# 动态链接库总结

## 一、动态链接库的分类

微软的VC++支持3种DLL，分别是**非MFC DLL（WIN32 DLL），MFC规则DLL，MFC扩展DLL**。

非MFC DLL不采用MFC类结构，其导出函数为标准的C接口，能被非MFC或MFC编写的应用程序所调用。如果建立的DLL不需要使用MFC，那么应该建立非MFC DLL，因为使用MFC会增大用户库的大小，从而浪费用户磁盘和内存空间。

MFC规则DLL采用MFC类库编写，其源文件中包含一个继承自CWinApp的类，但是没有消息循环，它又可细分为静态链接到MFC库的规则DLL和动态链接到MFC库的规则DLL。MFC规则DLL能够导出C风格的函数，一般情况下不能导出MFC类及其派生类、成员函数和重载函数等。

MFC扩展DLL采用MFC的动态链接版本创建，它只能被用于MFC类库所编写的应用程序调用。MFC扩展DLL支持C++接口，即该DLL可以导出定义在其内的类，应用程序可以构造这些类对象，也可以从这些类派生。

从下表我们可以看出三种DLL对DllMain入口函数的不同处理方式：

|  |  |
| --- | --- |
| DLL类型 | 入口函数 |
| 非MFC DLL | 编程者提供DllMain函数 |
| MFC规则DLL | CWinApp对象的InitInstance和ExitInstance |
| MFC扩展DLL | MFC DLL向导生成DllMain函数 |

## 二、WIN32 DLL的创建与使用

### 2.1 WIN 32导出函数

①新建Win32 DLL工程，已自动生成cpp文件，但没有h文件，添加h头文件，在h头文件中添加：

#include "stdafx.h"

//导出函数

方法一：

extern "C" float \_declspec(dllexport)calculate(float x,float y,int type);

方法二：

定义DEF文件，并在DEF文件中输入以下代码：

LIBRARY "WinDLL"

EXPORTS

calculate @1

**不需要写头文件**

②在cpp文件中正常实现函数：

#include "StdAfx.h"

#include "WinDLL.h"

float calculate(float x,float y,int type)

{

……

}

③DLL的使用：

1、新建工程，将编译成功的DLL文件复制到工程目录下。

2、在cpp文件中设置lib文件路径，并导出函数：

方法一：

#pragma comment(lib,"G:\\其它\\VC驿站\\WinDLL\\Debug\\WinDLL.lib")

extern "C" float \_declspec(dllimport)calculate(float x,float y,int type);

方法二：

#pragma comment(lib,"G:\\其它\\VC驿站\\WinDLL\\Debug\\WinDLL.lib")

extern float calculate(float x,float y,int type);

注意这里不需要：“extern "C" float \_declspec(dllimport)”

这样，就可以在该工程中使用DLL中的函数calculate。“#pragma comment……”这句代码可以免去在在工程属性中设置附加依赖项。

### 2.2 WIN 32导出类

①在h文件中声明类，格式类似于：

class \_declspec(dllexport) 类名

{

……

}

注意，这里没有extern “C”

然后在cpp文件中正常实现类。

类的使用

②新建工程，并复制DLL工程中的h文件到到新工程的h文件中，把dllexport换成dllimport。即:

class \_declspec(dllimport) 类名

{

……

}

在文件顶端指定lib文件位置：

#pragma comment(lib,"G:\\其它\\VC驿站\\WinDLL\\Debug\\WinDLL.lib")

实际上，无论是导出类还是导出函数，如果是\_declspec这种形式导出的，都可以把DLL工程中的头文件复制过去，然后把dllexport换成dllimport，再指定lib文件位置后便可正常使用。

### 2.3 实例

1. 以下给出一个实例，实例中既导出了一个函数，也导出了一个类：

DLL工程：

CPP文件：

// WinDLL.cpp : 定义DLL 应用程序的导出函数

#include "StdAfx.h"

#include "WinDLL.h"

//导出函数

float calculate(float x,float y,int type)

{

switch(type)

{

头文件：

#include "stdafx.h"

//导出函数

extern "C" float \_declspec(dllexport)calculate(float x,float y,int type);

//导出类

class \_declspec(dllexport) CColumn

{

public:

CColumn();

CColumn(double r,double h);

double GetArea();

double GetVolum();

double GetRadial();

double GetHeight();

void SetRadial(double r);

void SetHeight(double h);

protected:

private:

double radial;

double height;

};

case 1:

return x+y;

break;

case 2:

return x-y;

break;

case 3:

return x\*y;

break;

case 4:

if (y!=0)

return x/y;

else

return 0;

break;

}

}

//类实现

CColumn::CColumn()

{

radial=0;

height=0;

}

CColumn::CColumn(double r,double h)

{

radial=r;

height=h;

}

double CColumn::GetArea()

{

return radial\*radial\*3.1415926;

}

double CColumn::GetVolum()

{

return radial\*radial\*3.1415926\*height;

}

void CColumn::SetRadial(double r)

{

radial=r;

}

void CColumn::SetHeight(double h)

{

height=h;

}

double CColumn::GetRadial()

{

return radial;

}

double CColumn::GetHeight()

{

return height;

｝

double CColumn::GetRadial()

{

return radial;

}

double CColumn::GetHeight()

{

return height;

}

DLL测试工程：

H文件：

#include "stdafx.h"

//导出函数

#pragma comment(lib,"G:\\其它\\VC驿站\\WinDLL\\Debug\\WinDLL.lib")

extern "C" float \_declspec(dllimport)calculate(float x,float y,int type);

//导出类

class \_declspec(dllimport) CColumn

{

public:

CColumn();

CColumn(double r,double h);

double GetArea();

double GetVolum();

double GetRadial();

double GetHeight();

void SetRadial(double r);

void SetHeight(double h);

protected:

private:

double radial;

double height;

};

CPP文件：

#include "stdafx.h"

#include <iostream>

#include "WinDLLTest.h"

using namespace std;

void test1();

void test2();

int \_tmain(int argc, \_TCHAR\* argv[])

{

int choice;

bool flag=true;

while (flag)

{

cout<<"请选择将要进行的实验：导出函数的调用（1），导出类的调用（2），退出(0):";

cin>>choice;

switch (choice)

{

case 1:

test1();

break;

case 2:

test2();

break;

default:

flag=false;

break;

}

}

return 0;

}

void test1()

{

int type;

float a,b;

while(1)

{

type=10;

cout<<"请选择计算类型：+(1),-(2),\*(3),/(4),退出(0)"<<endl;

while(type!=0&&type!=1&&type!=2&&type!=3&&type!=4)

cin>>type;

if (type==0)

break;

cout<<"请输入第一个数值：";

cin>>a;

cout<<"请输入第二个数值：";

cin>>b;

cout<<"运算结果："<<calculate(a,b,type)<<endl;

}

}

void test2()

{

int c=0;

bool flag=true;

CColumn co(0,0);

while (flag)

{

cout<<"请选择：创建一个圆柱体(1),退出(0):";

cin>>c;

switch(c)

{

case 1:

double r,h;

cout<<"请输入圆柱半径:r=";

cin>>r;

co.SetRadial(r);

cout<<"请输入圆柱高:h=";

cin>>h;

co.SetHeight(h);

cout<<"圆柱面积："<<co.GetArea()<<endl;

cout<<"圆柱体积："<<co.GetVolum()<<endl;

break;

default:

flag=false;

}

}

}

## 三、MFC规则DLL的创建与使用

MFC规则DLL的概念体现在两方面，一是它是MFC的，意味着DLL内部可以使用MFC，二是它是规则的，意味着它不同于MFC扩展DLL，虽然在其内部可以使用MFC，但是其与应用程序的接口不能是MFC。MFC规则DLL又分为静态链接到MFC的规则DLL和动态链接到MFC的规则DLL。

静态链接到MFC的规则DLL与MFC库静态链接，将MFC库的代码直接生成在DLL文件中。在调用这种DLL的接口时，MFC使用DLL的资源。因此，在静态链接到MFC的规则DLL中不需要进行模块的切换。使用这种方法生成的规则DLL程序较大。

动态链接到MFC的规则DLL可以和使用它的可执行文件现时动态链接到MFC DLL和任何MFC扩展DLL。当DLL和应用程序中存在相同ID的资源时，系统可能不能获得正确的资源，因此，对于共享MFC DLL的规则DLL，我们必须进行模块切换以使得MFC能够找到正确的资源模板。

要点：

1）、在DLL中，使用类似于以下函数的申明方式：

extern "C" \_declspec(dllexport) void ShowDlg(void)；

在客户端程序中，使用类似于以下函数的声明方式：

extern "C" \_declspec(dllimport) void ShowDlg(void);

2）、隐式链接时：

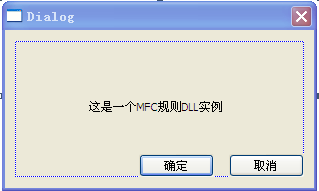
#pragma comment(lib,"RegularMFCDLL");

3）、在DLL函数内部开始时，使用如下语句：

AFX\_MANAGE\_STATE(AfxGetStaticModuleState());

实例：

3.1 创建一个MFC规则DLL工程，添加一个对话框资源，生成MFC类CRegMFCDlg。



3.2 添加一个函数，以显示该对话框：

extern "C" \_declspec(dllexport) void ShowDlg()

{

AFX\_MANAGE\_STATE(AfxGetStaticModuleState());

CRegMFCDlg dlg;

dlg.DoModal();

}

3.3 编译，生成LIB和DLL文件。

3.4 创建一个测试工程，为其添加菜单项，添加其响应函数，代码如下：

void CLoadRegularMFCDLLView::OnShowDlg()

{

ShowDlg();

}

3.5 在CPP文件中添加如下代码：

#pragma comment(lib,"RegularMFCDLL");

extern "C" \_declspec(dllimport) void ShowDlg(void);

编译运行即可。

由以上实例可见，MFC规则DLL在其内部可以使用MFC类，而导出的是函数，在客户端中调用导出的函数，可以访问DLL中的MFC类。

## 四、MFC扩展DLL的创建与使用

要点：

1）、导出类：在扩展DLL工程中添加资源，生成MFC类，并把Resource.h文件中的资源ID定义拷贝到类头文件中。导出变量和函数：

extern "C"

{

int AFX\_EXT\_DATA 变量名;

int AFX\_EXT\_API 函数名(参数);

}

2）、将DLL、LIB及类头文件拷贝到客户端工程目录下，并将头文件加入到工程中。

3）、静态链接DLL的方法：

#pragma comment(lib,"DLGDLL.lib")

实例：

4.1 创建一个MFC扩展DLL工程，并添加一个对话框资源，ID设置为IDD\_HELLOWWORLD\_DLG，添加自己须要的控件。



4.2 为该对话框资源生成一个对话框类，类名为CHelloDlg。将Resource.h中IDD\_HELLOWORLD及其它控件的ID的定义拷贝到H文件头上，并在类名前添加AFX\_EXT\_CLASS，得到类头文件如下：

#pragma once

#define IDC\_HELLOW\_TEXT 2000

#define IDD\_HELLOWORLD\_DLG 2000

class AFX\_EXT\_CLASS CHelloDlg : public CDialog

{

DECLARE\_DYNAMIC(CHelloDlg)

public:

CHelloDlg(CWnd\* pParent = NULL);

virtual ~CHelloDlg();

enum { IDD = IDD\_HELLOWORLD\_DLG };

protected:

virtual void DoDataExchange(CDataExchange\* pDX);

DECLARE\_MESSAGE\_MAP()

public:

CString m\_HelloText;

};

4.3 编译生成DLL和LIB文件。

4.4 新建一基于对话框的测试工程，取名为LoadDllDlg，将上述编译得到的HelloDlg.h文件、DLL文件、LIB文件拷贝到…LoadDllDlg\LoadDllDlg目录下。



4.5 将HelloDlg.h文件添加到工程中，这样新工程中会增加一个CHelloDlg的类。在主对话框类头文件中加入以下语句：

#include "HelloDlg.h"

#pragma comment(lib,"DLGDLL.lib")

4.6 设计主对话框，并为相应按钮添加响应函数：

void CLoadDllDlgDlg::OnBnClickedButton1()

{

CHelloDlg dlg;

dlg.DoModal();

}



4.7 编译运行程序，点击“调用DLL”按钮，弹出以下对话框：



可见，测试程序成功使用了DLL中CHelloDlg类。

注意：虽然MFC扩展DLL可以使用显示加载的方式，但不推荐。显示加载可以使用AfxLoadLibrary函数，卸载可以使用AfxFreeLibrary函数。