Delphi 串口及语音传真高级编程

温锦山　唐柱鹏　编著

北京航空航天大学出版社
http://www.buaapress.com.cn

内 容 简 介

本书是一本介绍 Delphi 串口及语音传真编程的专著。从介绍 Modem 直接 IO、TAPI 编程出发，到介绍全球著名的组件 Async Professional(APRO)的属性、方法、事件以及丰富的实例，特别是对语音、传真编程方法进行了深入的剖析，详尽地介绍了 APRO 的功能。

本书的精华是结合我国电信标准和 Modem 硬件的实际情况，部分修改了 APRO 组件的代码，用语音识别的方法解决了在"语音"模式中不能判断对方是否挂机、无人接听、忙音、静音等("数据"模式本来就可以判断)、修正了"传真"模式中部分内置 Modem 上不能收发传真的错误，成功地解决了国内语音、传真编程中的核心问题。

本书有丰富的实例，是作者多年来对电信编程的心得，是为解决实际问题而编写的，汇集了作者的经验和技巧。本书是电信应用编程中的得力助手，可为对 Delphi 语音、传真编程感兴趣者提供帮助，亦可供广大编程人员及各大专院校师生参考。

图书在版编目(CIP)数据

Delphi 串口及语音传真高级编程/温锦山等编著.
北京：北京航空航天大学出版社，2002.4
ISBN 7-81077-160-4

Ⅰ.D… Ⅱ.温… Ⅲ.Delphi 语言 程序设计
Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2002)第 012144 号

Delphi 串口及语音传真高级编程

温锦山 唐柱鹏 编著
责任编辑 张光斌 范曼华
北京航空航天大学出版社出版发行
北京市海淀区学院路 37 号(100083) 发行部电话：82317024 发行部传真：82328026
http://www.buaapress.com.cn
E-mail:pressell@publica.bj.cninfo.net
河北省涿州市新华印刷厂印刷 各地书店经销

*

开本：787×1 092 1/16 印张：30.25 字数：774 千字
2002 年 4 月第 1 版 2002 年 4 月第 1 次印刷 印数：5 000 册
ISBN 7-81077-160-4/TP·087 定价：50.00 元

前　言

编写一本串口及语音传真编程方面书籍的想法，是在编写语音自动应答系统时产生的。该系统的作用是通过语音 Modem 来把谈话内容录制下来，并能提示对方。虽然目前已有现成的软件，但用于商业用途的收费不菲，而且也有种种先天不足（如：不能判断对方是否挂机、无人接听、忙音、静音等），又不能集成在应用软件中，于是提出编写一个实用的自动应答系统。开始编写时头脑还是一片空白，只知道用语音 AT 命令可以实现这些功能，但是这些命令很繁杂，而且不同型号的调制调解器也许需要用不同的 AT 命令，很难实现兼用性；虽然 TAPI 可以实现，且兼用性不错，但找遍了书店和国内的网站都未找到详细完整的资料；另外在介绍语音、传真格式的转换等方面需要编写大量的代码；一些现成的免费控件如 MSCOMM、SPCOMM，虽然也可以实现一些简单的功能，但这只是 AT 命令的扩展，功能有限，很难满足语音、传真高级编程的需求，因此考虑到利用其他第三方组件。

组件 APRO 就是首选，它封装了 TAPI 函数，提供 APF 打印驱动环境，转换文件为传真文件格式、拨号控件、终端控件、FTP 控件等，可以实现呼叫中心、Fax 系统，实现语音 E-mail 更简单，为 GSM 提供 SMS 服务等。

虽然如此，结合我国电信标准和 Modem 硬件的实际情况，还是需要部分修改 APRO 组件的代码，用语音识别的方法解决了在“语音”模式中不能判断对方是否挂机、无人接听、忙音、静音等（“数据”模式本来就可以判断），修正了“传真”模式中部分内置 Modem 上不能收发传真的错误，成功地解决了国内语音、传真编程中的核心问题。

本书介绍利用 APRO 实现的电话自动应答系统，提供拨打电话或接听来电，播放、录制语音，检测、收集话机按键功能，还可以自动检测对方是否挂断等，是国内少有的功能较全的基于 Modem 电话语音软件；此外，还在修正了 APRO 传真模式的错误的前提下，编写了传真的收发、文件传输应用软件。

本书对组件的属性、方法、事件不是单纯的描述，而是有丰富的实例，并对实例进行深入剖析，更易于理解，以便让读者迅速地编出实用的程序，透彻地理解语音传真编程的实现方法。

本书共分为 9 章，第 1 章是引言，让读者能够对 Delphi 的功能以及所涉及到的基础知识有所了解。

第 2 章是直接 IO 及 TAPI 编程。本章主要对 Modem 编程的常用 API，多线程串口的例子，对控件 MSCOMM 的使用、输入 AT 命令测试串口和来电显示的例子做了详细的介绍；还通过对 TAPI 的实例剖析，理解直接使用 TAPI 的方法。

第 3 章是 APRO 基本控件。本章对部分 APRO 组件的属性、方法、事件做了详细介绍，其间也有一些代码的剖析，有利于对 APRO 的理解。

第 4 章是传真控件。本章主要介绍传真的原理，传真文件的转换、查看和收发 FAX 的 C/S管理等，并以实例的形式来概述，帮助理解组件的使用。

第 5 章是 TAPI 控件。本章详细描述了 TAPI 控件的属性、方法和事件。

第 6 章是 APRO 2.x 的简单介绍。这是为了兼容的目的而保留下来的控件。由于前面

介绍的控件已完全可以代替这些功能，因此只对其中的基本属性、方法、事件进行简介。

第 7 章是语音自动应答系统的设计。这里介绍了一个较简单的和一个较完整的功能齐全的语音自动应答程序，详细地分析了自动应答系统的源代码，并对 APRO 部分源代码进行修改，使用起来更灵活。在编译这个程序之前必须先用本书提供的源代码覆盖 APRO 的部分源代码，并重新编译 APRO 组件。

第 8 章是传真收发系统的设计。本章结合一个完整的传真收发系统的剖析，使读者能够迅速掌握控件的使用技巧。本章着重点在于收发传真程序的编写，填补了 Delphi 难以编写 Modem 传真收发程序的空白。如果不能正常收发传真，可看本章的疑难解答。

第 9 章介绍的是监视串口数据的 Windows 内核程序。编写语音、传真等与串口有关的程序时，了解与串口的通信数据是很重要的，利用提供的方法能很好地截取经过串口的数据，并保存为指定文件。

由于时间仓促，书中难免有不足之处，恳请读者批评指正。如果在阅读本书或使用书中例子的过程中有什么疑问，也欢迎与作者联系。

温锦山
wenjinshan@163.net
http://wenjinshan.yeah.net
唐柱鹏
tzhp@163.com
http://tzhp.yeah.net
2002.1

目　录

第1章 引 言

Delphi 是全新的可视化编程环境，为程序员提供了方便快捷的应用程序开发工具。Delphi 利用 Windows 图形化用户界面，采用面向对象的先进设计思想，是非常强大和易用的编程工具，可以很方便地编写标准的 GUI(Graphical User Interface，图形用户界面)或者控制台程序。当创建 GUI 应用程序时，利用 RAD 环境中的编译语言(Object Pascal)对 Windows 用户图形接口都以控件的形式包括在 Delphi 中。这意味着可以在应用程序中不用写一行代码，而仅仅是把控件放置于应用程序窗口或模块中、在 Form 中添加 ActiveX 控件等，就可以完成一定的功能。而对事件的处理更为简单，只要在相应的地方加入代码即可。基于 Object Pascal 可视化开发环境的对象向导与 C++相似，有些方面甚至优于 C++。Delphi 的模板都是 Pascal 程序结构，编译生成的应用程序不需要运行库，应用更广泛。

1.1 Delphi 组件和 VCL

VCL(Visual Components Library，可视化的控件库)是 Delphi 向导框架。在这个丰富的库中，会发现很多像 Windows 的对象，如 Windows、Buttons 等。当然，VCL 不仅仅包含这些控件的内容，还有一些定制的控件库，如 Timer 和 MultiMediaPlayer。另外，还有一些非可视对象，如 String Lists、Database Table 和 Streams。用户也可以编写自己的控件。

1.2 Delphi 开发数据库

Delphi 能够访问许多类型的数据库，它支持 BDE、ODBC 和 ADO 三种数据库引擎。BDE 可以访问本地的数据库(如 Paradox 和 Dbase)、网络的服务器(如 InterBase、SysBase、MS SQL 和 Oracle)等。

1.3 OOP 结构

数据类型 record 是普遍使用的对象，一个记录类型包含很多个字段，这些字段是记录的属性，事实上一个对象就像一个记录。对象与记录的差别是对象具有方法。一个对象包含属性和方法，方法是过程和函数的集合。

1.3.1 类

一个类定义了一个类型，其中包括状态和动作，对象是类的实例，和声明变量一样，如：

```
Var
  I:integer
```

I 就是类型 Integer 的实例。再如：

```
Var
  Car: Tverhicle;
```

Car 就是对象，事实上 Car 是指向对象的指针。

1.3.2 类定义

类的声明：

```
type
  TForm1 = class (Form)
  {列出类拥有的对象如 Tbutton、Tedit 等}
private
  {Private declarations}
public
  {public declarations}
end;
```

1.3.3 类的特性

方法和字段可以定义为以下的类型：

Private：只对本单元可用，其他单元不可见；

Public：字段和方法在其他单元也可以使用；

Protected：仅仅在当前类和继承的类可用。

以上分类声明的好处是：如果改变私有部分的属性和方法，对于其他的单元不会受到影响。

使用其他单元的变量有两种方法：

(1) 本单元使用其他单元的变量时，该变量在对方的单元中必须声明为全局变量。不推荐用这种方法，因为其不能确定变量的值。

(2) 封装在对象中，在私有部分声明该变量，可以用在公有部分声明的方法读写这些变量。推荐用这种方法，如 MyCounter 变量的例子：

```
type
Tform1 = class(Tform)
        {controls here}
    private
        {Private declarations}
        MyCounter: Integer;
    Public
      {public declarations}
      procedure SetMyCounter ( I: Integer);
      function GetMyCounter: Integer;
end;
```

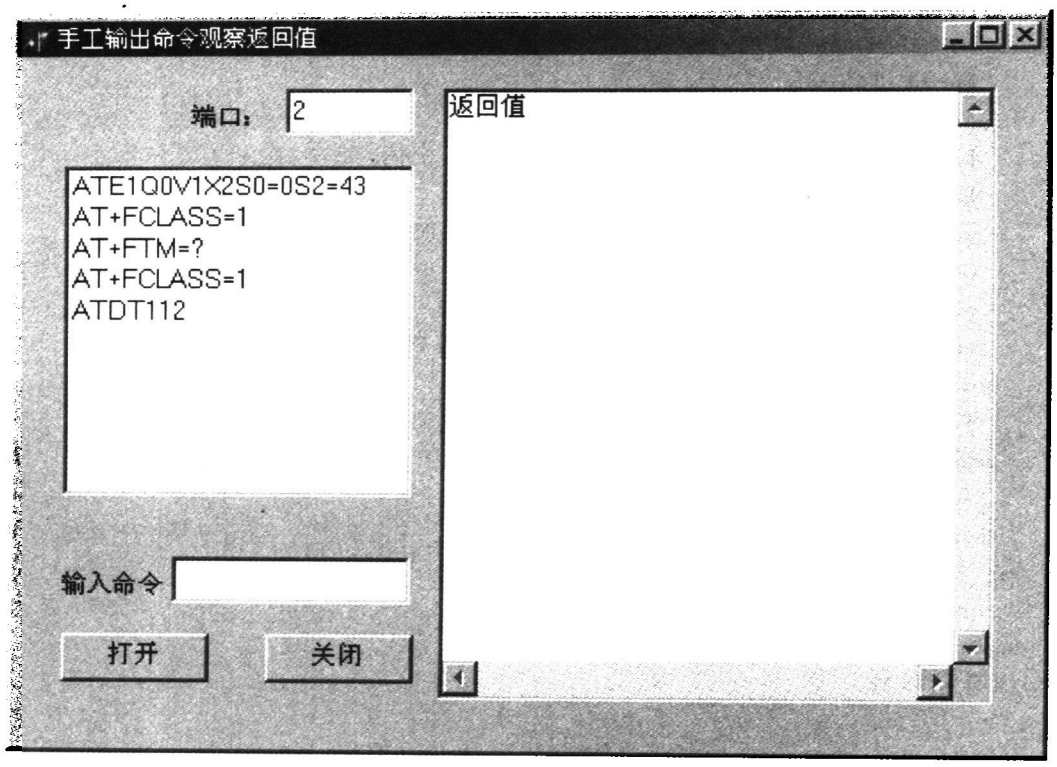

图 2.5　键盘输入 AT 命令并观察返回值

```
+FCLASS=1    进入数据模式
+FCLASS=8    进入语音模式
+FCLASS?     支持哪几种模式
+FCLASS=?    当前是什么模式
+VIP         初始化语音
+VCID=n      来电显示标识
+VCID?
+VCID=?
}
unit UnitComm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  OleCtrls, MSCommLib_TLB, StdCtrls;
type
  TForm1 = class(TForm)
    MSComm1: TMSComm;
    Edit1: TEdit;
```

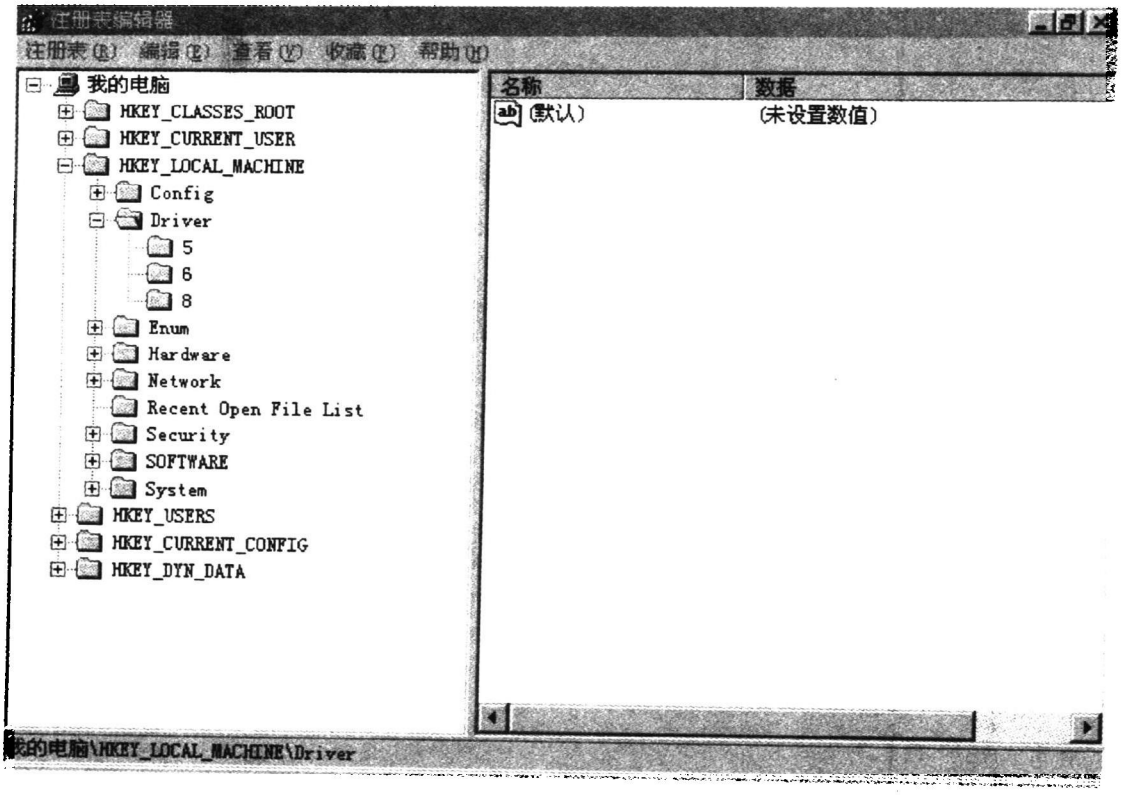

图 1.1 注册表编程器

操作注册表需要了解注册表的六个根键。下面是 Delphi 的定义：

```
const
  {$EXTERNALSYM HKEY_CLASSES_ROOT}
HKEY_CLASSES_ROOT=DWORD($80000000);
  {$EXTERNALSYM HKEY_CURRENT_USER}
HKEY_CURRENT_USER=DWORD($80000001);
  {$EXTERNALSYM HKEY_LOCAL_MACHINE}
HKEY_LOCAL_MACHINE=DWORD($80000002);
  {$EXTERNALSYM HKEY_USERS}
HKEY_USERS=DWORD($80000003);
  {$EXTERNALSYM HKEY_PERFORMANCE_DATA}
HKEY_PERFORMANCE_DATA=DWORD($80000004);
  {$EXTERNALSYM HKEY_CURRENT_CONFIG}
HKEY_CURRENT_CONFIG=DWORD($80000005);
  {$EXTERNALSYM HKEY_DYN_DATA}
HKEY_DYN_DATA=DWORD($80000006);
```

必须在 TRegistry 对象的 RootKey 属性中指定以上值。要取得某一个路径的某个键值，必须找到某一个主键，例如有如下一个路径存放着 APRO 的有关信息：

HKEY_LOCAL_MACHINE下的\Software\TurboPower\AsyncProfessional

其中：AsyncProfessional是键，在它前面的\Software\TurboPower便是主键，而这些键又是放在根键HKEY_LOCAL_MACHINE中。在这里设置AsyncProfessional的类型是字符串，所以需要一个字符串变量存放它。因此，读出数据的过程应该是：确定根键，进入主键（路径），读出键的数据值。

对注册表进行操作的方法：

```
procedure CloseKey;
constructor Create(Aaccess:LongWord); overload;
function CreateKey(const Key: String): Boolean;
function DeleteKey(const Key: String): Boolean;
function DeleteValue(const Name: String): Boolean;
destructor Destroy; override;
function GetDataInfo(const ValueName: String; var Value: TregDataInfo):
    Boolean;
function GetDataSize(const ValueName: String): Integer;
function GetDataType(const ValueName: String): TregDataType;
function GetKeyInfo(var Value: TregKeyInfo): Boolean;
procedure GetKeyNames(Strings: Tstrings);
procedure GetValueNames(Strings: Tstrings);
function HasSubKeys: Boolean;
function KeyExists(const Key: String): Boolean;
function LoadKey(const Key, FileName: String): Boolean;
procedure MoveKey(const OldName, NewName: String; Delete: Boolean);
function OpenKey(const Key: String; CanCreate: Boolean): Boolean;
function OpenKeyReadOnly(const Key: String): Boolean;
function ReadBinaryData(const Name: String; var Buffer; BufSize: Integer):
  Integer;
function ReadBool(const Name: String): Boolean;
function ReadCurrency(const Name: String): Currency;
function ReadDate(const Name: String): TdateTime;
function ReadDateTime(const Name: String): TdateTime;
function ReadFloat(const Name: String): Double;
function ReadInteger(const Name: String): Integer;
function ReadString(const Name: String): String;
function ReadTime(const Name: String): TdateTime;
function RegistryConnect(const UNCName: String): Boolean;
procedure RenameValue(const OldName, NewName: String);
function ReplaceKey(const Key, FileName, BackUpFileName: String): Boolean;
function RestoreKey(const Key, FileName: String): Boolean;
function SaveKey(const Key, FileName: String): Boolean;
function UnLoadKey(const Key: String): Boolean;
```

```
function ValueExists(const Name: string): Boolean;
procedure WriteBinaryData(const Name: String; var Buffer;BufSize:Integer);
procedure WriteBool(const Name: String; Value: Boolean);
procedure WriteCurrency(const Name: String; Value: Currency);
procedure WriteDate(const Name: String; Value: TdateTime);
procedure WriteDateTime(const Name: String; Value: TdateTime);
procedure WriteExpandString(const Name, Value: String);
procedure WriteFloat(const Name: String; Value: Double);
procedure WriteInteger(const Name: string; Value: Integer);
procedure WriteString(const Name, Value: String);
procedure WriteTime(const Name: String; Value: TdateTime);
```

主要属性：

Access：访问的安全属性。

CurrentKey：键。

CurrentPath：路径。

RootKey：根键。

下面的例子完成了对文件的关联操作：

```
uses Registry;  //一定要加入单元 Registry

procedure Tform1. Button1Click(Sender: Tobject);
const
    cMyExt='. apf';
    FileType='My. APF';
var
    Reg: Tregistry;
begin
    Reg:=Tregistry. Create;
    try
        Reg. RootKey:=HKEY_CLASSES_ROOT;  //设置根键 HKEY_CLASSES_ROOT
        Reg. OpenKey(cMyExt, True);  //打开键,True 表示如果键不存在则创建
        Reg. WriteString('', FileType);
            //写文件类型 APF 到注册表
            //HKEY_CLASSES_ROOT\. apf\(Default)='My. APF'
        Reg. CloseKey;  //关闭当前键

        //以下设置图标
        Reg. OpenKey(FileType, True);
        Reg. WriteString('','APF File');
            //添加 HKEY_CLASSES_ROOT\My. APF\(Default)='APF File' Reg. CloseKey;
        Reg. OpenKey(FileType+'\DefaultIcon', True);
        Reg. WriteString('', Application. ExeName+',0');
```

```
            //写文件类型的默认的图标
            //添加 HKEY_CLASSES_ROOT\My. APF\DefaultIcon
            //\(Default)='应用程序目录\Project1. exe,0'
        Reg. CloseKey;

        //以下设置关联
        Reg. OpenKey(FileType+'\Shell\Open', True);
            //写打开操作
        Reg. WriteString('','&Open');
        Reg. CloseKey;
        Reg. OpenKey(FileType+'\Shell\Open\Command', True);
        Reg. WriteString('','"'+Application. ExeName+'""%1"');
            //写打开类型文件的应用程序
            //添加 HKEY_CLASSES_ROOT\Project1. FileType\Shell\Open\Command
            //(Default)='"Application Dir\Project1. exe""%1"
        Reg. CloseKey;
        SHChangeNotify(SHCNE_ASSOCCHANGED, SHCNF_IDLIST, nil, nil);
            //使 Windows Explorer 知道所添加的类型
    finally
        Reg. Free;
    end;
end;

end.
```

上面是用 SHChangeNotify 来通知系统的，也可发送 WM_WININICHANGE 或 WM_SETTINGCHANGE 消息来通知，例如：

```
SendMessage(HWND_BROADCAST, WM_WININICHANGE, 0,
    LongInt(Pchar('你的注册表已经改变')));
```

1.5 多线程技术

1.5.1 线程的概念

调入内存准备执行的应用程序就是进程。每个进程可以有一个线程(即主线程)或多个线程，每个线程独立运行，保持自己的堆栈和寄存器。每个进程占据着 4 GB 的地址空间，这里的 4GB 的地址空间不是指物理地址空间，而是 Windows 的虚拟地址空间，其目的是使进程与进程互不干扰。由于每个进程都在自己的 4 GB 的地址运行，而不必理会其他的进程。所以每个进程在计算机中与其他进程的关系是完全独立的，它们互相覆盖的机会非常少。因此，一个程序对内存的无效访问导致覆盖另一个程序或操作系统的某些部分的情况是不可能的，每个进程都被封闭在一个安全的虚拟世界中，在其中它不能获得任何其他程序。这样的结果使进程间的通信与 Windows 3.1 中有很多不同。但是，这也使得计算机不会轻易崩溃，这就是

使用 Win32 的好处。

线程是一种操作系统对象，它代表一个进程中内部执行的代码的路径，是操作系统分配 CPU 时间的基本实体。对于单 CPU 来说并不是真正的多线程，而是把 CPU 的时间分成很短的时间片段分配给每个线程，给人的感觉是同时运行。应用程序为了实现多任务并行处理，可以采用创建成多个进程和在单一进程中创建多线程两种方法，但后者比前者更有效。通常，每个进程至少有一个线程（主线程）在执行自己的地址空间中的代码。如果没有线程执行进程地址空间中的代码，进程也就不存在，系统将自动清除进程及其地址空间。当进程终止，其创建的各种资源将被清除。

多线程同时执行，将会引起对共享资源的冲突。为避免冲突，有必要同步访问共享资源的多线程。而且线程同步也有利于确保相互独立的代码按照正确的顺序执行，如果不能保持多线程的同步，就会导致死锁和竞争的问题。

Win32 API 提供许多保持线程同步的对象（如互斥对象、文件句柄、线程句柄等），它们的句柄可以用来同步多个线程。一般采用创建事件对象来保持线程同步，当进程或线程终止时，进程和线程将被标记起来（signaled）。

1.5.2 线程的挂起和继续

使用线程首先要创建线程 CreateThread，其参数说明了线程开始执行的代码地址，还有一个可选的 32 位指针参数。通常起始地址是程序代码中定义的函数名，即使这个地址指向数据、代码或不可访问的区域，CreateThread 仍然成功。当执行时，如果起始地址出错，将出现异常，线程终止。如：

```
hThread:=CreateThread(nil,          //无安全属性
                      0,            //默认栈的大小
                      @ThreadFunc,  //线程函数
                      nil,          //线程参数
                      0,            //创建线程标志
                      ThreadID);    //返回线程 ID
 If hThread=0 then
     MessageBox(handle,'线程启动出错',nil,MB_OK);
```

参数说明：

第一个参数代表一系列安全属性。如果这个参数是 nil，将会使用默认的安全属性。在 Windows 95/98 下，把这个参数设置为 nil 是标准的用法，除非让子进程继承这个线程。如果线程的第二个参数是 0，那么线程堆栈的大小和应用程序堆栈的大小是一样的。换言之，也就是主线程和正在启动的线程具有相同大小的堆栈。如果必要的话，堆栈自动增长。第三个参数设置为线程起始执行地址，一般是可以完成一定功能的函数，为了取得函数的地址，需用“@”符号。

如果要在函数中传递一个参数，就要在线程参数中指定。当然可以创建一个结构或记录，然后把这个地址传递给这个参数，或者是字符串或其他类型的变量。线程标志能够传递与线程相关的标志，如果指定了 CREATE_SUSPENDED 标志，线程创建后立即挂起，直到调用 ResumeThread 函数；如果是 0，那么创建后直接执行。