在.Net平台下C#和Fortran的混合语言编程

Mixed-Language Programming of C# and Fortran under .Net Platform

林国勇 （合肥中国科学技术大学 230027）

董 洵 （北京奥博杰天软件有限公司 100022）

吴婉凡 （北京市2869信箱 100085）

摘要：本文介绍了如何在微软.NET平台上利用DllImport属性实现由C#访问Fortran的动态连接库（DLL）函数的功能，并针对实际应用中可能出现的问题提出相应的解决方案，同时给出了代码示例。

关键词：混合语言编程 C# Fortran DllImport 动态连接库

1 前言

.NET是Microsoft推出的完全面向对象开发的平台，用户可以在这个平台上快速建立企业级Web应用程序和高性能桌面应用程序。C#语言是由C++的发展和演化出来的，它是专门为.Net平台开发设计的一个先进的、安全的、面向对象的编程语言，其设计目的之一就是支持快速应用程序开发（Rapid Application Development）。与Visual Basic的开发方式相似，C#允许我们使用设计器快速而简单地设计Windows应用程序和Web应用程序。

在实际工程应用中存在着许多已经历时间的考验，成熟、稳定的Fortran计算程序，至今在工程计算中仍发挥着重要作用，但是Fortran语言本身并不适合用来开发Windows界面程序和Web应用程序。因此，为了提高程序的使用效率，提高代码的重用率，有必要对原FORTRAN程序进行改进，使得原有性能优良的FORTRAN代码可以被在C#下调用，最大限度的发挥两种语言各自的优点。

本文介绍了如何利用Microsoft的.NET平台使用DllImport属性支持.NET平台语言和普通编程语言的混合编程方法，来实现由C#访问Fortran的动态连接库（DLL）函数的功能。

2 混合编程的方法

使用C#与Fortran混合语言编程，就是利用C#实现友好的用户界面，用FORTRAN编写的过程进行所有的计算工作。在程序中，必须以C#为主程序来调用FORTRAN程序。实现混合编程的关键就是将FORTRAN计算程序编译为动态链接库，而后在C#中调用此动态链接库。下面将给出这种混合语言编程的方法和有关注意事项。

2.1 DllImport简介

.NET平台的DllImport是用来修饰从外界DLL导入函数的属性。通过这个功能，运行于虚拟平台之上的C#语言可以访问已有的所有Win32的API，从而大大扩展了.NET应用程序所适用的空间。

下面这个例子说明如何用DllImport使用Win32 API中的MoveFile函数

```
class FileManipulation
{
  [DllImport("KERNEL32.DLL",
  CallingConvention=CallingConvention.StdCall)]
  public static extern bool MoveFile(String src, String dst);
}
```

下面的代码演示了如何使用C#调用这个函数：

```
FileManipulation.MoveFile(@"c:\Hello.txt", @"c:\demo\Hello.txt");
```

这里public用来说明这个函数是公用的，可以在程序中的其他地方访问它；static则表示这个函数是静态的，即C#不会在调用的时候传入出参数以外的其他信息；extern则表示这个函数由程序以外的模块实现。方括号内的就是针对MoveFile函数设置的DllImport属性［C++, JScript］No example is available for C++ or JScript. To view a Visual Basic or C# example, click the Language Filter button in the upper-left corner of the page.。第一个参数表示这个函数是由Kernel32.dll导入的，第二个参数则指名的该函数的调用方式，在本文后面对此会有更进一步的介绍。

由于绝大部分的Win32API的库函数都使用C/C++语言编写，使用DllImport访问DLL中函数的编程难度很小。然而由于Fortran语言所不同于C/C++的种种特性，从C#语言中访问Fortran语言编写的DLL中的函数，还是有很多需要注意的地方，下面就将它们一一列出并依次给出解决的方案。

值得一提的是对于其中的绝大部分问题，本文都分别提供了从Fortran和C#出发的两种解决方案，供读者选择。一般而言，为了保证Fortran的DLL能够被更多的语言支持，首选的解决方案是使用Fortran语言提供的属性对导出函数的定义进行修改。

2.2 函数命名

从DLL中导出的Fortran函数的名字都被Fortran编译器自动转换为全部大写，这样在C#中就无法找到与Fortran源代码中声明相同的函数名。可以使用.NET Framework提供的dumpbin.exe工具查看DLL导出的函数名称。下面的命令将列出当前目录下math.dll中导出的所有函数信息。

```
Dumpbin /exports math.dll
```

对这个问题的来自Fortran方面的解决方案是使用ALIAS（别名）属性指定导出函数名。例如对于下面的Fortran函数：

```
SUBROUTINE Factorial (I)
INTEGER I
```

对应的C#声明为

```
[DllImport("Math.dll")]
public extern static void FACTORIAL(int i);
```

使用ALIAS修改后的定义如下，其中黑体部分表示修改处。

```
SUBROUTINE Factorial (I)
  !DEC$ ATTRIBUTES ALIAS:'_Factorial' :: Factorial
INTEGER I
```

对应的C#声明为

```
[DllImport("Math.dll")]
public extern static void Factorial(int i);
```

而来自于C#方面的解决方案是通过使用DllImport的EntryPoint属性指定导出的Fortran函数名。

```
SUBROUTINE Factorial (I)
INTEGER I
```

对应的C#声明为

```
[DllImport("Math.dll",
  EntryPoint="FACTORIAL")]
public extern static void Factorial(int i);
```

2.3 函数调用

不同的语言在函数调用返回时清理堆栈的的方式是不同的，一般分为由调用方清除和由被调用方清除两种。

C#语言采取的是与C相同的堆栈方式，由调用方清除;而Fortran采取的则是stdcall方式，即由被调用方清除。我们必须统一调用双方的堆栈清除方式才能让函数调用能够正常进行下去。

对于这个问题，来自于Fortran方面的解决方案是使用_cdecl声明

```
SUBROUTINE Factorial (I)
  !DEC$ ATTRIBUTES C, ALIAS:'_Factorial' :: Factorial
INTEGER I
[DllImport("Math.dll")]
public extern static void Factorial(int i);
```

来自于C#方面的解决方案是在DllImport的CallingConvention属性中指定调用方式。

```
SUBROUTINE Factorial (I)
  !DEC$ ATTRIBUTES ALIAS:'_Factorial' :: Factorial
INTEGER I
[DllImport("Math.dll",
  CallingConvention = CallingConvention.StdCall)]
public extern static void Factorial(int i);
```

3 混合编程中应注意的问题

用混合语言编程存在由于不同语言特点而导致的很多不匹配问题，解决这些问题是成功实现混合语言编程的关键。

3.1 传递数值参数

函数的参数传递由两种方式，一种是传值（Pass-by-value），被调用方得到的参数值调用方传入参数的拷贝而非传入参数本身。另一种是传参（Pass-by-reference），被调用方和调用方使用的同一个参数。在处理C#与Fortran之间的相互调用时必须妥善处理这个问题，否则很容易引起非法内存访问的问题。

默认情况下，Fortran对所有的参数使用Pass-by-reference的方式传递参数。但是对于C#来说，情况就复杂的多。在C#中，变量分为两种类型：值类型（Value Type）和引用类型（Reference Type）。顾名思义，值类型在作为函数参数调用时采用Pass-by-value而引用类型采用Pass-by-reference的方式。C#中所有的数值类型都是Value type而表示字符串的String则是Reference Type。

下面从这段引起内存问题的代码入手，演示如何解决这个问题。

3.1.1 有问题的代码

```
SUBROUTINE Factorial (I)
  !DEC$ ATTRIBUTES C, ALIAS:'_Factorial' :: Factorial
INTEGER I
Class MathLib
{
  [DllImport("Math.dll",
    CallingConvention = CallingConvention.StdCall)]
  public extern static void Factorial(int i);
}
```

如果C#用户使用下面的代码访问Factorial（阶乘）函数，

```
MathLib.Factorial(3);
```

Fortran函数会企图从内存地址0x00000003的地方读取参数，就会引

起非法内存访问的问题：

3.1.2 解决方案

在Fortran方面，可以利用Fortran的VALUE属性指定参数传递使用by-value的方式

```
SUBROUTINE Factorial (I)
  !DEC$ ATTRIBUTES C, ALIAS:'_Factorial' :: Factorial
INTEGER I    [VALUE]
[DllImport("Math.dll",
  CallingConvention = CallingConvention.StdCall)]
public extern static void Factorial(int i);
```

在C#方面，可以使用C#的ref关键字，确保函数传递给Fortran的参数时是该参数的地址。

```
SUBROUTINE Factorial (I)
  !DEC$ ATTRIBUTES C, ALIAS:'_Factorial' :: Factorial
INTEGER I
[DllImport("Math.dll",
  CallingConvention = CallingConvention.StdCall)]
public extern static void Factorial(ref int i);
```

3.2 传递并返回字符串

传递并返回字符串是混合编程中最为复杂，也是需要考虑的问题最多的部分。首先讨论单方向从C#将字符串传入Fortran函数的情况。

C#中字符串的表达方式与C语言相同，使用＼0表示字符串的结束；Fortran中字符串则采用在最右端添加空格的方式表示，并在最右端使用一个隐藏的参数表示实际长度。因此要向正确传递字符串，首先要解决如何正确表达字符串长度的问题。

3.2.1 解决方法

在Fortran方面，可以通过与声明函数调用相同的方式解决这个问题。一旦Fortran函数被声明为C attribute，函数就不再使用在最右端隐藏的方式表示字符串长度。

C#方面可以在C#的字符串参数声明后添加一个int类型的参数表达字符串的长度。

```
SUBROUTINE Print (Str)
  !DEC$ ATTRIBUTES ALIAS:'_ Print':: Print
CHARACTER*(*) Str
[DllImport("Math.dll",
  CallingConvention = CallingConvention.StdCall)]
public extern static void Factorial(string msg, int msgLength);
```

3.2.2 返回字符串

返回字符串的难度在于首先C#要讲自己的内存空间复制给Fortran函数使用，然后又需要将由Fortran修改过的字符串重新复制回C#的调用函数。这个来回中就牵涉到很多问题，大部分已经有DllImport自动解决了，DllImport不能自动处理的问题有下面两个，需要注意的是这部分的问题只能由C#单方面处理。

首先：Fortran默认使用Ansi的单字节方式存储和处理字符串，而C#则使用Unicode的双字节方式。这样产生的问题是C#会将Fortran的每两个字符合并为一个字符，结果在Fortran中的128个字符的字符串在C#中变成了64个垃圾字符。这个问题的解决方法就是针对函数参数使用MarshalAs和Out属性。MarshalAs告诉C#函数需要将获得的字符串从什么格式转到Unicode。常用的格式是LPStr（ANSI字符串）和LPWStr(双字节字符串)。Out则告诉C#这个函数需要从被调用函数修改后传回。

其次：C#中可以用System.String, System.Text.StringBuilder来表达字符串，两者所不同的在于string用来存储只读字符串，而StringBuilder专门设计用来处理字符串的修改。如果对string进行修改操作，实际上是重新分配了空间用来存储修改后的字符串。

```
String msg = "Hello";
msg += " World";
```

第二句执行后，C#的编译器会在内存中重新分配11个字符的空间（22byte），然后修改msg让它指向新分配的这块内存。

如果希望C#能够接受经Fortran修改后的字符串，就必须使用StringBuilder。

综合上面的因素，为了让C#能够正确使用由Fortran返回的字符串，函数声明应该为：

```
[DllImport(@"DECODE1.dll",
  CallingConvention=CallingConvention.Cdecl)]
public static extern void ToUpper(
[Out, MarshalAs(UnmanagedType.LPStr)] StringBuilder filename,
int fileLength);
```

4 结束语

C#制作的图形界面直观、清晰，有利于提高数据的输入效率，易于修改，节约时间，便于用户将工作重点放在方案比较上。实践证明，只要掌握了C#混合语言编程中的利用DllImport来修饰从外界DLL导入函数的属性和参数的传递技术，对Fortran计算程序稍加修改成为动态链接库，就可实现旧代码的升级利用，保护可贵的Fortran源程序资源，起着重要的作用。

参考文献

1 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/html/_core_mixed.2d.language_programming_with_c.2b2b.asp .

2 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemruntime interopservicesdllimportattributeclasstopic.asp.

3 http://msdn.microsoft.com/msdnmag/issues/03/07/NET/default.aspx.

4 K.Scott Allen等著，康博译，高效掌握C#，清华大学出版社，2002年10月。