能源与动力工程学院

混合语言编程

陈　斌

目录

混合语言编程的目的

- 混合语言编程：由两种或者两种以上的程序语言编写源代码，进行参数传递、共享数据结构和信息，实现子程序的互相调用，从而建立应用程序的过程。
- 使用混合语言编程，可以充分利用各种语言的优势，扬长避短。特别是可以调用已经存在的用其他语言编写的代码，避免重复劳动。
- Fortran具有独特而灵活丰富的数组操作，特别擅长处理高精度浮点数运算、复数运算、多维数组等，因此在数值计算、科学和工程技术领域具有强大的优势。Fortran自诞生以来积累了大量高效而可靠的源程序，目前广泛地应用于并行计算和高性能计算领域。
- C语言的特点是灵活，在内存的动态管理、函数调用与参数传递、字符串处理、图形处理以及软件开发环境和集成性等方面具有强大的优势。

调用约定

- 为变量和过程的命名、不同语言编写的例程（包括不同语言中的函数、子例程和过程）之间传递的参数等建立一套规则
- 命名约定：为了解决不同语言对变量名、参数名、过程名和函数名等标识符的不同处理，以及对名称标识符的不同长度限制等的有关规则、协议和约定。
  - 大小写、命名规则、标识符长度
  - 通过一定的书写规则和协议，用编译程序在将一个程序块放入目标文件之前改变它名称，采用一个兼容的、被调用语言认可的名称约定
- 参数传递
  - 值传递还是通过引用传递
  - 等价数据类型和数据结构
  - 传递变量的顺序

Fortran & C

Fortran	C
INTEGER(1)	signed char
INTEGER(2)	short int
INTEGER(4), INTEGER	int
INTEGER(8)	long int
REAL(4), REAL	float
REAL(8), DOUBLE PRECISION	double
COMPLEX(8)	无对应类型，可定义结构 struct complex {float r, i; }
CHARACTER	char
CHARACTER(n)	char x[n]
LOGICAL(1)	char
LOGICAL(2)	short
LOGICAL(4)	int
LOGICAL(8)	long int

Fortran & C

	有返回值调用	无返回值调用
Fortran	函数（FUNCTION）	子例程（SUBROUTINE）
C	函数（function）	空函数（void function）

- 对于有返回值调用，二者的返回值数据类型必须一致。
- Fortran调用例程时，实参和形参默认是引用传递，而C则是值传递。
- 如果二者均采用默认的调用方式，那么Fortran调用C函数时，C的形参应该声明为指针类型（*）；C调用Fortran函数时，C传递实参时须以&形式传递变量地址。
- 传递数组时，二者均使用数组首地址作为参数，因此无需转换。如果C的参数本身为指针变量，也不需要转换。

Fortran和C的混合编程方式

- 分别编译、独立运行。也就是将Fortran和C各自要实现的功能模块源代码在各自的开发平台上编译连接成可执行文件并独立执行，二者的数据通过数据文件交换。
- 函数级调用。即Fortran和C分别编译各自的功能模块源代码，得到各自的目标文件（.obj），然后集成链接这些obj文件生成一个统一的可执行文件，实现对对方函数的调用，数据交换通过约定接口来实现。（被调用模块一旦被修改，整个软件必须重新进行编译连接 ）
- 动态链接库方式。将需要被调用的功能模块源代码编译连接成动态连接库，然后通过约定的接口动态使用另外的语言调用该功能模块。 修改DLL无需重新编译主调程序，具有较好的移植性和复用性。

Fortran与C的函数级调用

语言	调用约定	例程名	obj文件中的例程名		对应的调用约定
			转化为	大小写	
Fortran	缺省	nAme	_NAME@n	全部大写	/
	C	nAme	_name	全部小写	_cdecl
	STDCALL	nAme	_name@n	全部小写	_stdcall
C	_cdecl (缺省)	nAme	_nAme	混合大小写	C
	_stdcall	nAme	_nAme@n	混合大小写	STDCALL

n代表所有形参的字节数之和，以十进制表示。

- 实参都是按照从左到右的顺序将形式参数表中的变量传递给例程
- Fortran被调用的例程名在obj文件中要么是全部大写（缺省模式），要么是全部小写
- C的例程则可以保留混合大小写

Fortran与C的函数级调用

- C调用Fortran例程时，Fortran需要在接口块的例程说明或例程头部加入编译伪指令告知编译器有关调用约定。
- !MS$ ATTRIBUTES
- !DEC$
- 对于固定格式，“！”可以写作“C”、“c”或“*”

属性名称		含 义	
例程	ALIAS	为函数或子程序指定其在调用方的名字	一般应为输出函数或子程序指明该属性
	C	指定用C方式代替缺省调用约定	若两者均未指定，则使用不同于二者的缺省方式
	STDCALL	指定用STDCALL方式代替缺省调用约定	
参数	REFERENCE	按引用方式传递参数或调用函数	
	VALUE	参数按值传递	
	VARYING	参数类型强制匹配	

Fortran与C的函数级调用

- 此时，C语言要在函数说明前面加上extern关键字说明函数来自外部或供外部使用，extern和函数说明中间加上_cdecl或_stdcall说明调用约定的属性

```
INTERFACE ! Fortran的接口块
    FUNCTION SUM(I, J)
        !MS$ATTRIBUTES C, ALIAS:'_Sum':: SUM !供C语言调用的Sum函数
        !MS$ATTRIBUTES REFERENCE::I, J
        INTEGER I, J, SUM
   END FUNCTION
END INTERFACE

/* C源程序*/
extern _cdecl int Sum(int *, int *)
{
   …
}
```

Fortran与C的函数级调用

- C调用Fortran例程时，C的形参必须传递与Fortran形参同类型变量的地址

```
! Fortran的例程
SUBROUTINE SUM(I, J)
    INTEGER I, J
…
END FUNCTION

/* C源程序*/
extern _cdecl int SUM(int *, int *);
int main()
{
   int a, b;
   …
   SUM(&a, &b);
   …
}
```

Fortran调用C语言函数

- Fortran调用C语言函数时，要在接口块中加入!MS$ ATTRIBUTES编译伪指令告知编译器调用C函数时使用的约定

```
INTERFACE
   例程说明语句      !定义函数FUNCTION或子例程SUBROUTINE
       [!MS$ ATTRIBUTES例程选项] !规定例程的调用、命名和参数传递约定
       [!MS$ ATTRIBUTES参数选项] !规定参数传递的方式
       例程参数声明
   END 例程名称
END INTERFACE
```

Fortran调用C语言函数

- 例程的调用约定：为了统一Fortran的例程声明和所调用的C语言函数声明的大小写，需要在例程调用约定语句中添加ALIAS（别名）属性声明，消除调用约定对例程名产生的影响，使C能用大小写混合形式声明被Fortran调用的函数。

调用的C函数

```
!MS$ ATTRIBUTES C, ALIAS:'_nAme' :: NAME
extern int nAme(int *p1, int *p2);
extern int _cdecl nAme(int *p1, int *p2);

!MS$ ATTRIBUTES STDCALL, ALIAS:'_nAme@n' :: NAME
extern int _stdcall nAme(int *p1, int *p2);
```

调用的C函数

Fortran调用C语言函数

- 参数的调用约定：Fortran缺省参数传递为引用传递，若在外部例程中施加了C或STDCALL调用约定，则缺省的引用传递改为值传递（数组参数除外）。为了消除调用约定对参数传递的影响，可以在外部例程中添加具体的参数传递属性（REFERENCE或VALUE）声明。需要注意，参数属性声明优先于调用约定声明，即参数属性声明最终决定参数的传递方式。

```
!MS$ ATTRIBUTES C::MYSUB
    ! C约定将例程MYSUB参数列表的引用传递改为值传递
!MS$ ATTRIBUTES VALUE :: a      !将a定义为值传递方式
!MS$ ATTRIBUTES REFERENCE :: a !将a定义为引用传递方式
```

Fortran调用C语言函数

例11-1 FORTRAN调用C的空函数

```
! FORTRAN 源文件：ForMain.f90
PROGRAM C4For1
  INTERFACE                          ! Fortran90与C的接口
      SUBROUTINE SUM(I, J, k)
       !MS$ ATTRIBUTES C, ALIAS: '_sum':: SUM ! 命名约定
       !MS$ ATTRIBUTES VALUE::I  ! 形参I使用值传递方式
       !MS$ ATTRIBUTES REFERENCE::J,K ! 形参J、k使用引用传递方式
       INTEGER I, J, K
      END SUBROUTINE
  END INTERFACE
  INTEGER I, J, K
  I=1; J=2
  WRITE(*, '(2(A, I2))') 'I=', I,', J=', J
  CALL SUM(I, J, K)
  WRITE(*, '(A, I2)') 'The sum of I and J is ', K
END
```

Fortran调用C语言函数

```
/* C 源文件：function.c  */
#include <stdio.h>

extern void _cdecl sum(int p1, int *p2, int *p3)
/* 调用C++程序时用 extern "C" */
{
    *p3 = p1 + *p2;
}
```

- 打开VF建立一个Console Application工程
- 先建立1个Fortran源文件ForMain.f90
- 再建立1个Text文件function.c
- 编译、连接，即可生成可执行文件

Fortran调用C语言函数

例11-2 FORTRAN调用C的有值函数

```
! FORTRAN 源文件：ForMain.f90
PROGRAM C4For2
   INTERFACE                          ! Fortran90与C的接口
        FUNCTION SUM(I,J)
               !MS$ ATTRIBUTES STDCALL, ALIAS:'_sum@8':: SUM
               !MS$ ATTRIBUTES VALUE:: I
               !MS$ ATTRIBUTES REFERENCE:: J
               INTEGER I, J, SUM
        END FUNCTION
   END INTERFACE
   INTEGER I, J
   I=1; J=2
   WRITE(*,'(2(A, I2))') 'I=', I, ', J=', J
   WRITE(*,'(A, I2)') 'The sum of I and J is ', SUM(I, J)
END
```

Fortran调用C语言函数

```
/* C 源文件：function.c  */
#include <stdio.h>
extern _stdcall int sum(int p1, int *p2)  !!!int应该
    在调用约定之前
{
    return  p1 + *p2;
}
```

- 打开VF建立一个Console Application工程
- 先建立1个Fortran源文件ForMain.f90
- 再建立1个Text文件function.c
- 编译、连接，即可生成可执行文件

C调用Fortran语言例程

- C语言调用Fortran例程时，要在源程序中对所调用的Fortran例程使用extern关键字说明该例程来自外部，extern和函数说明中间加上_cdecl或_stdcall说明调用约定的属性
- 如果Fortran例程使用缺省方式的调用约定（即不加任何说明），它被C调用的函数/例程名在其obj文件中全转化为大写。此时C在声明所调用的Fortran外部例程原型时，无论采用何种调用约定，例程名必须大写。
- 对于这种情形，Fortran的源文件中不用对被C调用的例程做任何说明。

```
/* 命名必须大写 */
extern double _cdecl DIS_FUNCTION(double *);
extern double _stdcall DIS_FUNCTION(double *);
```

C调用Fortran语言例程

例11-3 C调用FORTRAN例程，Fortran使用缺省的调用约定

```
/* C源文件 */
#include <stdio.h>
extern void _cdecl DIS_SUB(float *, float *, double *);/*命名必须大写*/
extern double _stdcall DIS_FUNCTION(double *);          /*命名必须大写*/
int main()
{
   float x = 1.0, y = 1.0;
   double distance, d;

    DIS_SUB(&x, &y, &d);              /* 传递变量的地址，即引用传递 */
    distance = DIS_FUNCTION(&d);
    printf("x=%3.1f  y=%3.1f  d=%f  distance=%f\n", x, y, d, distance);
   return 0;
}
```

C调用Fortran语言例程

```
!Fortran源文件
subroutine dis_sub(x, y, d)
   implicit none
   real :: x, y
   real(8) :: d
   d = x * x + y * y
end subroutine
real(8) function dis_function(d)
   implicit none
   real(8) :: d
   dis_Function = sqrt(d)
end function
```

- 打开VC建立一个Win32 Console Application工程
- 先建立1个C源文件CMain.c（Text文件）
- 再建立1个Fortran自由格式的源文件sub.f90
- 编译、连接，即可生成可执行文件

C调用Fortran语言例程

- 如果Fortran例程使用C或STDCALL调用约定，它分别对应于C的_cdecl方式和_stdcall方式。此时Fortran例程中必须通过!MS$ ATTRIBUTES编译伪指令说明C函数的别名，C就可以使用保留混合大小写的方式声明它所要调用的Fortran例程。

C调用Fortran语言例程

例11-4 C调用Fortran例程，Fortran使用C或STDCALL调用约定

```
/* C源文件 */
#include <stdio.h>

extern void _cdecl dis_Sub(float *, float *, double *);/* 命名无须大写 */
extern double _stdcall dis_Function(double *);          /* 命名无须大写 */

int main()
{
   float x = 1.0, y = 1.0;
   double distance, d;

   dis_Sub(&x, &y, &d);           /* 传递变量的地址，即引用传递 */
   distance = dis_Function(&d);
   printf("x=%3.1f  y=%3.1f  d=%f  distance=%f\n", x, y, d, distance);
   return 0;
}
```

C调用Fortran语言例程

```
!Fortran源文件
subroutine dis_sub(x, y, d)
   implicit none
   !MS$ ATTRIBUTES STDCALL, ALIAS:'_dis_Sub@12':: DIS_SUB
   !MS$ ATTRIBUTES REFERENCE::x, y, d
   real :: x, y
   real(8) :: d
   d = x * x + y * y
end subroutine

real(8) function dis_function(d)
   implicit none
   !MS$ ATTRIBUTES C, ALIAS:'_dis_Function':: DIS_FUNCTION
   !MS$ ATTRIBUTES REFERENCE:: d
   real(8) :: d
   dis_Function = sqrt(d)
end function
```

Fortran与C调用对方的动态链接库

- 静态库（*.lib）和动态库（*.dll）
  - LIB中的函数必须复制到EXE文件，DLL中的函数是EXE文件运行时动态调用的，与EXE放在同一目录下即可。当共享的函数放在DLL中时，应用程序能变得很小。同时，多个应用程序可以访问同一个DLL，提高了系统资源的利用效率。
  - DLL只在内存中加载一次，所有使用该DLL的进程会共享此块内存，所以使用DLL可以节省内存。
  - 如果函数参数和返回类型不变，也就是调用函数的接口保持不变，改变DLL中的函数代码后可以不重新编译或链接调用DLL的程序。因此，使用DLL可以提高软件的开发效率，通过改变DLL即可实现对应用程序的更新和升级。
  - DLL的通用性强，它的接口函数可被任何编程语言所编写的应用程序调用。

C调用Fortran DLL

- C在图形显示、数据结构等方面功能强大，而Fortran擅长科学计算。因此，通常是用C编写图形和用户交互界面，而将Fortran的科学计算例程作为DLL
- 在包含Fortran例程的DLL源文件中，需要在例程头部加入如下说明：

```
! FORTRAN DLL源文件
SUBROUTINE 例程名(参数列表)
    !MS$ ATTRIBUTES DLLEXPORT::函数名     !声明本函数为输出函数
     …
END SUBROUTINE OUTPUT
```

- 对应的C语言主程序中，需要对该函数做出说明：

```
    extern 函数类型 _cdecl 函数名(参数列表);
或  extern 函数类型 _stdcall 函数名(参数列表);
```

C调用Fortran DLL

- 为了能让C语言顺利地调用Fortran DLL，首先必须用Fortran编译器建立Fortran的DLL供C语言调用
- 打开Visual Fortran，选择Fortran Dynamic Link Library，新建一个项目ForDll

```
! FORTRAN 源文件 for4c.f90
SUBROUTINE OUTPUT(a, b, sum)
  !MS$ ATTRIBUTES DLLEXPORT::OUTPUT !声明本函数为输出函数
  IMPLICIT NONE
  INTEGER a, b, sum
  sum = a + b
END SUBROUTINE OUTPUT
```

- 编译、链接源文件即可在当前工程的Debug目录下生成ForDLL.lib和ForDLL.dll两个文件

C调用Fortran DLL

- C以静态方式调用Fortran DLL
- 又称隐式调用，即将Fortran的LIB文件直接加入到C的主程序，然后直接编译执行。该方法的实施简单，需要添加的码少。但由于需要装载LIB文件，会导致运算慢、占用系统资源过多。

例11-6 C隐式调用FORTRAN动态链接库

```
/* C 源文件 main.c */
#include <stdio.h>
/*声明函数OUTPUT为extern型的，即是从外部调用的。*/
extern void _stdcall OUTPUT(int *a, int *b, int *sum);
int main()
{
   int a = 1, b = 2, sum;
   OUTPUT(&a, &b, &sum);
   printf("%d + %d = %d\n", a, b, sum);
   return 0;
}
```

- 将Fortran建立的ForDLL.lib和ForDLL.dll文件复制到当前VC工程的Debug目录下，点击主菜单的Project，将ForDLL.lib文件加入VC的Project，然后编译运行。

C调用Fortran DLL

- C以动态方式调用Fortran DLL
- 也称显式调用或动态装载。首先通过Windows API函数LoadLibrary()装入DLL，再用GetProcAddress()函数来取得DLL中被调用的FORTRAN函数的地址，调用该函数后再用FreeLibrary()函数将DLL释放。这种调用方式的优点在于可以完全控制DLL的载入和释放， 无需将.lib文件加入VC工程，最有效地利用系统资源。
- 例11-7 C/C++显式调用FORTRAN动态链接库

C调用Fortran DLL

```
#include <stdio.h>
#include <windows.h>          /* 包含LoadLibray()等函数的头文件 */

int main()
{
    int a = 1, b = 2, sum;

    /* 声明dll中的要调用的函数指针，注意必须说明调用约定 */
    typedef int(_stdcall *p_output)(int *a,int *b, int *sum);
    p_output OUTPUT;          //用函数指针声明dll中要调用的函数

    HINSTANCE hInstance;      //声明一个实例句柄
    hInstance = LoadLibrary("ForDLL.dll"); //加载动态库文件，获得该文件的实例句柄
    if(hInstance == NULL)
    {
          printf("No DLL file exist!\n");
          return -1;
    }

    OUTPUT = (p_output)GetProcAddress(hInstance,"OUTPUT");  //得到dll中被调函数的地址
    if(OUTPUT == NULL)
    {
          printf("Can not fine the address of the function!\n");
          return -2;
    }

    OUTPUT(&a, &b, &sum);     //调用dll中的函数
    printf("%d + %d = %d\n", a, b, sum);

    FreeLibrary(hInstance);   //卸载动态库文件

    return 0;
}
```

Fortran调用C的动态链接库

- Fortran也可以调用C的DLL。如果C的DLL中使用了windows的API函数，Fortran可以通过这种途径间接调用API函数，给Fortran语言扩展了应用空间。
- C语言建立DLL源程序时，需要在文件头部加入如下说明：

```
   extern 函数类型 _declspec(dllexport) _stdcall nAme(参数列表);
或 _declspec (dllexport) extern 函数类型 nAme(参数列表);
```

- _declspec (dllexport)用来说明函数nAme作为DLL的输出函数。

```
! FORTRAN 主程序
  interface
       function name(参数列表)
       !MS$ATTRIBUTES DLLIMPORT, STDCALL, ALIAS: '_nAme@n' ::name
!      !MS$ATTRIBUTES DLLIMPORT, C, ALIAS: '_nAme' ::name
       参数说明
       end function name
  end interface
```

Fortran调用C的动态链接库

- 打开VC，选择Win32 Dynamic Link Library，新建一个项目CDLL

```
/* C 源文件 c4for.c */
#include <stdio.h>
#include <windows.h>              /* API函数MessageBox的头文件 */
_declspec (dllexport) extern int _stdcall sum(int *a, int *b);
/*也可写成 extern int _declspec (dllexport) sum(int *a, int *b);*/
int _stdcall sum(int *a, int *b)
{
   int c;
   char d[255];
   c = *a + *b;
   sprintf(d, "You are calling windows API function by Fortran to show the
              result '%d '", c);
   MessageBox(NULL, d, "C DLL called by Fortran", MB_OK);
        /* 调用消息弹出框函数显示计算结果 */
        /* d 表示待输出的字符串 */
        /* "C DLL called by Fortran" 将显示为弹出窗口的标题*/
        /* MB_OK 表示将弹出窗口的按钮显示为"OK" */
   return c;
}
```

Fortran调用C的动态链接库

- Fortran调用C的DLL　　例11-8 Fortran调用C的DLL

```
! FORTRAN 源文件 ForMain.f90
program main
   interface
        integer function sum(a, b)
        !MS$ ATTRIBUTES DLLIMPORT, STDCALL, ALIAS: '_sum@8' ::sum
        integer a, b
        end function sum
   end interface
   integer a,b
   a = 1; b = 2
   write(*, *) sum(a, b)
End
```

- 将前面VC建立的CDLL.lib和CDLL.dll文件拷贝至VF project的Debug目录下，点击主菜单Project选项，将CDLL.lib文件加入VF的project
- 编译并运行Fortran程序，可以调用C的DLL，间接使用windows的API函数MessageBox弹出消息框显示计算的结果

Fortran 2003与C的互相调用

- Fortran 2003的新标准提供了内部模块ISO_C_BINDING，定义了Fortran与C语言连接时必需的类型常量，从而规范了Fortran语言与C语言的连接方式。
- 对于支持Fortran 2003标准的编译器，源程序中简单引用ISO_C_BINDING模块后，就可以将Fortran变量定义成与C语言数据结构兼容的数据类型，C语言和Fortran语言就可以通过上述接口相互调用。
- Fortran使用iso_c_binding与C互相调用对方函数的前提是用接口块说明例程的协同性

Fortran 2003与C的互相调用

例11-9 Fortran 2003调用C的函数

```
! FORTRAN 源文件
PROGRAM FOR2003
   IMPLICIT NONE
   INTERGER X, Y
   INTERFACE
        SUBROUTINE SUM( A, B ) BIND( C )
               USE, INTRINSIC :: ISO_C_BINDING   !声明使用ISO_C_BINDING
               INTEGER( KIND=C_INT ), VALUE :: A
                                  !声明变量A相当于C的INT类型，使用值传递
               INTEGER( KIND=C_INT )        :: B
                                  !声明变量B相当于C的INT类型，使用引用传递
        END SUBROUTINE SUM
   END INTERFACE
   X = 1; Y = 2
   CALL SUM(X, Y)
END
```

Fortran 2003与C的互相调用

```
/* C 源文件 */
#include <stdio.h>

void sum(int a, int *b)
{
  printf("%s%d\n", "sum = ", a + *b);
}
```

C调用Fortran 2003的函数和子例程

例11-10 C调用Fortran 2003的函数和子例程

```c
/* C 源文件 */
#include <stdio.h>
extern void __cdecl DIS_SUB(float *, float *, double *);
/* Fortran函数或子例程的命名必须大写 */
extern double __stdcall DIS_FUNCTION(double *);
int main()
{
   float x = 1.0, y = 1.0;
   double distance, d;
   DIS_SUB(&x, &y, &d);
   distance = DIS_FUNCTION(&d);
   printf("x=%3.1f  y=%3.1f  d=%f  distance=%f\n", x, y, d, distance);
   return 0;
}
```

C调用Fortran 2003的函数和子例程

```fortran
! FORTRAN 源文件

subroutine dis_sub(x,y,d)
  use ISO_C_BINDING        !Fortran 2003 版本须使用本句说明
  implicit none
  real :: x, y
  real(8) :: d
    d = x * x + y * y
end subroutine

real(8) function dis_function(d)
  use ISO_C_BINDING        !Fortran 2003 版本须使用本句说明
  implicit none
  real(8) :: d
  dis_Function = sqrt(d)
end function
```

能源与动力工程学院

Thank You !