|  |
| --- |
| www.yesdata.net |
| Win32 DLL的创建与动态链接 |
| ——DLL运用实践谈 |
|  |
| **Yuan Aiqing** |
| **2014/1/26** |

|  |
| --- |
| Windows DLL的概念介绍及其导入导出方法的编译层面的详解，配以C++的实现。可以到wenku.baidu.com或者www.yesdata.net上获得本文的电子版。 |

目录

[前言 2](#_Toc378585863)

[一、 Windows DLL的发展历史 2](#_Toc378585864)

[二、 DLL类型：Win32 DLL 3](#_Toc378585865)

[三、 动态链接库与内存管理 3](#_Toc378585866)

[四、 动态链接DLL 3](#_Toc378585867)

[(一) 客户端可执行文件 4](#_Toc378585868)

[(二) 隐式链接 4](#_Toc378585869)

[1) 隐式链接方法 4](#_Toc378585870)

[2) C++隐式链接DLL示例代码——调用导出函数 4](#_Toc378585871)

[3) C++隐式链接DLL示例代码——调用导出类 5](#_Toc378585872)

[(三) 显式链接 5](#_Toc378585873)

[1) 显式链接方法 5](#_Toc378585874)

[2) C++显式链接DLL示例代码——调用导出函数 6](#_Toc378585875)

[3) VB显式链接DLL示例代码——调用导出函数 7](#_Toc378585876)

[4) C++显式链接DLL示例代码——调用导出类 8](#_Toc378585877)

[(四) 链接时的不同语言间参数传递的转换 9](#_Toc378585878)

[五、 创建DLL 10](#_Toc378585879)

[(一) 初始化DLL 10](#_Toc378585880)

[(二) 如何使用C++语言导出DLL 10](#_Toc378585881)

[(三) 使用其它语言导出DLL 11](#_Toc378585882)

[六、 11](#_Toc378585883)

# 前言

本文参考微软MSDN网站、维基百科等网络资料，对Windows DLL的导出、导入方法作了整理。文中的代码样本片段，基本以C++语言为主编写的，所以读者需有基础的C++编程语言知识背景，以及一些Windows运行机制相关的知识。另外，本文还从编译层面对DLL导出、导入过程作了浅显地深入了解，希望对相关读者有所帮助。

本文适用读者对象为希望使用C++来开发DLL的、或者是希望调用现有DLL的程序制作人员。

# Windows DLL的发展历史

现在网络发达，可以很轻松地从维基百科或百度搜索上面，找到Windows DLL的相关发展使。以下是截取自维基百科上的部分资料（http://zh.wikipedia.org/wiki/%E5%8A%A8%E6%80%81%E9%93%BE%E6%8E%A5%E5%BA%93）：

DLL的最初目的是节约应用程序所需的磁盘和内存空间。在一个传统的非共享库中，一部分代码简单地附加到调用的程序上。如果两个程序调用同一个子程序，就会出现两份那段代码。相反，许多应用共享的代码能够切分到一个DLL中，在硬盘上存为一个文档，在内存中使用一个实例（instance）。

DLL的广泛应用使得早期的视窗能够在紧巴巴的内存条件下运行。

DLL提供了如模块化这样的共享库的普通好处。模块化允许仅仅更改几个应用程序共享使用的一个DLL中的代码和数据而不需要更改应用程序自身。这种模块化的基本形式允许如Microsoft Office、Microsoft Visual Studio、甚至Microsoft Windows自身这样大的应用程序使用较为紧凑的补丁和服务包。

模块化的另外一个好处是插件的通用接口使用。单个的接口允许旧的模块与新的模块一样能够与以前的应用程序运行时无缝地集成到一起，而不需要对应用程序本身作任何更改。这种动态扩展的思想在ActiveX中发挥到了极致。

尽管有这么多的优点，使用DLL也有一个缺点：DLL地狱，也就是几个应用程序在使用同一个共享DLL库发生版本冲突。这样的冲突可以通过将不同版本的问题DLL放到应用程序所在的文件夹而不是放到系统文件夹来解决；但是，这样将抵消共享DLL节约的空间。目前，Microsoft .NET将解决DLL hell问题当作自己的目标，它允许同一个共享库的不同版本并列共存。由于现代的计算机有足够的磁盘空间和内存，这也可以作为一个合理的实现方法。

# DLL类型：Win32 DLL

用户可以创建“Win32 DLL”以及“MFC DLL”。

“Win32 DLL”中的导出函数可由可执行文件调用，这类DLL的导出函数，通常是通过标准C接口从DLL中导出的。可以参考Windows SDK中的[动态链接库](http://msdn.microsoft.com/zh-cn/library/ms682589(v=vs.100).aspx)部分。

“MFC DLL”是指内部使用了MFC的DLL。主要有“[静态链接到 MFC 的规则 DLL](http://msdn.microsoft.com/zh-cn/library/f22wcbea(v=vs.100).aspx)”、“[动态链接到 MFC 的规则 DLL](http://msdn.microsoft.com/zh-cn/library/30c674tx(v=vs.100).aspx)”、“[扩展 DLL](http://msdn.microsoft.com/zh-cn/library/20ytt2wa(v=vs.100).aspx)”三类，详情可以参考MSDN。

本文将主要介绍“Win32 DLL”。

# 动态链接库与内存管理

# 动态链接DLL

动态链接DLL，即：将可执行文件链接到DLL。或者称之为可执行文件加载DLL。

可执行文件可以以两种方式加载DLL：

第一种：隐式链接。

又称静态加载（static load）或加载时动态链接（load-time dynamic linking）。

在该方式下，可执行文件链接到该DLL创建者所提供的导入库文件（.lib文件）。在可执行文件加载时，操作系统加载此DLL。可执行文件调用DLL的导出函数，就好像这些函数包含在可执行文件内一样。

第二种：显式链接。

又称动态加载（dynamic load）或运行时动态加载（run-time dynamic linking）。

在该方法下，可执行文件必须进行函数调用，显式加载和卸载DLL，并访问该DLL中的导出函数。

对于同一个DLL，可执行文件可以采取以上两种链接方法。

对于可执行文件来说，可以以一种链接方法链接一个DLL，再以另一种链接方法链接另外一个不同的DLL。上述机制不是互斥的。

## 客户端可执行文件

客户端可执行文件，即调用DLL的可执行文件，它们可以用任何支持使用DLL的语言编写，包括C、C++、Pascal、Visual Basic、Delphi、C#等。

本文主要讲解动态链接DLL在C++语言下的实现，即假设客户端是用C++语言编写的。随着Windows下.NetFramework的日趋流行，托管程序如何调用本地DLL，也将在本文中被涉及到，客户端假设以托管程序C#语言开发。

## 隐式链接

### 隐式链接方法

为隐式链接到 DLL，可执行文件必须从 DLL 的提供程序获取下列各项：

●包含导出函数和/或 C++ 类的声明的头文件（.h 文件）。类、函数和数据均应具有 \_\_declspec(dllimport)；[参考 [dllimport和dllexport](http://msdn.microsoft.com/zh-cn/library/3y1sfaz2(v=vs.100).aspx)]

●要链接的导入库（.LIB files）。 （生成 DLL 时链接器创建导入库。）

●实际的 DLL（.dll 文件）。

### C++隐式链接DLL示例代码——调用导出函数

//包含导出函数和/或 C++ 类的声明的头文件（.h 文件）

#include “MyDll.h”

#pragma comment(lib,"MyDll.lib") //链接.lib文件

Int main(void)

{

Print(“some words.”);

Return 0;

}

//假设Print(char\*)是MyDll的导出函数

//可以看到就像调用普通函数一样调用导出函数

### C++隐式链接DLL示例代码——调用导出类

//包含导出函数和/或 C++ 类的声明的头文件（.h 文件）

#include “MyDll.h”

#pragma comment(lib,"MyDll.lib") //链接.lib文件

Int main(void)

{

MyClass myclass1;

myclass1.Print(“some words.”);

Return 0;

}

//假设MyClass是导出类，Print(char\*)是MyClass类的public成员函数

//可以看到就像调用普通类一样调用导出类

## 显式链接

### 显式链接方法

在显式链接下，应用程序必须进行函数调用以在运行时显式加载 DLL。 为显式链接到 DLL，应用程序必须：

●调用 LoadLibrary（或相似的函数）以加载 DLL 和获取模块句柄。

●调用 GetProcAddress，以获取指向应用程序要调用的每个导出函数的函数指针。 由于应用程序是通过指针调用 DLL 的函数，编译器不生成外部引用，故无需与导入库链接。

●使用完 DLL 后调用 FreeLibrary。

### C++显式链接DLL示例代码——调用导出函数

typedef UINT (CALLBACK\* LPFNDLLFUNC1)(DWORD,UINT);

...

HINSTANCE hDLL; // Handle to DLL

LPFNDLLFUNC1 lpfnDllFunc1; // Function pointer

DWORD dwParam1;

UINT uParam2, uReturnVal;

hDLL = LoadLibrary("MyDLL");

if (hDLL != NULL)

{

lpfnDllFunc1 = (LPFNDLLFUNC1)GetProcAddress(hDLL,

"DLLFunc1");

if (!lpfnDllFunc1)

{

// handle the error

FreeLibrary(hDLL);

return SOME\_ERROR\_CODE;

}

else

{

// call the function

uReturnVal = lpfnDllFunc1(dwParam1, uParam2);

}

}

### VB显式链接DLL示例代码——调用导出函数

' Enter the following Declare as one, single line:

Declare Sub getdiskinfo Lib "c:\dllartic\diskinfo.dll"

(ByVal mydrive As String, ByVal myvolume As String, free As Long)

Sub Command1\_Click ()

Dim drive As String \* 1

Dim volume As String \* 20

Dim free As Long

Call getdiskinfo(drive, volume, free)

Text1.Text = drive

Text2.Text = volume

Text3.Text = Str$(free)

End Sub

### C++显式链接DLL示例代码——调用导出类

//在客户端引用DLL中该类的定义（.h文件）

//class \_\_declspec(dllimport) CMyClass

//{

// Public:

// CMyClass();

// virtual ~CMyClass();

// virtual Print (); //必须是虚函数否则链接器提示找不到符号

//}

//并且DLL必须有一个导出函数以提供MyClass的实例指针

//\_\_declspec(dllexport) CMyClass\* CreateMyClass()

//{

// return new CMyClass();

//}

typedef CMyClass\* (CALLBACK\* LPFNDLLFUNC1)();

...

HINSTANCE hDLL; // Handle to DLL

LPFNDLLFUNC1 lpfnDllFunc1; // Function pointer

DWORD dwParam1;

UINT uParam2, uReturnVal;

hDLL = LoadLibrary("MyDLL");

if (hDLL != NULL)

{

lpfnDllFunc1 = (LPFNDLLFUNC1)GetProcAddress(hDLL,

" CreateMyClass ");

if (!lpfnDllFunc1)

{

// handle the error

FreeLibrary(hDLL);

return SOME\_ERROR\_CODE;

}

else

{

// call the function

CMyClass\* myclass = lpfnDllFunc1();

Myclass->Print();

}

}

## 链接时的不同语言间参数传递的转换

因为DLL通常是由C或C++编写的，所以DLL可以使用多种不直接支持客户端编程语言（如VB、Pascal）的参数。如此一来，程序员必须找到合适的数据类型来传递参数。

以下是C/C++的参数类型对应于VB时该采用哪种数据类型的整理表。

|  |  |  |
| --- | --- | --- |
| 编号 | C/C++参数类型 | VB事宜选用的参数类型 |
| 1 | 参数传递方式：值传递 | VB默认采用引用传递。但可以在参数前面加上ByVal关键字来实现。 |
| 2 | 8 到 16 位数字参数：  int, unsigned int, unsigned short, bool, WORD | ByVal Integer |
| 3 | 32 位数字参数:  long, unsigned long, DWORD | ByVal Long |
| 4 | 对象句柄:  所有句柄是与窗口相关联的 16 位的唯一整数值和通过值传递 | ByVal Integer |
| 5 | 字符串:  LPSTR , LPBYTE | ByVal String [[参考文档](http://support.microsoft.com/kb/118643/EN-US)] |
| 6 | 数值的指针 | Integer （不要用ByVal） |
| 7 | 结构  如果在 Visual Basic 用户定义的类型与所需的 DLL 结构相匹配，可以通过引用传递结构。  注意: 不能通过值传递结构。 | A struct identified in VB |
| 8 | 数组的指针 | 通过引用传递数组的第一个元素。 |
| 9 | 指向函数的指针 | Visual Basic 不支持回调函数，因此 Visual basic，不能使用具有指向函数的 DLL 函数。 |
| 10 | 空指针 | 如果 DLL 需要 Null 指针，将作为 （ByVal paramname As Any） 传递它。您可以使用 0 和 paramname 时调用该 DLL 中的值。 |

## Windows用来定位DLL的搜索路径

1、当前进程的可执行模块所在的目录。

2、当前目录。

3、Windows 系统目录。 （GetSystemDirectory 函数检索此目录的路径）

4、Windows 目录。 （GetWindowsDirectory 函数检索此目录的路径）

5、PATH 环境变量中列出的目录（未使用 LIBPATH 环境变量）。

# 创建DLL

## 初始化DLL

DLL 通常具有在 DLL 加载时必须执行的初始化代码（如分配内存）。在Win32 DLL类型中，你可以在被称为DllMain的函数中添加初始化代码和终止代码。

在 Win32 中，所有 DLL 都可能包含一个可选入口点函数（通常称为 DllMain），初始化和终止时都要调用此函数。 这使您有机会在需要时分配或释放其他资源。 Windows 在四种情况下调用入口点函数：进程附加、进程分离、线程附加和线程分离。

C 运行库提供了一个名为 \_DllMainCRTStartup 的入口点函数，并调用 DllMain。 根据 DLL 类型的不同，应在源代码中包含一个名为 DllMain 的函数，或应用 MFC 库中提供的 DllMain。

DllMain大概的样子：

BOOL APIENTRY DllMain(HANDLE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved)

{

switch( ul\_reason\_for\_call ) {

case DLL\_PROCESS\_ATTACH:

...

case DLL\_THREAD\_ATTACH:

...

case DLL\_THREAD\_DETACH:

...

case DLL\_PROCESS\_DETACH:

...

}

return TRUE;

}

注意：DllEntryPoint 的 Windows SDK 文档指出，必须在链接器命令行上用 /ENTRY 选项指定入口点函数的实际名称。 使用 Visual C++ 时，如果入口点函数的名称为 DllMain，则无需使用 /ENTRY 选项。 实际上，如果使用了 /ENTRY 选项，但没有将入口点函数命名为 DllMain，则 C 运行库将不会正确初始化。

## 如何使用C++语言导出DLL

## 使用其它语言导出DLL

## 创建纯资源DLL

# 