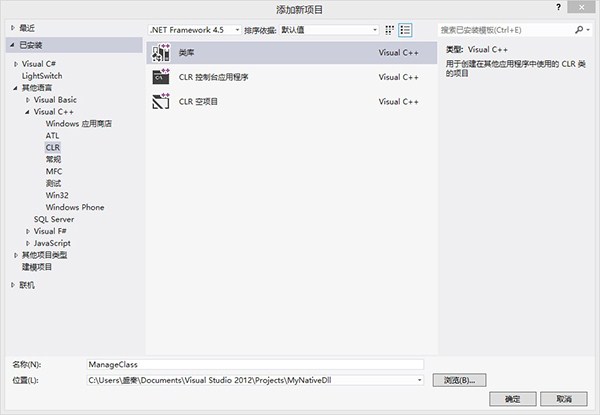
第二种方法就是使用Visual Studio的托管C++作为中介（wrapper），实现C#对原生C++的调用，因为托管C++能够很方便地调用原生C++中的各种h头文件、lib库等，类引用也非常方便，而且托管C++编译生成的DLL文件可以在C#工程中像C# DLL一样直接使用

Native C++   ==>> Managed C++ Wrapper ==>> C# GUI

具体方法为：

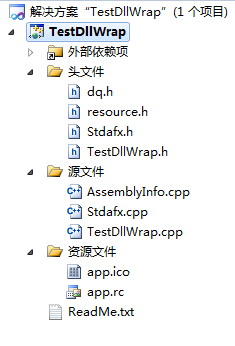
**一、建立CLR类库工程（即托管的C++类库工程）**

一幅图能说明的问题我就不多说，反正建立一个CLR类库工程，其命名暂定为TestDllWrap（图中名称为ManageClass，注意），这是工程名，请勿混淆，如下图，没什么注意事项可言的。



**二、在CLR类库中引用原生C++的内容**

我手上有一个用非托管C++写的dq.h，它本身是属于另外一个非托管C++工程，现在我直接将这个类文件拷贝到本工程的目录下去，简单起见，这个类我内联在一个头文件里，如果是其他比较大型的类，必要将NativeClass.h里#include到的其他文件也一并拷贝到本新建工程目录下，并引用其对应的lib等文件，然后将这些文件添加到VS的资源管理器下，如下图所示



上图中，除dq.h文件是我添加进去的，其他都是工程自带的东西，其中TestDllWrap.h及TestDllWrap.cpp是要生成dll所动用到的东西，暂时先不管，我们看一下dq.h里的内容:

#include &lt;blitz/array.h&gt;

#ifdef DQ\_EXPORTS

#define DQ\_API \_\_declspec(dllexport)

#else

#define DQ\_API \_\_declspec(dllimport)

#endif

class DQ\_API DataGridSet

{

public:

blitz::Array&lt;double, 2&gt; dataGrid;

double vmax;

double vmin;

};

DQ\_API void OpenDQ();

DQ\_API void \* StartCalc(int id, double q, int l, int d, int u, int s);

DQ\_API bool GetData(void \* pHnd, int t, DataGridSet &set);

DQ\_API void EndCalc(void \* pHnd);

DQ\_API void CloseDQ();

可以看到其中还有一个DataGridSet类定义，中间还用到一个外部库blitz++的array类，如果用简单的DllImport方法是很难将其导入C#中的

**三、封装成托管C++的内容**

引入该dq.h文件后，并将需要的lib等文件引用到该CLR项目中，编译无问题后，即可开始在托管C++代码中添加引用，代码如下：

#pragma once

using namespace System;

#include "dq.h";

namespace TestDllWrap {

public ref class dq\_Wrap

{

// TODO: 在此处添加此类的方法。

private:

DataGridSet \*datagridSet ;

double vmax,vmin;

int rows,cols;

double\*\* vals;

public:

dq\_Wrap(void)

{}

void OpenDQ\_wrap()

{

OpenDQ();

}

IntPtr StartCalc\_wrap(int id, double q, int l, int d, int u, int s)

{

return (IntPtr)StartCalc(id, q, l, d, u, s);

}

bool GetData\_wrap(IntPtr pHnd, int t)

{

DataGridSet set ;

bool tmpres = GetData( (void \*)pHnd, t, set);

vmax= set.vmax;

vmin = set.vmin;

rows = set.dataGrid.length(0);

cols = set.dataGrid.length(1);

vals = new double\*[rows];

for(int i=0;i&lt;rows;i++)

{

vals[i]=new double[cols];

for(int j=0;j&lt;cols;j++)

{

vals[i][j] = set.dataGrid(i,j);

}

}

return tmpres;

}

void EndCalc\_wrap(IntPtr pHnd)

{

EndCalc( (void \*)pHnd);

}

void CloseDQ\_wrap()

{

CloseDQ();

}

int GetSetRows()

{

return rows;

}

int GetSetCols()

{

return cols;

}

double GetSetVal(int i,int j)

{

return vals[i][j] ;

}

double GetSetMin()

{

return vmin;

}

double GetSetMax()

{

return vmax;

}

};

}

像上面这样，将原生C++中的方法重新进行了包装，最重要的是，将DataGridSet 这个不好处理的C++类以及其内部的一些值，全部使用C#中便于操作的中间类型及变量转存出来，包装成便于调用的函数。

注意这个工程一定要用RELEASE版本输出DLL

**四、生成托管dll**

保证无编译错误后，直接编译，即会在Release目录下生成dll及pdf文件

**五、在C#中引用**

将dll及pdb文件拷贝到C#的Debug目录后（pdb文件也可以不拷贝，不影响使用，拷贝过去后支持在C#中对C++代码进行调试），在解决方案中像引用C#类库一样引用该Dll文件，即可使用其中的函数，代码如下：

TestDllWrap.dq\_Wrap tmpdq = new TestDllWrap.dq\_Wrap();

tmpdq.OpenDQ\_wrap();

IntPtr resPtr = tmpdq.StartCalc\_wrap(0, 100000000.0, 4, 4, 6, 6);

bool res2 = tmpdq.GetData\_wrap(resPtr, 1);

int rows = tmpdq.GetSetRows();

int cols = tmpdq.GetSetCols();

double vmax = tmpdq.GetSetMax();

double vmin = tmpdq.GetSetMin();

double tmpVal = tmpdq.GetSetVal(10, 10);

tmpdq.EndCalc\_wrap(resPtr);

tmpdq.CloseDQ\_wrap();

代码见附件

托管C++[C:\Users\wyz\Documents\My Knowledge\temp\2f857635-1e3a-41fa-b14f-032314a9b2e5_4_files\2b359ff3-c7c1-482f-987b-ae83f4626757.png](wiz://open_attachment/?guid=a25c17a0-87fe-4065-8aa1-d9e0402877f2)

C#[C:\Users\wyz\Documents\My Knowledge\temp\2f857635-1e3a-41fa-b14f-032314a9b2e5_4_files\cf45a157-8d5d-4743-80fe-5c320082bc42.png](wiz://open_attachment/?guid=a8da6644-076e-414f-8e87-214f01e69461)