飞思科技
FECIT Sci-Tech
www.fecit.com.cn

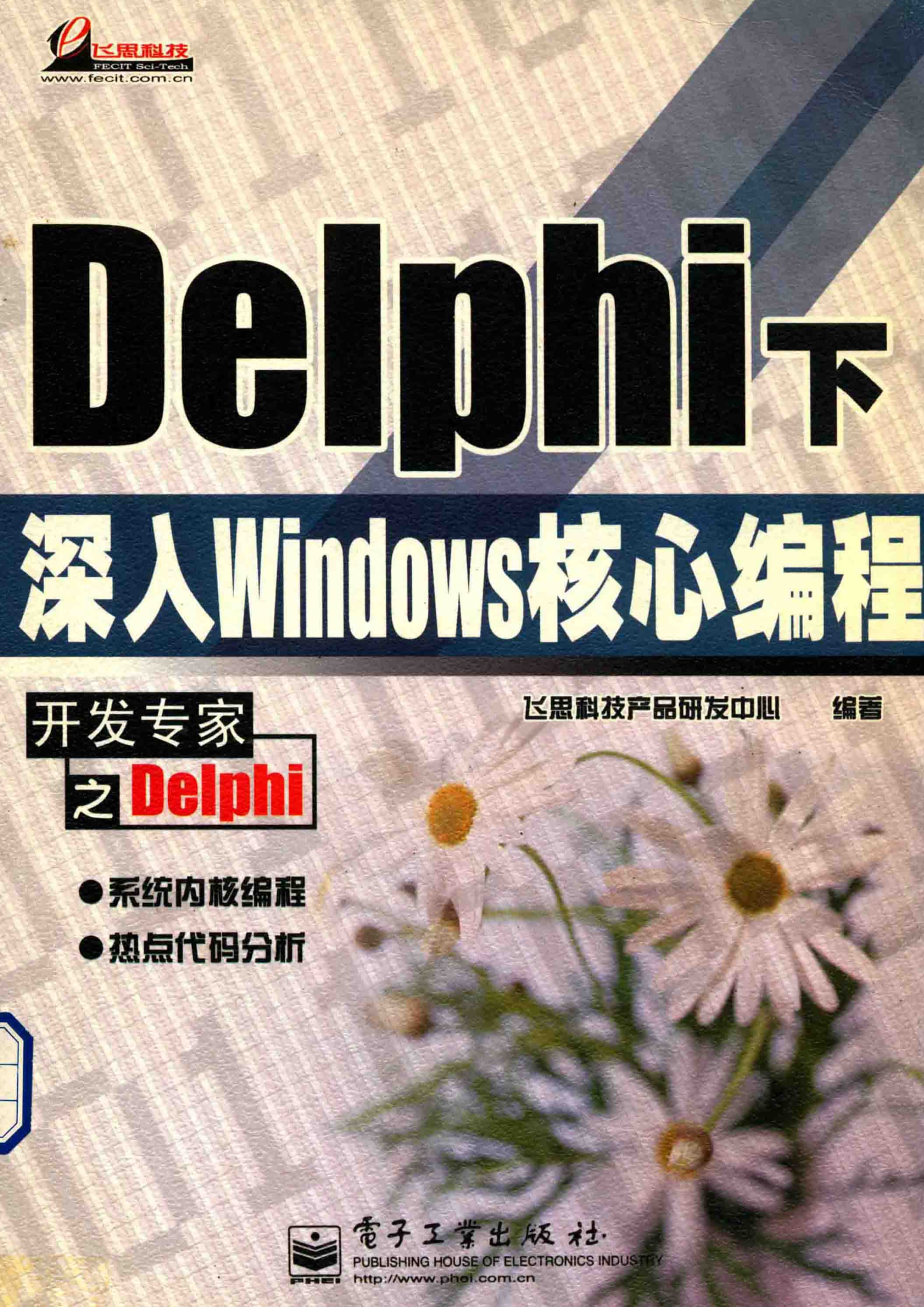
# Delphi 下

# 深入Windows核心编程

开发专家之Delphi

飞思科技产品研发中心 编著

- 系统内核编程
- 热点代码分析



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

开发专家之Delphi

# Delphi下深入Windows核心编程



飞思科技产品研发中心 编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

## 内 容 简 介

本书是一本介绍 Windows 核心技术及高级技巧的专著。从系统内核编程出发，使用大量的例子帮助读者理解这些编程技术，讲述了线程同步及隐藏、系统钩子深入分析、读写物理磁盘的关键技术、读写物理内存和其他进程内存的核心技术、Windows 9x 下调用 16 位实模式和保护模式代码的核心技术、直接读写端口技术、可执行文件加壳的技巧、PE 结构分析、Ring0 的实现、Windows API 截取技术、屏幕取词技术等方面的内容。全书对热点源代码进行了深入剖析和讲解，同时本书汇聚了作者利用 Soft-ICE 跟踪调试经验，作者多年的编程心得和技巧一览无遗。随书附送的光盘提供了书中涉及的程序源代码。

本书可对 Windows 核心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 下深入 Windows 核心编程 / 飞思科技产品研发中心编著. —北京：电子工业出版社，2003.1
（开发专家之 Delphi）
ISBN 7-5053-8402-3

Ⅰ.D... Ⅱ.飞... Ⅲ.①软件工具－程序设计②窗口软件，Windows－程序设计 Ⅳ.①TP311.56②TP316.7

中国版本图书馆 CIP 数据核字（2002）第 105039 号

责任编辑：王树伟 王 蒙
印 刷：北京市增富印刷有限责任公司
出版发行：电子工业出版社 http://www.phei.com.cn
北京海淀区万寿路 173 信箱 邮编：100036
经 销：各地新华书店
开 本：787×1092 1/16 印张：33.5 字数： 857.6 千字 附光盘 1 张
版 次：2003 年 1 月第 1 版 2003 年 1 月第 1 次印刷
印 数：6 000 册 定价：48.00 元（含光盘）

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077

“开发专家”是电子工业出版社计算机研发部长期以来精心培育的计算机科学技术类本版图书品牌。这个品牌是由多个专题系列组成的横向大系列，涵盖了计算机技术的各个方面，特别是一直受到极大关注的程序开发类系列，例如《开发专家之数据库》、《开发专家之网络编程》、《开发专家之 Delphi》及《开发专家之 Sun ONE》等。这些专题系列基于各自的角度，从纵向上包含了该专题的所有内容。因此，整个“开发专家”的品牌架构纵横交错，囊括了应用范围最广的计算机技术，海纳百川而又极具可扩展性。

“开发专家”的作者队伍主要依托于“飞思科技产品研发中心”。“飞思科技产品研发中心”是由专业的策划人员、权威的技术专家和资深的作者队伍共同构成的。在图书的出版上，形成了以研发为基础、以出版为中心、以服务为支持的专业化出版框架和流程。通过深入的市场调查和技术跟踪，在综合了技术需求和读者焦点等因素的基础上，形成各系列丛书的写作重点和大纲，然后聘请业界的有丰富实践经验及理论素养的资深专家及学者进行写作。同时，策划工作全程介入写作进程，严格控制写作质量，用最专业的技术背景、最深刻的理论基础、最具代表性的案例、最能为专业读者接受的形式，为读者提供品质最佳的图书产品，体现了出版者和著作者的完美结合。

多年来，计算机研发部始终把创造社会效益摆在首位，秉承一切为国内计算机技术专业读者服务的精神，为推动国内 IT 业发展、为体现国内技术的原创水平，穷尽所有的创意与努力，将出版者的命运与读者的支持紧紧地连在了一起。

在此，我们临出版行业之残酷竞争而不惧，异军突起，这与我们扎实的工作和广大读者的支持是分不开的。为使我们的脚步更坚实、使我们的队伍永葆活力和创造力，我们期待着您能为我们的前进贡献出您的意见和建议。同时，我们也在期盼着您的加入。

我们的联系方式如下：

咨询电话：（010）68134545　　68131648

答疑邮件：support@fecit.com.cn

飞思在线：http://www.fecit.com.cn　　http://www.fecit.net

答疑网址：http://www.fecit.com.cn/question.htm

通用网址：计算机图书、飞思、飞思教育、飞思科技、FECIT

电子工业出版社计算机研发部

# 前言

近些年计算机软件的发展很快，从工具软件到管理软件都有很成熟的产品。软件系统的开发不但需要扎实的软件基础知识及团队合作精神，还需要对 Windows 系统这个黑匣子有一定的了解，以编写出高性能的、高技术含量的代码来。如果仅凭对开发语言的掌握，不求甚解，不深入系统内核去分析、挖掘，就不会有 Soft ICE（系统反编译调试工具）、PartitionMagic（魔术化分区工具）、Norton Utilities（诺顿磁盘工具）等著名的软件。可是对于一些核心技术，大多数程序员还是知之甚少。本书介绍的技术大部分是作者及网友多年编程的积累和总结。把这些代码汇集在一起出版，希望能对读者有一定的帮助或启发。

本书讲述了线程同步及隐藏、系统钩子深入分析、读写物理磁盘的关键技术、读写物理内存和其他进程内存的核心技术、Windows 9x 下调用 16 位实模式和保护模式代码的核心技术、直接读写端口技术、可执行文件加壳的技巧、PE 结构分析、Ring0 的实现、Windows API 截取技术、屏幕取词技术等方面的内容。

其中的屏幕取词技术是网络上许多 BBS 的热点话题，遗憾的是 BBS 上涉及到关键技术时都含糊其词。本书将介绍一个完整的屏幕取词程序（包括 16 位的 Thunk 关键技术，适用于 Windows 9x/NT/2000）。

CIH 病毒曾经一度让全球的人们谈虎色变，这也是前几年的热门话题。CIH 病毒利用 Windows 9x 下保护模式编程的一个技巧（漏洞），从 Ring3 跳到 Ring0 执行一些系统服务函数，做了一些“见不得人”的操作：用垃圾数据覆盖硬盘扇区、读写破坏 BIOS 芯片。CIH 病毒源代码中有许多值得学习的地方，本书中的 Windows 9x 下的物理磁盘读写技术、时间变速器等就使用了这种技术。玩过网络游戏的人都应该听说过时间变速器，这种变速器运行在 Ring0，与操作系统同一特权级，使用特殊的指令改变计算机的时钟周期，骗过操作系统，造成计时器（时间）高速“前进”的假象，许多网络游戏就是用这样的变速器来作弊的。

在 DOS 下实现过汇编高级编程的人一定会对 EXE 文件很熟悉，可是 Microsoft 的产品未来将脱离 DOS，不再提供对 DOS 的支持。Win32 提供新的可执行文件格式 PE，本书将详细地说明 PE 的结构，并在 PE 文件结构的基础上实现截取 API、屏幕取词等技术。

全书分为 10 章，内容简介如下：

第 1 章 DLL 与数据共享，在 Win32 中通过使用内存映像文件、全局原子等实现多进程共享数据，这一章是基础，后面的很多章节都使用了本章介绍的数据共享技术。

第 2 章钩子原理，介绍了消息（包括键盘、鼠标消息）的截取、钩子的安装、DLL 注入其他进程的方法等技术。这一章所介绍的内容是后面截取 API、屏幕取词等技术的基础知识。

第 3 章系统内核，让读者对 Windows 内核有了一定的了解，介绍了进程的枚举、线程的同步、进程深度隐藏的种种方法及技巧、Windows NT/2000 的性能计数器等技术。

第 4 章低层操作，介绍内嵌汇编获取 Ring0 特权的技巧、Windows 9x/NT/2000 时间变速器的实现原理、16 位与 32 位代码的核心接口技术等方面的内容。

第 5 章磁盘读写，分别提供了 Windows 9x/NT/2000 下逻辑扇区和物理扇区的读写，

介绍了磁盘操作中鲜为人知的核心技术。

第 6 章回收站和 IE，介绍了回收站的物理存贮结构、监视回收站、IE 缓冲区数据、IE 窗口的监视等方面的内容。

第 7 章高级应用，介绍了 DDE、消息机制、剪贴板和目录的监控、程序运行后自动删除、只运一个实例的多种方法、移动正在使用的文件、类型转换与存储转换、可执行文件加壳等高级应用技巧。

第 8 章 PE 结构分析，介绍了 Windows 的 PE 文件结构，并详细分析了其中每部分的含义，这是截取 API 等技术的基础。

第 9 章内存管理，介绍了物理内存的存取、进程内存的存取、内存堆的枚举等核心技术，让读者对内存的结构有更深入的了解。

第 10 章 API Hook 及屏幕取词，介绍了 16 位和 32 位 Windows 9x 的 API Hook 技术、Windows NT/2000 的 API Hook 技术、Windows 9x/NT/2000 下的屏幕取词技术（该技术还包括 16 位的 Thunk 等关键技术）。其中，如果屏幕取词技术加入英文词库就可以实现独自开发词霸。

特别说明：在书中所有出现 Windows 9x 的地方，都适用于 Windows Me；所有出现 Windows NT/2000 的地方，都适用于 Windows XP。随书附送的光盘提供了书中涉及的程序源代码。

本书由飞思科技产品研发中心策划并组织编写，同时参与写作的还有温锦山、唐柱鹏、刘鉴澄、张恒、赵蕾等人。书中有三个例子由网友陈经韬（http://www.138soft.com/）提供，在此表示感谢。由于时间仓促，书中难免有不足之处，恳请读者批评指出。如果在阅读本书或使用书中例子的过程中有什么疑问，欢迎与我们联系。

当然，限于作者水平，加之时间仓促，书中不足之处难免，敬请读者批评指正。

我们的联系方式：

咨询电话：（010）68134545　68131648

答疑邮件：support@fecit.com.cn

飞思在线：http://www.fecit.com.cn　http://www.fecit.net

答疑网址：http://www.fecit.com.cn/question.htm

通用网址：计算机图书、飞思、飞思教育、飞思科技、FECIT

飞思科技产品研发中心

# 第 1 章　DLL 与数据共享

与 Win16（16 位 Windows）不同的是，在 Win32（32 位 Windows）中不能将全局变量放在 DLL 中让两个或两个以上的进程共享。这是由于 Win32 中 DLL 没有自己的局部堆，当一个进程装入 DLL 时，系统会自动将 DLL 的数据和代码映射到该进程的地址空间，DLL 中的任何函数的内存分配请求都是在被调用进程的地址空间中分配的，其他的进程无权访问这块内存。为了实现不同进程中的数据共享，可以使用内存映像文件、注册表或文件的读写、全局原子、声明一个共享数据段（只适用于 Windows 9x）、套接字、管道、远程过程调用等技术。其中，使用内存映像文件是最简单有效且实用的方法。

## 1.1　关于 DLL

动态链接库给应用程序提供了一种调用不在其执行代码中的函数的技术。函数全部封装在动态链接库中，动态链接库实际上是应用程序存储子程序的地方，可以把多个程序频繁使用的公共函数集中在一起，这样方便模块重用，减少内存空间的交换。DLL 可以拥有自己的数据段，但没有自己的堆栈，而是使用应用程序（EXE 文件）的堆栈。

Windows 系统平台提供了一种完全不同的编程和运行环境，可以把程序分为多个单独功能的模块（DLL），需要用到某一功能即调用相应的模块，不仅减少了 EXE 文件的大小和对内存空间的需求，而且使这些 DLL 模块可以同时被多个应用程序使用。可以多个模块同时使用一个 DLL，共享 DLL 在内存中的单一拷贝。例如，Windows 自己就将一些主要的系统功能以 DLL 模块的形式实现，再如，IE 中的一些基本功能就是由 DLL 文件实现的，可以被其他应用程序调用和集成。

如果与其他 DLL 之间没有冲突，该文件通常映射到每个进程虚拟空间的同一地址上。DLL 模块中包含各种导出函数，用于向外界提供服务。Windows 在加载 DLL 模块时将进程函数的调用与 DLL 文件的导出函数相匹配。

在 Win32 环境中，每个进程都复制了自己的读/写全局变量。如果想要与其他进程共享内存，必须使用存映像文件、注册表或文件的读写、全局原子、声明一个共享数据段（只适用于 Windows 9x）、套接字、管道、远程过程调用等技术。

### 1.1.1　DLL 的结构

每个 DLL 文件都包含一个导出函数表，这些导出函数由它们的函数名或函数编号与外界联系起来，函数表中还包含了 DLL 中函数的地址。当应用程序加载 DLL 模块时，应用程序并不知道在 DLL 中的调用函数的实际地址，只知道函数的名字或编号，系统在加载 DLL 模块时动态建立一个函数调用与函数地址的对应表。如果重新编译或重建 DLL 文

件，并不需要修改应用程序，除非改变了导出函数的名字或编号。简单的 DLL 文件只为应用程序提供导出函数，比较复杂的 DLL 文件除了提供导出函数以外，它本身还调用其他 DLL 文件中的函数。这样，一个特殊的 DLL 可以既有引入函数，又有导出函数。这并不会造成任何问题，因为动态链接过程可以处理交叉引用的情况。

#### 1.1.1.1 链接方式

应用程序导入函数与 DLL 文件中的导出函数进行链接有两种方式：隐式链接和显式链接。所谓的隐式链接是指在应用程序中不需指明 DLL 文件的实际存储路径，程序员不需关心 DLL 文件的实际装载，而显式链接则与此相反。

1. 隐式链接方式

采用隐式链接方式时，在 DLL 代码中可以像下面这样明确声明导出函数：

```
exports
    FunctionName Index 16 Name MyName,
    ProcedureName index 17 name YourName,
    ……
```

格式是“函数名 Index xx Name 导出名字”。当然，如果省略了“Name xxxx”，则意味着导出的名字就是函数名/过程名；如果省略了“Index xx”，则意味着由编译器自动产生编号。

在应用程序方面，要求像下面这样明确声明相应的引入函数：

```
procedure MyName; external 'MYLIB.DLL';
```

2. 显式链接方式

显式链接方式是直接调用 Win32 的 LoadLibary 函数，并指定 DLL 的路径作为参数。LoadLibary 返回 HINSTANCE 参数，应用程序在调用 GetProcAddress 函数时使用这个参数。GetProcAddress 函数将函数名或编号转换为 DLL 内部的地址。例如，有一个导出如下函数的 DLL 文件：

```
procedure DoSomething; external 'MYLIB.DLL';
```

下面是应用程序对该导出函数的显式链接的例子：

```
var
  prun:trun=nil;{Prun 为过程，默认为 nil}
  LibHandle:integer;
  CurPath:string;
begin
    {……}
  {装入 MYLIB.DLL}
  LibHandle:=LoadLibrary(pchar(CurPath+'MYLIB.DLL'));
  {如果装入成功}
  if libhandle<>0 then
  begin
      {获得 DoSomething 过程的地址}

      prun:=GetProcAddress(LibHandle,'DoSomething');
      if @prun<>nil then prun; {调用该 DLL 函数}
  end;
```

```
    {……}
end;
```

如果应用程序使用 LoadLibrary 显式链接，那么在这个函数的参数中可以指定 DLL 文件的完整路径。如果不指定路径，Windows 将遵循下面的搜索顺序来定位 DLL：

- 包含 EXE 文件的目录
- 进程的当前工作目录
- Windows 系统目录
- Windows 目录
- 列在 Path 环境变量中的一系列目录

在隐式链接方式中，所有被应用程序调用的 DLL 文件都会在应用程序 EXE 文件加载时被加载在到内存中；但如果采用显式链接方式，程序员可以决定 DLL 文件何时加载或不加载。例如，可以将一个带有字符串资源的 DLL 模块以英语加载，而另一个以西班牙语加载，应用程序在用户选择了合适的语种后再加载与之对应的 DLL 文件。因此，显式链接具有更大的灵活性，缺点是调用时比隐式链接需要编写更多的代码。

#### 1.1.1.2　调用方式

有如下三种调用方式：

- 通过过程、函数名。
- 通过过程、函数别名。
- 通过过程、函数的编号。

例如：

```
Function Getstring:string;stdcall;external 'Mydlls.dll' name 'Mygetstr';
```

Name 子句指定的函数名 Getstring 实际上是调用 Mydlls.dll 中的 Mygetstr，当程序调用 Getstring 时，实际上是调用 Mygetstr：

```
Function Getstring : string ; stdcall ; external 'Mydlls.dll' index 5;
```

Index 子句通过函数编号引入函数，这也可以减少 DLL 的加载时间。但是，建议程序员不用使用这种调用方式。

#### 1.1.1.3　调用约定

调用约定，是指调用例程时参数的传递顺序。Delphi 中 DLL 支持的调用约定如表 1-1 所示。

表 1-1　调用约定

| 调 用 约 定 | 参数传递顺序 |
| --- | --- |
| Register | 从左到右 |
| PASCAL | 从左到右 |
| Stdcall | 从右到左 |
| Cdecl | 从右到左 |
| Safecall | 从右到左 |

使用 Stdcall 方式，能保证不同语言写的 DLL 的兼容性，同时也是 Windows API 的约定方式；Delphi 的默认调用方式为 Register；Cdecl 是采用 C/C++的调用约定，适用于由 C++语言编写的 DLL；Safecall 是适合于声明 OLE 对象中的方法。

#### 1.1.1.4 DLL 中的变量和段

一个 DLL 声明的任何变量都为自己私有的，调用它的模块不能直接使用 DLL 中定义的变量。如果必须使用，只有通过过程或函数来完成。对 DLL 来说，永远都没有机会使用调用它的模块中的变量。一个 DLL 没有自己的堆栈段（SS），而是使用调用它的应用程序的堆栈。因此在 DLL 中的过程、函数不要假定 DS=SS（DS 为数据段）。

### 1.1.2 DLL 数据作用范围

DLL 可以与其相连的多个进程共享数据、变量和分配的内存，对于每个相关的进程来说也可以有其专用的数据、变量和分配的内存。因而，用户可以以每个线程为依据存储数据，从而使一个线程的多个实例具有其专用的数据。

DLL 函数中的局部变量的作用范围限于定义的函数之中。DLL 源代码中的全局变量对每一个与 DLL 相连的进程都是全局的。

为了实现不同进程中的数据共享，可以使用内存映像文件、注册表或文件的读写、全局原子、声明一个共享数据段（只适用于 Windows 9x）、套接字、管道、远程过程调用等技术。实际应用中，推荐使用内存映像文件的方式来实现数据共享。

## 1.2 内 存 映 像

在 Win32 中，通过使用映像文件在进程间实现共享文件或共享内存数据块，如果利用相同的映像名字或文件句柄，则不同的进程可以通过一个地址指针来读写同一个文件或同一个内存数据块，并把它当做该进程内地址空间的一部分。

在 Windows 9x/NT/2000 向内存中装载文件时，使用了特殊的全局内存区。在该区域内，应用程序的虚拟内存地址和文件中的相应位置对应。由于所有进程共享了一个用于存储映像文件的全局内存区域，因而当两个进程装载相同模块（应用程序 EXE 或 DLL 文件）时，它们实际上是在内存中共享其执行代码。

内存映像文件可以映射一个文件、一个文件中的指定区域或者指定的内存块，其中的数据就可以用内存读写指令来直接访问，而不必频繁地调用 ReadFile 或 WriteFile 这样的 I/O 系统函数，从而提高了文件存取速度和效率。

映像文件的另一个重要应用就是用来支持永久命名的共享内存。要在两个应用程序之间共享内存，可以在一个应用程序中创建一个文件并映射之，然后另一个应用程序可以通过打开和映射该文件，并把它作为自己进程的内存来使用（实际上是所有进程共享的）。

### 1.2.1　创建映像文件

其代码为：

```
CreateFileMapping(
  hFile: THandle;
  lpFileMappingAttributes: PSecurityAttributes;
  flProtect: DWORD;
  dwMaximumSizeHigh: DWORD;
  dwMaximumSizeLow: DWORD;
  lpName: Pchar
): THandle;
```

这个函数用于创建映像文件。

- hFile 是调用 FileOpen()或 FileCreate()函数后返回的文件句柄。如果不是共享文件，而是共享内存区域，在这里需要设为$FFFFFFFF。
- lpFileMappingAttributes 参数是文件映像的安全属性结构（一般设为 nil）。
- flProtect 参数是文件视图的保护类型（PAGE_READ 为可读、PAGE_WRITE 为可写、PAGE_REAdWRITE 为可读写）。
- dwMaxinumSizeHigh 参数用于指定文件映像大小的高 32 位（一般为 0，除非访问的文件大于 4GB）。
- dwMaxnumSizeLow 参数用于指定文件映像的大小的低 32 位。
- lpName 参数用于指定映像名。

如果函数调用成功，将返回文件映像的句柄。

### 1.2.2　打开映像文件

其代码为：

```
OpenFileMapping(
  dwDesiredAccess: DWORD;
  bInheritHandle: BOOL;
  lpName: PChar
  ): THandle;
```

这个函数用于打开已存在的映像文件。

- dwDesiredAccess 参数用于指定访问数据的模式（FILE_MAP_READ 为可读、FILE_MAP_WRITE 为可读写、FILE_MAP_ALL_ACCESS 为可读写）。
- bInheritHandle 参数指定 OpenFileMapping 函数返回的句柄在以后新建的子进程中是否得到继承。
- lpName 参数用于指定映像名。

如果函数调用成功，将返回文件映像的句柄。

### 1.2.3 映射到本进程中

其代码如下所示：

```
MapViewOfFile(
  hFileMappingObject: THandle;
  dwDesiredAccess: DWORD;
  dwFileOffsetHigh: DWORD;
  dwFileOffsetLow: DWORD;
  dwNumberOfBytesToMap: DWORD
): Pointer;
```

MapViewOfFile 可以将映像文件映射到本进程中。

- hFileMappingObject 参数通过 CreateFileMapping()或 OpenFileMapping()返回的文件映像的句柄。
- dwDesiredAccess 参数用于指定访问数据的模式（FILE_MAP_READ 为可读，FILE_MAP_WRITE 为可读写，FILE_MAP_ALL_ACCESS 为可读写）。
- dwFileOffsetHigh 参数用于指定数据在映像文件中的起始位置的的高 32 位。
- dwFileOffsetLow 参数用于指定数据在映像文件中的起始位置的的低 32 位。
- dwNumberOfBytesToMap 参数用于指定需要映射的字节数，设为 0 表示文件或内存区域的全部。

如果函数调用成功将返回数据映射的起始地址，这是本进程中可以直接访问的内存地址指针。

### 1.2.4 关闭内存映射

其代码为：

```
UnmapViewOfFile(
    lpBaseAddress: Pointer
): BOOL;
```

- UnmapViewOfFile 用于关闭由 MapViewOfFile 建立的内存映射。
- lpBaseAddress 参数是 MapViewOfFile 函数返回的内存地址指针。

Windows 95 环境下，这是两个或多个进程之间通过 Win32 API 实现内存共享的方法。在 Win32 上的所有共享内存依赖于 Win32 的内存映像文件 I/O 的支持。一个内存映像文件使用共享内存来提供公共的基于文件的共享数据。

内存映像文件 I/O 是 Win32 API 处理磁盘文件的一种方式。如果虚拟内存管理不控制数据缓冲和内存缓冲，则当一个文件被映像到内存区，从映像内存读写数据和从文件读写数据产生同样的效果。内存映像文件 I/O 效率很高，且使用非常方便。内存映像文件 I/O 允许两个或多个进程共享基于文件的数据。每个和共享有关的进程直接存取一组公共页。由于共享占用的资源最小，当大量的数据必须被共享时，通过内存映像文件 I/O 共享就显得非常有用。

内存映像文件的支持需要三个 Windows 内核对象：文件、文件映像和视图。为了映

像一个文件到内存，首先从磁盘打开一个文件，再建立映像文件，然后将映像文件连接到内存。为了在不同进程间同步数据，它提供了一个基于磁盘文件的逻辑连接，最终对数据块的访问通过一个数据地址指针来实现。一个内存映像文件对象可以创建多个视图，可以分别存取文件的不同部分。

使用内存映像文件 I/O 需遵循以下步骤。


（1）打开磁盘文件。这一步可采用 CreateFile 或 OpenFile 等函数。为了打开一个基于磁盘的文件，一般采用 OpenFile()函数：

```
function OpenFile(const lpFileName: LPCSTR; var lpReOpenBuff: TOFStruct;
uStyle: UINT): HFILE; stdcall;
  POFStruct = ^TOFStruct;
  TOFSTRUCT = record
    cBytes: Byte;
    fFixedDisk: Byte;
    nErrCode: Word;
    Reserved1: Word;
    Reserved2: Word;
    szPathName: array[0..OFS_MAXPATHNAME-1] of CHAR;
  end;
```


OpenFile 的返回值为–1 时，表示打开文件失败。参数 uStyle 定义了打开文件标志（OF_READ、OF_WRITE、OF_READWRITE 等）、文件被其他进程共享的方式（OF_SHARE_EXCLUTIVE、OF_SHARE_DENY_WRITE 等），以及文件打开时采取的动作（OF_CREATE、OF_DELETE、OF_EXIST 等）。在不再使用该文件时，记得调用函数 CloseHandle。

（2）创建文件映像对象。为了使用内存映像文件，要通过函数 CreateFileMapping 创建文件映像对象。CreateFileMapping 函数的第一个参数使用上一步得到的文件句柄；当然，也可以设置为$FFFFFFFF，这时文件映像对象是基于内存的，而不是基于文件的。lpszMapName 参数是给文件映像对象起的名字，必须确保其惟一性，因为与一个未知进程的名字冲突会产生非希望的共享。

除了 CreateFileMapping 函数之外，还有两个函数可以操作文件映像对象，分别是 OpenFileMapping 和 DuplicateHandle。

（3）创建视图对象。创建视图对象需要调用 MapViewOfFile()函数。


可以在一个文件中创建多个视图，以便分别访问文件的不同部分。调用 MapViewOfFile 函数时需传递视图在文件的起始位置偏移和要映射的字节数。