VC ++ 6.0 环境中 LIB 与 DLL 文件的使用

作者：合肥工业大学 杨铸
网址：www.200yi.com 电邮：fairyfar@msn.com
2008.01 春节前夕
Scripted by FairyFar.

1. 关于本文

首先声明，这不是教程，也不是论文，仅当工作和学习札记。

作为程序员，我们经常要与别人合作完成编程工作。现代软件工程规模越来越大，一个程序可能被分配给不同程序员，不同部门，甚至不同单位合作完成。这与一个人独立完成一个程序的工作流程大不相同，“因为你不是一个人”在编程——你的程序最终要和其他人的代码集成到一个系统中。

想一想吧，作为程序员，最痛苦的经历不是写程序，也不是阅读程序，最痛楚莫过于深陷于一大堆没有秩序的代码“泥沼”之中。

记住：我们是程序员，我们为程序员写程序。像热爱我们的生活一样去打理我们的程序吧——I love this program!

2. 案例分析

下面是一系列案例。需要说明的是，这些案例不具有实用性，只是为了能够说明和简化问题而设计的，实践中的软件工程要复杂得多。编程环境：VC ++ 6.0 Enterprise Edition 英文版。

2.1 最初的工程

案例 1 说明：在下面的 Test_Project_1 工程中，仅有一个源代码文件 Test_Main.cpp。该工程完成以下功能：

max 子程序首先显示版权信息，然后比较两个整型数的大小，并返回较大者，同时还要记录和显示 max 函数被调用了多少次。在 main 主程序中，调用 max 函数 2 次。

这是我们最初的工程，没有考虑软件集成问题。建立案例 1 步骤：

1）File → New → Projects → Win32 Console Application。在 Location 中指定工程保存位置，在 Project name 中输入工程名 Test_Project_1，建立“An empty project”。

2）File → New → Files → C++ Source File。在 File 中输入文件名 Test_Main，建立 cpp 文件，Test_Main.cpp 文件内容见表 2.1。

表 2.1 Test_Main.cpp 文件内容

```
#include <iostream.h>

//定义宏。版权信息。
#define COPYRIGHT "Scripted by FairyFar."

//全局变量。记录 max 函数被调用的次数。
int GLOBAL_NUM = 0;

/*********************************************************
函数功能：显示版权信息，返回两个整型变量中较大者。
入口参数：x，待比较大小的变量 1；y，待比较大小的变量 2。
返 回 值：两个整型变量中较大者。
```

```
*********************************************************/
int max(int x, int y)
{
    cout<<COPYRIGHT<<endl;      //显示版权信息
    ++GLOBAL_NUM;               //函数被调用次数加 1
    cout<<"Call max function "<<GLOBAL_NUM<<" time(s)."<<endl;
    return x>y ? x : y;
}

/*********************************************************
主程序
*********************************************************/
void main(void)
{
    cout<<COPYRIGHT<<endl;      //显示版权信息
    cout<<max(19, 49)<<endl;    //第 1 次调用 max 函数
    cout<<max(200, 8)<<endl;    //第 2 次调用 max 函数
}
```

这只能算是教科书上的例题，没有体现软件工程的设计思想。下面的文字，我们将以此案例为基础，要求程序功能不变，重新设计工程结构。以满足模块化程序设计要求。

2.2 重新设计工程结构

案例 2 说明：在下面的 Test_Project_2 工程中，将建立多个文档。咱们先来建立工程，稍后再解释。

（1）File → New → Projects → Win32 Console Application。在 Location 中指定工程保存位置，在 Project name 中输入工程名 Test_Project_2，建立“An empty project”。

（2）File → New → Files → C++ Source File。在 File 中输入文件名 Test_Main，建立.cpp 程序源文件，Test_Main.cpp 文件内容见表 2.2。

表 2.2 Test_Main.cpp 文件内容

```
#include <iostream.h>
#include "Macro_Define.h"       //宏定义头文件
#include "Global_Value.h"       //全局变量声明和定义头文件
#include "Max_Func.h"           //max 函数声明头文件

/*********************************************************
主程序
*********************************************************/
void main(void)
{
    cout<<COPYRIGHT<<endl;      //显示版权信息
    cout<<max(19, 49)<<endl;    //第 1 次调用 max 函数
    cout<<max(200, 8)<<endl;    //第 2 次调用 max 函数
}
```

（3）File → New → Files → C/C++ Header File。在 File 中输入文件名 Max_Func，建立.h 头文件，Max_Func.h 文件内容见表 2.3。

表 2.3 Max_Func.h 文件内容

```
/*********************************************************
函数功能：显示版权信息，返回两个整型变量中较大者。
入口参数：x，待比较大小的变量 1；y，待比较大小的变量 2。
返 回 值：两个整型变量中较大者。
*********************************************************/
#ifndef Max_Func_H_98HUT67RCZOOGHM4   //避免该文件被重复 include
#define Max_Func_H_98HUT67RCZOOGHM4
```

```
int max(int x, int y);                  //max 函数声明

#endif
```

（4）File → New → Files → C++ Source File。在 File 中输入文件名 Max_Func，建立.cpp 程序源文件，Max_Func.cpp 文件内容见表 2.4。

表 2.4 Max_Func.cpp 文件内容

```
#include <iostream.h>
#include "Macro_Define.h"          //宏定义头文件
#include "Extern_Value.h"          //引用全局变量的头文件
#include "Max_Func.h"              //max 函数声明头文件

int max(int x, int y)
{
    cout<<COPYRIGHT<<endl;         //显示版权信息
    ++GLOBAL_NUM;                  //函数被调用次数加 1
    cout<<"Call max function "<<GLOBAL_NUM<<" time(s)."<<endl;
    return x>y ? x : y;
}
```

（5）File → New → Files → C/C++ Header File。在 File 中输入文件名 Macro_Define，建立.h 头文件，Macro_Define.h 文件内容见表 2.5。

表 2.5 Macro_Define.h 文件内容

```
//宏定义。版权信息。
#ifndef Macro_Define_H_IU89T78CVC23ZQPO
#define Macro_Define_H_IU89T78CVC23ZQPO

#define COPYRIGHT "Scripted by FairyFar."

#endif
```

（6）File → New → Files → C/C++ Header File。在 File 中输入文件名 Global_Value，建立.h 头文件，Global_Value.h 文件内容见表 2.6。

表 2.6 Global_Value.h 文件内容

```
//全局变量。记录 max 函数被调用的次数。
#ifndef Global_Value_H_PPP78RA7893H75DZ
#define Global_Value_H_PPP78RA7893H75DZ

int GLOBAL_NUM = 0;

#endif
```

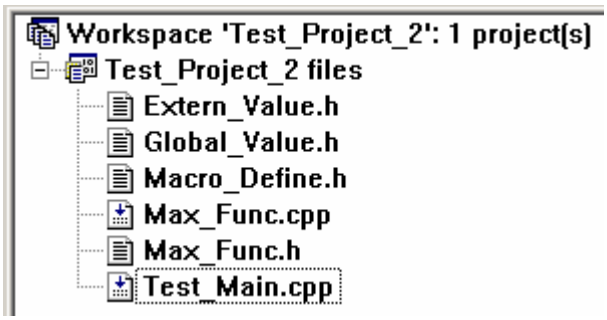
（7）File → New → Files → C/C++ Header File。在 File 中输入文件名 Extern_Value，建立.h 头文件，Extern_Value.h 文件内容见表 2.7。

表 2.7 Extern_Value.h 文件内容

```
//引用全局变量
#ifndef Extern_Value_H_2PPI89RQ12M9VB89
#define Extern_Value_H_2PPI89RQ12M9VB89

extern int GLOBAL_NUM;

#endif
```

经过以上步骤，我们建立的 Test_Project_2 工程包含了多达 6 个程序文件（见图 2.1）。是不是有些罗嗦？那么，有必要给个合理的解释。

图 2.1 工程 Test_Project_2 文件结构



观察以上步骤（2）～（7）建立的 6 个文件，你会发现它们是相对独立的，按代码功能各负其责。来看看这几个文件的用途：

Test_Main.cpp：主程序。实际应用中，它通常是其它工程与 Test_Project_2 工程的接口，即在此文件中调用 Test_Project_2 对应的模块。

Max_Func.h：只是声明 max 函数，函数具体实现放在 Max_Func.cpp 中。但是应该给出详细的注释，尤其是接口参数与调用方法。

Max_Func.cpp：max 函数具体实现。

Macro_Define.h：宏定义文件。宏与全局变量实现机制迥异，因此，我们单独设立此文件。

Global_Value.h：全局变量声明与定义。

Extern_Value.h：引用全局变量文件，只有那些使用了全局变量的文件才需要 include 此文件。

2.3 链接库

链接库分静态链接库和动态链接库两种：

LIB 是静态链接库（对应.lib 文件），其用于程序编连（编译和连接）时静态链接。也就是说，使用静态链接库的程序，在程序编连时，.lib 文件中的函数就已经被连接到了最终的可执行文件中。

DLL（Dynamic Linking Library）是动态链接库（对应.dll 文件），它是程序运行期必需的。程序编连时，.dll 文件中存放实际的函数代码，它并不被插入到最终的可执行文件的。可执行文件中只是一些地址调用信息。

使用链接库有什么作用？

程序开发期，有利于程序模块化设计。各模块开发单位各自维护其源代码，某个模块修改时，只需重新提交其链接库文件和相应头文件即可，代码实现部分是怎么修改的，其它合作单位无需了解。这也达到了知识产权保护的目的。

使用了动态链接库的程序，有利于软件发布后的维护与升级，大部分的维护与升级工作只需要更新相应得.dll 文件即可。

如果要使用动态链接库技术，就得了解.def 文件。

.def 是模块定义文件，供生成动态链接库时使用。动态链接库中定义有两种函数：导出函数（Export Function）和内部函数（Internal Function）。导出函数可以被其它模块调用，内部函数只能在库内部使用。我们可以用 C++定制动态库文件，需要编写的是包含导出函数表的.def 模块定义文件和实现导出函数功能的 C++文件。

以下简要介绍.def 文件的结构：

（1）第一条语句必须是 LIBRARY，指出 DLL 名字。

（2）EXPORTS 语句列出被导出函数的名字。

（3）DESCRIPTION 语句描述 DLL 的用途。

（4）注释行符“;”。

2.4 LIB 静态链接库的使用

还是以案例来说明。案例 3：以案例 2 为基础。假定单位 A 负责开发模块 A，单位 B 负责开发模块 B。模块 A 需要调用模块 B 的功能，模块 A 相当于案例 2 的 Test_Main.cpp，模块 B 负责实现 max 函数功能。两个单位只是合作关系，也就是说项目组 B 不想也没有义务把自己编写的 max 函数源代码提供给项目组 B。LIB 可以解决这个矛盾。单位 B 只需要将 max 使用的头文件和.lib 库文件提供给单位 A 即可。

首先来看看单位 B 的工作。单位 B 建立工程 Max_Func：

（B1）File → New → Projects → Win32 Console Static Library。在 Location 中指定工程保存位置，在 Project name 中输入工程名 Max_Func，建立一个空工程。

（B2）File → New → Files → C/C++ Header File。在 File 中输入文件名 Max_Func，建立.h 头文件，Max_Func.h 文件内容见表 2.8。

表 2.8 Max_Func.h 文件内容

```
/*************************************************************
函数功能：显示版权信息，返回两个整型变量中较大者。
入口参数：x，待比较大小的变量 1；y，待比较大小的变量 2。
返 回 值：两个整型变量中较大者。
*************************************************************/
#ifndef Max_Func_H_AZ00899NNZMF7CXD
#define Max_Func_H_AZ00899NNZMF7CXD

int max(int x, int y);            //max 函数声明

#endif
```

（B3）File → New → Files → C++ Source File。在 File 中输入文件名 Max_Func，建立.cpp 程序源文件，Max_Func.cpp 文件内容见表 2.9。

表 2.4 Max_Func.cpp 文件内容

```
#include <iostream.h>
#include "Macro_Define.h"         //宏定义头文件
#include "Global_Value.h"         //全局变量声明和定义头文件
#include "Extern_Value.h"         //引用全局变量的头文件
#include "Max_Func.h"             //max 函数声明头文件

int max(int x, int y)
{
    cout<<COPYRIGHT<<endl;        //显示版权信息
    ++GLOBAL_NUM;                 //函数被调用次数加 1
    cout<<"Call max function "<<GLOBAL_NUM<<" time(s)."<<endl;
    return x>y ? x : y;
}
```

（B4）File → New → Files → C/C++ Header File。在 File 中输入文件名 Macro_Define，建立.h 头文件，Macro_Define.h 文件内容见表 2.10。

表 2.10 Macro_Define.h 文件内容

```
//定义宏。版权信息。
#ifndef Macro_Define_H_JK9AAYU34VCVCU19
#define Macro_Define_H_JK9AAYU34VCVCU19

#define COPYRIGHT "Scripted by FairyFar."

#endif
```

（B5）File → New → Files → C/C++ Header File。在 File 中输入文件名 Global_Value，建立.h 头文件，Global_Value.h 文件内容见表 2.11。

表 2.11 Global_Value.h 文件内容

```
//全局变量。记录 max 函数被调用的次数。
#ifndef Global_Value_H_4AQBBPII95XCOYKM
#define Global_Value_H_4AQBBPII95XCOYKM

int GLOBAL_NUM = 0;

#endif
```

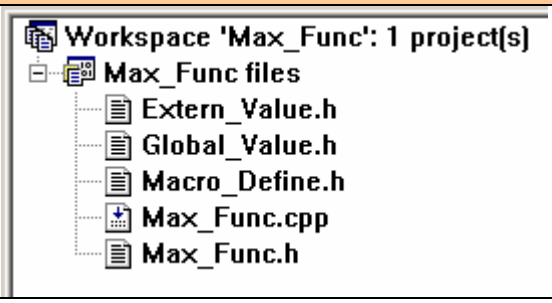
（B6）File → New → Files → C/C++ Header File。在 File 中输入文件名 Extern_Value，建立.h 头文件，Extern_Value.h 文件内容见表 2.12。

表 2.12 Extern_Value.h 文件内容

```
//引用全局变量
#ifndef Extern_Value_H_MZ999TY4ZXQM7IPL
#define Extern_Value_H_MZ999TY4ZXQM7IPL

extern int GLOBAL_NUM;

#endif
```

你可能已经注意到了，以上步骤（B2）～（B6）与 2.2 节步骤（3）～（7）相同，共建立 5 个文件（图 2.2 所示）。注意，因为全局变量是由各模块开发组统一命名的，因此 Macro_Define.h，Global_Value.h，Extern_Value.h 文件是各单位共有的。

图 2.2 工程 Max_Func 文件结构



编连 Max_Func 工程，我们得到了 LIB 文件：Max_Func.lib。

好了，单位 B 现在可以把以下文件发给单位 A 了：

Max_Func.h，Max_Func.lib。

接下来，我们到单位 A 去看看他们的工作：

（A1）首先建立工程 Test_Project_3。File → New → Projects → Win32 Console Application。在 Location 中指定工程保存位置，在 Project name 中输入工程名 Test_Project_3，建立“An empty project”。

（A2）将单位 B 发来的文件拷贝到 Test_Project_3 工程目录下，用“Add File to Project”菜单将这些文件加入到工程中。

（A3）File → New → Files → C++ Source File。在 File 中输入文件名 Test_Main，建立.cpp 程序源文件，Test_Main.cpp 文件内容见表 2.13。

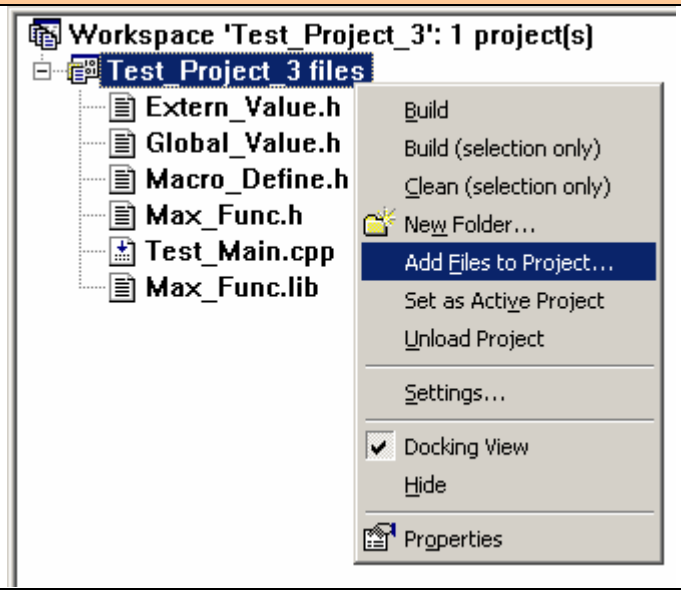
表 2.13 Test_Main.cpp 文件内容

```
#include <iostream.h>
#include "Macro_Define.h"        //宏定义头文件
#include "Global_Value.h"        //全局变量声明和定义头文件
#include "Max_Func.h"            //max 函数声明头文件

/*********************************************************
```

```
主程序
*******************************************************/
void main(void)
{
    cout<<COPYRIGHT<<endl;       //显示版权信息
    cout<<max(19, 49)<<endl;     //第 1 次调用 max 函数
    cout<<max(200, 8)<<endl;     //第 2 次调用 max 函数
}
```

图 2.3 所示，为 Test_Project_3 工程的文件结构。上面已经说过了，Macro_Define.h，Global_Value.h，Extern_Value.h 文件是各单位共有的。

图 2.3 工程 Test_Project_3 文件结构



如果单位 B 现在修改（譬如改进）了 max 函数，所要做的只是将新生成的.lib 文件发给单位 A。单位 A 无需知道修改的具体细节。

2.5 DLL 动态链接库的使用

2.3 节已经对 DLL 作了介绍，本节我们仍然通过案例来介绍 DLL 的使用方法。案例 4：以案例 2 为基础，建立工程 Test_Project_4。最终发布一个可执行文件 Test_Project_4.exe 和与其配套使用的动态链接库 Max_Func.dll，前者是主程序，运行期需要调用 max 函数，该函数指令在 Max_Func.dll 动态链接库中。我们将该案例拆分成两个子任务。

先创建动态链接库文件 Max_Func.dll 及其导出库文件 Max_Func.lib：

（1）File → New → Projects → Win32 Dynamic-Link Library。在 Location 中指定工程保存位置，在 Project name 中输入工程名 Max_Func，建立 “An empty DLL project”。

（2）～（6）同 2.3 节的步骤（3）～（7）。

编连 Test_Project_4 工程生成动态链接库文件 Max_Func.dll，该文件是供主程序运行期调用的。编连时还需要一个 DLL 的导出函数库文件 Max_Func.lib，因此我们需要编写包含导出函数表的.def 模块定义文件（参阅 2.3 节）。

（7）File → New → Files → Text File。在 File 中输入文件名 Max_Func.def，建立.def 模块定义文件，Max_Func.def 文件内容见表 2.14。

表 2.14 Max_Func.def 文件内容

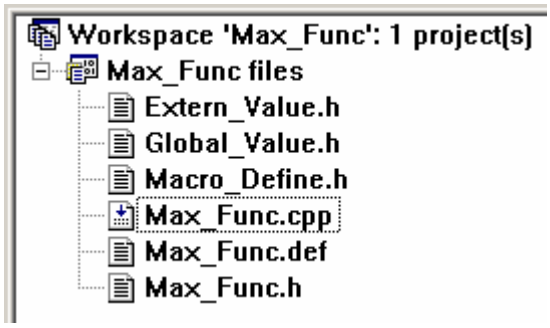
```
LIBRARY Max_Func

EXPORTS
max
```

编连 Max_Func 工程，生成我们需要的文件：Max_Func.dll，Max_Func.lib。顺便提一下，编连时，“output”窗口提示“Creating library Debug/Max_Func.lib and object Debug/Max_Func.exp”，在生成.lib 文件同时还产生了一个相应得.exp 文件。

.exp 是由 lib.exe 工具（参阅 2.6 节）从.def 文件生成的输出文件，其中包含了函数和数据项目的输出信息，link 工具将使用.exp 文件来创建动态链接库。

图 2.4 所示，为 Max_Func 工程的文件结构。

图 2.4 工程 Max_Func 文件结构



接下来的任务就是使用以上创建的文件啦。

（1）首先建立工程 Test_Project_4。File → New → Projects → Win32 Console Application。在 Location 中指定工程保存位置，在 Project name 中输入工程名 Test_Project_4。建立“An empty project”。

（2）将以下文件拷贝到 Test_Project_4 工程目录下：Max_Func.h、Max_Func.dll、Max_Func.lib、Global_Value.h、Extern_Value.h，用“Add File to Project”菜单将这些文件加入到工程中，Max_Func.dll 文件也可以不“Add”进来。

（3）File → New → Files → C++ Source File。在 File 中输入文件名 Test_Main，建立.cpp 程序源文件，Test_Main.cpp 文件内容见表 2.15。

表 2.15 Test_Main.cpp 文件内容

```
#include <iostream.h>
#include "Macro_Define.h"          //宏定义头文件
#include "Global_Value.h"          //全局变量声明和定义头文件
#include "Max_Func.h"              //max 函数声明头文件

/*********************************************************
主程序
*********************************************************/
void main(void)
{
    cout<<COPYRIGHT<<endl;         //显示版权信息
    cout<<max(19, 49)<<endl;       //第 1 次调用 max 函数
    cout<<max(200, 8)<<endl;       //第 2 次调用 max 函数
}
```

OK 了！编连 Test_Project_4 工程，直接运行，结果和以上的各个案例完全一样。实际上，本案例已经有了本质改变。来做个测试，我们将 Max_Func.dll 改个名字，譬如改成 Max_FuncX.dll，再运行本程序，提示图 2.5 所示的错误。这说明了说明？很显然，运行期 Test_Project_4.exe 程序的确依赖于 Max_Func.dll 的存在。

图 2.5 找不到 DLL 时的运行期错误

这样做，对于模块化程序设计和软件分包大有裨益。Max_Fun.dll 提供了相对独立的功能，当需要维护该功能模块时，所要做得只是重新发布该.dll 文件。

2.6 关于 dumpbin.exe 和 lib.exe 工具

2.5 节中，我们在 VC ++ 6.0 集成编程环境中从 Max_Func.dll 导出对应的 Max_Func.lib 文件。实际上 VC ++ 6.0 提供了两个命令行工具：dumpbin.exe 和 lib.exe。利用这两个工具我们也可以从.dll 手工导出其对应的.lib。如同 VC ++ 6.0 在编连程序时会自动调用 link.exe 一样，在案例 4 的第二个子任务中，VC ++ 6.0 自动调用了 lib.exe 命令行工具。

以案例 4 的第二个子任务为例，简单了解一下 dumpbin.exe 和 lib.exe 两个工具的使用方法：

（1）生成.def 文件。在命令行执行：

```
dumpbin /exports Max_Func.dll > Max_Func.def
```

编辑 Max_Func.def 文件，文件内容同表 2.14。

（2）生成.lib 文件。在命令行执行：

```
lib /def:Max_Func.def /machine:i386 /out:Max_Func.lib
```

行文止此，感谢你的阅读！