实 验 六　动态链接库的创建与使用

6.1　实验目的

学习和掌握动态链接库机制，学习它的的创建与使用方法。

6.2　动态链接库简介

比较大的应用程序都由很多模块组成，这些模块分别完成相对独立的功能，它们彼此协作来完成整个软件系统的工作。可能存在一些模块的功能较为通用，在构造其它软件系统时仍会被使用。在构造软件系统时，如果将所有模块的源代码都静态编译到整个应用程序 EXE 文件中，会产生一些问题：一个缺点是增加了应用程序的大小，它会占用更多的磁盘空间，程序运行时也会消耗较大的内存空间，造成系统资源的浪费；另一个缺点是，在编写大的 EXE 程序时，在每次修改重建时都必须调整编译所有源代码，增加了编译过程的复杂性，也不利于阶段性的单元测试。

动态链接库 (DLL) 意思为 Dynamic Link Library，这是 Windows 系统平台上提供的一种较有效的编程和运行机制，用户可以将独立的程序模块创建为较小的 DLL(DynamicLinkable Library) 文件，并可对它们单独编译和测试，DLL 就是一个包含可由多个程序同时使用的代码和数据的库。在运行时，当 EXE 程序确实要调用这些 DLL 模块的情况下，系统才会将它们装载到内存空间中。这种方式不仅减少了 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。DLL 实现了代码封装性，它的编制与具体的编程语言及编译器无关，不同编程语言生成的 DLL 函数可以互相调用。

Windows 操作系统平台将一些主要的系统功能以 DLL 模块的形式实现，并且可供用户调用其中的函数，也就是人们常说的 API 接口，因此 windows 具有很好的可扩展特性。程序员要用到的大量基础工作都可以调用 API 来实现的，比如文件拷贝，程序不必实际操作文件的每个字节，只要向 Windows 平台发送一个 Copy 命令就可以了。表6-1列出了 windows 系统常用的一些动态链接库文件及它们的作用：

6.2.1　动态链接库运行机制

DLL 是一种磁盘文件，系统中非常重要的如 KERNEL32.DLL、USER32.DLL 和 GDI32.DLL 就是动态链接文件，各种驱动程序文件如 KEYBOARD.DRV、SYSTEM.DRV 和 MOUSE.DRV 和视讯及打印机驱动程序也都是动态链接库，还有以.FON、.SYS 和许多以.EXE 为扩展名的系统文件都可以是 DLL。这些动态链接库能被所有 Windows 应用程序使用。

表 6-1 系统 DLL 文件说明

文件名	功能
KERNEL32.DLL	低级内核函数，用于内存管理、任务管理、资源控制等
USER32.DLL	与 windows 管理有关的函数，消息、菜单、光标、计时器、通信，钩子等
GDI32.DLL	图形设备接口库。于设备输出有关的函数：大多数绘图、显示场景、图元文件、坐标及其字体函数
ODBC32.DLL	实现 ODBC 功能
Ws2_32.dll	实现 socket 通信功能

动态链接文件标准扩展名是.DLL，以.DLL 扩展名的动态链接库能被 Windows 自动加载；如果文件有其它扩展名，则程序要使用 LoadLibrary 或者 LoadLibraryEx 函数加载该模块。一个 DLL 在内存中只有一个实例，系统为每个 DLL 维护一个线程级的引用计数，一旦一个线程载入了该 DLL，引用计数将会加 1。而程序终止或者引用计数变为 0（仅指运行时动态链接库），DLL 就会释放占用程序的虚地址空间。DLL 文件中的内容由全局数据、服务函数和资源组成，在运行时被系统加载到进程的虚拟空间中，成为调用进程的一部分。Windows 将提供内部的地址映的工作，例如一个 DLL 文件被加载后在物理内存中只占一个固定区域，有多个进程使用同一个 DLL 文件，Windows 将这个 DLL 的内存地址空间通过地址映射后提供给各个进程，进程代码地址与 DLL 映射后地址构成的是进程的虑地址空间，进程在自己的虚地址空间中好像是自己独自在使用这个 DLL 文件，使用 DLL 中的函数与程序自身的函数没有区别。DLL 可以有自己的数据段，但没有自己的堆栈，使用与调用它的应用程序相同的堆栈模式，它在运行时需要分配的内存是属于调用它的进程的，不同程序即使调用相同函数所分配的内存互相也不会影响，DLL 函数中的代码所创建的任何对象（包括变量）都归调用它的线程或进程所有。在 Windows 环境中，每个进程都复制了自己的读/写全局变量，如果想要与其它进程共享内存，可以使用内存映射文件或者声明一个共享数据段。

程序如果依赖其它 DLL 模块，则 DLL 文件必须存在，在没有设定文件完整路径情况下，Windows 将遵循下面的搜索顺序来定位 DLL：

1. 包含 EXE 文件的目录
2. 进程的当前工作目录
3. Windows 系统目录
4. Windows 目录
5. 列在 Path 环境变量中的一系列目录，ActiveX 控件或 COM 组件的 DLL 是需要注册的，Windows 普通 DLL 不用注册。

程序调用动态链接库的方式有两种，分别是静态调用方式和运行时的动态调用方式：

1. 静态调用方式；
2. 动态调用方式；

静态调用方式由编译系统完成对 DLL 的加载和应用程序结束时 DLL 卸载的编码（如还有其它程序使用该 DLL，则 Windows 对 DLL 的应用记录减 1，直到所有相关程序都结束对该 DLL 的使用时才释放它），简单实用，但不够灵活。

动态调用方式是由编程者用 API 函数加载和卸载 DLL 来使用 DLL，操作略复杂，但能有

效地使用内存，是编制大型应用程序时的重要方式。它是指应用程序中用 LoadLibrary(windows 平台的系统 API) 显式的将自己所做的动态连接库调进来，动态连接库的文件名即是上面两个函数的参数，再用 GetProcAddress() 获取想要引入的函数指针，然后通过转换变为可以在代码中使用的函数，用户就可以像使用应用程序自定义的函数一样来调引入的外部函数。在应用程序退出之前，应该用 FreeLibrary 释放动态连接库。

如果使用载入时动态链接，程序启动时发现 DLL 不存在，系统将终止程序并给出错误信息；在使用运行时动态链接方式时，LoadLibrary 函数返回空指针，需要调用的函数指针将无法获得。

动态链接具有下列优点：节省内存和减少交换操作。很多进程可以同时使用一个 DLL，在内存中共享该 DLL 的一个副本。相反，对于每个用静态链接库生成的应用程序，Windows 必须在内存中加载库代码的一个副本。节省磁盘空间。许多应用程序可在磁盘上共享 DLL 的一个副本。相反，每个用静态链接库生成的应用程序均具有作为单独的副本链接到其可执行图像中的库代码。升级到 DLL 更为容易。DLL 中的函数更改时，只要函数的参数和返回值没有更改，就不需重新编译或重新链接使用它们的应用程序。相反，静态链接的对象代码要求在函数更改时重新链接应用程序。

方便提供售后支持。例如，可修改显示器驱动程序 DLL 以支持当初交付应用程序时不可用的显示器。支持多语言程序。只要程序遵循函数的调用约定，用不同编程语言编写的程序就可以调用相同的 DLL 函数。程序与 DLL 函数在下列方面必须是兼容的：函数期望其参数被推送到堆栈上的顺序，是函数还是应用程序负责清理堆栈，以及寄存器中是否传递了任何参数。

使国际版本的创建轻松完成。通过将资源放到 DLL 中，创建应用程序的国际版本变得容易得多。可将用于应用程序的每个语言版本的字符串放到单独的 DLL 资源文件中，并使不同的语言版本加载合适的资源。使用 DLL 的一个潜在缺点是应用程序不是独立的；它取决于是否存在单独的 DLL 模块。

6.2.2 托管代码与非托管代码

托管代码与非托管代码是微软针对运行中的 windows 程序与公共语言运行库的关系进行的一种划分，先来看看如何将程序划分为托管代码与非托管代码。

Managed Code —托管代码

由公共语言运行库环境（而不是直接由操作系统）执行的代码。托管代码应用程序可以获得公共语言运行库服务，例如自动垃圾回收、运行库类型检查和安全支持等。这些服务帮助提供独立于平台和语言的、统一的托管代码应用程序行为。微软以命名空间的方法来管理框各种框架类，命名空间都是以名称 System 或 Microsoft 开头的，大量的框架类调用简单，运行安全为用户节省开发时间，降低程序开发难度。托管代码支持多种语言编写，如 C# 和 VB.NET 语言，代码被编译成中间语言 (IL)，在托管环境中运行。

Unmanaged Code —非托管代码

非托管代码与公共语言运行库环境无关。编写这些程序代码使用专用语言编译工具如 C++ 与 VB，生成的是机器可以直接执行的二进制代码，在这些程序中，用户必须自己提供内存的申请和释放。要保证指针引用的正确性，程序不仅要进行类型检查功能，稍有不慎即发生地址越界，引用非法空间，内存泄露等错误，机器也难以由这些错误中恢复回来。

托管的代码就是把有关内存管理（内存申请，内存释放，垃圾回收之类的）任务由.NET 的 CLR 来管理，CLR 把底层的内存操作封装起来了，不能直接进行内存的读取和硬件相关的操作，它的优点是运行比较安全，不会出现诸如内存泄露之类的问题，缺点也很明显，不能直接读取内存，运行性能上会有损失。非托管的代码以 C 和 C++ 为典型，它们可以直接进行硬件操作，其运行性能比较高，但是程序运行风险大，对开发人员的要求也比较高。编写托管代码的程序员可以花更多的精力关注程序的应用逻辑设计并可以减少代码的编写量，更短的开发时间和更健壮的程序。这两种代码最直观的区别就是托管代码中不推荐使用指针 (也可使用)，而非托管代码可以使用指针来直接读取内存。

6.2.3 动态链接库相关工具

通过运行各种命令行工具，为特定 DLL（例如 user32.dll）列出函数名。例如，可以使用 dumpbin /exports user32.dll 或 link /dump /exports user32.dll 来获取函数名。

6.3 函数参数与返回值

函数先定义后调用，函数定义时使用的参数说明称为形式参数，它实现占位作用，调用函数时由实际的值作为实参代替形参。设计函数时要考虑三个基本问题：输入参数、输出和函数功能，参数个数可以是 0 个或几个，如果希望参数数目可变，还可在函数形参定义时采用 params 关键字，在实际调用时参数个数可任意。

函数运算结果有 return 方式和引用方式，return 方式只能返回一个值，引用方式比较特殊，它使用了输入参数，实参的值是由函数执行过程中赋值。引用方式的形式参数添加 ref 或 out 关键字修饰，在函数内对参数的赋值在函数执行完仍然是保留的。在使用 windows 的内核 API 时，某些函数调用必须遵守引用参数 (out) 的方式使用，将运行结果返回。

函数参数有传值方式和引用方式，在 C# 中规定函数参数有三种方式：a. 传值、b.ref 、c.out，如果函数在声明形式参数时无修饰符则是传值方式，实参变量的值在函数执行后不受影响，传值的使用方式相当于实参的值被复制到函数中，不会对原始的实参变量产生影响。引用方式的参数形参与实参为同一内容，函数体内对参数的赋值被保留。

传值方式和 ref 引用方式参数要求调用函数前实参需要赋初值，而 out 方式的形式参数，其实参变量可不必初始化，变量的值会在函数执行过程中被设置。

函数在编写时，函数定义时使用的参数说明称为形式参数，形式参数是一种占位的作用，在调用函数时就需要以实际的值作为实参，而函数调用时参数可以有三种使用方式，a. 传值，b.ref，c.out，ref 和 out 方式都叫引用方式。传值的使用方式相当于参数的值只进不出，也不会对原始的实参产生影响；参数以引用的方式使用时，在函数体中，当有代码对参数进行赋值，赋值的效果会在函数调用完成以后影响到实参的值，在这种情形中，函数中操作的参数量与在调用时传入参数变量相同，这种方式函数可以把运行结果传回，比仅仅使用 return 的方式要更灵活，因为 return 只能返回一个值，而函数的参数列表没有个数的限制。在使用 windows 的内核 API 时，某些函数调用是必须遵守引用的方式使用。

6.4 托管代码动态链接库

创建和使用 C# DLL，用户新建 Visual C# 项目时，已经引用了最常用的基类 DLL（程序集），如果需要使用的类尚未引用，则要添加对此 DLL 的引用。类库由命名空间组成，命名空间都包含可在程序中使用的类型：类、结构、枚举、委托和接口。VS2008 工具提供使用 C# 创建 DLL 的模板，在创建新项目时，选择模板为“类库”，项目说明为用于创建 C# 类库 (.dll) 的项目，设项目位置为 D:
xue，名称为 ClassLibrary1，可参考图6-1：

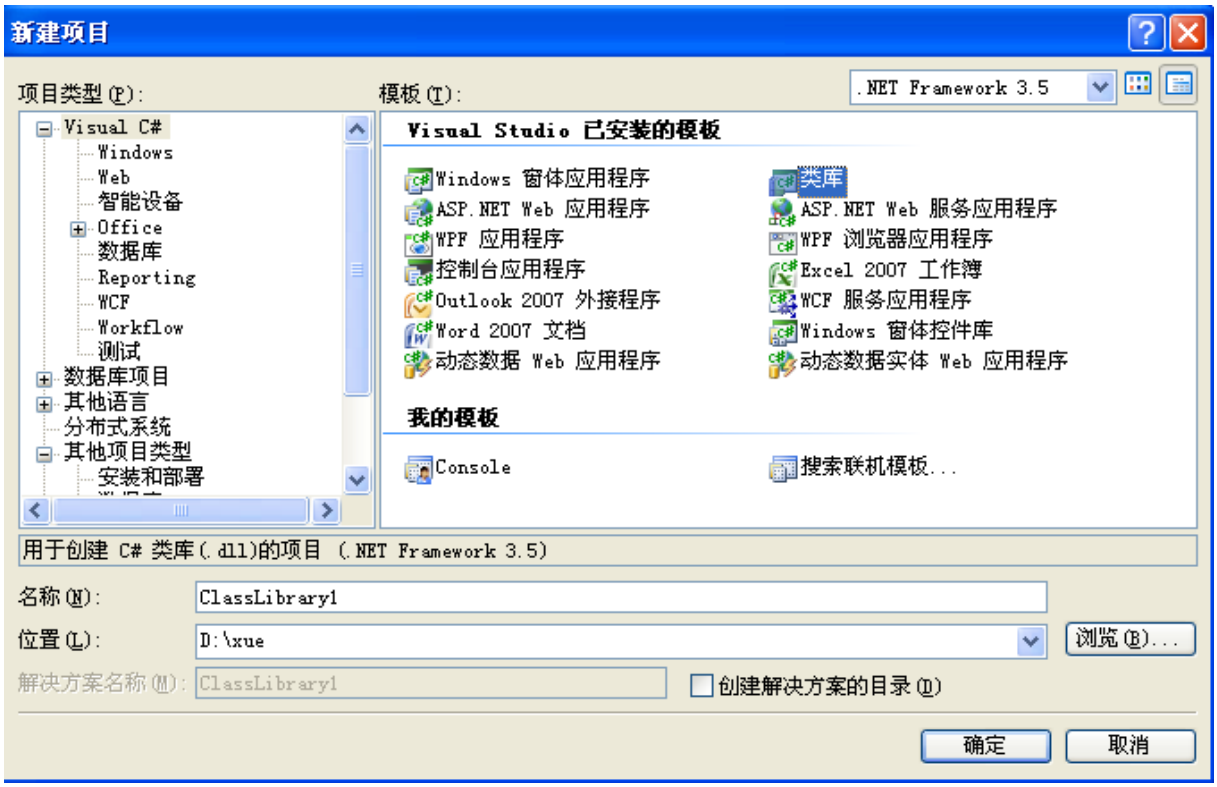

图 6-1 使用 C# 创建类库 (DLL)

使用 C# 编写托管的库根编写普通的类差不多，它与其它应用程序最大的区别是程序文件没有 Main 这样的入口函数，因为它本身用于调用，不需要有入口函数，编写类代码如下：

```
public class Class1
{
    public static long factorial(long num)
    {
        long faresult=1;
        if(num>1)
        {
            for (int i = 1; i <= num; i++)
            {
            faresult = faresult * i;
```

```
        }
      }

      return (faresult);
    }
  }
```

类中 factorial 是个静态函数，它根据提供的整数计算其阶乘的功能。编译后这个项目将生成名为 ClassLibrary1.dll 的文件，factorial 函数可被其它程序使用。接下来再新建一个名为 facttest 的控制台程序用于调用 dll 文件中的 factorial 函数。把 ClassLibrary1.dll 文件拷贝到 facttest 可执行文件同一目录下，在 facttest 项目文件中加命名空间：

```
using ClassLibrary1;
```

项目用添加引用的方式引用 ClassLibrary1.dll 文件，主函数参考代码如下：

```
static void Main(string[] args)
{
  StreamWriter sw = new StreamWriter("out.txt");
  sw.WriteLine(" 数字 {0} 的阶乘值为 {1}",10,Class1.factorial(10));
  sw.Close();
}
```

程序运行时将数的阶乘值输出在 out.txt 文件中。dll 文件可以与调用程序分别编译，用户可将 factorial 函数的实现代码重写，例如实现为累加功能，而 facttest 程序则不变，再次运行 facttest 程序可得到不同的结果，但是要注意的是 factorial 函数名称和参数列表不能变。

6.4.1 动态调用 DLL 中的类方法和属性

如果以动态方式调用 DLL 中的函数，可使用下面的参考代码：

```
StreamWriter sw = new StreamWriter("out.txt");
Assembly a = Assembly.LoadFrom("ClassLibrary3.dll");//装载组件
foreach (Type t in a.GetTypes())
  {
  if (t.IsClass && !t.IsAbstract)
  {
    MethodInfo mi = t.GetMethod("factorial");//获得类型方法
    //创建实例 (无参构造器)
    object o = Activator.CreateInstance(t);
    Object[] parameters = new Object[1];
    parameters[0] = 10;
    //调用实例方法
    sw.WriteLine(" 函数结果为：{0}", mi.Invoke(o, parameters));
  }
```

```
}
sw.Close();
```

. 使用这种方法，尝试将库函数实现代码修改后进行编译，而调用程序不重新编译，查看是否得到新的运行结果，在使用反射机制时，需要添加反射的命名空间：

```
using System.Reflection;
```

6.4.2 提取 DLL 中的类方法和属性

使用 C# 生成的类库，即使使用者没有 dll 文件的源代码，也可以很方便的应用反射的机制得到 dll 文件中的类方法和属性，参考代码如下：

```
StreamWriter sw = new StreamWriter("out.txt");
Assembly a=Assembly.LoadFrom("ClassLibrary1.dll"); //装载组件
foreach(Type t in a.GetTypes())
{
  if(t.IsClass &&!t.IsAbstract)
  {
    MethodInfo[] miArr=t.GetMethods(); //获得类型的公有方法
    object o=Activator.CreateInstance(t);
    //创建实例 (无参构造器)
    foreach (MethodInfo mi in miArr)
    {
      //object re = mi.Invoke(o, null);//调用实例方法
      sw.WriteLine(" 类成员方法：{0}", mi.Name);
    }
  }
}
sw.Close();
```

6.5 非托管的动态链接库

微软的.NET FrameWork 框架类很多，整体功能非常强大，但也有部分功能用框架类不太合适，例如与窗体消息处理密切相关的功能，这些涉及到 windows 核心的运作，要开发这些功能的程序还是要依赖 Windows 的 API。Windows 的 API 提供对操作系统的扩展和功能调用，它功能很强大，自从诞生以来变化较小，一直是广大程序员最值得研究和学习的内容，这部分的内容将在 windows 程序原理部分讲述。API 以 C++ 函数的形式提供，包含在系统的多个 dll 文件中，这些函数都有直接访问内存的操作，属于非托管的动态链接库。经过操作系统市场长期的考验，API 的健壮性得到用户的认可，.NET 平台也支持对这些非托管链接库的调用，调用非托管的动态链接库需要使用 Interop 服务，调用过程包括下面几点：

1. extern 修饰符用于声明在外部实现的方法，
2. 与 DllImport 属性一起使用
3. 将方法声明为 static

下面是一个简单的示例，它接收用户输入的字符串并显示在消息框中，它使用了 User32.dll 库导入的 MessageBox 方法。

```
[DllImport("User32.dll")]
public static extern int MessageBox(int h, string m, string c, int type);
static void Main(string[] args)
{
    string myString;
    Console.Write("Enter your message: ");
    myString = Console.ReadLine();
    MessageBox(0, myString, "My Message Box", 0);
}
```

6.5.1 调用非托管代码参数的封送处理

CLR 常用简写词语，CLR 是公共语言运行时，Common Language Runtime) 和 Java 虚拟机一样也是一个运行时环境，它负责资源管理（内存分配和垃圾收集），并保证应用和底层操作系统之间必要的分离。

CLR 进行数据参数封送有两种方式：多数情况下进行数据值复制，特定情况下将对钉入数据，数据在内存堆中将不被移动，数据指针值被传入非托管代码中。

在必须调用非托管代码而且没有别的方式可选，参数不能使用引用方式传递，调用与被调用处于同一线程内。

A blittable type is one that has a common representation in both managed and unmanaged memory.

char 类型即可看作是 blittable 类型。

Blittable types are data types in software applications which have a unique characteristic. Data are often represented in memory differently in managed and unmanaged code in the Microsoft .NET framework. However, blittable types are defined as having an identical presentation in memory for both environments, and can be directly shared.

分配非托管内存、复制非托管内存块、将托管类型转换为非托管类型，此外还提供了在与非托管代码交互时使用的其他杂项方法。

Marshaling 使用 MarshalAsAttribute 属性和 Marshal 类显式封送数据。

```
    [DllImport("user32.dll")]
private static extern int GetWindowText(int hWnd, StringBuilder title, int size);
```

使用 StringBuilder 应遵循几个标准： 1 . 不采用引用方式， 2 . 保证托管代码使用 Unicode，或者使用 LPWSTR 3. 调用前应设置 StringBuilder 合适尺寸，提供 StringBuilder 的尺寸参数避免溢出。

调试 DLL 自生成动态链接库输出函数。

6.5.2 C# 引用内核 API

介绍 C# 编程语言中如何引入内核 API 为了能够在 windows 平台上开发应用程序，普通的程序开发人员所面临的问题是如何将任务需求用代码表达出来。人们所面对的是无生命的机器，它必定会有本身的局限性，所以我们无法像人与人间的交流那样，可以用形形色色的方式来表达。windows 平台是一个非常具有的操作系统产品，windows 在提供给用户较好用户体验效果的同时，也将各种计算机的硬件功能经过抽象提供给了开发者。那么这些抽象的形式就是 API，程序开发人员的工作也就是将现实的问题用 window 的 API 表达的一个过程，就像建造一座如何漂亮的大楼，所使用的钢筋水泥都是一样的。API 作用，意义 windows 运行平台是所有程序执行的基础，他提供了大量的 API 供普通用户调用来获取系统服务，这些 API 涉及了系统中所有的功能。windows 应用程序正是由这些 API 经过一定的组合连接构造出来的，在 windows 平台上无论使用何种编程语言，都最终调用的是同一个实体实现。windows 平台 API 的存在形式就是众多的 DLL 文件，只要我们能够找到需要的 DLL 文件，我们就可以方便得使用其中的 API，在用户的程序中，API 的概念就是函数指针的意思，程序会按照这个函数指针的位置去执行相应的已经编译好的可执行代码。使用 C# 编程来使用同核 API 来实现一些功能。

6.5.3 DLL 地狱问题

DLL 提供程序复用给用户带来便利的时候也带来一个比较麻烦的问题，生成的 DLL 在维护和修改时可能造成意想不到的后果，比如增加了变量声明，修改了 DLL 中基类定义等。用户程序使用新版本的 DLL 库后，程序再也不能工作了这称为 DLL 地狱 (DLL Hell) 问题。特别是 COM 组件被编译为 DLL 文件，当 COM 组件被升级后，很多其它的软件都无法使用了。

为什么新版本的 DLL 文件替换旧版本 DLL 文件后会造成 DLL 地狱问题呢？这要由 DLL 编译运行方式说起，假如 DLL 中定义了一个变量 m_i，程序调用时将从 DLL 占据的内存位置加上 m_i 变量的偏移量访问，假如新版本的 DLL 在变量 m_i 添加了其它内容例如新的变量 m_j，而调用这个 DLL 文件的其它程序无法知道这种变化，根据原来的偏移量访问得到错误的引用。假如 DLL 添加了新的函数，其它程序将调用错误的函数。

在.Net 平台中采用自我描述与版本管理功能，实现 Side by Side 技术，应用程序安装成功就不必担心 DLL 的更新问题，它允许一个 DLL 的多个编译版本在同一台机器上运行，每一个应用程序可使用指定的 DLL 编译版本，不再发生 DLL Hell 问题。

6.6 实验作业

1. 完成实验内容。
2. 阶乘算法如果直接使用基本数值类型，其最大能表示的计算结果是 2^{64}，请设计算法能表示最大结果为 10^{100} 的程序。
3. 不使用框架类 (例如 File 类) 采用 windows 平台 ReadFile 与 WriteFile 方法实现文件拷贝。
4. ListView 控件中文本实现多行显示。