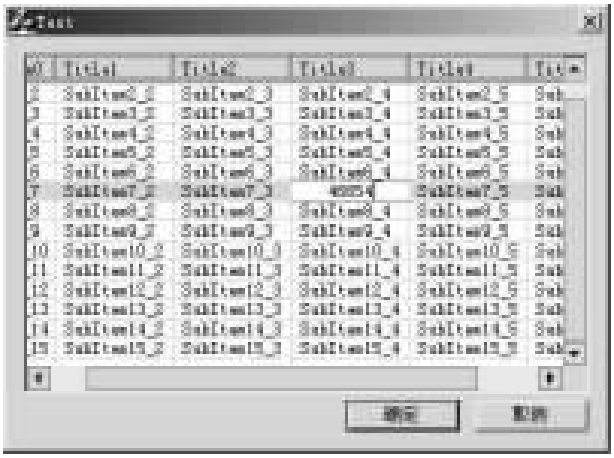


图 1

MFC 提供的 CListCtrl 虽然可以勉强使用，但是需要设置的选项太多，而且只能对首列编辑，使用起来有较大的不便。本文介绍的 CNewListCtrl 将 MFC 提供的 CListCtrl 和 CEdit 进行封装重组，可以轻松实现表格的显示和编辑，如图 1 所示。

二、实现方法

第一步，创建一个工程。

启动 Visual C + + ，选择菜单 File - >New，选择 Projects，选择 MFC AppWizard EXE ，在 Location 中填入工程存放的路径，在 Project name 中填入工程名 例如 Test ，单击 OK，选择 Dialog based，单击 Finish 完成工程的创建。

第二步，编辑 Test 对话框。

添加一个 List 控件，调整到合适的大小，在控件上单击右键，在弹出的菜单中选择 Properties，选择 General，在 ID 中填入控件的 ID 号 IDC_LIST_TEST ，不需要更改其它的选项。在控件上单击右键，在弹出的菜单中选择 ClassWizard，在弹出的对话框中选择 Member Variables，双击 List 控件的 ID 号 IDC_LIST_TEST ，在弹出的对话框中的 Member variables name 中填入与该 List 控件对应的变量名 m_mylist，单击 OK，再单击 OK 关闭 ClassWizard 对话框。

第三步，编辑 Test 对话框对应的源文件。

在 Workspace 窗口中选择 ClassView，展开 CTestDlg 分支，双击 m_mylist，跳到它在源文件中的定义位置，将 CListCtrl 改为 CNewListCtrl，按 CTRL + HOME 键跳到文件头，键入#include < < NewListCtrl. h>>。在 Workspace 中双击 OnInitDialog 跳到该函数的定义位置，在 SetIcon 函数的后面加上以下黑体的代码，用以测试 List 控件。

```
SetIcon(m_hIcon, FALSE);            // Set small icon
int numTitle = 8;      //列数
int numItem = 20;      //每行项数
CString str;
// 1. 先产生 numTitle 列
for( int i = 0; i < numTitle; i + + )
{
str. Format( "Title%d", i + 1 );
m_mylist. InsertColumn( i, str );
}
// 2. 插入 numItem 行
for( i = 0; i < numItem; i + + )
{
str. Format( "Item%d", i + 1 );
      m_mylist. InsertItem( i, str );
for( int j = 1; j < numTitle; j + + )
      {
      str. Format( "SubItem%d_%d", i + 1, j + 1 );
      m_mylist. SetItemText( i, j, str );
      }
}
// 3. 调整列宽
for( i = 0; i < numTitle; i + + )
      m_mylist. SetColumnWidth( i, LVSCW_AUTOSIZE );
```

第四步，编写 CNewListCtrl 类。

4. 1 生成 CNewListCtrl 类和 CMyEdit 类。

在 Workspace 中的 TestClass 上单击右键，选择 New Class 弹出 New class 对话框，在 Name 中填入 CNewListCtrl，在 Base class 中选择 CListCtrl，单击 OK 关闭对话框。此时，在工程中多了 NewListCtrl. cpp 和 NewListCtrl. h 两个文件。再打开 New class 对话框，在 Name 中填入 CMyEdit，在 Base class 中选择 CEdit，单击 Change. . . 按钮，弹出 Change Files 对话框。将 MyEdit. h 和 MyEdit. cpp 分别改为 NewListCtrl. h 和 NewListCtrl. cpp，单击 OK，再击 OK 关闭 New class 对话框。在

Workspace 中双击 CMyEdit，跳到 class CMyEdit 的定义处，将该定义移到 class CNewListCtrl 的定义之前。

4.2 实现 List 控件风格的自动设置。

需要重写 CNewListCtrl 的虚函数 PreSubclassWindow。PubClassWindow HWND hWnd 函数在 MFC 中的含义是将句柄为 hWnd 的窗口作为自己的子类，该窗口的需要由父类来处理的消息让自己来处理。这里是指由 CNewListCtrl 类的实例 m_mylist 来处理对话框中 ID 号为 IDC_LIST_TEST 的 List 控件窗口的需要由父类来处理的消息。PreSubclassWindow 函数在 SubclassWindow 之前执行，允许程序员添加自己的需要在这时执行的代码。这些函数只在初始化时执行一次。这里用 PreSubclassWindow 来修改 List 控件的风格。

在 Workspace 中的 CNewListCtrl 上单击右键，选择 Add Virtual Function...，在弹出的对话框中选择 PreSubclassWindow，单击 Add and Edit 编辑 PreSubclassWindow 函数，在函数中加上下面的黑体代码：

```
void CNewListCtrl::PreSubclassWindow()
{
ModifyStyle(LVS_EDITLABELS, 0L);          //禁止标题编辑
ModifyStyle(0L, LVS_REPORT);              //设为 Report 类型
ModifyStyle(0L, LVS_SHOWSELALWAYS);//始终高亮度被选中的表项
SetExtendedStyle(LVS_EX_FULLROWSELECT |//允许整行选中
LVS_EX_HEADERDRAGDROP |  //允许整列拖动
LVS_EX_GRIDLINES |                        //画出网格线
LVS_EX_ONECLICKACTIVATE |  //单击选中表项
LVS_EX_FLATSB );                          //扁平风格的滚动条
CListCtrl::PreSubclassWindow();
}
```

如果现在编译运行程序，可以看到，此时 List 控件虽然不能编辑表项，但已自动具有图 1 所示的风格。

4.3 实现表项的编辑。

基本思想是在需要编辑的时候在表项的位置创建一个编辑控件，编辑控件中用表项的文字填充；当编辑完成时，取出编辑控件中的文字填到表项中，然后销毁编辑控件。

在 Windows 中，编辑列表项的传统操作方法是先左键单击选中该表项，再次单击开始编辑表项。当按下回车键或者将输入焦点转移到别的控件上来结束编辑时，编辑结果有效，此时应当用编辑控件中的文字替换表项中的文字；当按下 ESC 键来结束编辑时，编辑结果无效，此时应当保持表项中的文字不变。现在按此规则来实现表项的编辑。

首先要为 CNewListCtrl 类添加成员变量，在 Workspace 中双击 CNewListCtrl 跳到 class CNewListCtrl 的定义处，添加下列黑体代码：

```
protected:
    BOOL m_bEditing;    //是否有表项正在被编辑状态
    CMyEdit m_edit;          //编辑控件
    int m_nItem;             //被编辑表项的行号
    int m_nSubItem;          //被编辑表项的列号
```

在构造函数 CNewListCtrl CNewListCtrl 函数中加上

```
    m_bEditing = FALSE;
```

想知道用户何时想编辑表项，必须监视鼠标左键的按下消息，为此，需要重写 OnLButtonDown 函数。在 Workspace 中的 CNewListCtrl 上单击右键，选择 Add Windows Message Handler...，在弹出的对话框中找到 WM_LBUTTONDOWN，单击 Add and Edit 编辑 OnLButtonDown 函数，加上下列黑体代码：

```
void CNewListCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
  POSITION pos;
  BOOL bSelected = FALSE;
  // 检查是否有 Item 正被编辑
  if( m_bEditing ==TRUE)
       goto defalt_session;
  // 检查是否有 Item 被选中，没有时不进入编辑
  pos = GetFirstSelectedItemPosition();
  if( pos )
  {
  // 得到被点击的 Item
  LVHITTESTINFO testinfo;
  testinfo.pt.x = point.x;
  testinfo.pt.y = point.y;                  //点击时的鼠标位置
testinfo.flags = LVHT_ONITEMLABEL; //点击的必须是标题
       if( SubItemHitTest(&testinfo) <0 )
goto defalt_session;     //没有点在有效区域，不进入编辑
m_nItem = testinfo.iItem;                 //被点击表项的行号
  m_nSubItem = testinfo.iSubItem;    //被点击表项的列号
       // 检查该表项是否被选中，没被选中不进入编辑
       while( pos )
            if( m_nItem ==GetNextSelectedItem(pos) )
            {
                 bSelected = TRUE;
                 break;
            }
       if( bSelected ==FALSE )
goto defalt_session;     //没有点在有效区域，不编辑
       // 开始编辑
       m_bEditing = MyBeginEdit();
       return;
  }
defalt_session:
  CListCtrl::OnLButtonDown(nFlags, point);
}
```

其中 MyBeginEdit 函数完成开始编辑前的准备工作 在 Workspace 中单击右键 选择 Add Member Function... 在弹出的对话框中 Fuction Type 填入 BOOL Function Declaration 填入 MyBeginEdit 选择 Protected 单击 OK 编辑 MyBeginEdit 函数 添加以下的黑体代码

```
BOOL CNewListCtrl::MyBeginEdit()
{
  // 得到被编辑表项的区域
  CRect rect;
  if( GetSubItemRect(m_nItem, m_nSubItem, LVIR_LABEL,
```

```
rect) = =FALSE )
        return FALSE;
    // 创建编辑控件
int style =WS_CHILD|WS_CLIPSIBLINGS|WS_EX_TOOLWIN-
DOW |WS_BORDER;
if( m_edit. Create(style, rect, this, ID_MYEDIT) = =FALSE )
        return FALSE;
    // 取被编辑表项的文字
CString txtItem = GetItemText( m_nItem, m_nSubItem );
// 取出的文字填写到编辑控件
    m_edit. SetWindowText( txtItem );
    m_edit. SetFocus();
    m_edit. SetSel( 0, -1 );
    m_edit. ShowWindow( SW_SHOW );
return TRUE;
}
```

然后在该函数前面加上

```
#define ID_MYEDIT   101  //编辑控件的 ID 号
```

为了处理编辑完成时的工作，还要添加 MyEndEdit 函数，在 Workspace 中单击右键，选择 Add Member Function. . . ，在弹出的对话框中，Fuction Type 填入 void，Function Declaration 填入 MyEndEdit BOOL bValidate ，选择 Public，单击 OK 编辑 MyEndEdit 函数，添加以下的黑体代码。参数 bValidate 指明本次编辑结果是否有效。

```
void CNewListCtrl: : MyEndEdit( BOOL bValidate )
{
    // 编辑结果是有效的,重设被编辑表项的文字
    if( bValidate )
    {
        CString txtItem;
        m_edit. GetWindowText( txtItem );
        SetItemText(m_nItem, m_nSubItem, txtItem);
    }
// 销毁编辑窗口
    m_edit. DestroyWindow();
    m_bEditing = FALSE;
}
```

此时再编译运行程序，可以看到如图 2 的结果。这个结果仍有缺陷，首先，一旦进入了编辑过程，编辑控件就不能去除了，这是因为没有告诉程序何时结束编辑；其次，在编辑控件中按下 ESC 键或回车键，对话框直接就关闭了，这是因为这两个键按下的消息被对话框截取当成了关闭消息处理了；再次，编辑控件中的文字明显地与周围表项中的文字不协调，还需要对字体进行设置。这些都需要在编辑控件内部解决。

图 2

第五步，编写 CMyEdit 类。

5. 1 先解决字体问题，方法是在编辑控件创建时将字体设成与列表框的表项字体一样。

先为 CMyEdit 添加成员变量 m_font。在 Workspace 中 CMyEdit 上单击右键，选择 Add Member Variable. . . ，在弹出的对话框中 Variable Type 填入 CFont，在 Variable name 中填入 m_font，单出 OK 加入该变量。然后需要处理 CMyEdit 控件窗口的创建消息 WM_CREATE。在 Workspace 中 CMyEdit 上单击右键，选择 Add Windows Message Handle. . . ，在弹出的对话框中找到 WM_CREATE，单击 Add and Edit 编辑 OnCreate 函数，加上下面的黑体代码：

```
int CMyEdit: : OnCreate(LPCREATESTRUCT lpCreateStruct)
{
  if (CEdit: : OnCreate(lpCreateStruct) = = -1)
      return -1;
  // 改变字体
  LOGFONT logfont;
  logfont. lfHeight = -12;
  logfont. lfWidth = 0;
  logfont. lfEscapement = 0;
  logfont. lfOrientation = 0;
  logfont. lfWeight = 400;
  logfont. lfItalic = 0;
  logfont. lfUnderline = 0;
  logfont. lfStrikeOut = 0;
  logfont. lfCharSet = 134;
  logfont. lfOutPrecision = 0;
  logfont. lfClipPrecision = 0;
  logfont. lfQuality = 0;
  logfont. lfPitchAndFamily = 2;
  strcpy( logfont. lfFaceName, "宋体" );
  m_font. DeleteObject();
  if( m_font. CreateFontIndirect(& logfont) )
      SetFont( & m_font );
  return 0;
}
```

5. 2 结束编辑过程。编辑控件失去输入焦点的时候，编辑过程应当结束，因此应该重写 OnKillFocus 函数。在 Workspace 的 CMyEdit 上单击右键，选择 Add Windows Message Handle. . . ，在弹出的对话框中找到 WM_KILLFOCUS，单击 Add and Edit 编辑 OnKillFocus 函数，加上下面的黑体代码：

```
void CMyEdit: : OnKillFocus(CWnd * pNewWnd)
{
  // 得到父窗口,并通知父窗口结束编辑过程
  CNewListCtrl * parent = (CNewListCtrl * )GetParent();
  if( parent )
```

```
        parent->MyEndEdit( m_bInputValid );
m_bInputValid = TRUE;
    CEdit::OnKillFocus(pNewWnd);
}
```

其中 m_bInputValid 是成员变量，只有在 ESC 键按下时被置为 FALSE，表示编辑结果无效。添加这个变量，在 Workspace 中 CMyEdit 上单右键，选择 Add Member Variable...，在弹出的对话框中的 Variable Type 中填入 BOOL，在 Variable Name 中填入 m_bInputValid，选择 private，单击 OK 加入该变量。然后在构造函数 CMyEdit　CMyEdit　中加入：

```
m_bInputValid = TRUE;
```

至此，已经完全实现了对 List 表项的编辑。但美中不足的是，如果在编辑过程中用户一不小心按下了 ESC 键或回车键，对话框会立刻关闭，给用户带来不便。

5.3 处理 ESC 键和回车键按下的消息。

在通常情况下，对话框中的控件不能得到处理 ESC 键和回车键按下的消息的机会。在 MFC 的消息处理机制中，消息从消息队列里取出后，并不直接发送给它的目标窗口进行处理，而是首先按照从目标窗口到应用程序主窗口的次序进行传递，在传递过程中由各个窗口对消息进行预解释 PreTranslate 。只有在所有窗口都没有对该消息作出预解释的情况下，该消息才能被发送给目标窗口；否则，该消息被第一个能预解释它的窗口截流并处理，目标窗口将失去对消息的处理机会。这个过程参见图 3。

缺省的情况下，控件的消息预解释函数不解释 ESC 键和回车键按下的消息，而对话框窗口则将它们解释为用户单击

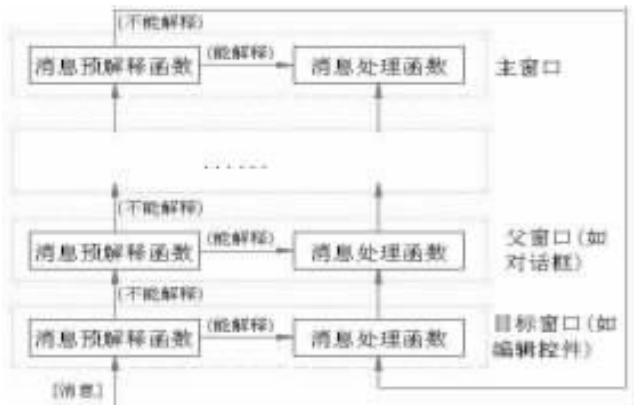


图 3　MFC 的消息传递流程

了 Cancle 或 OK 按钮的消息，并加以处理，所以控件没有机会再处理这个消息。

很明显，现在必须对 ESC 键和回车键按下的消息作预解释，这个过程通过重写 PreTranslateMessage 函数来实现。在 Workspace 中 CMyEdit 上单击右键，选择 Add Virtual Function...，在弹出的对话框中找到 PreTranslateMessage，单击 Add and Edit 编辑它，加入下面的黑体代码：

```
BOOL CMyEdit::PreTranslateMessage(MSG * pMsg)
{
  //拦截 ESC 键和 Enter 键按下的消息，解释为 WM_KILLFOCUS 消息
  if( pMsg->message == WM_KEYDOWN )
  {
      if( pMsg->wParam == 13 )          //回车键
          pMsg->message = WM_KILLFOCUS;
      else if( pMsg->wParam == 27 )     //ESC 键
      {
          m_bInputValid = FALSE;
          pMsg->message = WM_KILLFOCUS;
      }
  }
  return CEdit::PreTranslateMessage(pMsg);
}
```

编译运行程序，可以看到 List 控件的表现完全达到预期的目的，如图 1 所示。

打开 NewListCtrl.cpp，留下 StdAfx.h 和 NewListCtrl.h 文件的#include 包含语句，去掉其余的#include 语句，保存文件。现在 NewListCtrl.h 和 NewListCtrl.cpp 可以在任何其它的 MFC 工程中使用，只须将它们添加到工程，包含 NewListCtrl.h 文件，使 CNewListCtrl 类的实例与 List 控件的窗口相关联即可 如第二、三步所示 。

该控件对列表项的插入、删除、添加等调用接口与 CListCtrl 的接口完全一样。

小结

本文介绍了如何改写 MFC 的 CListCtrl 控件，使其成为一个方便实用的报表形式控件。同时也介绍了 MFC 中消息传递流程及 PreSubclassWindow 函数和 PreTranslateMessage 函数的作用。

（收稿日期：2001 年 3 月 26 日）

（上接第 45 页）

```
na.com">Copyright(C) Kenny, 2001 </A> </p></TD>
    <td><P  align=right><INPUT  tabindex=1  style="
FONT: 9pt 宋体; WIDTH: 6em" type="submit" value="确定" onClick="jscript:returnParameter();"></P></td>
  </TR>
</TABLE>
</body>
</html>
```

在 HTML 文件中可以将按钮指定为 Submit 类型来接收回车键的输入。可以调用 window.Close　来关闭窗口。Tab 键顺序可以用 tabIndex 来指定，对话框的大小可以用 <HTML>标签的属性来指定，标题由<TITLE> 标签来指定。更多的不再详述，请参阅有关 HTML 的资料。

（收稿日期：2001 年 2 月 8 日）

在 Delphi 中动态创建和拖动控件

赖 炜 齐 欢

一、引言

Delphi 很好地将面向对象编程技术和 Windows 可视化编程技术相结合，并建立在 VCL（Visual Component Library）的基础上，提供了一种崭新而且完备的软件开发工具。无论是数据库、客户机/服务器、多级、Internet/Intranet 解决方案，还是控件与功能，程序员都可以使用 Delphi 并采用适当的技术得以实现。

Delphi 4 的特性包括：速度非常快的 AppBrower 编辑器，对 Windows 常用控件与 Windows 98 的改进支持，改进的 OLE（Object Linking and Embedding）与 COM（Component Object Model）支持，扩展的数据库组件与 DBE 以及 VCL 核心类的新特性，包括对停放、约束和位置点控件的支持。

二、动态创建控件

以按钮（Tbutton）为例，Tbutton 类的 Create 方法可以动态地创建并初始化一个 Tbutton 的实例，它的说明如下：

```
constructor Create(AOwner: TComponent); override;
```

其中 Tcomponent 是按钮的父对象。

Tbutton 的 Create 方法会：(1)调用了继承的 Create 方法，(2)把按钮的 ControlStyle 属性设置为 csSetCaption、csOpaque 和 csDoubleClicks，(3)将按钮的长度和宽度分别设置为 75 pixels 和 25 pixels，(4)将按钮的 TabStop 属性设置为 True。

动态地添加按钮以后，可以按照需要对按钮的其他属性（Caption、Left、Top、Width、Height 等）赋值，从而得到程序所需要实现的功能。例如以下的程序段：

```
var btnClose: Tbutton;
. . . . . .
Procedure Tform1. FormActive (sender: TObject);
begin
btnClose: =Tbutton. Create;
btn. Width: =form1. ClientWidth;
btn. Height: =form1. ClientHeight;
btn. Top: =0;
btn. Height: =0;
btn. Caption: =´Press here to close this form´;
. . . . . .
end;
```

上面的程序段在窗口 form1 被激活时创建一个名为 btnClose 的 Tbutton 对象，并且用它充满整个窗体，btnClose 的 Caption 被设置为“Press here to close this form”。当然在窗体事件中也必须释放该按钮，即必须在 FormClose 事件中添加如下代码：

```
btnClose. Free;
```

否则下次激活该窗体时会发生错误。

如何实现 btnClose 按钮关闭窗口的功能呢？这需要在 btnClose 的 OnClick 事件中添加 Form1. Close 代码，但是由于按钮 btnClose 是动态创建的，不可以直接在 btnClose 的 OnClick 事件中添加代码，因此可以采用如下方法：

首先，在 Form1 中声明一个 btnClick 过程，并添加如下代码：

```
Procedure Tform1. btnClick (Sender: TObject);
begin
 Form1. Close;
end;
```

然后在创建 btnClose 按钮对象时同时对它的 OnClick 过程进行处理：

```
btnClose. OnClick: =btnClick;
```

则可以实现 btnClose 关闭窗体的功能。

三、控件的拖动

在控件的拖放中，最关键的两个事件是 OnDragOver 和 OnDragDrop 事件。通过重载 DragOver 过程并添加适当的代码，可以在 OnDragOver 事件句柄被调用前做程序员自己想做的工作。它的形式如下：

```
Procedure DragOver(Source: TObject; X, Y: Integer; State:
TDragState; var Accept: Boolean);
```

其中 Accept 如果设置为 True，就表明用户可以放下正在拖动的控件，反之就是不允许放下控件；Source 是正在被拖动的控件。同样的，通过重载 DragOver 过程并添加适当的代码，可以在 OnDragOver 事件句柄被调用前做程序员自己想做的其他工作。

以下面的代码为例：

```
procedure TForm1. ListBox1DragOver(Sender, Source: TOb-
ject; X, Y: Integer; State: TDragState; var Accept: Boolean);
begin
  Accept : = Source is TLabel;
end;
procedure TForm1. ListBox1DragDrop(Sender, Source: TOb-
ject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font : = (Source as TLabel). Font;
      Color : = (Source as TLabel). Color;
    end;
```

（下转第 55 页）

在 Visual C++ 6.0 下应用 Win32 系统钩子技术

郎 锐

摘 要 Windows 系统的核心是事件驱动的运行机制，整个操作系统都是通过消息的传递来实现的。而 Win32 系统钩子（HOOKS）则是 Windows 系统中的一个非常重要的系统接口，通过这个系统接口可以截获并处理排在消息循环队列中准备发送到其他应用程序的各种消息，以实现某些特定的功能。本文着重讨论了 Win32 系统钩子的运行机制，特别是对全局钩子。并通过用 VC6 编制的一个简单的鼠标钩子示例程序，对 Win32 全局钩子的运行机制、Win32 DLL 的特点以及本机进程间数据共享等相关问题进行了简单的阐述。

关键字 Visual C++ 系统钩子 DLL 数据共享

一、 引言

钩子的本质是一段用以处理系统消息的程序，通过系统调用，把它挂入系统。钩子的种类很多，每种钩子可以截获并处理相应的消息，每当特定的消息发出，在到达目的窗口之前，钩子程序先行截获该消息、得到对此消息的控制权。此时钩子函数可以对截获的消息进行加工处理，甚至可以强制结束消息的传递。这有点类似于 MFC 中的 PreTranslateMessage 函数，所不同的是该函数只能用于拦截本进程中的消息，而对系统消息则无能为力。

二、 Win32 系统钩子的实现

每种类型的钩子均由系统来维护一个钩子链，最近安装的钩子位于链的开始，拥有最高的优先级，而最先安装的钩子则处在链的末尾。要实现 Win32 的系统钩子，首先要调用 SDK 中的 API 函数 SetWindowsHookEx 来安装这个钩子函数，其原型是：HHOOK SetWindowsHookEx int idHook HOOKPROC lpfn HINSTANCE hMod DWORD dwThreadId ，其中，第一个参数是钩子的类型，常用的有 WH_MOUSE、WH_KEYBOARD、WH_GETMESSAGE 等；第二个参数是钩子函数的地址，当钩子钩到任何消息后便调用这个函数；第三个参数是钩子函数所在模块的句柄；第四个参数是钩子相关函数的 ID 用以指定想让钩子去钩哪个线程，为 0 时则拦截整个系统的消息，此时为全局钩子；如果指定确定的线程，即为线程专用钩子。

全局钩子函数必须包含在 DLL（动态链接库）中，而线程专用钩子则可包含在可执行文件中。得到控制权的钩子函数在处理完消息后，可以调用另外一个 SDK 中的 API 函数 CallNextHookEx 来继续传递该消息。也可以通过直接返回 TRUE 来丢弃该消息，阻止该消息的传递。

使用全局钩子函数时需要以 DLL 为载体，VC6 中有三种形式的 MFC DLL 可供选择，即 Regular statically linked to MFC DLL（标准静态链接 MFC DLL）、Regular using the shared MFC DLL（标准动态链接 MFC DLL）以及 Extension MFC DLL（扩展 MFC DLL）。第一种 DLL 在编译时把使用的 MFC 代码链接到 DLL 中，执行程序时不需要其他 MFC 动态链接类库的支持，但体积较大；第二种 DLL 在运行时动态链接到 MFC 类库，因而体积较小，但却依赖于 MFC 动态链接类库的支持（这两种 DLL 均可被 MFC 程序和 Win32 程序使用）；第三种 DLL 的也是动态连接，但做为 MFC 类库的扩展，只能被 MFC 程序使用。

三、Win32 DLL

Win32 DLL 的入口和出口函数都是 DLLMain，这同 Win16 DLL 是有区别的。只要有进程或线程载入和卸载 DLL 时，都会调用该函数，其原型是：

BOOL WINAPI DllMain HINSTANCE hinstDLL DWORD fdwReason LPVOID lpvReserved

其中，第一个参数表示 DLL 的实例句柄；第三个参数系统保留；第二个参数指明了当前调用该动态链接库的状态，它有四个可能的值：DLL_PROCESS_ATTACH（进程载入）、DLL_THREAD_ATTACH（线程载入）、DLL_THREAD_DETACH（线程卸载）、DLL_PROCESS_DETACH（进程卸载）。在 DLLMain 函数中可以通过对传递进来的这个参数的值进行判别，根据不同的参数值对 DLL 进行必要的初始化或清理工作。由于在 Win32 环境下，所有进程的空间都是相互独立的，这减少了应用程序间的相互影响，但大大增加了编程的难度。当进程在动态加载 DLL 时，系统自动把 DLL 地址映射到该进程的私有空间，而且也复制该 DLL 的全局数据的一份拷贝到该进程空间，每个进程所拥有的相同的 DLL 的全局数据其值却并不一定是相同的。当 DLL 内存被映射到进程空间中，每个进程都有自己的全局内存拷贝，加载 DLL 的每一个新的进程都重新初始化这一内存区域，也就是说进程不能再共享 DLL。因此，在 Win32 环境下要想在多个进程中共享数据，就必须进行必要的设置。一种方法便是把这些需要共享的数据单独分离出来，放置在一个独立的数据段里，并把该段的属性

设置为共享，建立一个内存共享的 DLL。

四、全局共享数据的实现

可以用#pragma data_seg 建立一个新的数据段并定义共享数据，其具体格式为：

```
#pragma data_seg ( ″shareddata″)
HWND sharedwnd =NULL; //共享数据
#pragma data_seg()
```

所有在 data_seg pragmas 语句之间声明的变量都将在 shareddata 段中。仅定义一个数据段还不能达到共享数据的目的，还要告诉编译器该段的属性，有两种方法可以实现该目的（其效果是相同的），一种方法是在 . DEF 文件中加入如下语句：

```
SETCTIONS
shareddata READ WRITE SHARED
```

另一种方法是在项目设置链接选项中加入如下语句：

```
/SECTION: shareddata, rws
```

五、鼠标钩子程序示例

本示例程序用到全局钩子函数，程序分两部分：可执行程序 MouseDemo 和动态链接库 MouseHook。首先编制 MFC 扩展动态链接库 MouseHook. dll

（一）选择 MFC AppWizard DLL 创建项目 Mousehook

（二）选择 MFC Extension DLL （MFC 扩展 DLL）类型

（三）通过 Project 菜单的 AddToProject 子菜单的 “New. . . ”添加头文件 MouseHook. h

（四）在头文件中建立钩子类

```
class AFX_EXT_CLASS CMouseHook: public CObject
{
public:
CMouseHook();        //钩子类的构造函数
~CMouseHook();       //钩子类的析构函数
BOOL StartHook(HWND hWnd);        //安装钩子函数
BOOL StopHook();                  //卸载钩子函数
};
```

（五）在 MouseHook. cpp 文件中加入#include “MouseHook. h ”语句

（六）加入全局共享数据变量

```
#pragma data_seg(″mydata″)
HWND glhPrevTarWnd =NULL;  //上次鼠标所指的窗口句柄
HWND glhDisplayWnd =NULL;  //显示目标窗口标题编辑框的句柄
HHOOK glhHook =NULL;              //安装的鼠标勾子句柄
HINSTANCE glhInstance =NULL;  //DLL 实例句柄
#pragma data_seg()
```

（七）在 DEF 文件中定义段属性

```
SECTIONS
mydata READ WRITE SHARED
```

（八）在主文件 MouseHook. cpp 的 DllMain 函数中加入保存 DLL 实例句柄的语句

```
extern ″C″ int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{)
  UNREFERENCED_PARAMETER(lpReserved);
       if (dwReason = = DLL_PROCESS_ATTACH
       {
  if (!AfxInitExtensionModule(MouseHookDLL, hInstance))
                  return 0;
       new CDynLinkLibrary(MouseHookDLL);
  lhInstance =hInstance;        //插入保存 DLL 实例句柄
       }
  else if (dwReason = = DLL_PROCESS_DETACH)
       {
  AfxTermExtensionModule(MouseHookDLL);
  }
       return 1;   // ok
}
```

这个函数最重要的部分是调用 AfxInitExtensionModule ，它初始化 DLL，使它在 MFC 框架中正确的工作。它需要传递给 DllMain 的 DLL 实例句柄和 AFX_EXTENSION_MODULE 结构，结构中存在着对 MFC 有用的信息。

九 类 CMouseHook 的成员函数的具体实现

```
Cmousehook: : Cmousehook() //类构造函数
{
Cmousehook: : ~Cmousehook() //类析构函数
{
     stophook();
}
BOOL Cmousehook: : starthook(HWND hWnd)  // 安装钩子并设定接收显示窗口句柄
{
BOOL bResult =FALSE;
glhHook =SetWindowsHookEx(WH_MOUSE, MouseProc, glhInstance, 0);
if(glhHook! =NULL)
bResult =TRUE;
glhDisplayWnd =hWnd;    //设置显示目标窗口标题编辑框的句柄
return bResult;
}
BOOL Cmousehook: : stophook()    //卸载钩子
{
BOOL bResult =FALSE;
if(glhHook)
{
bResult =  UnhookWindowsHookEx(glhHook);
if(bResult)
{
glhPrevTarWnd =NULL;
glhDisplayWnd =NULL; //清变量
 glhHook =NULL;
}
}
return bResult;
}
```

十 钩子函数的实现

```
LRESULT WINAPI MouseProc(int nCode, WPARAM wparam, LPARAM lparam)
{
   LPMOUSEHOOKSTRUCT pMouseHook = (MOUSEHOOKSTRUCT FAR * ) lparam;
 if (nCode>=0)
 {
HWND glhTargetWnd = pMouseHook ->hwnd;    //取目标窗口句柄
HWND ParentWnd = glhTargetWnd;
while (ParentWnd != NULL)
{
glhTargetWnd = ParentWnd;
ParentWnd = GetParent(glhTargetWnd);  //取应用程序主窗口句柄
}
if(glhTargetWnd! = glhPrevTarWnd)
{
char szCaption[100];
GetWindowText(glhTargetWnd, szCaption,100);  //取目标窗口标题
if(IsWindow(glhDisplayWnd))
SendMessage(glhDisplayWnd, WM_SETTEXT, 0, (LPARAM)(LPCTSTR)szCaption);
glhPrevTarWnd = glhTargetWnd;     //保存目标窗口
}
}
return CallNextHookEx(glhHook, nCode, wparam, lparam);  //继续传递消息
}
```

编译完成便可得到运行时所需的鼠标钩子的动态链接库 MouseHook. dll 和链接时用到的 MouseHook. lib。

六、集成

下面新建一调用鼠标钩子动态链接库的钩子可执行程序：

（一）用 MFC 的 AppWizard EXE 创建项目 MouseDemo

（二）选择 “基于对话应用”，其余几步均为缺省

（三）在对话框上加入一个编辑框 IDC_EDIT1

（四）在 MouseDemo. h 中加入对 Mousehook. h 的包含语句：#Include “Mousehook. h”

（五）在 CMouseDemoDlg. h 的 CMouseDemoDlg 类定义中添加私有数据成员：

CMouseHook m_hook;

（六）在 OnInitDialog 函数的 “TODO 注释”后添加：

CWnd * pwnd = GetDlgItem(IDC_EDIT1); //取得编辑框的类指针

m_hook. StartHook(pwnd ->GetSafeHwnd()); //取得编辑框的窗口句柄并安装钩子

（七）链接 DLL 库，即把 Mousehook. lib 加入到项目设置链接标签中

（八）把 MouseHook. h 和 MouseHook. lib 复制到 MouseDemo 工程目录中，MouseHook. dll 复制到 Debug 目录下。编译执行程序即可。当鼠标滑过窗口时便会在编辑框中将此窗口的标题显示出来。

结论

系统钩子具有相当强大的功能，通过这种技术可以对几乎所有的 Windows 系统消息进行拦截、监视、处理。这种技术可以广泛应用于各种软件，尤其是需要有监控、自动记录等对系统进行监测功能的软件。本程序只对鼠标消息进行拦截，在 Win32 环境下对键盘、端口等应用此技术完成特定的功能。

参考文献

1. [美]David Bennett 著 徐军等译. Visual C++5 开发人员指南. 机械工业出版社

2. 山东大学威海分校. Microsoft Windows 环境与编程基础. 电子系统工程系出版

3. [美] Kate Gregory 著，康博工作室译. Visual C++5 开发使用手册. 机械工业出版社

4. 王华编著. Visual C++ 6.0 编程实例与技巧. 机械工业出版社

收稿日期：2001 年 3 月 26 日）

（上接第 52 页）

```
  end;
end;
```

该窗体有一个 TListBox 控件和三个 TLabel 控件，每个控件的 Fonts 和 Color 属性各不相同。用户可以随意选择一个 Label 控件，把它拖到 ListBox 控件上并且放下，当 Label 放下以后，ListBox 中的各项就与放下的 Label 具有相同的字体和颜色属性。

在 ListBox1 的 ListBox1DragOver 过程中运用了逻辑表达式 Source is Tlabel 表明如果试图放下的控件是 Tlabel 控件，则允许放下，否则不允许。

在 ListBox1 的 ListBox1DragDrop 过程中用到了 as 运算符，它执行一个带有校验的类型选择。如果没有上面的 if 语句的话，当 Sender 确实是一个 TListBox 对象 （Sender as TLabel）时，将返回一个 Sender 的类型为 Tlabel 的引用；当 Sender 不是 TlistBox 对象时，将引发一个异常。为了避免引发异常，上述程序加上 if 语句先行校验。

参考文献

1. Marco Cantu. Mastering Delphi 4. 电子工业出版社，1999

2. Charlie Calvert. Delphi 4 Unleashed. 机械工业出版社，1999

收稿日期：2001 年 3 月 8 日）

C + + Builder5. 0 多层数据库应用程序设计

王忠贵

摘 要 本文介绍了 C + + Builder5. 0 中多层数据库的实现原理和方法，并结合一个简单的实例程序加以具体说明。

关键词 C + + Builder，多层数据库

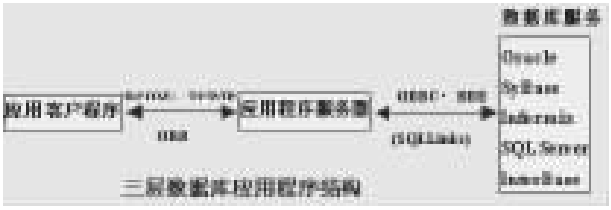
一、多层数据库应用程序设计简单原理

1. 多层数据库的概述

网络数据库或分布式数据库应用程序大多采用多层数据库体系结构。多层数据库体系结构中服务器端集中实现了应用逻辑，应用客户程序主要在于显示数据和与用户交互上。这样，在实现分布式数据处理中，可以把一个应用程序分布在几个机器上运行，可以提高应用程序的性能，有利于安全，同时又不至于使用户界面变得非常复杂。本文先对 C + + Builder5. 0 多层数据库设计方法和原理作简要说明，再举一个多层数据库动态查询例子，具体阐明 C + + Builder5. 0 多层数据库应用程序设计。

2. C + + Builder5. 0 三层数据库应用程序结构

Borland 公司的 Delphi 和 C + + Builder 典型的三层数据库应用程序结构如下：第一层是数据库服务器，第二层是应用程序服务器，第三层是应用客户程序。其中，最关键的是应用程序服务器，它在三层数据库应用程序体系中起承上启下的作用。数据库服务器是诸如 InterBase 、Oracle 、Sybase 、MS SQL Server 等网络数据库系统。应用程序服务器通过 BDE 数据库引擎）、ODBC 开放数据库互联）、 SQL * NET 等软件访问数据库服务器，而应用客户程序则通过 DCOM 分布式组件对象）、TCP / IP 或 ORB 与应用程序服务器通讯。结构框图如下：



三层数据库应用程序结构

3. C + + Builder5. 0 中实现多层数据库的控件及属性、各控件间的关系、客户程序与应用服务器之间的消息传递过程

（1） 客户端多层数据库控件

● TDCOMConnection 通过 DCOM 与应用程序服务器连接，是一种最直接的连接方式，它不需要专门的运行期软件，但必须安装 DCOM95 或 DCOM98，运行 Dcomcnfg. exe 作相应的设置。TDCOMConnection 的 ComputerName 属性设置应用程序服务器所在的计算机名，ServerName 属性设置应用程序服务器名称。

● TSocketConnection 通过 TCP / IP 与应用程序服务器连接，适合范围非常广，但需要专门运行期软件 ScktSrvr. exe 或 ScktSrvc. exe。TSocketConnection 的 Host 属性设置应用程序服务器所在的计算机名，ServerName 属性设置应用程序服务器名称。

● TClientDataSet TClientDataSet 功能类似单层数据库程序中的 TTable、TQuery. 但它不依赖 BDE，只需要一个动态链接库 DBCLIENT. DLL。

（2） 服务器端多层数据库控件

● TRemoteDataModul 是一个支持双重接口的自动化服务器，这种远程数据模块适应于使用 DCOM、TCP / IP、 OLEEnterprise 连接。

● TMTSDataModule 是一个支持双重接口的自动化服务器，用这种类型的远程数据模块创建的应用服务器是动态链接库，它提供了基于角色的安全机制，每个客户都扮演一个角色，决定否能访问这种远程数据模块。适应于使用 DCOM、TCP / IP、 OLEEnterprise 连接

● TDataSetProvider 客户程序是通过应用程序服务器提供 IProvider 的接口获得数据的，有一数据集就需一个 TDataSetProvider 控件。

（3） 消息传递过程

在多层数据库应用程序中，客户请求应用服务器数据是根据客户端 TClientDataSet 控件的 FetchOnDemand 属性值确定的，分两种情况：1. 客户端 TClientDataSet 控件的 FetchOnDemand 属性为 true 时：客户请求数据是自动进行的，应用服务器对客户请求的响应也是自动的，它自动检索数据、把数据打包，然后把数据包传递给客户。2. 客户端 TClientDataSet 控件的 FetchOnDemand 属性为 false 时：客户端应用程序必须显示调用 TClientDataSet 控件的 GetNextPacket 函数。此函数用于向应用服务器检索下一个数据包，并把检索到的数据包添加到前一次检索到的数据包后面，并返回实际检索到的记录数，其中 TClientDataSet 控件的 PacketRecords 属性用于设置一个数据包中最多能容纳的记录数，设为 - 1 表示容纳数据集中的所有记录。

当客户端应用程序调用 AppUpdate 函数向数据库服务器申请更新数据时，它把消息传递给应用服务器端，应用服务器端调用 TDataSetProvider 控件的 AppUpdate 函数向数据库服务器更新数据。更新记录是逐条进行的，每更新一条记录前会触发 TDataSetProvider 控件 OnUpdateRecord 事件 因此应用服务器端还有机会对其中的数据进行编辑加工。

在多层体系结构中，客户程序要与应用服务器交换数据，首先必须获得应用服务器的 IProvider 接口，这就要用到 TClientDataSet 控件的 RemoteServer 属性、ProviderName 属性。RemoteServer 属性用于指定客户端的 MIDAS 连接控件，设置 RemoteServer 属性，就可以选择一个相应 IProvider 接口。

二、动态传递 SQL 语句的三层数据库应用示范程序

编程思路：应用服务器端数据模块添加三个方法 GetAliasNames TVariant * AliasNames 、SetAliasName BSTR AliasName 、SetSQL BSTR SQLString 。GetAliasNames 实现获得服务器端所有数据库别名，SetAliasName 设置应用服务器端 TQuery 的 AliasName 值，SetSQL 设置应用服务器端 TQuery 的 SQL 属性值。当客户端应用程序启动时调用应用服务器端数据模块的 GetAliasNames 函数获得服务器端所有数据库别名，客户端程序选择其中一个别名，并调用 SetAliasName BSTR AliasName 函数设置服务器端数据模块中数据库控件 TQuery 的 AliasName 值，然后调用 SetSQL BSTR SQLString 函数设置应用程序服务器端 TQuery 的 SQL 属性。当应用程序客户端 TClientDataSet 的 Active 设为 true 时，应用程序服务器自动把查询的结果传到客户端显示。此程序的要点是在服务器端数据模块添加上述三个方法。

1. 创建一个简单的数据库应用服务程序

按下列步骤创建简单的数据库服务程序。

1 创建新的应用程序，选择 File | NewApplication 菜单，然后在 New| Items 对话框 Multitier 页，单击 RemoteDataModule 图标，输入一个合适的类名 DataServer。然后进入类型库编辑器（Type Library Editor）。

2 给应用服务程序添加方法，选择所要使用的接口 IDataServer，单击类型库编辑器工具栏中的 Method 按钮添加三个方法 GetAliasNames、SetAliasName、SetSQL，并设置相应参数名。

3 在数据模块中加入 TQuery、TDataSetProvider 2 个数据库控件，其属性 Name 分别为 Query1、DataSetProvider1。设 DataSetProvider1 的 DataSet 属性为 Query1。

4 将窗体单元文件另存为 DataServerMain 自动化对象（DataServer）实现单元文件另存为 DataServer 项目另存为 AppDataServer。

在自定义 IDataServer 对象的 TDataServerImpl. cpp 文件中实现上述三个方法。其它文件的代码均可自动生成。

```
STDMETHODIMP TDataServerImpl: : GetAliasNames (TVariant * AliasNames)
{ int i;
  HRESULT hr;
  TStrings * DBNames;
  DBNames = new TstringList;
  _try
  { try
{Session - >GetDatabaseNames(DBNames); //获取数据库别名
Variant V = VarArrayCreate(OPENARRAY(int, (0, DBNames - >Count - 1)), varOleStr);
//通过变量数组传递数据库别名
     for (i = 0; i < DBNames - >Count - 1; i + + )
     V. PutElement(WideString(DBNames - >Strings[i]), i);
      * AliasNames = V;
     delete DBNames;
     hr = S_OK;
    }
    catch (. . . )
    {hr = E_FAIL;
    }
  }
  _finally
  {return hr;
  }
}
STDMETHODIMP TDataServerImpl: : SetAliasName(BSTR AliasName)
{ _try
  {m_DataModule - >Query1 - >Close();
       m_DataModule - >Query1 - >DatabaseName = AnsiString (AliasName);
   return S_OK;
  }
 catch (. . . )
  {return E_FAIL;
  }
}
STDMETHODIMP TDataServerImpl: : SetSQL(BSTR SQLString)
{ try
  {m_DataModule - >Query1 - >Close();
          m_DataModule - >Query1 - >SQL - >Text = AnsiString (SQLString);
   return S_OK;
  }
  catch (. . . )
  {return E_FAIL;
  }
}
```

2. 客户程序设计

客户端应用程序比较简单，按一般应用程序创建即可。在窗体中加入 TMemo、TComboBox、TButton、TDatasource、TClientDataSet 、TDOMConnection、TDBGrid。TMemo 用于输入 SQL 语句、TComboBox 显示所有别名，TDBGrid 显示查询结

果。当客户端应用程序激活时，调用应用数据库服务器提供的 GetAliasNames 函数，获得所有的数据库别名，并在 TComboBox 中显示，用户选择一个别名后在 TComboBox 事件中加入代码，在 TMemo 中输入查询语句，按 RunSQL 执行查询功能。在客户端应用程序中必须加入服务器端数据模块类型库的头文件#include "AppDataServer_TLB. h" 和其它几个头文件 ComObj. hpp、 axctrls. hpp、utilcls. h (AppDataServer_TLB. h 通过 Project|Import Type Library |Add 引进并通过 File|Include Unit Hdr 加入）。各控件的属性值如下表：

控件	属性	属性值
TDataSource	Name	DataSource1
	DataSet	ClientDataSet1
TDBGrid	DataSource	DataSource1
TClientDataSet	Name	ClientDataSet1
	RemoteServer	TDCOMConnection1
	ProviderName	DataSetProvider1
TDCOMConnection	ServerName	AppDataServer. DataServer
	ComputerName	应用程序服务器所在的计算机
	Name	TDCOMConnection1
TComboBox	Name	AliasComboBox
TButton	Name	RunSQL

客户端应用程序的主要源代码如下：

```
void __fastcall TForm1::FormCreate(TObject * Sender)
{int i;
Variant DBNames;
IDataServerDisp SQLServ; //应用程序服务器的 IDISPATH 接口
 Label3->Width=105;
 __try
  {if (!DCOMConnection1->Connected);
DCOMConnection1->Connected = true
//获取应用程序服务器的 IDISPATH 接口
SQLServ. Bind(DCOMConnection1->GetServer());
AliasComboBox->Items->Clear();
//获取应用程序服务器端的别名
SQLServ->GetAliasNames(DBNames);
    for(i=0; i<VarArrayHighBound(DBNames, 1); i++)
AliasComboBox->Items->Add(DBNames. GetElement(i));
  }
  __finally
  {if(SQLServ. IsBound())
   SQLServ. Unbind();
  }
}
void __fastcall TForm1:: AliasComboBoxClick(TObject *
Sender)
{int i;
 IDataServerDisp SQLServ;
 __try
  {if (!DCOMConnection1->Connected);
       DCOMConnection1->Connected = true
       SQLServ. Bind(DCOMConnection1->GetServer());
       //设置应用程序服务器端 Query1 的数据库别名
SQLServ->SetAliasName(WideString (AliasComboBox->
Items->Strings[AliasComboBox->ItemIndex]));
  }
  __finally
  {if(SQLServ. IsBound())
   SQLServ. Unbind();
  }
}
void __fastcall TForm1::RunSQLClick(TObject * Sender)
{ static WideString WSQL;
  IDataServerDisp SQLServ;
  __try
  {if (!DCOMConnection1->Connected);
       DCOMConnection1->Connected = true
    SQLServ. Bind(DCOMConnection1->GetServer());
    ClientDataSet1->Close();
    if(Memo1->Lines->Text = = NULL)
    {ShowMessage("查询语句为空, 请重新输入 SQL 语句");
     return ;
    }
    WSQL=WideString(Memo1->Lines->Text);
    //设置应用程序服务器端 Query1 的 SQL 查询语句
SQLServ->SetSQL(WSQL);
    ClientDataSet1->Open();
   }
  __finally
  {if(SQLServ. IsBound())
   SQLServ. Unbind();
  }
}
}
```

客户端程序运行结果如下图：

本程序在 Win98 和 Win2000 下 C++Builder5. 0 环境中调试通过。同时，运行程序时必须先运行服务器程序，以便在系统中注册。选择别名时请选择不需要用户名和密码的别名 因本程序没有提供输入用户名和密码的功能）。BCDEMOS 是 C++Builder5. 0 提供的，不需要用户名和密码。如果将服务程序和客户程序运行在不同的计算机上，运行服务程序的计算机必须用 DCOMCNFG. EXE 做相应配置。

收稿日期：2001 年 2 月 20 日）

编程自动生成 ODBC 数据源

万 春 刘丽莉

摘 要 本文介绍了编程自动生成 ODBC 数据源的三种方法，给出各种方法的具体实现，并比较了各种方法的优缺点。

关键词 数据源，ODBC，自动生成

用 ODBC 作为数据库的接口是客户（浏览器）/服务器模式常用的方法，数据库开发者一般经常采用建立一个数据源来与 ODBC 服务器端的数据库相连。虽然数据源的建立可以采用 Windows 控制面板的“ODBC 数据源”手工建立，但数据库软件开发者经常需要在程序中编程建立满足系统要求的数据源，以简化最终用户的操作以及降低应用系统对最终用户专业知识的要求。

以下是几种编程建立数据源的方法，使用前，注意先安装相应的 ODBC 驱动程序。

一、用 Microsoft Jet 生成数据源

1. 方法

Microsoft Jet 引擎提供了一个名为 RegisterDatabase 的方法，可以向系统添加数据源。其语法格式为：

DBEngine. RegisterDatabase DNS Driver Silent Attributes

DBEngine——是在 DAO 对象模型中处于最顶层的对象。

Dbname——是一个字符串，表示数据源名。

Driver——是 ODBC 驱动程序名，例如：对 Microsoft SQL Server，其驱动程序名是“SQL Server”。

Silent——是一个逻辑量。如果为 True，则该方法运行时不弹出 ODBC 驱动程序对话框，此时下一个参数包含所有驱动程序需要的信息；如果为 False，则弹出 ODBC 驱动程序对话框。

Attributes——是一个字符串，包含了准备加入到 Windows 注册表的一系列关键字，各关键字用回车符作分隔符。不同厂家的数据库驱动程序所需的 Attributes 的设置是不同的，下面是 Microsoft SQL Server 驱动程序的 Attributes。

SQL Server 的 Attributes 关键字描述：

ADDRESS——SQL Server 服务器端的网络地址。该项省略，则默认为服务名。

DATABASE——SQL Server 数据库名 。

DESCRIPTION——对数据源的文字描述 。

LANGUAGE——SQL Server 所用的国家语言 。

NETWORK——所使用的网络库。例如：采用命名管道，则设置该值为 DBNMPNTW；采用 TCP/IP 协议，该值为 DBMSSOCN；采用 APPLETALK 协议，该值为 DBMSADSN。如果该项省略，则系统使用客户端默认的网络库。

OEMTOANSI——如果 SQL Server 客户端和 SQL Server 服务器端使用相同的非 ANSI 字符集，该关键字决定是否将 OEM 字符集转换为 ANSI 字符集。合法的关键字值有两个，“YES”表示转换，“NO”表示不转换。

SERVER——数据源所在的计算机名。如果该项省略，计算机名默认为数据源名。特别是“local”指本地数据库。

USEPROCFORPREPARE——允许（禁止）SQLPrepare 存储过程。该值为 NO，则禁止；该值为 YES，则允许。

2. 实现例子

```
Dim mystring As String
mystring = "description = 这是用 DAO 自动生成数据源的例子" & Chr$(13) & "address = 127. 0. 0. 1" & Chr$(13) & "useprocforprepare = yes" & Chr$(13) & "oemtoansi = no" & Chr$(13) & "database = master" & Chr$(13) & "network = DBMSSOCN"
DBEngine. RegisterDatabase "DAO 数据源", "SQL Server", True, mystring
```

以上代码运行后，点击控制面板“ODBC 数据源”图标，会看到一个新生成的数据源“DAO 数据源”，该数据源连接本地的 SQL SERVER，默认数据库为 master，采用 TCP/IP 协议。

二、用 RDO 数据库引擎

采用 RDO 数据库引擎的 rdoRegisterDataSource 方法也可以建立数据源。其语法如下：

rdoEngine. rdoRegisterDataSource DSN, Driver, Silent, Attributes

各个参数的含义、设置方法与用第一种方法相同。

下面是用 rdoRegisterDataSource 方法建立数据源的例子：

```
Dim myAttribs As String
myAttribs = "Description =" & "这是用 RAO 自动生成数据源的例子" & Chr$(13) & "OemToAnsi = No" & Chr$(13) & "server = (local)" & Chr$(13) & "Network = DBNMPNTW" & Chr$(13) & "Database = master" & Chr$(13) &
```

″useprocforprepare = yes″
rdoEngine. rdoRegisterDataSource ″RDO 数据源″, ″SQL Server″, True, myAttribs

以上代码运行后，点击控制面板“ODBC 数据源”图标，会看到一个新生成的数据源“RAO 数据源”,该数据源连接本地的 SQL SERVER，默认数据库为 master，采用命名管道协议。

如图 1 所示是用以上两种方法建立的数据源的情况。

三、用 ODBC API 建立数据源

ODBC API 提供了一个名为 SQLConfigDataSource 的函数，能用来向系统中添加、删除和修改数据源。其语法格式如下：

Private Declare Function SQLConfigDataSource Lib ″odbccp32. dll″ (ByVal hwind As _Integer, ByVal request As Integer, ByVal driver As String, ByVal attrib As String) As Integer

其中：hwind 通常为 0；

request 表示操作的种类，含义如下：

Request 值	操作的含义
1	增加一个新用户数据源
2	修改一个已经存在的用户数据源
3	删除一个已经存在的用户数据源
4	增加一个新系统数据源
5	修改一个已经存在的系统数据源
6	删除一个已经存在的系统数据源

driver 驱动程序描述，例如：如果采用 Microsoft SQL Server，此处应为“SQL Server”。

attrib 是一个字符串，包含了准备加入到 Windows 注册表的一系列与具体厂家驱动程序相关的关键字，各关键字的含义与前述两种方法相同。

1. 用 ODBC API 建立一个数据源

将如下代码加入到 Visual Basic 工程中。如下代码运行后，点击控制面板“ODBC 数据源”图标，会看到一个新生成的数据源“API 数据源”，该数据源连接本地的 SQL SERVER，默认数据库为 master，采用命名 TCP / IP 协议。

```
Dim mystring As String * 255
Dim driver As String * 64
Dim retcode As Integer
driver = ″SQL Server″ & Chr$(0)
mystring = ″dsn = API 数据源″ & Chr$(0) & ″DESCRIPTION = 这是用 API 自动生成数据源的例子″ & Chr$(0) & ″Server = 127. 0. 0. 1″ & Chr$(0) & ″Useprocforprepare = yes″ & Chr$(0) & ″OEMTOANSI = no″ & Chr$(0) & ″DATABASE = master″ & Chr$(0) & ″network = DBMSSOCN″ & Chr$(0) & Chr$(0)
retcode = SQLConfigDataSource(0, 1, driver, mystring)
If retcode Then
   MsgBox ″成功建立数据源!″
Else
   MsgBox ″建立数据源失败!″
End If
```

2. 用 ODBC API 删除上述建立的数据源

将如下代码加入到 Visual Basic 工程中。如果在运行如下代码之前，系统存在一个名为“API 数据源”的数据源，则运行后，点击控制面板“ODBC 数据源”图标，会看到一个数据源“API 数据源”已经被删除。

```
Dim mystring As String * 255
Dim driver As String * 64
Dim retcode As Integer
driver = ″SQL Server″ & Chr$(0)
mystring = ″dsn = API 数据源″ & Chr$(0) & ″DESCRIPTION = 这是用 API 自动生成数据源的例子″ & Chr$(0) & ″Server = 127. 0. 0. 1″ & Chr$(0) & ″Useprocforprepare = yes″ & Chr$(0) & ″OEMTOANSI = no″ & Chr$(0) & ″DATABASE = master″ & Chr$(0) & ″network = DBMSSOCN″ & Chr$(0) & Chr$(0)
retcode = SQLConfigDataSource(0, 3, driver, mystring)
If retcode Then
   MsgBox ″成功删除数据源!″
Else
   MsgBox ″删除数据源失败!″
End If
```

3. 运行图示

图 2 是用 ODBC API 生成数据源的情况。

图 1 数据源生成情况

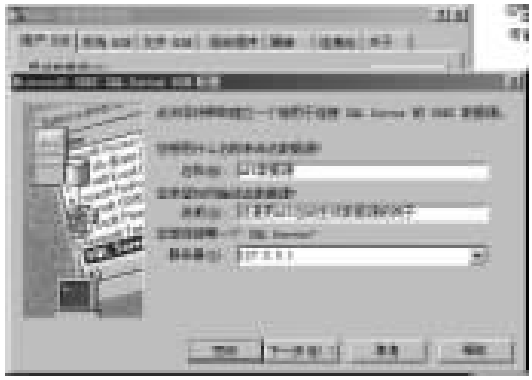


图 2 API 数据源的配置情况

四、结束语

以上三种方法，ODBC API 实现方法上最灵活，适用范围较广，但使用稍麻烦一些。前两种方法功能少，但易用。

（收稿日期：2001 年 3 月 5 日）

用 OLE 存取 blob 类型数据

郭小兵

摘 要 本文比较系统地介绍了在 PB 中用 OLE 技术如何对 blob 类型数据进行存取，举例说明了常用的三种存取方法的实现过程，并且对各种方法的优缺点加以比较分析。

关键词 PB，OLE，BLOB，二进制数据，OLE 存储

前言

在数据库的开发过程中，经常需要在数据库中存储一些备注信息，而这些备注信息的内容一般较大，格式多样，如有可能是语音文件、视频文件、图片文件、文本文件等，怎样在 PB 中实现这些格式不同的备注文件的存取及预览，一直是 PB 开发人员比较关心的一个问题，本文系统地介绍了三种存取备注二进制信息的方法。

对备注二进制信息的存储可以采用以下三种方式：

方法一：文件保存在固定的路径下，数据库中存取文件路径和名称。

方法二：数据库中用 blob 类型或者 varbinary 类型字段存储备注文件。

方法三：在本地用 OLE 存储结构存储备注文件。

一、OLE 的基本概念

OLE 是 Object Linking Embedding 对象链接与嵌入）的缩写，它可以使 Windows 应用程序共享数据和程序。

二、OLE 控件

在 PB 中 OLE 控件是一个 OLE 对象的包容器，可以使用服务器应用程序提供的功能和命令来编辑对象，也可以使用自动化 OLE 交互，在程序中激活对象和向服务器应用程序发送命令；在 PB 的 Windows 画板中的 OLE 控件允许用户从多个应用程序嵌入和链接组件。

1. 建立和设置 OLE 控件

从 Windows 画板中选择 OLE 控件插入 Windows。

当建立一个 OLE 控件并且插入一个对象时，PB 将激活服务器应用程序以允许对对象进行编辑和修改；在使 OLE 中的对象称为非活动状态后，可以使用控件属性选项卡来设置控件的属性。

2. 激活修改 Windows 画板中的 OLE 对象

在 OLE 控件的弹出菜单中选择 open 可以激活画板中 OLE 对象

使用服务器应用程序修改 OLE 对象

结束修改：使对象恢复为非活动状态，只要单击服务器应用对象之外的任何区域即可，也可以直接关闭服务器应用程序的窗口。

3. 嵌入和链接 OLE 控件

可以将 OLE 对象嵌入或者链接到自己的应用程序中。嵌入对象的数据放在应用程序中，在开发过程中这些数据放在应用程序的 PBl 库中，当生成应用后，这些数据将存放在 exe 或 PBd 文件中，虽然在程序的运行过程中可以修改，但修改的数据不会保存；链接对象的数据存放在 PB 应用程序以外，当链接一个对象时，在 PB 应用程序中不存放数据文件，而是存放引用数据的指针 使用链接的数据，对数据的管理和保存都由服务器应用程序负责。这样可以用服务器应用程序修改处理数据，处理后的数据可以保存回原文件中。链接方式应用于需要多个应用程序共享的数据文件，任何一个应用程序修改了数据文件，都将影响到所有链接该文件的应用程序。

4. OLE 控件的激活方式

OLE 控件的激活方式有 offsite 和 in - place 两种激活方式，offsite 激活方式是指在 PB 应用程序的界面以外单独打开 OLE 对象，in - place 激活方式是指 PB 应用程序的界面的原位置打开 OLE 对象。在数据窗口中的 DBOLE 默认的是 offsite 激活方式，而 Windows 中的 OLE 默认的激活方式是 in - place。

在 PB 应用程序中可以用命令 OLE_control. active offsite 或者 OLE_control. active in - place 设置 OLE 对象的以何种方式打开。

5. 设置和插入 OLE 对象

在程序运行时可以用函数：

OLE_control. insertfile soucefile 插入对象

OLE_control. objectdata = blobvar 设置对象

三、OLE 存储

1. OLE 存储 OLEstorage 的概念

OLE 存储 OLEstorage 是 OLE 数据的一个仓库，存储就象磁盘上的目录结构，它可以是一个 OLE 对象，也可以包含在 OLE 对象中，每个对象都包含在 OLE 存储或者存储内的子存储内。保存在 OLE 存储中的数据称作 OLE 流 OLE stream ，

OLE 流同 OLE 对象的关系就象文件同目录的关系。含有 OLE 对象的存储或子存储可以看做是属于特殊服务器的信息，在该层次之下的各部分都可以被相应的服务器程序操作。OLE 存储对象是类用户对象，可以说明相应类型的变量，建立与之相应的实例和打开存储等，在使用完存储后需要关闭存储，释放分配的内存。

2. OLE 存储的打开和保存

OLE 存储可以用 open 函数打开，open 函数的格式为：

Olecontrol. Open （OLEsourcefile）

此函数在 OLEsourcefile 不存在时，自动创建该文件，所以创建 OLE 文件也用该函数； OLE 存储可以用 save 函数保存，save 函数的格式为：

```
OLEcontrol. save()  // 保存 OLE 控件
OLEstorage. save()  //保存 OLE 存储
```

另外，可以使用 saveas 函数存储 OLE 控件中的内容到 OLE 存储的存储对象中。Saveas 函数的格式为：

olecontrol. SaveAs OLEtargetfile

四、处理 blob 类型数据

对于大二进制数据，在 PB Script 中是用 blob 数据类型表示并加以处理。标准 SQL 语句中的 select、insert 和 update 语句无法直接查询 blob 类型的数据，在 PB 中操作 blob 类型的数据只能用专用的语句，从数据库中查询 blob 类型的数据的命令是：

selectblob restofselectstatement {using transactionobject}

更新数据库中 blob 类型数据的格式是：

```
updateblob tablename
set blobcolumn = blobvarible
restofupatestatement {using transctionobject};
```

如连接的数据库是 sybase 或者 Sql，则 selectblob 和 updateblob 语句要求数据库的自动提交方式为 true，所以在每次调用 selectblob 和 updateblob 语句以前必须用命令 Sqlca. autocommit = true，把数据库的自动提交方式设置为 true，在 updateblob 语句的结束后，再用命令 Sqlca. autocommit = false，把自动提交方式设置为 false。

五、数据窗口的 blob 列

1. 数据窗口 blob 列的功能

在 PB 的 datawindow 画板中 DBOLE 控件允许用户利用这个控件浏览和操作数据库中的大二进制数据，即通过 DBOLE 控件可以作如下操作：

- 往数据库中存储大二进制数据，如：excel 工作表、word 文档、视频文件、图片文件等各种格式的文件；
- 从数据库中检索数据到 datawindow 对象；
- 使用 OLE 服务器程序察看修改数据；
- 将修改后的数据保存回数据库；

2. 在数据窗口中添加 blob 列的的步骤

1 选择具有二进制字段的数据表作为数据源建立一新的数据窗口 该窗口可以至少需要包含非数据库表的标识列）

2 选择 insert - control - OLE database blob 菜单，在数据窗口的 detail 节中要插入 blob 列的位置，单击鼠标，这时将显示如图 1 所示的对话框。

图 1 defination 属性框

下面解释这些属性的具体含义

1 client class： 客户类名，默认为 datawindow。

2 client name：客户名，默认为 untitled。

3 table 选择含有 blob 列的数据库表，所选表的字段将出现在右侧的 large binary / text column 列表框中。

4 large binary / text column：选择一个 blob 类型的字段列。

5 key clause：检索和更新 blob 数据的关键字表达式，其中使用带冒号前缀的变量指出是数据窗口对象的列，如表达式 id = id，id 是数据库表中的列，变量指出数据窗口对象的列。

6 filetemplate ：如果需要 OLE 应用服务器每次打开相同的文件，则在 filetemplate 框中输入文件名。

7 OLE class ：如果不需要 OLE 应用服务器每次打开相同的文件，则在 OLE class 框中选择一个 OLE 类，如 Pbrush。

8 Client name expression：显示在 OLE 服务器应用程序窗口标题的文字，可以输入为： ‘对应记录的 id 号是 + id ”。

单击 ok 按钮关闭对话框，将 dbole 列添加到适当的位置，保存数据窗口。

预览则可以对数据库中的 blob 数据进行存取，但是在新建的记录中只能存取 OLE class 框中选择的一种格式的 blob 数据，不能存储多种格式的数据；但如果数据库中存有多种格式的数据，可以预览各种格式的数据。

六、源程序建立

1　首先在数据库中建立如下结构的表 blobsave：

字段名称	数据类型	备注
id	char　(4)	primairy key　index
s_path	char　(50)	
pic	binary　50	

2　在 PB 建立 PBl 库 blobsave. PBl

3　在 PBl 库 blobsave. PBl 中建立应用 blobsave

在应用的 open 事件中设置数据库连接程序　本程序中采用的是 odbc 方式连接数据库，读者可根据自己所建立的数据库的不同选用不同的连接方式，以下连接数据库的代码也有所改动，至于连接不同的数据库的方法，请参考有关资料，本文不做详细介绍）：

```
SQLCA. DBMS　=　″ODBC″
SQLCA. AutoCommit　=　False
SQLCA. DBParm　=　″Connectstring = ′DSN = blob′″
connect;
　　open(w_main)
```

w_main 是为应用程序建立的主窗口，如图 2 所示：

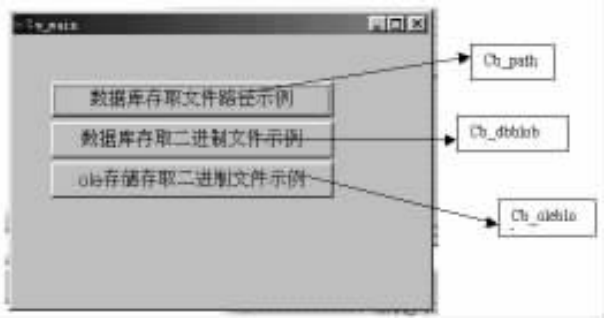


图 2　出口 w_main

其中命令按钮 cb_path 的 clicked 中的代码格式如下：open w_path

其中命令按钮 cb_dbblob 的 clicked 中的代码格式如下：open w_dbblob

其中命令按钮 cb_OLEblob 的 clicked 中的代码格式如下：open w_OLEblob

4　建立数据窗口 dw_blobsave

按照上文中建立数据窗口的 blob 列的方法建立数据窗口 dw_blobsave 如图 3 所示：

其中：add、del、save、cancel、retrieve 等分别为数据窗口 dw_blobsave 的 append row，delete row，update，retrieve 动作按钮。

5　新建窗口 w_path，如图 4 所示：

首先创建实例变量 OLEstorage　stor1

然后建立窗口 w_path，其中数据窗口控件 dw_1 的 rowfocuschanged 中的代码如下：

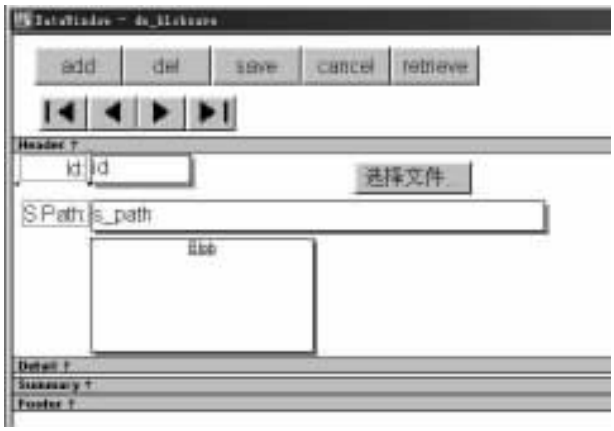


图 3　数据窗口 dw_blobsave

图 4　窗口 w_path

```
long row_num
row_num = dw_1. getrow()
if　row_num >0 then
　ole_1. insertfile(dw_1. object. s_path[row_num])
end if
```

其中数据窗口 dw_1 的 buttonclicked 中的代码如下：

```
if dwo. name = ″cbselect″ then
　long row_num
　row_num = dw_1. getrow()
　string filepath, filename
　getfileopenname(″请选择备注文件″, filepath, filename)
　dw_1. object. s_path[row_num] = filepath
　ole_1. insertfile(filepath)
end if
```

保存窗口 w_path。

6　建立窗口 w_dbblob

打开 w_path，把其另存为 w_dbblob，改变数据窗口 dw_1 的 rowfocuschanged 中的代码如下：

```
blob text1
long row_num
row_num = dw_1. getrow()
if row_num>0 then
　string　id
　id　=　dw_1. object. id[row_num]
　sqlca. autocommit = true
　selectblob pic into : text1 from blobsave where id　=　: id;
　ole_1. objectdata = text1
　sqlca. autocommit = false
end if
```

改变数据窗口 dw_1 的 buttonclicked 中的代码如下

```
long row_num
```

```
if dwo. name = ″cbselect″ then
        row_num =dw_1. getrow( )
        string filepath, filename
    getfileopenname(″请选择备注文件″, filepath, filename)
        dw_1. object. s_path[row_num] =filepath
        ole_1. insertfile(filepath)
end if
if dwo. name = ″cbsave″ then
   string id
   sqlca. autocommit  =  true
   blob text1
  text1  =  ole_1. objectdata
  dw_1. update( )
  commit;
       row_num =dw_1. getrow( )
       id =dw_1. object. id[row_num]
  updateblob  blobsave
  set pic  =  : text1
  where id  =  : id ;
    commit;
    sqlca. autocommit  =  FALSE
  dw_1. retrieve( )
  dw_1. scrolltorow(row_num)
end if
```

保存窗口 w_dbblob

7 **建立窗口** w_OLEblob

打开 w_path，**把其另存为** w_OLEblob，**在窗口** w_OLEblob **的** open **事件中写入以下代码：**

```
stor1   = create olestorage
stor1. open(″c: \p1. ole″)   //打开或创建 ole 文件
```

在窗口 w_OLEblob **的** close **事件中写入以下代码**:

```
destroy stor1
```

改变数据窗口 dw_1 **的** rowfocuschanged **中的代码如下：**

```
blob text1
long row_num
row_num =dw_1. getrow( )
if row_num>0 then
       string  id
       id  =  dw_1. object. id[row_num]
   ole_1. open(stor1, ″w″ +id)
end if
```

改变数据窗口 dw_1 **的** buttonclicked **中的代码如下：**

```
long row_num
if dwo. name = ″cbselect″ then
        row_num =dw_1. getrow( )
        string filepath, filename
    getfileopenname(″请选择备注文件″, filepath, filename)
        dw_1. object. s_path[row_num] =filepath
        ole_1. insertfile(filepath)
end if
if dwo. name = ″cbsave″ then
          string id
       row_num =dw_1. getrow( )
       id =dw_1. object. id[row_num]
   ole_1. saveas(stor1, ″w″ +id)
       stor1. save( )
end if
```

保存窗口 w_OLEblob

运行应用程序即可。

8 **三种方法的优缺点**

方法一：文件保存在固定的路径下，数据库中存取文件路径和名称可以节省数据空间，避免了数据库过分膨胀，但备注文件必须在一定的目录下，不能丢失，且同一目录下文件不能重名，对文件的管理造成一定的困难。另外，在 OLE **控件中浏览显示备注文件时，由于每次都要调用服务器程序，所以速度较慢。**

方法二：在数据库中用 blob **类型或者** varbinary **类型字段存储备注文件，当文件存储在数据库中以后，就可以删除硬盘上原来的临时文件，不需要复杂的二进制文件管理，且数据库可以存储在网络服务器上，对数据的共享非常方便。**

方法三：在本地用 OLE **存储结构存储备注文件中可以把所有的二进制文件信息存储在一个** OLE **存储文件中，管理比较方便。当二进制文件信息存储后，可以删除原来的临时文件；因为打开存储文件后不需要每次执行服务器程序来显示存储信息，所以存取速度较快。**

说明：本文在 PB6. 5，Sql anywhere **数据库和** PB6. 5，Sql Server **数据库下，**Windows98，Windows Me，NT4. 0 **平台上试验通过。**

收稿日期：2001 **年** 3 **月** 7 **日）**

瑞星引领杀毒软件高速增长

近日，连邦软件宣布了 2001 **年上半年销售涨幅最大和降幅最大的软件类别，其中杀毒软件以** 60% **的增长率名列涨幅三甲。其余进入涨幅前三位的还有游戏软件与教育软件，而字处理软件和电子图书则成为降幅最大的两种软件。**

杀毒软件销售量的高速增长，瑞星公司功不可没。作为杀毒软件产业领导厂商的北京瑞星公司拥有超过 65% **的市场占有率，正是由于瑞星公司在** 2001 **年上半年以良好的增长势头，超过** 100% **的销售增长率，带动了整个杀毒软件市场的高速增长。**

近年来，瑞星公司的产品技术一直处于国内领先地位，在网络版和单机版杀毒软件双双荣获“公安部评测第一名”后，其单机版产品再次被评定为中国杀毒软件第一个“一级品”。年内 IT **产品的传统销售淡季期间，在业内被誉为邮件病毒 克星”的瑞星杀毒软件** 2001 **版依然保持热卖。今年暑期，瑞星公司紧抓机遇大力推出了百万巨奖酬宾活动，再次掀起杀毒软件市场销售的高潮。**

今天，网络已成为我们生活中不可缺少的部分。但是在为人类带来便利的同时，互联网也使病毒插上了飞翔的翅膀，病毒的传播速度更加惊人，编程手段也日渐高明。认识信息安全的重要性，加强对计算机病毒的防范已被提到了史无前例的高度，人们逐渐意识到了通过杀毒软件来保护自己的计算机系统安全，于是 2001 **年杀毒软件市场呈现出一片火热。**

如何用 XML 在浏览器中显示图像

叶兴茂

扩展标记语言（XML）以其结构规范、扩展性强、数据存储和表现分离、自定义标签等特点而备受世人注目，自从其问世以来，发展的步伐，一刻也没有停止。目前其应用已经延伸到信息发布、多媒体展示、电子商务、通用数据交换、图形的矢量化表示等多种领域。但是，由于该语言不像 HTML 那样有许多开发工具支持，目前在 Web 信息发布方面还不是非常普及，但由于其优良特性，取代 HTML 成为新一代的数据交换标准和 Internet 语言，已经是大势所趋。下面我们就用该语言来实现一个显示图像的 XML 程序，和大家交流和学习。

XML 语言的核心包括三个方面，即 XML 大纲、XML 文档和 XSLT。其中 XML 大纲定义了 XML 文档的数据结构、语法及所使用的标签集合，它是一种规则的规则，类似于 C、C++和 Java 等高级语言中的 struct 和 class 定义，非常重要。而 XML 文档则是 XML 大纲的实例化，用于描述具体的实例数据。由于 XML 语言中，数据的结构和数据的表现是分离的，要想在浏览器中正确地显示具体的 XML 文档，就必须有一种机制来表现它，这就是 XSLT 的作用，即扩展级联样式单。按照 XML 语言的特点，我们把用 XML 显示图像的方法，分为以下三步来实现。

第一步，设计并开发图像的 XML 大纲

XML 大纲中，重要的有几个地方，一是正确理解“名域”，说白了，“名域”实际上对元素和元素属性的限定，以防止多个文档组合时，元素及其属性的重名冲突，它的作用和 java 中的包（package）、C#中的命名空间（namespace）差不多，下列代码中，Schema 标签中的 xmlns = “urn schemas - microsoft - com xml - data”和 xmlns dt = “urn schemas - microsoft - com datatypes”都是名域。其次是如何定义元素、元素的属性以及怎样使用处理指令等语法规则。XML 大纲中，元素用 ElementType 标签定义，用 element 标签引用。而元素的属性用 AttributeType 定义，用 attribute 标签引用。

要在 Internet 网上描述一幅图像，最基本的有图像的宽度、高度、图像源（即 URL）等要素，以下便是根据 XML 大纲的规则定义的图像 XML 大纲。

```
<?xml version = ″1.0″ encoding = ″gb2312″?>
<Schema xmlns = ″urn: schemas – microsoft – com: xml – data″
xmlns: dt = ″urn: schemas – microsoft – com: datatypes″>
<! – –This schema defines XML document mainly used for show a image – –>
<! – –define image element, including child nodes comment, source, width and height – –>
<ElementType name = ″comment″/>
<ElementType name = ″source″/>
<ElementType name = ″width″/>
<ElementType name = ″height″/>
<ElementType name = ″image″ content = ″eltOnly″>
  <element type = ″comment″/>
  <element type = ″source″/>
  <element type = ″width″/>
  <element type = ″height″/>
</ElementType>
<! – –define link element, including child nodes href and text – –>
<ElementType name = ″href″/>
<ElementType name = ″text″/>
<ElementType name = ″link″>
  <element type = ″href″/>
  <element type = ″text″/>
</ElementType>
<! – –define message element, including child nodes E_Mail and content – –>
<ElementType name = ″E_Mail″/>
<ElementType name = ″content″/>
<ElementType name = ″message″>
   <element type = ″E_Mail″/>
   <element type = ″content″/>
</ElementType>
<ElementType name = ″myWeb″ content = ″eltOnly″>
  <element type = ″link″/>
  <element type = ″message″/>
  <element type = ″image″/>
</ElementType>
</Schema>
```

图像 XML 大纲中，定义的主要标签有 image、comment、source、width 和 height 等，用于描述图像及其宽度和高度等属性。同时也定义了类似于 HTML 语言中 a 标签的 link 超链接标签，用于定位具体的网络资源。显然，这些自定义的标签要比那些死板的 HTML 标签易懂得多，望文知意，一目了然。

第二步，编写规范的图像 XML 文档

使用上面定义的图像 XML 大纲，就能够描述具体的图像对象，编写 XML 文档了，代码如下：

```
<?xml version = ″1.0″ encoding = ″gb2312″?>
<?xml: stylesheet type = ″text/xsl″ href = ″image.xsl″?>
<myWeb xmlns = ″x – schema: image_schema.xml″>
```

```
<link>
  <href>http://www.mlr.gov.cn</href>
  <text>欢迎浏览国土资源部网站</text>
</link>
<image>
  <comment>show a image by XML</comment>
  <source>images/scenery.jpg</source>
  <height>250</height>
  <width>680</width>
</image>
<message>
  <E_Mail>mailto:info_xmye@mail.mlr.gov.cn</E_Mail>
  <content>如需源代码请与作者联系</content>
</message>
</myWeb>
```

这里的图像 XML 文档描述了一个超链接，一幅图像和一个 E_Mail 地址。超链接指向国土资源部网站，而图像描述的是一幅非常美丽的大陆风光，其数据源是 images/scenery.jpg，宽度 680 像素，高度 250 像素。E_Mail 地址，即是作者的电子邮箱，点击它可与作者联系。看到这些标签，略懂英语的人，即使不明白 XML 的语法，也能知道其基本含义。

第三步，XSLT 表现图像的 XML 文档

虽然定义了图像 XML 大纲和图像 XML 文档，但是，浏览器并不认识这些东西，要想让浏览器显示出来，还必须为其定义表现方式，这时候，XSLT 就发挥作用了，它把 XML 文档翻译成其它形式的数据结构，供浏览器显示，目前如微软的 ie5.0 支持的格式是 EHTML　即扩展超文本标记语言，它也是符合 XML 规范、结构良好的 XML 文档。

```
<?xml version="1.0"?>
<html xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <body style="font-family: Arial, helvetica, sans-serif;
font-size: 9.5pt;
      border: 1em;
      background-color: #FEEEEE">
    <div style="margin-left: 10px; margin-bottom: 1em;
font-size: 9pt">
      <!--<xsl:for-each select="myWeb/image">-->
        <center>
        <a style="font-size: 2em; color: red">
          <xsl:attribute name="href">
          <xsl:value-of select="myWeb/link[0]/href"/>
          </xsl:attribute>
          <xsl:value-of select="myWeb/link[0]/text"/>
        </a>
        <br/><br/>
        <img>
          <xsl:attribute name="src">
        <xsl:value-of select="myWeb/image/source"/>
          </xsl:attribute>
          <xsl:attribute name="width">
          <xsl:value-of select="myWeb/image/width"/>
          </xsl:attribute>
          <xsl:attribute name="height">
          <xsl:value-of select="myWeb/image/height"/>
          </xsl:attribute>
        </img>
        <br/><br/>
        <span>
          <p style="font-size: 2em">
<xsl:value-of select="myWeb/image/comment"/>
          </p>
        </span>
        <a style="font-size: 1.5em; color: blue">
          <xsl:attribute name="href">
<xsl:value-of select="myWeb/message/E_Mail"/>
          </xsl:attribute>
<xsl:value-of select="myWeb/message/content"/>
        </a>
        </center>
      <!--</xsl:for-each>-->
    </div>
  </body>
</html>
```

上述代码和 HTML 差不多，只要熟悉 HTML，很容易掌握。这里使用了名域 xmlns:xsl=http://www.w3.org/TR/WD-xsl，其局部名称是 xsl。标签 attribute、value-of、for-each 等均受其约束。其中 for-each 标签类似于高级语言中的循环语句，用来通过模式匹配遍历并转化满足条件的 XML 源树。XSLT 的语法完全是规范的 XML，这也体现了 XML 的简单性，一种文法走天下。要注意的是 EHTML 是结构良好的文档，而 HTML 却不是。两者最主要的区别是 EHTML 中的所有标签都是封闭的，型如
 等格式，像 HTML 中 、
等标签在 XML 文档中，再也行不通了。但它们转换起来并不麻烦和复杂，只要使这些不封闭的标签封闭起来，就可以了，如把 变成 ，
 变成
。

本文中涉及的代码在 IE5.0 浏览器中，经过严格测试。

（收稿日期：2001 年 2 月 16 日）

KV3000 用户福音连篇

随着 KV3000 用户越来越多，江民公司每一次重大升级和增加功能都处处为 KV3000 用户着想。比如：至少保证一年内免费为 KV3000 用户升级，KV300+至 KV3000，江民公司为最早的 KV300+用户免费升级时间长达四年多，四年后才用 60 元的优惠价为用户升级到 KV3000。江民公司还免费为 KV3000 用户增加防黑客攻击软件，免费为 KV3000 用户升级到 KV3000 FOR WIN9X-XP 多语言版　即将推出）；不到半年时间为用户升级 33 次；为北京地区的 KV 用户免费修复硬盘等等。

最近，江民公司又决定，对 KV3000 用户的密钥盘使用损坏更换费从 10-20 元降为 5 元/片，各地门市代理商直接向江民公司更换的费用为 3 元/片。这次江民公司降价为用户更换损坏了的盘片，再一次体现了江民公司一贯为用户着想，不断提供更优惠服务的经营策略。

采用 ActiveX 技术加强互联网软件的显示功能

龚本灿　李庆华　王少蓉

摘　要　本文介绍了用 VB 开发 ActiveX 控件，开放 ActiveX 控件属性、方法、过程的技术，并且以 Dephi 的 CGI 程序为例，介绍了如何调用开放的属性、方法、过程来对 ActiveX 控件进行编程，以实现互联网上方便、灵活的数据显示功能。

关键词　ActiveX 控件，CGI，互联网

一、 引言

在安琪集团“产品质量互联网数据查询系统”的开发过程中遇到这样一个情况，产品质量数据库包括品名、规格、批号、检验依据、产品数量及其它检验项目共 27 个字段，在互联网上编程对该数据库进行查询，如果采用 HTML 的 <Table>元素来显示数据，当字段、记录数很多，屏幕显示不下时，只能通过屏幕的上、下、左、右滚动条来浏览数据，这种浏览方式有两个弊端：①向下滚动时会将表格表头滚出屏幕，导致用户不知道显示数据对应哪个字段；②向右滚动时会将品名、规格、批号滚出屏幕，导致用户不知道显示数据对应哪个品名、规格、批号。欲实现上、下滚动时表格表头固定，只记录上、下滚动；左、右滚动时左边的品名、规格、批号三列固定，只让其它字段左、右滚动。为实现这一功能采取的办法是用 VB 开发 ActiveX 控件，用 CGI 编程对 ActiveX 控件进行设置。

二、 用 VB 开发 ActiveX 控件

2.1 启动 VB6.0，在“新建工程”对话框中，选择“ActiveX 控件”。

2.2 在 UserControl 中添加 MSFlexGrid 控件，将 UserControl 命名为 grid，MSFlexGrid 控件命名为 MSFlexGrid。

2.3 程序代码

```
'开放 ActiveX 控件的 rows 属性，用来设置 MSFlexGrid 控件的行数
Public Property Let rows(ByVal New_rows As Integer)
    MSFlexGrid.rows() = New_rows
    PropertyChanged "rows"
End Property
'开放 ActiveX 控件的 cols 属性，用来设置 MSFlexGrid 控件的列数
Public Property Let cols(ByVal New_cols As Integer)
    MSFlexGrid.cols() = New_cols
    PropertyChanged "cols"
End Property
'开放 ActiveX 控件的 fixedcols 属性，用来设置 MSFlexGrid 控件的左边固定列数
Public Property Let fixedcols(ByVal New_fixedcols As Integer)
    MSFlexGrid.fixedcols() = New_fixedcols
    PropertyChanged "fixedcols"
End Property
'开放 ActiveX 控件的 setvalue 过程，用来向 MSFlexGrid 控件中写入数据
Public Sub setvalue(ByVal r As Integer, ByVal c As Integer, ByVal s As String)
  MSFlexGrid.Row = r
  MSFlexGrid.Col = c
  MSFlexGrid.Text = s
End Sub
```

说明：① setvalue 用来向 MSFlexGrid 指定的单元格填入数据，例 setvalue 1 1 “abcd” 表示在第一行，第一列单元格中填入 abcd 。②在 CGI 程序中通过 Vbscript 脚本来调用开放的 rows、cols、 fixedcols 属性和 setvalue 过程。

2.4 将项目保存为 grid.vbp，并编译 。

2.5 用 VB6.0 工具包在 Internet 上发布 ActiveX 控件，发布完成后，在发布目录下包含有 grid.HTM、grid.CAB 文件和 support 目录。grid.HTM 中主要内容如下：

```
<OBJECT ID="grid"
CLASSID="  CLSID:  F2E5CA4E－F8F8－11D3－906F－000021EA2A2F"
CODEBASE="grid.CAB#version=1,0,0,0">
</OBJECT>
```

含义：ID 控件的名称
CLASSID 对象的唯一识别 ID 号
CODEBASE 所创建的 ActiveX 控件发布文件的位置

三、 用 Dephi 4.0 的 CGI 程序对 ActiveX 控件进行编程

3.1 启动 Dephi 后，选“新建 Web Server Application”，在选项框中选“CGI Stand - alone executable 。

3.2 打开窗体，并在窗体上添加如下部件：

设置好 Database1 和 Table1 的数据库连接，并将 Pageproducer1 的 HTMLDoc 属性设为：<#tag>

3.3 **主要程序如下：**

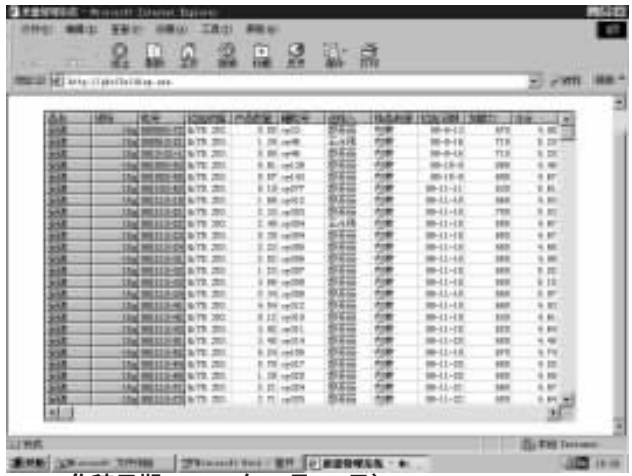
```
procedure TWebModule1.WebModule1WebActionItem1Action
  (Sender: TObject; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean);
begin
  response.content: = Pageproducer1.Content;
end;
procedure TWebModule1. Pageproducer1HTMLTag(Sender:
TObject; Tag: TTag; const TagString: String; TagParams:
TStrings; var ReplaceText: String);
var
    s, text1: string;
    v: variant;
    i, j: integer;
    k: integer;
    d: real;
begin
  table1.open;
  text1: =´<html><head><title>质量管理系统</title>´+
chr(13)+chr(10);
  {插入脚本，对 ActiveX 控件进行设置，并从数据库中取数到
ActiveX 控件中显示}
  text1: =text1+´<script language=vbscript>´+chr(13)+
chr(10);
text1: =text1+´Sub window_onload()´+chr(13)+chr(10);
     text1: =text1+´  grid.rows=52´+chr(13)+chr(10);
{设置 ActiveX 中 MSFlexGrid 行数}
     text1: =text1+´  grid.cols=27´+chr(13)+chr(10);
{设置 ActiveX 中 MSFlexGrid 列数}
    text1: =text1+´  grid.fixedcols=3´+chr(13)+chr(10);
{设置 MSFlexGrid 左边固定列数}
  {取数据库字段名到 MSFlexGrid 的第一行}
  for j: =0 to table1.fieldcount-1 do
    begin
      s: =trim(table1.fields[j].fieldname);
       text1: =text1+´  Call grid.setvalue(0,´+inttoStr(j)
+´,´´´+s+´´´)´+chr(13)+chr(10);
    end;
  {取数据库记录到 MSFlexGrid 中}
  i: =1;
  table1.first;
  while not table1.eof do
    begin
      for j: =0 to table1.fieldcount-1 do
        begin
          v: =table1.fieldvalues[table1.fields[j].fieldname];
          if varisnull(v) then
            s: =´´
          else  {将各数据项转换成字符串}
            begin
              k: =vartype(v);
              if (k=4) or (k=5) then {是数值，保留两位小数}
                begin
                  d: =v;
                  s: =floattostrf(d, fffixed, 10, 2)
                end
              else {是字符};
                s: =v;
            end;
          text1: =text1+´  Call grid.setvalue(´+inttoStr(i)
+´,´+inttoStr(j)+´,´´´+s+´´´)´;
          text1: =text1+chr(13)+chr(10);
        end;
      i: =i+1;
      table1.next;
    end;
  text1: =text1+´  End Sub´+chr(13)+chr(10);
  text1: =text1+´  </script></head>´
  {脚本完毕}
     text1: =text1+´  <body  background=images/back-
ground.jpg>´;
  text1: =text1+´  <p align=center>湖北安琪集团质量报
告单</p>´;
  {嵌入 ActiveX 控件}
  text1: =text1+´  <center>´;
     text1: =text1+´  <OBJECT  ID=grid  WIDTH=685
HEIGHT=400´;
  text1: =text1+´ CLASSID=CLSID: F2E5CA4E-F8F8-
11D3-906F-000021EA2A2F´;
text1: =text1+´CODEBASE=grid.CAB#version=1,0,0,0>´;
  text1: =text1+´</OBJECT>´;
  text1: =text1+´</center>´;
  text1: =text1+´</body></html>´;
  replacetext: =text1;
end;
```

3.4 **将以上程序编译生成** disp.exe。

说明：①从数据库中取数到 ActiveX **控件的脚本语句是动态生成的。②通过** ActiveX **中** MSFlexGrid **的左、右、上、下滚动条，可以很方便地进行数据浏览。③在服务器上安装** Dephi **的** BDE **并配置** ODBC **系统** DSN**，使其连接到数据库** db1.mdb**。④将** ActiveX **发布目录下的所有内容，数据库文件** db1.mdb **和** CGI **执行文件** disp.exe **拷到** Web **服务器上，通过浏览器调用** disp.exe **即可。**

四、 显示结果

如下所示



收稿日期：2001 **年** 2 **月** 23 **日）**

软件实现微机远程启动

廖金祥

摘　要　本文简要分析了局域网中远程（无盘）启动的过程，说明了用软件代替 BOOTROM 芯片实现远程启动的方法，最后给出了具体的实例。

关键词　远程启动，BOOTROM 芯片，INT19H 中断，BOOTCODE

一、问题的提出

在局域网中，无论是有盘站还是无盘站要实现远程（无盘）启动都是靠网卡上的 BOOTROM 芯片来实现的。近年来，随着网络带宽的增加，高性能网卡多被使用，一方面有些网卡上根本没有 BOOTROM 插座；另一方面一些网卡的 BOOTROM 芯片专用化，其价格已达到 30 元（如 DLINK、3COM 网卡），数十个站点的局域网购买 BOOTROM 芯片就得花去 1～2 千元。能否不用 BOOTROM 芯片而由纯软件的方法实现远程启动呢？答案是肯定的。经过研究，笔者已经在多个单位的局域网上基于不同的网卡（NE2000、TPLINK、DLINK、ACCTON、3COM 等）用纯软件的方法实现了远程启动，节约了大笔开支。本文将此项技术发表与读者共享。

二、远程启动代码的获取

要用软件实现远程启动，首先必须获取 BOOTROM 芯片中的启动代码（简称为 BOOTCODE 下同）从而在微机启动过程中执行 BOOTCODE 而不要 BOOTROM 芯片。目前 BOOTROM 芯片多为 EPROM 或 E2PROM，BOOTCODE 烧录在其中，当网卡插上此芯片后 BOOTCODE 被装载到上位内存的 C800H～F000H 段中。在 BOOTCODE 的头部一般有 NetWare、NW、BootWare 等字样。我们可用 DEBUG 找到 BOOTCODE 并将他以文件的形式存放在磁盘上。方法如下：

在网卡插座上插入与网卡匹配的 BOOTROM 芯片，在 DOS 界面下进入 DEBUG。

① 键入：S C800：0 FFFF 55 AA←┘ ；搜索扩展 ROM 块的标志 55AA，若成功则显示 ROM 块地址，记下这些地址，然后搜索 S D800：0 FFFF 55 AA←┘直到 S E800：0 FFFF 55 AA←┘。假设其中的一个地址是 XXXX：YYYY，转②。

② 键入：D XXXX：YYYY←┘ ；查找 NetWare 等字串，若找到转③，否则查找下一个 ROM 块地址。

③ 查看 55AA 后 1 字节的值，此值为 BOOTCODE 的长度，它是 512 字节的倍数，设此值为 20H 则表示 BOOTCODE 长 16K，值为 40H 则长度为 32K，类推。转④。

④ 键入 R CX←┘

键入 4000←┘ ；4000 表示长度 = 16K，若长度 = 32K 则此处改为 8000　64K 时改为 0 同时 BX 改为 1

键入 N BOOTCODE←┘

键入 W XXXX：YYYY←┘ ；此时在当前目录下生成了 BOOTCODE 文件。

三、软件实现远程启动

首先，我们分析一下硬件（BOOTROM 芯片）实现远程启动的过程，微机开机自检后，BIOS 要搜寻扩展槽中的插件，当确认插件有效时便执行其上的扩充 ROM 块中的代码并使之融入系统。BOOTROM 芯片就是作为扩充 ROM 而融入系统的。BOOTROM 中的代码首先修改 INT19H 中断，使 INT19H 中断指向 BOOTROM 自身的启动代码段，然后将控制权返还给 BIOS。当 BIOS 发 INT19H 中断引导磁盘上的操作系统时 BOOTROM 又取得了控制权。接着在屏幕上显示一个远程启动菜单（或提示一个选择，如：按‘H’键），由用户选择是本地启动还是远程启动（是何种远程启动 NOVELL、NT、NETWARE ？当选择了远程启动后，便向服务器发出一个“远程启动请求”并送出本网卡的 ID 地址（6 个字节），当服务器在客户数据库中找到该 ID 地址后便发出应答信号，BOOTROM 再次发出要求传送操作系统文件的请求，服务器则根据该工作站的配置情况将相关的操作系统文件送到工作站的内存中，于是 BOOTROM 便将控制交给内存中的操作系统从而完成远程系统引导。

上述过程表明，BOOTROM 是在 INT19H 中断引导操作系统时取得控制权并实施远程启动的。我们知道，INT19H 中断会读出硬盘（若不是从软盘启动的话）的 0 柱 0 头 1 扇区（简称为 001 扇）上的主引导记录，这就给我们提供了一个机会。我们可以编写一段代码（简称为 BOOTREC）替换 001 扇上的主引导记录，而将原来的主引导扇存放到其他地方（比如：0 柱 0 头 3FH 扇上），这样 BOOTREC 在操作系统引导前得以执行。BOOTREC 要完成如下任务：首先将保存在 C：盘尾柱上的 BOOTCODE 读到内存空闲处（比如 7000H 段），用 CALL FAR 远调用执行 BOOTCODE，BOOTCODE 在修改 INT19H 中断后返回。然后 BOOTREC 发 INT19H 中断再次进入 BOOTCODE，BOOTCODE 会显示远程启动菜单，用户可选择启动方式。若是选择远程启动一切很顺利，然而当选择了本地启动时，BOOTCODE 就先恢复 INT19H 中断然后要调

用 INT19H 中断引导操作系统，这样 BOOTREC 再次被执行从而进入了死循环。怎么办？我们可以在 BOOTREC 执行一次后设置一个标志，再次进入 BOOTREC 后遇此标志就不再执行 BOOTCODE 而直接读入原 001 扇引导操作系统。一个实用的 BOOTREC 见程序一，阅读时注意程序中的注释。将程序一汇编连接并转换成 . COM 文件再改名为 BOOTREC。

程序一　BOOTREC 汇编源程序 B. ASM

```
code segment
   assume cs: code, ds: code
   org 7c00h ; BOOTREC 装入 0000: 7C00H
start:    cli
   xor ax, ax
   mov ss, ax
   mov sp, 7c00h
   mov si, sp
   mov ds, ax
   mov es, ax
   sti
   mov di, 0E00h
   mov cx, 0100h
   repnz movsw ; 本代码移到 0: E00H 处
 db 0eah, 1dh, 0eh, 0, 0 ; jmp 0000: 0E1CH
   cmp byte ptr ds: [0c0h], 0ffh; 再次进入吗?
   jz exit    ; 是, 转本地硬盘启动
   mov bx, 7000h; 否, 将 C: 盘尾柱 0DH 头处的
   mov es, bx    ; BOOTCODE 读到 7000: 0000 处
   xor bx, bx
   mov cx, ds: [1BAH +0E00h]; 1BAH 字节存放着 C: 盘的尾柱扇号
   mov dx, 0d80h
   mov ax, 023fh
   int 13h
   jc exit
   add bx, 7e00h
   inc dh
   mov ax, 023fh
   int 13h
   jc exit
   add bx, 7e00h
   inc dh
   mov ax, 0202h ; 共读入了 128 扇 =64K 字节
   int 13h
   jc exit
   mov byte ptr ds: [0c0h], 0ffh; 第一次执行    mov word ptr ds: [4f2h], 0  后设此标志
 db 9ah, 3, 0, 0, 70h; CALL 7000: 0003 执行 BOOTCODE
   xor ax, ax ; 寄存器清 0 准备 INT19H 调用
   mov dx, ax
   mov cx, ax  mov bx, ax
   int 19h  ; 再次进入 BOOTCODE
exit: mov bx, 7c00h; 读入原 001 扇到 0: 7C00H  mov ax, 0
   mov ds, ax  mov es, ax  mov cx, 003fh
   mov dx, 0080h
   mov ax, 0201h
   int 13h
boot: db 0eah, 0, 7ch, 0, 0 ; 跳转到 0: 7C00H
patitab:  org 7dbeh
  db 64 dup(0) ; 预置分区表 64 字节, 实际数    org 7dfeh
; 据由原 001 扇传递过来 db 55h, 0aah  ; 主引导扇有效标志
code ends
   end
```

四、BOOTCODE 的安装及 001 扇的替换

第 3 节提到 BOOTREC 从 C：盘尾柱 0DH 头处读入 BOOTCODE 并从 0 柱 0 头 3FH 扇读入原主引导记录，这就要求我们设计一个安装程序，事先将 BOOTCODE 和原主引导记录存放在该处。这个安装要完成如下任务：

读出 001 扇上的主引导扇并将它保存在 0 柱 0 头 3FH 扇上；将分区表中 C：盘的尾柱号 （C4H ~ 1C5H 字节）减 1 使尾柱前移，空出尾柱装 BOOTCODE，新的尾柱扇号保存在 1BAH 字节中供 BOOTREC 使用。读入 BOOTREC 文件，将主引导扇中的 1BAH 字节开始的 68 个字节　含 1BEH 处的 64 字节分区表）传送到 BOOTREC 的同一位置，再将 BOOTREC 写入 001 扇。读入 BOOTCODE 文件，将它写入 C：盘尾柱 0DH 头 1 扇区开始的 128 扇区中，这表明 BOOTCODE 的长度不能超过 64K。读入 0 柱 1 头 1 扇区上的分区引导扇并将它保存在 0 柱 0 头 3E 扇区上，该扇区第 18H 字节为每磁头的扇区数、第 1AH 字节为每柱磁头数，二者的乘积就是每柱扇区总数。第 20H 字节处的 4 字节是 C：盘扇区总数，从 C：盘扇区总数中减去 1 个柱的扇区数，然后重写分区引导扇。只有这样 C：盘尾柱才真正不属于 C：盘，其上保存的 BOOTCODE 安全可靠，这一点非常重要。为了避免重复安装，在 BOOTREC 的第 1BDH 字节上设置一个标志 FFH，安装程序读入 001 扇区检测到此标志时就不再安装。

一个实用的安装程序见程序二。要提请读者注意的是：本文适用于逻辑 C：盘小于 8. 4G 情况，若大于 8. 4G ，文中所有对扇区的读写均不能用常规 INT13H，要用扩展的 INT13H。然而扩展 INT13H 中断的调用接口参数并未见有关文档公布，笔者将专门撰文论述大容量硬盘扇区的直接读写问题。

程序二　安装程序　INST. ASM 的汇编源程序

```
DATA SEGMENT PARA PUBLIC ´DATA´
  ROMBUF DB 0FFFFH DUP(0)
DATA ENDS
code segment para public ´code´
   assume cs: code, ds: data, ss: code
start proc far
   jmp bg
rwbuf db 512 dup(0)
fbuf  db 512 dup(0)
fname db ´bootrec´, 0
romname db ´bootcode´, 0
errts db ´READ OR WRITE ERR´, ´$´
stack db 200h dup(´ ´)
bg: lea ax, bg
   mov sp, ax
   mov ax, cs
   mov ss, ax
   push cs
   pop ds
   push cs
   pop es
   lea bx, rwbuf
   mov cx, 1
```

```
    mov dx, 0080h
    mov ax, 0201h ; 读入 001 扇
    int 13h
    jnc ok
    jmp rwerr
ok: cmp byte ptr [bx+1bdh], 0ffh; 安装过吗?
    jnz ok1
    jmp exit
ok1: mov cx, 003fh
    mov ax, 0301h ; 保存原 001 扇
    int 13h
    mov ax, [bx+1c4h]
    mov si, bx
    mov bx, ax
    and ax, 0ffc0h
    add ax, 1
    mov [si+1bah], ax ; 保存尾柱扇号
    mov ax, bx
    and al, 0c0h
    mov cl, 6
    shr al, cl
    mov ah, al ; 柱号高 2 位
    mov al, bh
    sub ax, 1  ; 柱号减 1
    shl ah, cl
    and bl, 3fh
    add bl, ah
    mov bh, al
    mov cx, bx ; CX=柱扇号
    mov [si+1c4h], cx
    mov ax, 3d02h; 打开 BOOTREC 文件
    lea dx, fname
    int 21h
    jnc ok2
    jmp rwerr
ok2: mov bx, ax
    mov cx, 200h
    lea dx, fbuf
    mov ah, 3fh  ; 读入文件
    int 21h
    mov ah, 3eh
    int 21h
    lea si, rwbuf+1bah
    lea di, fbuf+1bah
    mov bx, di
    mov cx, 68   ; 传送尾柱扇号、安装标志及
    cld          ; 分区表共 68 字节
    repnz movsb
    mov byte ptr [bx+3], 0ffh ; 置安装标志
    mov ax, 0301h
    lea bx, fbuf
    mov cx, 1
    mov dx, 0080h
    int 13h
    mov ax, 3d00h ; 打开 BOOTCODE 文件
    lea dx, romname
    int 21h
    jnc n1
    jmp rwerr
n1: mov bx, ax
    mov dx, data
    mov ds, dx
    lea dx, rombuf
    mov cx, 0ffffh
    mov ah, 3fh  ; 读入 BOOTCODE
    int 21h
    jnc n2
    jmp rwerr
n2: mov ah, 3eh
    int 21h
    push ds
    pop es
    push cs
    pop ds
    mov ax, 033fh; 写到尾柱 0dH 头 1 扇区处
    lea bx, rombuf
    lea si, fbuf
    mov cx, [si+1bah]
    mov dx, 0d80h
    int 13h
    add bx, 7e00h
    inc dh
    mov ax, 033fh
    int 13h
    add bx, 7e00h
    inc dh
    mov ax, 0302h
    int 13h
    push cs
    pop es
    lea bx, rwbuf
    mov ax, 0201h
    mov cx, 1
    mov dx, 0180h
    int 13h  ; 读入 0 柱 1 头 1 扇
    jc rwerr
    mov ax, 0301h
    mov cx, 003eh
    mov dx, 0080h
    int 13h
    mov si, bx
    mov ax, [si+18h]
    mov bx, [si+1ah]
    mul bx    ; 乘积=每柱扇区总数
    mov bx, dx ; bx=乘积高位, ax=乘积低位
    mov dx, [si+20h] ; dx=C: 盘扇区总数低位
    mov cx, [si+22h] ; cx=C: 盘扇区总数高位
    sub dx, ax       ; 扇区总数减去尾柱扇区数
    sbb cx, bx
    lea bx, rwbuf
    mov [si+20h], dx ; 更新扇区总数
    mov [si+22h], cx
    mov cx, 1
    mov dx, 0180h
    mov ax, 0301h
    int 13h
    jmp exit
rwerr: lea dx, errts
    mov ah, 9
    int 21h
exit: mov ax, 4c00h
    int 21h
start endp
code ends
    end start
```

收稿日期：2001 年 3 月 9 日）

关于微软 Web 服务器 IIS 5 的几个高级主题的讨论（一）

魏 臻 路 军

摘 要 本文主要介绍了微软 Windows 2000 Server 集成的 IIS 5 Web 服务器，详细地探讨四个较为专业、高级的主题：虚拟站点配置、ASP 编程、ASP 扩展和 IIS 5 的安全问题。为网管、开发人员等广大同行提供参考。

关键词 Windows 2000 Server，DNS，IIS，Web，TCP / IP，HTML，ASP，VBScript，JavaScript，Visual InterDev，SOM，DTC，COM，XML，Windows Script Component，VB，VC Unicode，注册表，漏洞

引用一位资深 IT 作家的话：“除非过去的五年你一直与世隔绝，住在山洞中，否则你不可能不意识到 Internet 正成为世界运行的主要方式”。现在，Internet 对世界的影响越来越大，它已经渗透到我们生活的每个角落：在我们的工作当中，在我们的生活当中。而且，这种趋势将来还会发展下去。任何人随时随地都可以获得大量的信息，这实在令人瞠目结舌。

微软绝不会对如此精彩的部分袖手旁观：IE 浏览器捆绑，NT 的 BackOffice 集成。现在，微软又在其跨世纪战略产品——Windows 2000 中集成了 Internet Information Server（Internet 信息服务器，以后简称 IIS）的最新版本 5. 0，这无疑是最有意义的一件事。

IIS 是一个可以提供 HTTP（Web）、FTP（文件传输）、NNTP（网络新闻）、SMTP（E - Mail）服务的全功能平台。IIS 的安装、配置和管理简单，界面友好，功能强大——微软自己的 Web 站点运行的就是 IIS，它在平均每天百万次的访问量下保持稳定、顺畅的运行，这就是对 IIS 最好的肯定。

据 Netcraft（www. netcraft. com / survey）对 Web 服务器的调查统计，IIS 荣登最受欢迎的 Web Server 第二位宝座。IIS 的各种版本在这次抽样调查中使用率占 23%。那么谁是 Web 服务器的冠军呢？也许我们大家都心知肚明了，它就是 Apache——NCSA HTTPd Web Server 的一个原代码公开版本，它占了 50% 以上，而且还在增长。为什么呢？如果我们对 Microsoft 产品和 Linux 都有所了解，就会明白：Windows 产品虽然界面华美，但是它不稳定，GUI（图形用户界面）耗费资源，以致于系统效率不高，故而对系统的硬件配置要求比较苛刻。另外，使用微软产品您就得向比尔·盖茨交钱，而且价格不菲。恰恰这些缺点正是 Apache 的长处：稳定、高效、免费而且原代码完全公开。

但是微软绝不会甘心屈居第二，它会不断地给 IIS 添加新的特性，不断增强、完善它的功能。现在，就让我们就微软 Web 服务器的最新产品——IIS 5. 0，略去一般安装、日常维护等常见事务，来讨论一下几个比较高级的主题（我们假设您目前使用的是 Windows 2000 Server。当然，Professional 版同样包含 IIS 5）：

- 虚拟站点配置技术
- ASP（Active Server Pages）编程问题
- ASP 扩展
- IIS5. 0 安全问题探讨——攻击与防御。

第一部分 虚拟站点配置技术

一、一机多站

所谓一机多站，是在一个物理系统上存放多个 Web 站点。规模较小的公司的站点托管，学校实验室和一些特殊场合可以使用这种方式架设网站。为了实现一机多站，我们通常有三种方法：

1. 多 IP 地址：

这是最普通常用的方法。不管你给服务器安装多个网卡还是单网卡，你都可以给服务器分配多个 IP 地址，并且为各个虚拟站点（一机多站的情况下，通常称这种站点为虚拟站点）分配不同 IP。如果同时有域名服务，还得将各 IP 正确匹配 DNS 记录。举例如下：

我同时给我的 Web 服务器分配三个 IP：192. 168. 1. 1；192. 168. 1. 100；192. 168. 1. 200。操作步骤为：控制面板→网络和拨号连接（或右击网上邻居→属性）→右击本地连接→属性；选择 TCP / IP 协议，单击“属性”按钮，在 TCP / IP 属

图 1

图 2

性对话框中点击“高级 V . . . ”按钮，在“IP 设置”选项卡中添加 IP。

然后在“Internet 信息服务设置”中给各个虚拟站点分配 IP：管理工具→Internet 服务管理器　或运行% WinDir% hsystem32 hinetsrv hIIS. msc 等方法）打开 Internet 信息服务控制台，在其中某个虚拟站点上右击，再点击“属性”　在打开的对话框中选取“Web 站点”选项卡，然后在“Web 站点标识”的“IP 地址”下拉列表中选取对应的 IP，完成后确定退出。用同样的方法依次为所有虚拟站点分配各自的 IP。

最后，您还得为虚拟站点定义 DNS 名称：管理工具→DNS　或运行 DNSMGMT. msc 等方法）打开 DNS 控制台，然

图 3

图 4

后在合适的域层次中新建域、新建主机等等，这里不再赘述。如图 3 示例，创建了“www. KJGG_MIS. LuJun. net”，其 IP 地址为：192. 168. 1. 200。

2. 单 IP 地址，多端口号：

给服务器分唯一 IP 地址，同时给服务器上的每个虚站分配不同的 TCP 端口号。这种方法主要用在 IP 资源比较紧张，并且除了默认站点外　默认站点使用 80 号端口），其余各个站点不需要公开访问的情况。其设置方法同图 2 所述相似，更改“Web 站点标识”的“TCP 端口”即可。

3. 使用主机头名：

和上面两种相比较，使用主机头名是一种较容易被忽视的方法。什么叫主机头名呢？机头名是将浏览器所要求的主机名

例如 www. microsoft. com）包含在传送到 Web 服务器的 HTTP 头中的一种途径。当我们在浏览器中键入“www. microsoft. com”时，浏览器会查看 URL 里包含的主机的 IP 地址，然后在 80 端口　默认）建立一个 TCP 连接。一旦连接建立后，它会把一个页面请求传送到 Web 服务器，并把主机域名信息　主机头名）包含在请求中。IIS 收到该请求后，会查看该主机头

名，并且和服务器清单中的名称进行比较，然后将正确的页面发送回去。

让我们延续上面的例子，假如我们不但要在 IP 地址 192. 168. 1. 200 上提供虚拟站点 www. KJGG_MIS. LuJun. net，而且同时提供虚拟站点 www2. KJGG_MIS. LuJun. net （可以是别的名称，如：www. Test. LuJun. net，只要在 DNS 设置中做相应变化，同时主机头名改为 www. Test. LuJun. net 即可）。首先打开 DNS 设置控制台，展开域 LuJun. . net，展开子域 KJGG_MIS，添加主机 www2，将其 IP 地址同样设置成 192. 168. 1. 200，如图 5 所示。

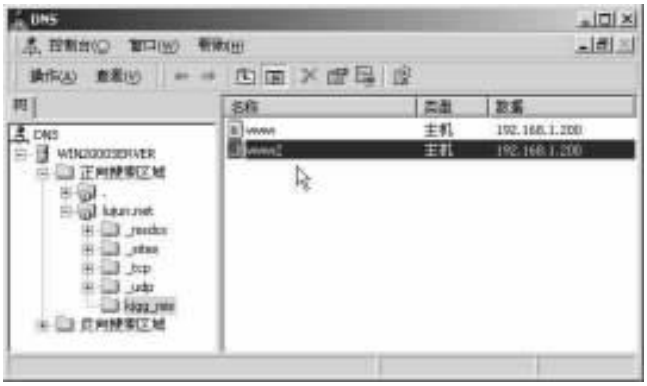


图 5

图 6

关闭 DNS 设置，然后打开 Internet 信息服务控制台，新建站点 KJGG_MIS2（对应图 2），将其 IP 地址设为 192. 168. 1. 200，再点击“高级 D . . . ”，编辑该站点，将其主机头名设置成“www2. KJGG_MIS. LuJun. net”，如图 6 所示。

这样，我们在一个 IP 192. 168. 1. 1 上同时提供两个站点：www. KJGG_MIS. LuJun. net 和 www2. KJGG_MIS. LuJun. net。注意：只有 Microsoft Internet Explorer 3. 0、Netscape Navigator 2. 0 以及更高版本才支持使用主机头名。如果浏览器不支持主机头名，那么将传回默认站点的默认页面（例子中的 www. KJGG_MIS. LuJun. net）。

二、多机一站

一般的商业网站，尤其是像我们熟悉的“微软中国”、“新浪网”、“CCTV”等大型网站，都不可能是在一台机器上提供整个的 Web 服务，而是由多台服务器同时运行，提供浏览服务，甚至跨地区还有镜像。这就是多机一站。

要实现多机一站，也同样有许多种方法。总的来说，大体上可分成两类：

● 第一类是高成本的“解决方案”，也是比较高级和专业的方式，例如采用群集技术，这种解决方案需要大量的资金购买硬件及群集软件；

● 第二类是廉价的解决方法，比如说在网站编程开发上动脑筋，整个网站分成几大类，首页放在默认站点，而将其余各类分别重新定向到另外的服务器上去。这种方法是将 Web 服务的整个内容进行了拆分。还比如对访问进行“分流”——由若干服务器同时提供同样的 Web 服务，每个服务器承担部分访问量。这里介绍一种“DNS 轮询法”：

假如现在有三台 IIS 服务器，IP 地址分别为 192. 168. 1. 100、192. 168. 1. 101、192. 168. 1. 102，我们可以在 DNS 中建立名称都为“www. KJGG_MIS. LuJun. net”的主机，如图 7 所示。

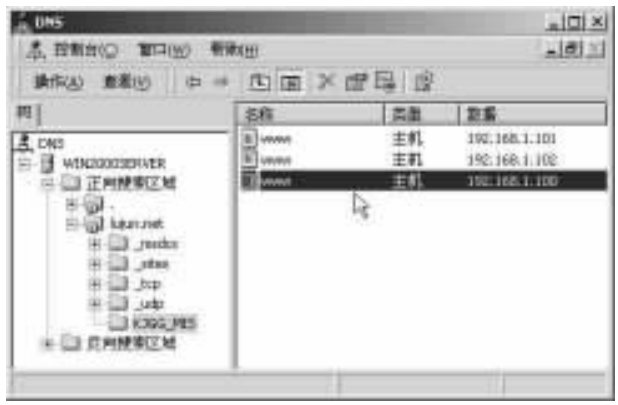


图 7

这样，当第一个人访问“www. KJGG_MIS. LuJun. net”时，DNS 会返回给他 192. 168. 1. 100，即由第一台服务器提供服务，第二个访问者会被定向到 192. 168. 1. 101，即第二台服务器，同样第三个访问者会被定向到第三台服务器 192. 168. 1. 102。而第四个访问者则会被定向到 192. 168. 1. 100，即轮回到第一台服务器。这种进程被成为“循环 DNS”，这种多机一站的实现方法称为“DNS 轮询法”。这样，每台服务器承担了 1 / 3 的访问量。

注意：如果其中一台服务器出现故障，则 DNS 会毫不知情地仍然将它的 IP 地址返回给浏览者，当然，该访问者就会收到出错信息。

注：微软在 Windows 2000 Advanced Server 中集成了 TCP / IP Network Load Balancing 软件组件（NLB），这是一个最多支持 32 个主机的软件负载平衡解决方案。由于本篇基于 Windows 2000 Server，所以不准备详细介绍这个组件。

（未完待续）

（收稿日期：2001 年 2 月 8 日）

用 VC 编写窗口化 PING 应用程序

赵　谦

摘　要　本文阐述了在 Windows 下编写 ICMP 网络协议应用程序的原理，并通过实例介绍了在 VC 中编写窗口化 PING 应用程序的方法。

关键词　Visual C + +，ICMP 协议，Socket API

一、引言

PING 应用程序是 Windows 下的一个非常有用的网络工具程序。它通过向网络中发送 ICMP 报文包，然后通过分析网络中返回的 ICMP 报文中的信息来达到判断诸如：路由器故障，路由选择失败，目的网络未知等的目的。

二、ICMP 协议简介

由于在 Internet 协议软件中提供了不可靠的无连接数据报传送服务，当网络中出现了诸如路由器不能正确选择路由或传送数据报，或者它检测到一个异常条件影响它转发数据报，路由器需要通知源站点采取措施避免或纠正出现的问题。为了使互联网中的路由器报告差错或提供有关意外情况的信息，在 TCP / IP 中设计了一个特殊用途的报文机制，称为 Internet 控制报文协议（ICMP）。它是 IP 的一部分，并在每个 IP 实现中都是必需的。

ICMP 报文为两级封装，如图 1 所示。每个 ICMP 报文放在 IP 数据报的数据部分中，而数据报本身放在帧的数据部分中通过物理网络。携带 ICMP 报文的数据报与携带用户信息的数据报具有完全相同的路由选择，没有增加可靠性或优先级。ICMP 报文由三个字段组成，即一个 8 位整数的报文类型（Type）字段用来标识报文，一个 8 位代码（Code）字段提供有关报文类型的进一步信息、以及一个 16 位校验和（CheckSum）字段。此外，报告差错的 ICMP 报文总是包括产生问题的数据报报头及开头的 64 位数据。后者使接收方能更精确地判断是哪个协议及哪个应用程序对该数据报负责。

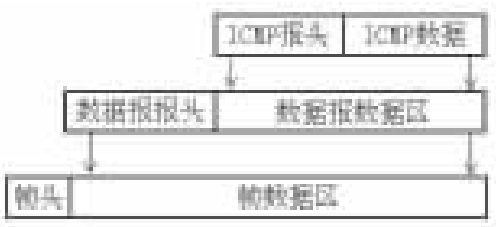


图 1　ICMP 报文的两级封装

在 Windows Socket 中，我们可以利用原始套接字 Raw Socket 来控制 ICMP 报文。而 Windows Socket 的各个版本中只有版本号 2. 1 以上的版本才支持这项功能。因此我们 Socket2. 1 的 Socket 构造器来创建一个 Socket 对象：

WSASocket (int af, int type, int protocol, LPWSAPROTOCOL_INFO lpProtocolInfo, GROUP g, DWORD dwFlags)

其含义分别为 af 为地址族描述；type 新套接字的类型描述，protocol 套接字使用的特定协议；lpProtocolInfo 一个指向 WSAPROTOCOL_INFO 结构的指针，该结构定义所创建套接字的特性；g 套接字组的描述符；dwFlags 套接字属性描述。

三、编程实例

1. 创建一个新的 AppWizard 应用工程，名称为 PingWin，选择基于 Dialog based 的应用程序框架。

2. 然后单击“Next”按钮，进入第 2 步对话框，去掉 ActiveX Controls 选项，去掉 Windows Sockets 选项（如图 2）。

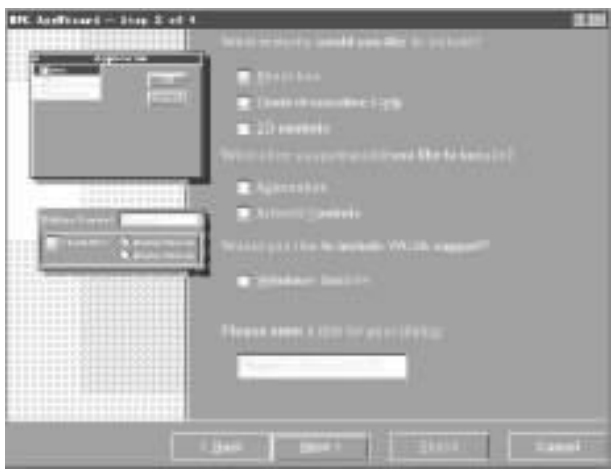


图 2

3. 然后单击“Finish”按钮，完成框架程序的创建。

4. 在 StdAfx. h 头文件中加入 Socket2 头文件：#include < winsock2. h>

5. 添加 Socket2 的输入库文件，选择菜单 Project 中的 Set-

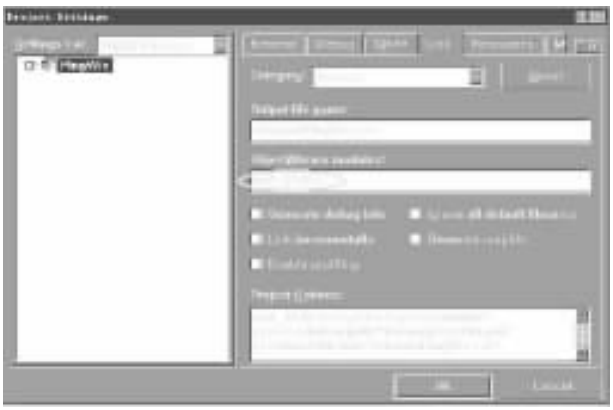


图 3

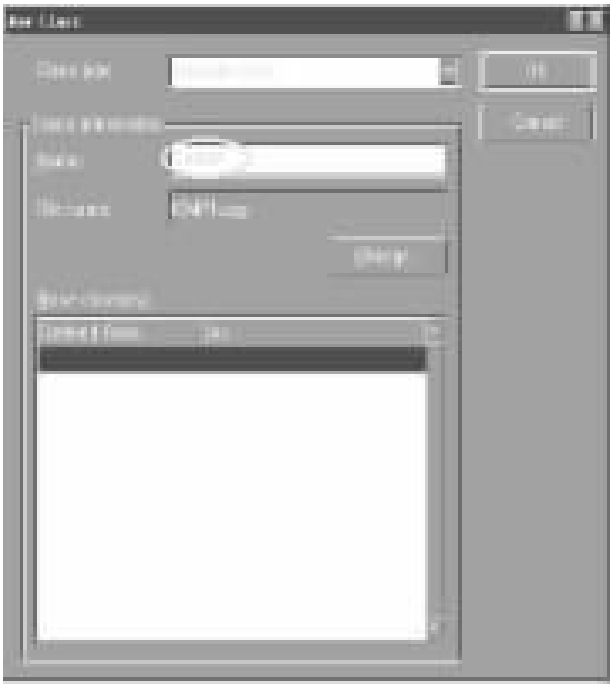


图 4

ting... 进入 Project Setting... 对话框中（如图 3），在 Object / library modules 框中输入 ws2_32. lib，然后点击“确定”。

6. 建一个自定义类，点击菜单中的 Insert 中的 New Class...，进入 New Class... 对话框（如图 4）。在 Class type 中选择 Generlc Class，在 Name 中输入你的自定义类名，在这里是 CICMP。然后点击“确定”。

在自定义类中添加函数

```
BOOL Initialize(void);                          //初始化
void Uninitialize(void);                        //反初始化
USHORT CheckSum(USHORT * buffer, int size); //检查和
BOOL SendICMP(int nMsg, char * pAddr);          //发送报文
BOOL SendICMP(int nMsg, sockaddr_in * pAddr); //发送报文
BOOL RecvICMP(void);                            //接收报文
```

7. 在对话框模板设置内将对话框设成图 5 所示的样子。

设置地址框的 ID 为 IDC_ADDRESS，类型框的 ID 为 IDC_COMBO1，其 DATA 域数据为：回送请求、目的站不可达、源站抑制；发送按钮的 ID 为 IDC_BTNSEND；接收按钮的

图 5

ID 为 IDC_BTNRECV；退出按钮的 ID 为 IDCANCEL；状态栏的 ID 为 IDC_STATIC_STATE。

启动 ClassWizard，在 Member Variables 标签中添加变量：对 IDC_BTNSEND 添加 CButton 型变量 m_conBtnSend、对 IDC_BTNRECV 添加 CButton 型变量 m_conBtnRecv、对 IDC_STATIC_STATE 添加 CStatic 型变量 m_conState、对 IDC_ADDRESS 添加 CString 型变量 m_strAddress、对 IDC_COMBO1 添加 int 型变量 m_nMsg。

四、添加代码

ICMP. H

```
//////////////////////////////////////
// ICMP. h: interface for the CICMP class.
//////////////////////////////////////
#if !defined(AFX_ICMP_H__FFBF406F_1036_11D5_856C_AFBBB6690115__INCLUDED_)
#define AFX_ICMP_H__FFBF406F_1036_11D5_856C_AFBBB6690115__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#define EXITPROCESS 0xFFFF
#define DEF_PACKET 32
#define MAX_PACKET 1024
#define ICMP_MIN          8     //ICMP 包的最小尺寸
//ICMP 消息
#define ICMP_ECHO         8       //回送请求
#define ICMP_ECHOREPLY    0       //回送应答
#define ICMP_CANTTO 3             //目的站不能到达
#define ICMP_DELAY  4             //源站抑制
#define ICMP_TIME         13      //时间请求
#define ICMP_TIMEREPLY    14      //时间应答
#define ICMP_ADD          17      //地址掩码请求
#define ICMP_ADDREPLY     18      //地址掩码应答
//IP 及 ICMP 数据结构
typedef struct ip_hander{
  unsigned int    IHL: 4;     //用 32 位字表示的报头长度
  unsigned int    ver: 4;     //协议版本
  unsigned char   level;      //优先级
  unsigned short  len;        //数据长度
```

```
    unsigned short  ID;              //标识
    unsigned short  mflag;           //其它标记
    unsigned char   ttl;             //生命期
    unsigned char   prot;            //协议
    unsigned short  cksum;           //检查和
    unsigned int    sourceIP;        //源 IP 地址
    unsigned int    destIP;          //目的 IP 地址
}IP_HANDER;
typedef struct icmp_hander{
    unsigned char   type;                //类型
    unsigned char   code;                //编码
    unsigned short  cksum;           //检查和
    unsigned short  ID;              //标识
    unsigned short  number;          //计数值
    unsigned int    time;                //时间
}ICMP_HANDER;
class CICMP
{
public:
    CICMP();
    virtual ~CICMP();
    IP_HANDER *         m_pRec;
    ICMP_HANDER *       m_pIcmp;
    SOCKET              m_Sock;
    CString m_strInfo;
    sockaddr_in m_sockAddr;
    int m_nMsg;
    BOOL Initialize(void);                   //初始化
    void Uninitialize(void);                 //反初始化
USHORT CheckSum(USHORT * buffer, int size); //检查和
BOOL SendICMP(int nMsg, char *  pAddr); //发送报文
BOOL SendICMP(int nMsg, sockaddr_in *  pAddr); //发送报文
    BOOL RecvICMP(void);                     //接收报文
};
#endif // !defined(AFX_ICMP_H__FFBF406F_1036_11D5_85-
6C_AFBBB6690115__INCLUDED_)
```

ICMP. CPP

```
////////////////////////////////////
// ICMP. cpp: implementation of the CICMP class.
////////////////////////////////////
#include "stdafx. h"
#include "PingWin. h"
#include "ICMP. h"
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] =__FILE__;
#define new DEBUG_NEW
#endif
////////////////////////////////////
// Construction/Destruction
////////////////////////////////////
CICMP: : CICMP()
{
    m_Sock  = 0;
    m_pRec  = NULL;
    m_pIcmp  = NULL;
    m_pRec  =(IP_HANDER *)new BYTE[MAX_PACKET];
m_pIcmp  = (ICMP_HANDER *)new BYTE[MAX_PACKET];
}
CICMP: : ~CICMP()
{
  delete [] m_pRec;
  delete [] m_pIcmp;
}
BOOL CICMP: : Initialize()
{
  //初始化 Socket2. 1
  WSADATA wsa;
  if( WSAStartup(MAKEWORD(2, 1), & wsa) )
  {
      m_strInfo = "错误: 网络问题导致程序初始化失败!";
      return FALSE;
  }
  //建立 socket
  m_Sock = WSASocket (AF_INET, SOCK_RAW, IPPRO-
TO_ICMP, NULL, 0, 0);
  if(!m_Sock)    {
  m_strInfo = "错误: 网络问题导致 Socket 初始化失败!";
      return FALSE;
  }
  //设置超时时间
  int timeout = 1000;
  setsockopt(m_Sock, SOL_SOCKET, SO_RCVTIMEO, (char
*)& timeout, sizeof(timeout));
  timeout = 1000;
  setsockopt(m_Sock, SOL_SOCKET, SO_SNDTIMEO, (char
*)& timeout, sizeof(timeout));
  return TRUE;
}
void CICMP: : Uninitialize()
{
  if(m_Sock)
      closesocket(m_Sock);
  WSACleanup();
}
USHORT CICMP: : CheckSum(USHORT *buffer, int size)
{
 unsigned long cksum = 0;
 while(size > 1) {
  cksum + = * buffer + +;
  size  - =sizeof(USHORT);
 }
 if(size ) {
  cksum  + = * (UCHAR * )buffer;
 }
 cksum = (cksum >> 16) + (cksum & 0xffff);
 cksum + = (cksum >>16);
 return (USHORT)( ~cksum);
}r
BOOL CICMP: : SendICMP(int nMsg, cha  * pAddr)
{
  //取出目的地址
  sockaddr_in sockAddr;
  memset((void *)& sockAddr, 0, sizeof(sockAddr));
```

```
    sockAddr.sin_family = AF_INET;
    sockAddr.sin_port = 0;
    sockAddr.sin_addr.S_un.S_addr = inet_addr(pAddr);
    return SendICMP(nMsg, &sockAddr);
}
BOOL CICMP::SendICMP(int nMsg, sockaddr_in *pAddr)
{
    //填充 ICMP 数据各项
    char *p_data;
    m_plcmp->type = nMsg;
    m_plcmp->code = 0;
    m_plcmp->ID = (USHORT)GetCurrentProcessId();
    m_plcmp->number = 0;
    m_plcmp->time = GetTickCount();
    m_plcmp->cksum = 0;
    //填充数据区
p_data = ((char*)m_plcmp + sizeof(ICMP_HANDER));
    memset((char*)p_data, 'E', DEF_PACKET);
    //填充检查和项
  m_plcmp->cksum = CheckSum((USHORT*)m_plcmp,
DEF_PACKET+sizeof(ICMP_HANDER));
    //发送数据
    int state;
state = sendto(m_Sock, (char*)m_plcmp, DEF_PACKET+
sizeof(ICMP_HANDER), NULL, (struct sockaddr*)pAddr,
sizeof(sockaddr));
    //如果有错误
    if(state == SOCKET_ERROR) {
        if(GetLastError() ==WSAETIMEDOUT)
            m_strInfo = "连接因超时而失败!(发送)";
        else
            m_strInfo = "出现未知发送错误!";
        return FALSE;
    }
    //如果发送的数据长度不正确
    if(state < DEF_PACKET) {
        m_strInfo = "发送数据不正确!";
        return FALSE;
    }
    //保存数据
    m_nMsg = nMsg;
    memcpy((void*)&m_sockAddr, (void*)pAddr,
        sizeof(sockaddr_in));
    return TRUE;
}
BOOL CICMP::RecvICMP()
{
    //接收数据
    int state;
    int len = sizeof(sockaddr_in);
  state=recvfrom(m_Sock, (char*)m_pRec, MAX_PACKET,
0, (struct sockaddr*)&m_sockAddr, &len);
    if (state == SOCKET_ERROR) {
        if (WSAGetLastError() == WSAETIMEDOUT)
            m_strInfo = "连接因超时而失败!";
        else
            m_strInfo = "未知接收错误!";
        return FALSE;
    }
    //分析数据
    int iphdlen;
    iphdlen = m_pRec->IHL * 4;
    if (state < (iphdlen+ICMP_MIN)) {
        m_strInfo = "目的地址的响应数据不正确";
        return FALSE;
    }
    ICMP_HANDER *p_recicmp;
p_recicmp=(ICMP_HANDER*)((char*)m_pRec+iphdlen);
    if(p_recicmp->type == ICMP_CANTTO) {
        m_strInfo = "路由器无法转发";
        return FALSE;
    } {
    if(p_recicmp->type == ICMP_DELAY)
        m_strInfo = "源站抑制";
        return FALSE;
    }
    if(m_nMsg == ICMP_ECHO) {
        if (p_recicmp->type != ICMP_ECHOREPLY) {
            m_strInfo = "接收到非正常数据";
            return FALSE;
        }
        char* addr;
        addr = inet_ntoa(m_sockAddr.sin_addr);
    m_strInfo.Format("从%s 接收到%d 个字节的数据, 响应时
间: %dms.", addr, len, GetTickCount()-p_recicmp->time);
        return TRUE;
    }
}
```

PingWinDlg.h

```
// PingWinDlg.h : header file
#if !defined(AFX_PINGWINDLG_H__C06B1347_65D1_11D4_
B8C0_444553540000__INCLUDED_)
#define AFX_PINGWINDLG_H__C06B1347_65D1_11D4_B8C0_
444553540000__INCLUDED_
#include "ICMP.h"      // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
/////////////////////////////////
// CPingWinDlg dialog
class CPingWinDlg : public CDialog
{
// Construction
public:
  CICMP m_icmp;
CPingWinDlg(CWnd* pParent=NULL); //standard con-
structor
// Dialog Data
  //{{AFX_DATA(CPingWinDlg)
  enum { IDD = IDD_PINGWIN_DIALOG };
  CButton   m_conBtnSend;
  CButton   m_conBtnRecv;
  CStatic   m_conState;
```

```
    CString    m_strAddress;
    int        m_nMsg;
    //}}AFX_DATA
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPingWinDlg)
    protected:
virtual void DoDataExchange(CDataExchange * pDX);
// DDX/DDV support
//}}AFX_VIRTUAL
// Implementation
protected:
    HICON m_hIcon;
    // Generated message map functions
    //{{AFX_MSG(CPingWinDlg)
    virtual BOOL OnInitDialog();
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    virtual void OnCancel();
    virtual void OnOK();
    afx_msg void OnBtnsend();
    afx_msg void OnBtnrecv();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.
#endif // !defined(AFX_PINGWINDLG_H__C06B1347_65D1_
11D4_B8C0_444553540000__INCLUDED_)
```

PingWinDlg.cpp

```
// PingWinDlg.cpp : implementation file
#include "stdafx.h"
#include "PingWin.h"
#include "PingWinDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////t
// CAboutDlg dialog used for App Abou
class CAboutDlg : public CDialog
{
public:
    CAboutDlg();
// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);
// DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
protected:
  //{{AFX_MSG(CAboutDlg)
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
  //{{AFX_DATA_INIT(CAboutDlg)
  //}}AFX_DATA_INIT
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CAboutDlg)
  //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
  //{{AFX_MSG_MAP(CAboutDlg)
      // No message handlers
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////
// CPingWinDlg dialog
CPingWinDlg::CPingWinDlg(CWnd* pParent /* =NULL */)
  : CDialog(CPingWinDlg::IDD, pParent)
{
  //{{AFX_DATA_INIT(CPingWinDlg)
  m_strAddress = _T("");
  m_nMsg = -1;
  //}}AFX_DATA_INIT
  // Note that LoadIcon does not require a subsequent De-
stroyIcon in Win32
  m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
void CPingWinDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CPingWinDlg)
  DDX_Control(pDX, IDC_BTNSEND, m_conBtnSend);
  DDX_Control(pDX, IDC_BTNRECV, m_conBtnRecv);
  DDX_Control(pDX, IDC_STATIC_STATE, m_conState);
  DDX_CBString(pDX, IDC_ADDRESS, m_strAddress);
  DDX_CBIndex(pDX, IDC_COMBO1, m_nMsg);
  //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CPingWinDlg, CDialog)
  //{{AFX_MSG_MAP(CPingWinDlg)
  ON_WM_SYSCOMMAND()
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
  ON_BN_CLICKED(IDC_BTNSEND, OnBtnsend)
  ON_BN_CLICKED(IDC_BTNRECV, OnBtnrecv)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////
// CPingWinDlg message handlers
```

```
BOOL CPingWinDlg: : OnInitDialog()
{
    CDialog: : OnInitDialog();
// 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
    m_nMsg = 0;
    UpdateData(FALSE);
    if(! m_icmp. Initialize()) {
        m_conBtnSend. EnableWindow(FALSE);
        m_conBtnRecv. EnableWindow(FALSE);
        m_conState. SetWindowText(m_icmp. m_strInfo);
    }
    return TRUE;
// return TRUE unless you set the focus to a control
}
void CPingWinDlg: : OnSysCommand(UINT nID, LPARAM
lParam)
{
    if ((nID & 0xFFF0) = = IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout. DoModal();
    }
    else
    {
        CDialog: : OnSysCommand(nID, lParam);
    }
}
// If you add a minimize button to your dialog, you will
need the code below to draw the icon. For MFC applica-
tions using the document/view model, this is automatically
done for you by the framework.
void CPingWinDlg: : OnPaint()
{
    if (IsIconic())
    {;
        CPaintDC dc(this); // device context for painting
        SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc. GetSafeHdc(), 0);
        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(& rect);
        int x = (rect. Width() – cxIcon + 1) / 2;
        int y = (rect. Height() – cyIcon + 1) / 2
        // Draw the icon
        dc. DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog: : OnPaint();
    }
}
// The system calls this to obtain the cursor to display while
the user drags the minimized window.
HCURSOR CPingWinDlg: : OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}
void CPingWinDlg: : OnCancel()
{
    // TODO: Add extra cleanup here
    m_icmp. Uninitialize();
    EndDialog(TRUE);
}
void CPingWinDlg: : OnOK()
{
    // TODO: Add extra validation here
    OnBtnsend();
}
void CPingWinDlg: : OnBtnsend()
{
    // TODO: Add your control notification handler code here
    CWnd * pWnd;
    pWnd = GetDlgItem(IDC_ADDRESS);
    CString str;
    pWnd – >GetWindowText(str);
    if(str. IsEmpty()) {
        MessageBox("请输入地址!");
        pWnd – >SetFocus();
    }
    else
    {
        UpdateData();
        int nMsg;
        switch(m_nMsg)
        {
        case 0: nMsg = 8; break;
        case 1: nMsg = 3; break;
        case 3: nMsg = 4;
        }
```

下转第 86 页)

位图的高倍放大与平滑拖动

谢经荣

摘 要 在位图显示中，通过使用一个介于位图数据和显示设备之间的内存位图，作为向显示设备显示位图的后备缓冲，很好地解决了位图的高倍放大与平滑拖动的问题，对于 Windows 平台上其他图形的处理也具有一定参考价值。

关键词 高倍放大，平滑拖动，双缓冲

一、问题的提出

图形处理是计算机应用中一个十分活跃的领域，Windows 平台所提供的 GUI 为程序员提供了十分完善的编程接口，尤其是它对位图提供直接支持。图形的放大、缩小、拖动应该是图形处理中的一个必备功能了，而对于一副较大的位图，下面的讨论将看到，直接用位图拷贝和缩放拷贝函数都会存在一些问题，因此有必要对此进行专门的设计。

二、问题的分析

对位图处理的通常做法是将位图点阵数据及调色板信息一次读入，需要在图形设备上显示时只需以此设备的设备文本句柄作为参数调用位图函数即可。显示图形的设备通常是视图窗口，位图有缩放时采用滚动窗口更合适。

将位图放大可以用位图缩放拷贝函数 StretchDIB 等，但是在高倍放大时，如果我们将整个位图放大到滚动视图，目标位图会很大，这些函数根本不能正确工作，这主要是因为大尺寸的位图得不到需要的巨大缓冲区；即使放大比率不是很高，但是目标位图尺寸较大时，每次拖动滚动条时的重新绘制消耗的资源也很大，拖动中我们会发现有明显的时延。解决第一个问题的办法就是只拷贝需要显示在滚动视图的用户可见区域的部分位图；而第二个问题的解决需要用到双缓冲：在位图数据和显示缓存中间再开辟一块适当大小的内存缓冲，首先读取位图数据并画到内存位图中，然后再将内存位图的适当部分拷贝到显示缓冲区。如果使这个内存区域大于滚动视图的可见窗口尺寸，那么在拖动的过程中我们不必每次都重新用位图数据绘制位图，只需计算一下应拷贝的内存位图位置再作一次位图拷贝即可。这正是计算机技术中广泛应用的 Cache 技术的具体应用。

三、实现

关于位图的读取、调色板的使用、设备无关、设备相关位图等的讨论已有许多的讨论，这里不赘述。程序中直接采用设备无关位图类 CDibImage[1]，另外视图是从滚动视图类 CScrollView 派生的。设计时将第二块存放放大或缩小位图的内存缓冲区抽象为一个类，与 CDibImage 以及 CBmpView 类相互作用，完成相关功能。

1. CBmpBuffer 类的声明

```
// BmpBuffer.h: interface for the CBmpBuffer class.
typedef int RatioIndex;
class CDibImage;
class CBmpBuffer
{
public:
  CBmpBuffer();
  virtual ~CBmpBuffer();
public:
  static double RATIO(int index); //
  BOOL NeedRestruct(); //是否需要重新构造内存位图
  void RestructBuf(); //重新构造内存位图
  void DumpBuf();     //将内存位图绘到目的设备
  void SetScrollParam(CRect rc1, CRect rc2) {rcTotal = rc1; rcVisible = rc2;}
  void GetScrollParam(CRect & rTotal, CRect & rVisible) {rTotal = rcTotal; rVisible = rcVisible;}
  void SetRatio(RatioIndex rrr){nratio = rrr;}
  void GetRatio(RatioIndex& rr){rr = nratio;}
  void SetTargetDC(CDC * pDC) {m_targetDC = pDC;}
  void GetTargetDC(CDC * pDC) {pDC = m_targetDC;}
  void SetPDib(CDibImage * dib) {m_pDib = dib;}
  void GetPDib(CDibImage * dib) {dib = m_pDib;}
  void SetCacheMode(int i) {m_nCacheMode = i;}
  void GetCacheMode(int& i) {i = m_nCacheMode;}
protected:
  RatioIndex nratio;  //比率索引
  SIZE rawsize;  //原始尺寸
  POINT rawoff;  //原始偏移量
  CRect rcTotal;  //放大后的尺寸
  CRect rcVisible; //放大后可视的部分
private:
```

```
    CDibImage * m_pDib;
    CDC * m_targetDC;
private:
    int m_nCacheMode;
    CDC * memDC;
    CBitmap * bmp;
    BOOL PointInRaw(CPoint& pt);
};
```

2. CBmpBuffer 类的实现

```
//BmpBuffer.cpp: implementation of the CBmpBuffer class
//这里只给出主要方法的实现,对于构造函数等省略
BOOL CBmpBuffer::PointInRaw(CPoint & pt)
{
    CPoint pt1, pt2;
    pt1 = rawoff;
    pt2.x = rawoff.x + rawsize.cx;
    pt2.y = rawoff.y + rawsize.cy;
    if(pt.x >= pt1.x && pt.x <= pt2.x && pt.y>=
pt1.y && pt.y <= pt2.y )
        return TRUE;
    return FALSE;
}
BOOL CBmpBuffer::NeedRestruct()
{
    CPoint pt1 = rcVisible.TopLeft();
    CPoint pt2 = rcVisible.BottomRight();
    double ratio = CBmpBuffer::RATIO(nratio);
    pt1.x /= ratio;
    pt1.y /= ratio;
    pt2.x /= ratio;
    pt2.y /= ratio;
    if(!PointInRaw(pt1))
        return TRUE;
    if(!PointInRaw(pt2))
        return TRUE;
    return FALSE;
}
void CBmpBuffer::RestructBuf()
{;
    CSize DrawScope = rcVisible.Size()
    if(DrawScope.cx>rcTotal.Width()) DrawScope.cx =
rcTotal.Width();
    if(DrawScope.cy>rcTotal.Height()) DrawScope.cy =
rcTotal.Height();
    CPoint pt1 = rcVisible.TopLeft();
    CPoint pt2 = rcVisible.BottomRight();
    double ratio = CBmpBuffer::RATIO(nratio);
    //(1)第二块内存位图大小
    if(m_nCacheMode == 3)
    {
        rawsize.cx = 800;
        rawsize.cy = 600;
    }
    else
    {
        double CacheMagnify = m_nCacheMode + 1;
        rawsize.cx = CacheMagnify * DrawScope.cx;
        rawsize.cy = CacheMagnify * DrawScope.cy;
    }
    //(2)区域适当调整
    CPoint bbpt1, bbpt2;
    bbpt1.x = pt1.x - (rawsize.cx - DrawScope.cx)/2;
    bbpt1.y = pt1.y - (rawsize.cy - DrawScope.cy)/2;
    bbpt2.x = pt1.x + (rawsize.cx + DrawScope.cx)/2;
    bbpt2.y = pt1.y + (rawsize.cy + DrawScope.cy)/2;
    if(bbpt1.x <0) bbpt1.x=0;
    if(bbpt1.y <0) bbpt1.y=0;
if(bbpt2.x >rcTotal.Width() ) bbpt2.x = rcTotal.Width();
if(bbpt2.y >rcTotal.Height() ) bbpt2.y = rcTotal.Height();
    //(3)原始偏移量& 原始尺寸
    rawoff.x = bbpt1.x;
    rawoff.y = bbpt1.y;
    rawoff.x /= ratio;
    rawoff.y /= ratio;
    rawsize.cx = (bbpt2.x - bbpt1.x)/ratio;
    rawsize.cy = (bbpt2.y - bbpt1.y)/ratio;
    //(4)向内存位图绘制
    if(memDC->GetSafeHdc()):: DeleteObject(memDC->
Detach());
    memDC->CreateCompatibleDC(m_targetDC);
    if(HBITMAP(bmp))
        ::DeleteObject(bmp->Detach());
    bmp->CreateCompatibleBitmap(m_targetDC, ratio * raw-
size.cx, ratio * rawsize.cy);
    CBitmap * pOldbmp = memDC->SelectObject(bmp);
HPALETTE hPal = ::CreateHalftonePalette(*m_targetDC);
    ::SelectPalette(*memDC, hPal, TRUE);
    CRect rcDst = CRect(0,0, ratio * rawsize.cx, ratio * raw-
size.cy);
    CRect rcSrcc = CRect(rawoff.x, rawoff.y, rawoff.x +
rawsize.cx, rawoff.y +rawsize.cy);
    CRect rcSrc = rcSrcc;
    rcSrc.bottom = m_pDib->Height() - rcSrcc.top;
    rcSrc.top = m_pDib->Height() - rcSrcc.bottom;
    m_pDib->Draw(*memDC, &rcDst, &rcSrc);
}
void CBmpBuffer::DumpBuf()
{;
    CSize DrawScope = rcVisible.Size()
if(DrawScope.cx>rcTotal.Width()) DrawScope.cx = rcTo-
tal.Width();
if(DrawScope.cy>rcTotal.Height()) DrawScope.cy = rcTo-
tal.Height();
    CPoint pt1 = rcVisible.TopLeft();
    CPoint pt2 = rcVisible.BottomRight();
    double ratio = CBmpBuffer::RATIO(nratio);
    CPoint bbpt1, bbpt2;
    bbpt1.x = rawoff.x * ratio;
    bbpt1.y = rawoff.y * ratio;
```

(下转第 92 页)

尺寸公差的查询和自动标注

陈华光 陈 多

摘 要 本文介绍了在 AutoCAD 2000 中用 VBA 的数据库方法实现尺寸公差的实时查询，用 ActiveX 方法实现尺寸公差的自动标注，并将新增命令组成下拉菜单。

关键词 AutoCAD 2000，数据库，VBA，尺寸公差

AutoCAD 是 Autodesk 公司开发的一个大型计算机辅助绘图软件，功能强大，性能稳定，广泛应用于建筑、机械等设计领域。在绘制机械图时，离不开对尺寸公差的标注， AutoCAD 虽然提供了强大的尺寸标注命令，但直到 AutoCAD 2000 还没有提供方便的尺寸公差的标注命令。要标注公差，必须先查公差手册，找出上、下偏差值，然后用繁锁方法标注。本文用 AutoCAD 自带的 VBA 实现数据库连接，在 VBA 中用编写宏的方法通过菜单方式实现尺寸公差的实时查询和自动标注。

一、用数据库实现尺寸公差的实时查询

1. 用 VBA 实现数据库的连接

要实现尺寸公差的实时查询和自动标注，必须将手册中的公差表转换成数据库，本文数据库选用 Microsoft Access 97。建一个名为“公差库”的数据库，该库由轴 轴的极限差 GB1801 - 79）和孔 孔的极限差 GB1801 - 79）两个表组成。将基本尺寸上限、下限、每项公差的上偏差、下偏差都转换成表的一个列 字段 。

AutoCAD R14. 01 以上版本的软件为其提供了 VBA 语言接口。要实现 VBA 与数据库 Microsoft Access 97 的连接 选主菜单条上[工具]菜单项下的[引用]命令，激活[引用]对话框，从可使用的引用列表中选择 Microsoft DAO 2. 5 / 3. 5 Compatibility Libray 一项，然后单击[确定]按钮，把它引用到工程中来。

2. 尺寸公差的实时查询

尺寸公差查询模块的输入参数为基本尺寸、公差带代号 包括基本偏差代号和公差等级数字），输出参数为上偏差、下偏差。用 DBEngine. OpenDatabase “公差库. mdb” 语句打开数据库。同查公差手册一样，根据输入的基本偏差代号 英文字母 的大小写来选择查找轴表还是孔表。通过基本尺寸与表中基本尺寸上限、下限比较来确定表中的行 记录 ，通过公差带代号来确定该行中列的位置，返回上偏差值、下偏差值。

使用 InputBox 函数输入待查询的基本尺寸和公差带代号，调用查询模块，利用 MsgBox 函数可在 AutoCAD 环境中实时显示出该公差带的上偏差、下偏差值。

二、用 VBA 实现公差查询和自动标注的方法

AutoCAD 2000 直接嵌入了 VBA，在 VBA 中用户可以访问和操作 AutoCAD 内所有的绘图对象和非绘图对象，编写出可直接在 AutoCAD 中运行的宏命令。在 VBA 中可以用 ThisDrawing 表示 AutoCAD 中的 ActiveDocument 当前文档 子对象，用 ThisDrawing. ModelSpace 表示 Model Space 模型空间）子对象，它由当前图形文件中图形实体组成，可通过改变图形实体的属性或方法来修改图形实体或用 Add 方法来生成新实体。用 ThisDrawing. utility 表示 Utility 功能）子对象，它使用户在 AutoCad 命令行与 CAD 交互成为可能，通过它可以处理整型、浮点型、字符型等用户输入，可以接受点 Point 、角 Angle 等特殊量的输入，还可以接受实体的选择等等。

1. 标注尺寸公差的方法

要完成带公差的尺寸标注一般要在“Dimension Style”对话框中填写好上下公差再进行标注，或使用尺寸公差变量来进行，这些方法都比较麻繁，而且它们的尺寸文本只能采用自动测量值。本文介绍一种简单方法，即在提示输入尺寸文本时输入‘基本尺寸 hH0. 7x hS 上偏差 下偏差”即可。式中 H 为大写，x 为小写，S 为大写，0. 7x 表示上下偏差文字高度为基本尺寸文字高度的 0. 7 倍。

2. 给已标注的尺寸标注添加公差标注

对于已完成的尺寸标注可用 ThisDrawing. Utility. GetEntity 方法，提示用户选择一个尺寸标注，对所选对象可用它的 ObjectName 属性来判断是否为尺寸标注，如果是尺寸实体，用 Measurement 属性或 TextOverride 属性找出该尺寸的尺寸文本字符，根据尺寸文本字符，提示用户输入公差带代号，调用查询模块，返回上偏差、下偏差值。为了使标注的尺寸符合机械制图标准，通过转换模块将上、下偏差值转为满足制图标准的字符串，比较上、下偏差值的绝对值是否相等，如果不相等 将尺寸实体的 TextOverride 属性修改为 ‘基本尺寸”+ “hH0. 7x ”+ “hS”+ “上偏差”+ “ ”+ “下偏

差”；如果相等，将尺寸实体的 TextOverride 属性修改为‘基本尺寸”+ ‘%%p”+ 偏差的绝对值，最后对尺寸实体用 Update 方法更新。

3．开发带尺寸公差的尺寸标注

为了使它的提示与 AutoCAD 标注该类尺寸相同，要使用 ThisDrawing. Utility 子对象。用它的 GetPoint 方法在当前图形提示用户选择点，用它的 GetEntity 方法提示用户选择实体，用它的 PolarPoint 方法进行点坐标计算等等。用 ThisDrawing. ModelSpace. Add 的方法增加一个尺寸实体。对所标注的尺寸实体用改变 TextOverride 属性来完成尺寸公差的标注，方法同上节。

4. 用 VBA 菜单组织有关尺寸公差的各种命令

AutoCAD 2000 也将菜单视为对象 直接位于 Application AutoCAD 本 之下 在 VBA 中用 ThisDrawing. Application. MenuGroups 表示，它包括了 CAD 所装的全部菜单。它的子对象 AcadPopupMenu 包括了所有的下列菜单，本文用它的 add 方法增加一个尺寸公差的下列菜单 用 AddMenuItem 方法在新增的下拉菜单项下增加公差查询等实用菜单命令，用 InsertMenuInMenuBar 方法将尺寸公差的下列菜单插入到 AutoCAD 的主菜单中。

三、源程序清单

```
Dim pt1, pt2, lot, bPt As Variant
Dim dimObj, entry As AcadEntity
Dim px1(0 To 2) As Double
Dim tt As Single
Dim x, y, z, dist As Double
'带公差的的直径标注模块
Sub diat()
 ditxt = ThisDrawing.GetVariable("dimtxt")
 ThisDrawing.SetVariable "dimtxt", ditxt
 arsize = ThisDrawing.GetVariable("dimasz")
 On Error Resume Next
RETRY:
ThisDrawing.Utility.GetEntity entry, bPt, "选择圆或圆弧"
If Err <>0 Then
   Err.Clear
   MsgBox "谢谢使用公差标注程序 V1.0" + (Chr(13)) + "湖南工程学院" + (Chr(13)) + "作者:陈华光"
   Exit Sub
Else
   If entry.ObjectName <> "AcDbCircle" And entry.ObjectName <>"AcDbArc" Then
   ThisDrawing.Utility.Prompt "所选取的实体不是一个圆弧,重选或按回车退出"
  GoTo RETRY
  End If
End If
pt1 = entry.Center
If entry.ObjectName = "AcDbCircle" Then
  d1 = entry.Diameter
 Else
  d1 = entry.Radius * 2
End If
arsize = ThisDrawing.GetVariable("dimasz")
pt2 = ThisDrawing.Utility.GetPoint(, vbCrLf & "指定尺寸线的位置点:")
x = pt2(0) - pt1(0): y = pt2(1) - pt1(1)
z = pt2(2) - pt1(2)
dist = Sqr((x ^ 2) + (y ^ 2) + (z ^ 2)) - d1 / 2
ang = Atn(y / x)
ang2 = 3.1415926232 + ang
pPt1 = ThisDrawing.Utility.PolarPoint(pt1, ang, d1 / 2)
pPt2 = ThisDrawing.Utility.PolarPoint(pt1, ang2, d1 / 2)
If x > 0 Then
  Set dimObj = ThisDrawing.ModelSpace.AddDimDiametric(pPt1, pPt2, dist)
Else
  Set dimObj = ThisDrawing.ModelSpace.AddDimDiametric(pPt2, pPt1, dist)
End If
dist = dimObj.Measurement
js1 = InputBox("基本尺寸为" + Mid$(Str$(dist), 1, 7) + "请输入公差带代号:")
If Asc(Mid(js1, 1, 1)) > 58 Or Asc(Mid(js1, 1, 1)) < 48 Then
  js1 = RTrim(LTrim$(Mid$(Str$(dist), 1, 8))) + js1
End If
Call getgz(js1, tt1, tt2, tt)
If tt1 = 0 And tt2 = 0 Then
  Exit Sub
End If
cc = Str$(tt)
Call chgtt(pp1, pp2, tt1, tt2)
If Abs(Val(pp1)) = Abs(Val(pp2)) Then
 dimObj.TextOverride = cc + "%%p" + Mid(pp1, 2)
Else
 dimObj.TextOverride = cc + "\H0.7x;" + "\S" + pp1 + "^" + pp2
End If
dimObj.TextHeight = ditxt
dimObj.ArrowheadSize = arsize
dimObj.TextOutsideAlign = False
dimObj.TextInsideAlign = False
dimObj.VerticalTextPosition = acAbove
dimObj.Update
End Sub
'带公差的对齐型标注模块
Sub alit()
ditxt = ThisDrawing.GetVariable("dimtxt")
ThisDrawing.SetVariable "dimtxt", ditxt
arsize = ThisDrawing.GetVariable("dimasz")
pt1 = ThisDrawing.Utility.GetPoint(, vbCrLf & "指定尺寸的起点:")
pt2 = ThisDrawing.Utility.GetPoint(pt1, vbCrLf & "指定尺寸的终点:")
px1(0) = (pt1(0) + pt2(0)) / 2
px1(1) = (pt1(1) + pt2(1)) / 2
px1(2) = (pt1(2) + pt2(2)) / 2
lot = ThisDrawing.Utility.GetPoint (px1, vbCrLf & "指定尺
```

```
寸线的位置点: ")
 x = pt1(0) - pt2(0) : y = pt1(1) - pt2(1)
 z = pt1(2) - pt2(2)
 dist = Sqr((x ^ 2) + (y ^ 2) + (z ^ 2))
 js1 = Mid$(Str$(dist), 1, 7)
 js1 = InputBox("基本尺寸为" + js1 + "请输入公差带代号:", "输入窗口")
 If Asc(Mid(js1, 1, 1)) > 58 Or Asc(Mid(js1, 1, 1)) < 48 Then
  js1 = RTrim(LTrim$(Mid$(Str$(dist), 1, 8))) + js1
 End If
 Call getgz(js1, tt1, tt2, tt)
 If tt1 = 0 And tt2 = 0 Then
  Exit Sub
 End If
 cc = Str$(tt)
 Call chgtt(pp1, pp2, tt1, tt2)
  Set dimObj = ThisDrawing.ModelSpace.AddDimAligned(pt1, pt2, lot)
 If Abs(Val(pp1)) = Abs(Val(pp2)) Then
  dimObj.TextOverride = cc + "%%p" + Mid(pp1, 2)
 Else
  dimObj.TextOverride = cc + "\H0.7x;" + "\S" + pp1 + "^ " + pp2
 End If
 dimObj.TextHeight = ditxt
 dimObj.ArrowheadSize = arsize
 dimObj.TextOutsideAlign = False
 dimObj.TextInsideAlign = False
 dimObj.VerticalTextPosition = acAbove
 dimObj.Update
End Sub
'上、下偏差值转换模块
Sub chgtt(pp1, pp2, tt1, tt2)
 pp1 = 0.001 * tt1 : pp2 = 0.001 * tt2
 If pp1 > 0 Then
  pp1 = "+0" + Trim(Str$(pp1))
 ElseIf pp1 < 0 Then
  pp1 = "-0" + Mid$(Trim(Str$(pp1)), 2, 4)
  Else: pp1 = " " + Trim(Str$(pp1))
 End If
 If pp2 > 0 Then
  pp2 = "+0" + Trim(Str$(pp2))
  ElseIf pp2 < 0 Then
   pp2 = "-0" + Mid$(Trim(Str$(pp2)), 2, 4)
  Else: pp2 = " " + Trim(Str$(pp2))
 End If
 Do While (Len(pp1) < 6) And (Len(pp1) <> 2)
  pp1 = pp1 + "0"
 Loop
 Do While (Len(pp2) < 6) And (Len(pp2) <> 2)
  pp2 = pp2 + "0"
 Loop
End Sub
'用数据库查上、下偏差的模块
Sub getgz(ByVal tt, tt1, tt2, cc As Single)
 js = tt
 Set md = DBEngine.OpenDatabase("公差库.mdb")
 h = Len(js)
 For m = 1 To 8
  Str1 = Mid(js, m, 1)
  If Asc(Str1) > 96 Then
   Set rs = md.OpenRecordset("轴", dbOpenDynaset)
   cc = Left(js, m - 1)
   ph = Right(js, h - m + 1)
   Exit For
  ElseIf Asc(Str1) > 64 And Asc(Str1) < 97 Then
   Set rs = md.OpenRecordset("孔", dbOpenDynaset)
   cc = Left(js, m - 1)
   ph = Right(js, h - m + 1)
   Exit For
  End If
 Next m
 ph1 = ph & "1": ph2 = ph & "2"
 rs.MoveLast
 k = rs.RecordCount
 For j = 1 To k
  rs.AbsolutePosition = j
  If cc >= rs.Fields(1) And cc <= rs.Fields(2) Then
   Exit For
  End If
  On Error GoTo doerror
 Next j
 n = 3
 Do Until rs.Fields(n).Name = ph1
   n = n + 1
   On Error GoTo doerror
 Loop
 tt1 = rs.Fields(n)
 tt2 = rs.Fields(n + 1)
doerror:
 If Err Then
 MsgBox "公差数据库没有建立相应的公差项"
  Exit Sub
 End If
End Sub
'公差查询模块
Sub query()
'Dim tt1, tt2 As Single
 js1 = InputBox("请输入基本尺寸,公差代号和等级:", "输入窗口")
 Call getgz(js1, tt1, tt2, tt)
 pp1 = Str$(tt1): pp2 = Str$(tt2)
 MsgBox "基本尺寸和公差带代号为" + js1 + " 的上偏差是" + pp1 + " 下偏差是" + pp2, , "公差查询结果"
End Sub
'添加公差菜单模块
Sub addmenu()
 Dim currMenuGroup As AcadMenuGroup
 Dim newMenu As AcadPopupMenu
Dim newMenuItem, MenuSeparator As AcadPopupMenuItem
Dim addgzMacro, queryMacro, diatMacro, alitMacro As String
     Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)
Set newMenu = currMenuGroup.Menus.Add("公差标注")
 addgzMacro = "ESC ESC -vbarun addgz "
```

```
 queryMacro = "ESC ESC -vbarun query "
 diatMacro = "ESC ESC -vbarun diat "
 alitMacro = "ESC ESC -vbarun alit "
 Set newMenuItem =newMenu. AddMenuItem (newMenu.
Count + 1, "公差查询", queryMacro)
 Set MenuSeparator = newMenu. AddSeparator("")
Set newMenuItem = newMenu. AddMenuItem (newMenu.
Count + 1, "添加公差", addgzMacro)
Set newMenuItem = newMenu. AddMenuItem (newMenu.
Count + 2, "直径公差标注", diatMacro)
Set newMenuItem = newMenu. AddMenuItem (newMenu.
Count + 3, "对齐公差", alitMacro)
currMenuGroup. Menus. InsertMenuInMenuBar"公差标注", ""
End Sub
'对已标尺寸实体添加公差标注模块
Sub addgz()
 On Error Resume Next
RETRY:
 ThisDrawing. Utility. GetEntity entry, bPt, "选择一个尺寸标
注实体"
 If Err <>0 Then
    Err. Clear
    MsgBox "谢谢使用公差标注程序 V1. 0" + (Chr(13))
+ "湖南工程学院" + (Chr(13)) + "作者：陈华光"n
    Exit Sub
 Else
   If entry. ObjectName <>"AcDbDiametricDimension" And
entry. ObjectName <>"AcDbRadialDimension" And entry.
ObjectName <>"AcDbAlignedDimension" And entry. Object-
Name <>"AcDbRotatedDimension" The
    ThisDrawing. Utility. Prompt "所选取的实体不是尺寸标
注, 重选或按回车退出"
    GoTo RETRY
   End If
 End If
 dist = entry. TextOverride
 If dist = "" Then
    On Error GoTo doerror
doerror:
 If Err Then
  MsgBox "所选取尺寸标注不支持该属性，可能是低版本所
标注"
  Exit Sub
  End If
  dist = entry. Measurement
 End If
 h = Len(dist)
 For m = 1 To 10
  Str1 = Mid(dist, m, 1)
  If Asc(Str1) >48 And Asc(Str1) < 58 Then
   str2 = Mid$((dist), 1, m - 1)
   js2 = Mid$((dist), m, h)
   Exit For
  End If
 Next m
 js1 = InputBox("基本尺寸为" + js2 + "; 请输入公差带
代号：", "输入窗口")
 If Asc(Mid(js1, 1, 1)) > 58 Or Asc(Mid(js1, 1, 1)) < 48
Then
   js1 = js2 + js1
 End If
 Call getgz(js1, tt1, tt2, tt)
 If tt1 = 0 And tt2 = 0 Then
  Exit Sub
 End If
 cc = Trim(Str$(tt))
 If Asc(Mid(dist, 1, 1)) >58 Or Asc(Mid(dist, 1, 1)) < 48
Then
  cc = str2 + cc
 Else
  If entry. ObjectName = "AcDbDiametricDimension" Then
   cc = "%%C" + cc
  End If
  If entry. ObjectName = "AcDbRadialDimension" Then
   cc = "R" + cc
  End If
 End If
 Call chgtt(pp1, pp2, tt1, tt2)
 If Abs(Val(pp1)) = Abs(Val(pp2)) Then
  entry. TextOverride = cc + "%%p" + Mid(pp1, 2)
 Else
   entry. TextOverride = cc + "\H0. 7x;" + "\S" + pp1 +
"^ " + pp2
 End If
End Sub
```

（收稿日期：2001 年 2 月 27 日）

（上接第 80 页）

```
    if(m_icmp. SendICMP(nMsg, (char * )(LPCSTR)str))
            m_icmp. RecvICMP();
        m_conState. SetWindowText(m_icmp. m_strInfo);
    }
}
void CPingWinDlg: : OnBtnrecv()
{
// TODO: Add your control notification handler code here
    m_icmp. RecvICMP();
    m_conState. SetWindowText(m_icmp. m_strInfo);
}
```

五、结束语

本程序只设计了回送请求功能，感兴趣的朋友可以在此代码的基础上开发出功能更强的 ICMP 协议应用程序来，希望能够起到抛砖引玉的作用。

参考文献

1. 胡道元 . 计算机网络 . 清华大学出版社，1999 年 8 月

2. 蒋东兴、林鄂华、陈棋德、印敏、刘启新 . Windows Sockets 网络程序设计大全世界 . 清华大学出版社 2000 年 4 月

（收稿日期：2001 年 3 月 5 日）

高精度数字地震仪中的嵌入式软件开发

邹建国

一、序言

PC/104 的应用在我国目前正处于方兴未艾的局面，由于它拥有体积小、功耗低、功能全、与台式计算机兼容等显著特点，因此在自动化、仪器仪表、机电一体化应用中广泛使用。我国前几代地震仪器基本上使用硬件直接完成数据采集工作，或者使用单片机进行数据采集，它们共同的缺点是记录数据时间短，而且数据采集精度较低，一般为 16 位，仪器智能化水平较低。为了改进我国数字地震仪器目前的缺点，中国科学院地球物理研究所及国家地震局地球所都开始开发先进的 24 位数字地震仪，其设计的方案都使用了 PC/104 计算机，我负责其中控制软件及数据采集部分的实现。由于仪器的软件开发与一般的软件开发有许多不同，许多的编程都是对硬件控制的编程，要对计算机的底层硬件要十分了解，对于操作系统也要有比较深入的认识，在软件的开发过程中，我们遇到了许多的问题，通过不断的研究开发，最终实现了预定的目标。下面的论述，读者将可以了解 PC/104 在智能仪器中所面临的一些问题及解决方法，以及能源管理与数据采集的实现。

二、PC/104 的特点

由 PC/104 模块制成的嵌入式系统，和台式机不同，一般都是为满足某一项特别任务而设计的。PC/104 嵌入式系统的一般应用在医疗仪器、航空设备、售货机、测试设备、通信设备、交通车辆、数据登录、以及工业控制等方面。设计 PC/ 104 的主要目的是解决专用软件价格太贵问题。由于处理器和 PC 机兼容，可以采用在台式机上运行过的软件，其中重复使用的子程序十分多，而且价格便宜。在台式机运行过的工具，例如集成的开发环境、编译器、以及查错工具等，如果运用到相应的嵌入式系统上去，比起其它的竞争对象，可以节约相当多的项目开发费用。

PC/104 系统上的操作系统，DOS 是主要的选择，许多方面都在应用。现在，几乎所有的台式机上运行的操作系统都可以选用，包括 Windows 98 和 Windows NT。也可以选用 Linux。PC/104 处理器还可以运行嵌入式系统用的操作系统，例如 Windows CE 和 嵌入式）Windows NT。如果系统有实时的要求，则可以选择更为传统的 RTOS，例如，WindRiver System 公司的 VxWorks，或 QNX Software Systems 公司的 QNX。

三、数字地震仪的软件设计

24 位数字地震仪中的 PC/104 配置为：25MHZ 386/SX 处理器、4 兆字节 RAM、两个串行端口 一个可配置为 RS-485 或 RS-232 、一个并行端口、一个 IDE 硬盘接口、电子盘 4M-16M、支持硬盘容量最大 8.4G。由于仪器在野外使用，只能用电瓶或太阳能电池供电，希望仪器功耗尽量低，运行以后不需要显示器、键盘等对数据采集无用的设备，因此对于标准输入输出要使用其它通讯方法。

由于资源有限，操作系统选择了 Windows95 版的 MS-DOS，占用空间几百 K 字节。DOS 操作系统短小精悍，占用内存小，并且使用广泛，开发工具很多，相关书籍资料容易找到，可使项目开发更容易成功。对于标准输入、输出，美国 AMPRO 公司的 PC/104 产品中的 BIOS 软件已经固化了串口控制台的功能，我们只要在 BIOS 中进行相应的设置，然后在控制计算机上运行终端仿真软件即可仿真仪器本地的显示和键盘输入。虽然可以控制仪器，但无法在控制计算机上获得仪器磁盘中的数据，因此还需要一种比串口速度高的通讯手段来获得数据，我们使用了台式机上广泛使用 PCTOOLS 中的一个 DOS 下的网络通讯软件 DRIVEMAP 来完成数据传递，它可以使用串口、并口或者网卡进行通讯 并且可以编程使它自动运行。这样使我们直接使用经过市场考验的软件，集成到我们的系统中提高了开发效率。仪器要求几十天上百天的连续运行，发生故障要能自行监测，并能处理，因此编制的采集软件要有专门针对故障、干扰的处理部分，使数据的丢失减少到最小。

四、能源管理软件模块的开发

野外使用的仪器要求的低功耗，也是软件要实现的重点，虽然 PC/104 的功耗已经比较低，软件还要进一步发掘系统的节能潜力，英特尔公司和微软公司共同制定了一个能源管理标准 APM，它有一个中断调用 INT 15H，只要按规范设定入口参数，调用中断 15H 即可实现。但在我试验的 PC/104 上此中断调用没有起作用，于是我直接对 CPU 和 8259 芯片及实时时钟芯片编程，在试验过程中，我发现将 CPU 休眠，可以节约 PC/104 大部分的功耗，其他的外设，由于没有连接，所以发出休眠指令也没有降低多少功耗。结合对实时钟的定时编程，我实现了 CPU 休眠指定时间自动唤醒，并继续运行安排的程序。硬盘也是一个功耗较大的设备，由于通过电子盘缓存几兆字节数据，硬盘也可以休眠很长时间，通过对硬盘控制器的直接编程，可以实现直接关闭硬盘的电机，从而节省很大功耗。

我开发的能源管理程序，实际上使用了 CPU 的停机控制指令 HLT，并结合编程 8259 中断控制器，将无关中断关闭，

留下 RTC 实时钟中断。这样即使没有 APM 的 BIOS 接口，我的程序也可实现需要的功能，并且掌握了能源管理的本质。相关程序篇幅较长，本篇只摘录一部分。

五、仪器实现自动化、无人值守的软件设计方法

要实现自动化、无人值守，软件必须能够自动判断运行状态，并根据不同情况采取不同的措施。在无人值守时，仪器的电源可能会波动，或者停电又来电，在正常供电后仪器应自动按原设定控制参数运行采集数据，而不能停机。要实现这个功能，可以通过编写 DOS 操作系统的 Autoexec. bat 自动批处理程序来上电自动启动用户编写的程序，用户也可能启动仪器要人为干预，这时又不能自动启动采集程序，我的解决方案是在启动过程中设定几十秒的等待时间，在这段时间如有键盘输入，系统便进入人机对话模式，否则进入自动运行模式。由于各种干扰或硬件故障，系统可能会死机，必须使用硬件‘看门狗”技术，在计算机死机一段时间内复位计算机。这样只会丢失几十秒的数据。

嵌入式软件设计包括许多方面，我上面只是涉及其中的一小部分，希望能给读者一定的启发。

六、程序附录

下面列出通过 INT 15H 调用的 APM 能源管理功能的中断调用部分接口规范，以便大家可以自己编程实现其接口。

1. 设置各外设的四种节能方式

入口: AX = 5307h BX = 设备号
00xxh = 系统 00h = BIOS
01h = 由系统 BIOS 管理的所有设备
01xxh = 显示器
02xxh = 第二存储器
03xxh = 并口 04xxh = 串口
这里 xx 代表物理设备号, 当为 ff 时代表所有设备
CX = 系统状态号
0 已准备好 1 等待
2 挂起 3 关闭
返回值: CF = 0 成功 CF = 1 错误
AH = 01H 电源管理功能已关闭
09H 不能识别的设备号
0AH CX 中的参数值越界
60H 无法进入请求的状态

2. 设置 BIOS 能源管理功能的使能 / 使不能

入口: AX = 5308h BX = FFFFh
CX = 0 使不能 1 使能
返回值: CF = 0 成功 CF = 1 错误
AH = 01H 电源管理功能已关闭
09H 不能识别的设备号
0AH CX 中的参数值越界

3. 恢复 BIOS 上电时的默认值

入口: AX = 5309h BX = FFFFh
返回值: CF = 0 成功 CF = 1 错误
AH = 09H 不能识别的设备号

4. 获得电源状态

入口: AX = 530Ah BX = 0001h
返回值: CF = 0 成功
BH = 交流电源状态
0 关闭 1 在线
– 1 未知状态
BL = 电池状态:
0 高 1 低
2 临界 3 充电
– 1 未知

5. 获得临近的能源管理事件

入口: AX = 530Bh
返回值: CF = 0 成功
BX = 1 请求系统等待
2 请求系统挂起
3 标准恢复系统消息
4 临界恢复系统消息
5 电池电压低
CF = 1 错误
AH = 03h 未建立接口连接
80h 没有临近事件

```
//= = = = = = = = = = = = = = = = = = = = = =
// 测试能源管理的 C 语言程序:
// 使用 HLT 指令测试 CPU 暂停的能源管理程序
// 作者: 邹建国
//    此程序在 PIII – 533EB 兼容计算机上上使用
//    Borland C/C + + 3. 1 编译, 在 DOS 环境下调试通过
// 编译方法: BCC HLT. C ZOOLIB. LIB
//= = = = = = = = = = = = = = = = = = = = = =
#include <stdlib. h>
#include <stdio. h>
#include <dos. h>
#include <conio. h>
#include "zoulib. h"
void main()
{
  unsigned char temp_21, temp_a1;
  disable();
       temp_21 = inp(0x21);
       temp_a1 = inp(0xa1);
       outp(0xa1, 0xFe);
       outp(0x21, 0xfb); //设置唤醒中断为实时钟 IRQ8
  enable();
//  设置实时钟 10 秒后产生报警中断 IRQ8, 你也可自己编写使用其他中断
  set_wakenup(10);
  printf("CPU Halted! \n");
  asm  hlt
//成功时系统应停在这里, 并且电源功耗马上就会下降
  printf("CPU Runing! \n");
  disable();
       outp(0x21, temp_21);
       outp(0xa1, temp_a1);
  enable();
}
```

APM 中断调用实例参见 APM. C 程序

（收稿日期：2001 年 6 月 16 日）

用‘明码’记忆逻辑加密存储卡 SLE4442 的‘密码’

杨瑞生

SLE4442 是由德国西门子（SIEMENS）公司设计的逻辑加密存储卡（Smart Card with Security Logic）。该卡采用密码控制逻辑来控制对 2K 位 EEPROM 存储器的访问和改写，不能象普通存储卡那样可以被任意改写或复制，不仅容量大，而且安全保密，加之价格低廉，可多次重复使用，是目前国内应用较多的一种 IC 卡芯片。

SLE4442 IC 卡芯片内部主要有 3 个存储器，即 256 × 8 位 EEPROM 型存储器、32 × 1 位 PROM 型保护存储器和 4 × 8 位 EEPROM 型加密存储器。在加密存储器中，存放有长度为 3 个字节的可编程加密代码（PSC）和长度为 3 个比特位（Bit）的密码输入错误计数值（EC）。可编程加密代码在比较成功前是不可读的，只能进行比较操作，其‘写入、擦除’操作也受自身‘比较’操作结果的控制，只有在‘比较’成功后，才能对其进行读出、写入和擦除操作，也只有‘比较’成功后，才能对主存储器和保护存储器进行写入和擦除操作。密码输入错误计数值规定了允许输入密码出错的次数，当连续 3 次输入错误密码后，密码输入错误计数值减为零，芯片的存储单元将会全部被锁死。所以，记忆可编程加密代码，无论对于 IC 卡的安全，还是方便日后对 IC 卡的维护，都具有重要意义。

IC 卡芯片在出厂时，都将加密存储器中编入一个专用的可编程加密代码。开发使用 IC 卡时，必须首先合法得到这个代码，然后再将代码根据开发工程项目的用途重新进行编程。如果一批 IC 卡共用一个加密代码，方便发行的同时，也容易埋下安全隐患。比较可靠的办法应该是‘一卡一码’。然而，加密代码是由 6 位 16 进制数所组成，即有 3 × 8 = 24 个比特位，可能的加密代码有 2 的 24 次方个之多，要记忆这么多的加密代码实属不易。因此，笔者在工程开发实践中摸索出了一种用‘明码’记忆‘密码’的有效办法。就是为每张 IC 卡规定一个‘序列号’，并将此序列号就存放在 SLE4442 的主存储区中，作为‘明码’，需要时，通过自编的一段程序，由此‘明码’导出‘密码’。

序列号可由 16 个字符组成。如图 1 所示。分成 5 个部分，各部分含义可以根据编程者的意图确定，这里只是一个例子。‘引导符’作为序列号的标志。‘操作说明’指出对‘变体 PSC’采取什么类型的操作才能生成正确的可编程加密代码，“Ax”对应“8”，“Bx”对应“9”，“C2”对应“6”，操作类型可规定为 4 种或更多：

图 1

Ax x 无意义 = = = = 字节的高低四位对调

Bx x 无意义 = = = = 字节内容取反

Cm m 为次数 = = = = 将字节左环位移 m 次

Dm m 为次数 = = = = 将字节右环位移 m 次

‘变体 PSC’是伪可编程加密代码，只有进行相应变化后才能生成正确的 PSC。‘分隔符’隔开‘流水号’。‘流水号’是 IC 卡的流水编号。

序列号可以有灵活的组合，例如：“No. D5C3B0YTH - 003”、“No. C5A7D2F68 - 004”等等。序列号虽然是‘明码’，但是，直接观之并不能得到真正的密码。

图 2

由序列号合成 PSC 的程序，可用 Microsoft Visual Basic 5. 0 编程。该程序的界面如图 2 所示，源程序如下：

```
' FILENAME: T51_4442. EXE
' 根据序列号合成可编程加密代码(PSC)
' Yangruisheng
' 2001. 2. 6
Option Explicit
Dim R031 As Byte
Dim R032 As Byte
Dim R033 As Byte
Dim R034 As Byte
Dim R035 As Byte
Dim R036 As Byte
```

```
Dim R037 As Byte
Dim R038 As Byte
Dim R039 As Byte
Dim R011 As Byte
Dim R012 As Byte
Dim R013 As Byte
Dim BITW(0 To 7) As Integer
' 合成可编程加密代码(PSC)函数 HPSC. 入口参数: R040 =变
PSC
R041 =操作类型
R042 =操作数据
Private Function HPSC(R040 As Byte, R041 As Byte, R042
As Byte) As Byte
   Dim R000 As Byte
   Dim iibb As Byte
Select Case R041                          ' 区分操作类型
Case & H41                    ' "A" = =R040 高低四位对调
          R000 = R040
          R000 = R000 And & HF
          R000 = R000 * & H2
          R000 = R000 * & H2
          R000 = R000 * & H2
          R000 = R000 * & H2
          R000 = R000 And & HF0
          R040 = R040 And & HF0
          R040 = R040 / & H2
          R040 = R040 / & H2
          R040 = R040 / & H2
          R040 = R040 / & H2
          R040 = R000 Or R040
          HPSC = R040
     Case & H42     ' "B" = =R040 内容取反
          HPSC =Not(R040)
     Case & H43     ' "C" = =R040 左环位移
          CBITW (R040)
     R042 = R042 And & HF' 屏蔽高四位取出位移次数
          For iibb = 1 To R042
             ZHWY
          Next iibb
          HPSC = HCZJ
       Case & H44' "D" = =R040 右环位移
          CBITW (R040)
     R042 = R042 And & HF' 屏蔽高四位取出位移次数
          For iibb = 1 To R042
             YHWY
          Next iibb
          HPSC = HCZJ
       Case Else
          HPSC = R040' 操作类型不对
   End Select
End Function
' 将字节拆成 Bit 位
Private Function CBITW(ZJSJ As Byte) 1 1 1 1
      Dim II00 As Integer
      For II00 = 0 To 7
         BITW(II00) = 0
      Next II00
      If (ZJSJ And & H80& ) <>0 Then BITW(7) =
      If (ZJSJ And & H40& ) <>0 Then BITW(6) =
      If (ZJSJ And & H20& ) <>0 Then BITW(5) =
      If (ZJSJ And & H10& ) <>0 Then BITW(4) =
      If (ZJSJ And & H8& ) <>0 Then BITW(3) = 1
      If (ZJSJ And & H4& ) <>0 Then BITW(2) = 1
      If (ZJSJ And & H2& ) <>0 Then BITW(1) = 1
      If (ZJSJ And & H1& ) <>0 Then BITW(0) = 1
End Function
' 将 Bit 位合成字节
Private Function HCZJ() As Byte
      HCZJ = 0
      If BITW(0) = 1 Then HCZJ = HCZJ + 1
      If BITW(1) = 1 Then HCZJ = HCZJ + 2
      If BITW(2) = 1 Then HCZJ = HCZJ + 4
      If BITW(3) = 1 Then HCZJ = HCZJ + 8
      If BITW(4) = 1 Then HCZJ = HCZJ + 16
      If BITW(5) = 1 Then HCZJ = HCZJ + 32
      If BITW(6) = 1 Then HCZJ = HCZJ + 64
      If BITW(7) = 1 Then HCZJ = HCZJ + 128
End Function
' 左环位移一次
Private Function ZHWY()
      Dim BBBB As Integer
      Dim II00 As Integer
      BBBB = BITW(7)
      For II00 = 7 To 1 Step -1
         BITW(II00) = BITW(II00 - 1)
      Next II00
      BITW(0) = BBBB
End Function
' 右环位移一次
Private Function YHWY()
      Dim BBBB As Integer
      Dim II00 As Integer
      BBBB = BITW(0)
      For II00 = 0 To 6
         BITW(II00) = BITW(II00 + 1)
      Next II00
      BITW(7) = BBBB
 End Function
Private Sub Command1_Click()
      R031 = Asc(Mid(Text1. Text, 1, 1))     '操作 1
      R032 = Asc(Mid(Text1. Text, 2, 1))     '操作 1
      R033 = Asc(Mid(Text2. Text, 1, 1))     '操作 2
      R034 = Asc(Mid(Text2. Text, 2, 1))     '操作 2
      R035 = Asc(Mid(Text3. Text, 1, 1))     '操作 3
      R036 = Asc(Mid(Text3. Text, 2, 1))     '操作 3
      R037 = Asc(Mid(Text4. Text, 1, 1))     '变 PSC1
      R038 = Asc(Mid(Text5. Text, 1, 1))     '变 PSC2
      R039 = Asc(Mid(Text6. Text, 1, 1))     '变 PSC3
      R011 = HPSC(R037, R031, R032)
      R012 = HPSC(R038, R033, R034)
      R013 = HPSC(R039, R035, R036)
   Text7. Text = Hex(R011) & Hex(R012) & Hex(R013)
End Sub
```

序列号及其合成 PSC 的算法，还可以灵活多样，形成各种组合，一个 PSC 可以对应一个序列号，一个 PSC 也可以对应若干个序列号。

收稿日期：2001 年 2 月 12 日）

提高 Linux 的安全等级的简单办法

崔同杰

Linux 缺省的安全等级是 0，如果将其升到 1，就可以一定程度上提高系统的安全性。安全等级为 1 的时候，它会禁止修改 ex2fs 系统中文件的 immutable 和 append - only 位，同时禁止装入 / 移除 module。所以我们可以先用 chattr + i < file>将大部分的可执行文件、动态连接库、一些重要的系统文件 inetd. conf、securetty、hosts. allow、hosts. deny、rc. d 下的启动 script. . . 加上 immutable 位，这样 ‘黑客 ” 就很难在你的机器上放置木马和留后门了 即便他已经得到了 root 权限，当然通过直接硬盘读写仍然可以修改，但比较麻烦而且危险 。 ‘黑客 ” 们一旦进入系统获得 root，首先会清除系统的记录文件。你可以给一些系统记录文件 wtmp、messages、syslog. . . 增加 append - only 位，使 ‘黑客 ” 不能轻易地修改它们。要抓他们就容易多了。 修改安全等级比较直接的办法是直接修改内核源码，将 linux / kernel / sched. c 中的 securelevel 设成 1 即可。不过如果要改变安全等级的话需要重新编译内核，我太懒，不想那么麻烦。为什么不用 module 呢？我写了个很简单的 lkm 和一个 client 程序来完成安全等级的切换。

方法 insmod lkm clt - h

注意 普通用户也可以执行 clt 来切换安全等级，所以最好是在 clt 和 lkm 中加段密码检查，如果密码不对就不允许执行。

这两个程序在 Redhat 5. 2 2. 0. 36 下编译运行通过。对于 2. 2. x 的内核，securelevel 变成了 securebits，简单地将它改到 1，会连 setuid 都被禁止了，这样普通用户就不能登陆了。如果谁对 2. 2. x 比较熟悉，请不吝赐教，共同提高。

在测试这些程序以前，请备份重要数据。本人不为运行此程序带来的任何损失负责。）

一旦 securelevel = 1，kernel 将不允许装入 modlue，所以你的 kerneld 可能不能正常工作，而且禁止你访问 / dev / kmem，所以有些用到 svgalib 的程序也不能正常工作，像 zgv 什么的。不过这本来就是安全隐患，所以不工作就不工作好了

关于 chattr、lsaddr 请参见 man chattr 和 man lsattr

```
/ * * * * * * * * lkm. c * * * * * * /
/ * Simple lkm to secure Linux. This module can be used to change the securelevel of Linux. Running the client will switch the securelevel.
 * gcc -O3 -Wall -c lkm. c
 * insmod lkm
 * It is tested in Redhat 5. 2 (2. 0. 36).
 * (It should be modified if you want to run it in 2. 2. x kernel).
 * It is really very simple, but we just for educational purposes. : -)
 * warning3@ hotmail. com
 * /
#define MODULE
#define __KERNEL__
#include <linux/config. h>
#include <linux/module. h>
#include <linux/version. h>
#include <linux/errno. h>
#include <linux/types. h>
#include <linux/fs. h>
#include <linux/string. h>
#include <linux/mm. h>
#include <linux/proc_fs. h>
#include <asm/segment. h>
#include <asm/unistd. h>
#include <linux/dirent. h>
#include <asm/unistd. h>
#include <linux/sockios. h>
#include <linux/if. h>
#define __NR_secureswitch 250
extern void * sys_call_table[];
int sys_secureswitch(int secure)
{
if(secure = =0) securelevel =0;
if(secure = =1) securelevel =1;
return securelevel;
}
int init_module(void)
{
  sys_call_table[__NR_secureswitch] = (void *) sys_secureswitch;
return 0;
}
void cleanup_module(void)
{
sys_call_table[__NR_secureswitch] = NULL;
return;
}
/ * * * * * clt. c * * * * * * /
/ * This client can switch the secure level of Linux.
 * gcc -O3 -Wall -o clt clt. c
 * Usage: clt -h/ -l
```

```
* -h switch to the high secure level.
* -l switch to the low secure level.
* Most of codes are ripped from smiler@tasam.com, thanks smiler.:)
* warning3@hotmail.com
*/
#include <asm/unistd.h>
#include <stdio.h>
#include <errno.h>
#define __NR_secureswitch 250
static inline _syscall1(int, secureswitch, int, command);
int main(int argc, char **argv)
{
int ret, level = 0;
if (argc < 2)
{
fprintf(stderr, "Usage: %s [-h/-l]\n", argv[0]);
exit(-1);
}
if (argv[1][1] == 'h') level++;
else if (argv[1][1] != 'l')
{
fprintf(stderr, "Usage: %s [-h/-l]\n", argv[0]);
exit(-1);
}
ret = secureswitch(level);
if (ret < 0)
printf("Hmmm... It seemed that our lkm hasn't been loaded.;-)\n");
else {
if (ret == 0) {
puts("Now the secure level is changed to 0!\n");
}
else {
puts("Now the secure level is chagned to 1!\n");
}
}
return(1);
}
```

收稿日期 2001 年 2 月 19 日

上接第 82 页）

```
    bbpt2.x = bbpt1.x + ratio * rawsize.cx;
    bbpt2.y = bbpt1.y + ratio * rawsize.cy;
m_targetDC->BitBlt(rawoff.x * ratio, rawoff.y * ratio, ratio * rawsize.cx, ratio * rawsize.cy, memDC, 0, 0, SRCCOPY);
}
```

3. 使用方法：

在视图类中，每次放大和缩小命令发出后，只需要设置 CBmpBuffer 对象的系列参数，通过 NeedRestruct 方法判断以决定是否重新构造这个内存位图，然后将类存位图拷贝到目的设备即可。注意下面第一个判断语句用于展示不用双缓冲时的图形效果。

```
void CBmpView::OnDraw(CDC * pDC)
{
    if(m_bNoUsingCache)
    {
        CSize size = GetTotalSize();
        CRect rect(0, 0, size.cx, size.cy);
        m_pDib->Draw(*pDC, &rect);    }
    else
    {
        m_BmpBuf.SetPDib(m_pDib);
        m_BmpBuf.SetTargetDC(pDC);
        this->SetScrollParam();
        if(m_bForce || m_BmpBuf.NeedRestruct())
        {
            m_BmpBuf.RestructBuf();
            m_bForce = FALSE;
        }
        m_BmpBuf.DumpBuf();
    }
}
```

4. 性能分析：

通过运行程序，我们可以看到：（1）如果不采用双缓冲而直接将位图放大到屏幕，则放大到一定程度（例如：400×300 的位图在笔者的机器上放大到 7 倍以后）就没法工作了。采用上文的技术则可以解决高倍放大的问题（理论上可以任意高放大比率）。（2）当我们的内存位图等于视图可见部分大小时，每次拖动位图都要重新构造整个内存位图，时延非常明显，当然如果我们仅仅用页滚动，其效果也不错，和 PhotoED 近似；当我们的内存位图尺寸 2 倍于视图可见部分大小时，一次拖动最多重新构造一次位图，在此重新构造时有明显的停顿，其效果与 ACDSee3.2 相当；当我们的内存位图尺寸 3 倍于视图可见部分大小时，任何一次拖动都不必重新构造一次位图，而只需要在拖动放开时重新构造一次即可，这时的拖动效果可以与 PhotoShop 媲美。

四、小结

在位图显示中，通过使用一个介于位图数据和显示设备之间的内存位图，作为向显示设备显示位图的后备缓冲，很好地解决了位图的高倍放大与平滑拖动的问题，对于 Windows 平台上其他图形的处理也具有一定参考价值。

参考文献

1. http://www.csdn.net/dev/visual C++/source code/bitmap/dibedit.zip

2. D J. Kruglinski. Visual C++技术内幕 第四版

收稿日期：2001 年 3 月 27 日）

博士信箱 QUESTION&ANSWER

微机系统内存的合理使用与常见问题解答

开机后内存检测时间长怎么办

通过设置 Quick Power On Self Test 和 Esc 键来解决。操作如下：

(1) 开机时，按 Del 键进入 Setup；

(2) 选择 BIOS Features Setup 回车；

(3) 使用 PgDn 键把 "Quick Power On Self Test" 设置为 "Enabled"；

(4) 使用 F10 键退出 回答 "Y"）；开机自检内存时，按 Esc 键跳过自检。

随着内存价格急剧下降，电脑基本配置内存容量的增加，开机内存自检的时间越来越长，即使使用快速检测，把三遍检测改成一遍检测，时间也不短，因此需要使用 Esc 键直接跳过检测。以后 Setup 一定会有完全不检测内存的开关。

怎样挑选内存条芯片

目前，PC - 100 的 SDRAM 已成为装机选购内存的标准，但在市面上很多内存并不真正符合 PC - 100 规范。在表面上看，目前仍有不少内存不带 SPD，或者是带 SPD 的所谓 "PC - 100" 使用的芯片面性符合规范。如常见的 LGS 的 10K、现代的 TC - 10 等，这些芯片并不符合 PC - 100 内存规范的要求，其中主要是由于其 tAC 大于 6ns。但实际上许多非 PC - 100 内存在 100MHZ 甚至更高的外频下工作得很正常，可以理解为内存被超频了。在价格相差并不很大的情况下，应该尽量选购性能较好的内存。因为非 PC - 100 的内存本身虽然可能可以在 100MHZ 下工作，但超过额定的指标往往不能保证完全没问题，所谓的正常可能只是在有限的范围内不出错而已。因而在选购内存芯片时要注意以下几点：

(1) 认准类型

现在许多人认为 168 线内存就是 SDRAM，这是不对的。因为同样有 168 线的 EDO DRAM。目前市场上 168 线的 EDO DRAM 还能经常看到。168 线的 EDO DRAM 最明显的一点是 EDO DRAM 的针脚比较少，标准的 16Mbit SDRAM 的针脚为 44 根，而 16Mbit EDO DRAM 的为 28 根，疏密程度明显不同；64Mbit SDRAM 的针脚为 54 根，而 EDO DRAM 的针脚为 32 或 44 根，针脚不是连续排列，因而中间会留下几个针脚的空隙。但最保险的方法是看芯片的编号，从内存芯片的编号上可以辨认出其类型、性能等。

(2) 认清芯片的品牌

不同的品牌的质量不同，一些品牌的内存芯片的检测比较严格，在质量和性能上留的富余度也比较大，而一些厂商可能由于品质管理或自身的技术条件限制了其产品的品质。这种品牌区别一般不会影响正常的使用，但在超频的时候就有比较大的影响。

(3) 追求芯片的品质

内存芯片上的标号只能是一个参考，芯片本身的实质并不会完全在标号上面体现出来，有可能会遇到名不符实的芯片。有两种可能，一种情况是芯片本身是次品，但通过不明途径流入市场；另一种情况是 Remark 的芯片，将低质芯片的标识打磨掉之后，重新打上标识，以冒充较优质的芯片。

(4) 小心次品

这种情况使用的一般是原厂的芯片，芯片的外观一般比较完美。这类内存的价格一般都比较低，在遇到芯片不错，价格低廉的内存时一定要多长几个心眼，看看这根内存的印刷电路板是否做得比较简陋、粗糙。如果感觉印刷电路板比较差，如果是好的芯片配这样不起眼的印刷电路板就非常令人担心。要仔细查看其上标的生产日期是否一致。如果有几个不同的日期就说明同一条内存上的芯片有几个不同的批次，这时我们就需要小心。

此外有瑕疵的产品还有一种表现形式是芯片的数目和实际容量不符，常见的就是所谓的 '补位片' 或 '补位条'，如看芯片上的标识，知道每颗芯片的容量是 64Mbit，除 8 得知每颗芯片是 8MB，则 64MB 一条的内存上应该有 8 颗芯片，若带校验的内存就有 9 颗，否则芯片的数目与实际容量不符。

(5) 必须警惕 Remark

这种情况由于要打磨或腐蚀芯片的表面，一般都会在芯片的外观上表现出来。正品的芯片表面一般都很有质感，要么有光泽或荧光感要么就是哑光的。如果觉得芯片的表面色泽不纯甚至比较粗糙、发毛，那么这颗芯片的表面一定受到了磨损。刮擦留下的痕迹通常比较粗，打磨的芯片上的痕迹主要集中在标识附近，细看后可以发现芯片的这些区域与其他部分的质感不大一致，如颜色偏浅、泛白等。另外 Remark 的芯片还可能出现芯片表面有明显的磨损面，通常芯片的角上会有凹陷的小圆圈，侧光观察小圆圈的深度是否均匀。打磨过的芯片上的小圆圈可能还会出现某部分边缘缺失的情况。

怎样从内存颗料编号识别内存

可以通过内存颗粒上的编号识别内存的生产厂家、种类、使用的电压、芯片密度、电气接口、容量、速度、芯片封装方式、功耗、版本等重要参数。

三星内存颗粒识别方法如下：

编号：KM X XX X XX X X X X X - X XX
1 2 3 4 5 6 7 8 9 10 11 12

其中

1. KM 表示三星内存

2. RAM 的种类，4 = DRAM

3. 内存芯片组成：4 = x4；8 = x8；16 = x16 分别代表 4 位、8 位、16 位）

4. S = SDRAM

5. 内存芯片密度：1 = 1M；2 = 2M；4 = 4M . . .

6. 刷新：0 = 4K；1 = 2K；2 = 8K

7. 内存排数：2 = 2 排；3 = 4 排

8. 内存电气接口：0 = LVTTL；1 = SSTL

9. 内存版本：空白 = 第 1 版；A = 第二版 . . . . . .

10. 封装类型：T = TSOP Ⅱ

11. 电源供应：C = 自动调节；F = 低电压自动调节低功耗

12. 最少存取周期：7 = 7ns 143MHz ；8 = 8ns 125MHz 10 = 10ns 100MHz ；H = 100MHz @ CAS 值为 2；L = 100MHz @ CAS 值为 3。

例如：KM416S16230A - G10 指的是三星 16M × 16 = 256Mbit SDRAM 内存芯片，刷新为 8KB，内存 Banks 为 3，内存接口 LVTTL，第 2 代内存，自动刷新，速度是 10ns 100MHz 。其他品牌的内存也依此类推……

现代 Hyundai SDRAM 内存颗粒分老版本和新版本，其编号的识别方法如下：

现代 SDRAM 内存老版本：

编号：XY XX X XXX XX X X X X XX XX
1 2 3 4 5 6 7 8 9 10 11

其中：

1：HY 代表现代产品。

2：代表产品类型。57 代表 DRAM；5D 代表 DDR SDRAM。

3：代表电压。V 代表 3. 3V；U 代表 2. 5V。

4：代表密度和刷新，4 代表 4MB （K 刷新）；16M 4K 刷新）；64 代表 64M （8K 刷新）；65 代表 64M （4K 刷新）；128 代表 128M （8K 刷新）；129 代表 128M （4K 刷新）；257 代表 256MB （8K 刷新）。

5：代表数据带宽。40 代表 4 位；80 代表 8 位；16 代表 16 位；32 代表 32 位。

6：代表芯片组成，1 代表 2BANK；2 代表 4BANK。

7：代表电器接口，0 代表 LVTTL；1 代表 SSTL；2 代表 SSTL2。

8：代表芯片修正版本；空白代表第 1 版，A 代表第 2 版，B 代表第 3 版，C 代表第 4 版，D 代表第 5 版。

9：代表功耗，空白代表普通；L 代表低功耗。

10：代表封装方式；JC 代表 400mil SOJ；TC 代表 400mil TSOP Ⅱ；TD 代表 13mm TSOP - Ⅱ TG 代表 16 mm TSOP - Ⅱ；TQ 代表 100Pin TQF - PI。

11：代表内存的速度，5 代表 5ns 200MHz ；55 代表 55ns 183MHz ；6 代表 6ns 166MHz ；7 代表 7ns 143MHz ；75 代表 75ns 133MHz ；8 代表 8ns 125MHz ；10P 代表 10ns 100MHz @ CL = 2 或 3 ；10S 代表 10ns 100MHz@ CL = 3 ；10 代表 10ns （00 MHz ；12 代表 12ns （3 MHz ；15 代表 15ns （6 MHz 。

现代 SDRAM 内存新版本：

编号： HY XX X XX XX X X X XX X XX
1 2 3 4 5 6 7 8 9 10 11

其中：

1：HY 代表现代产品。

2：代表产品类型，57 代表 SDRAM。

3：代表电压，V 代表 3. 3V。

4：代表密度和刷新，64 代表 64M （4K 刷新）；65 代表 64M （8K 刷新）；28 代表 128M （4K 刷新）；56 代表 156M （8K 刷新）。

5：代表数据代带，4 代表 4 位；8 代表 8 位；16 代表 16 位；32 代表 32 位。

6：代表芯片组成，1 代表 2BANK；2 代表 4BANK。

7：代表意义不详，一般为 0。

8：代表电气接口，0 代表 LVTTL；1 代表 SSTL_3。

9：代表芯片修正版本；空白或 H 代表第 1 版；A 或 HA 代表第 2 版；B 或 HB 代表第 3 版；C 或 HC 代表第 4 版。

10：代表封装方式；T 代表 TSOP；Q 代表 TQFP；I 代表 BLP；L 代表 CSP （F - CSP）。

11：代表内存的速度，5 代表 5ns （00MHz ；55 代表 5. 5ns 183MHz 6 代表 6ns （66MHz 7 代表 7ns （43MHz K 代表 7. 5 ns PC133@ CL = 2 H 代表 7. 5 ns PC133@ CL = 3 8 代表 8ns （25MHz ；P 代表 10ns PC100@ CL = 2 S 代表 10ns PC100@ CL = 3 10 代表 10ns （00MHz 。

诸如 LGs SDRAM、胜利 （KingMax）、三菱 （MITSUBISHI） SDRAM、东芝 （TOSHIBA） SDRAM、富士通 （FUJISU SDRAM、西门子 （Siemens） SDRAM、日立 （Hitachi） SDRAM、美光 （Micro） SDRAM、金帮 GeⅡSDRAM 等内存，颗粒编号识别方法与类似，具体每个编号的含义可查该产品的使用说明的有关资料。

Windows 能够管理的最大内存是多少

Windows 9x 必须运行在 Intel 80486 以上的 CPU 之上，Windows 3. x 可以运行在 80386 之上。80386 至 Pentium Ⅲ处理器都是 32 位处理器，所谓 32 位的含义之一是指 CPU 在以字节为单位的前提下 字节的英文单词为 Byte，缩写时用大写字母 B 来表示，一个字节由 8 个比特组成，比特的英文单词为 bit，缩写时用小写字母 b 来表示）它能访问和管理的内存地址为 32 位的二进制数，也就是说，这些 CPU 能访问和管理的内存容量是 232 个字节，即 4GM。