文章编号:1671 - 6833(2013)02 - 0064 - 03

Fortran 90 借用 C ++ 函数模板

毕苏萍[1], 周振红[2], 赫晓慧[2]

(1.郑州大学 土木工程学院,河南 郑州 450001; 2.郑州大学 水利与环境学院,河南 郑州 450001)

摘 要: 函数模板作为一种泛型编程方法,对科学与工程计算具有重要现实意义. 首先,探讨了 C ++ 函数模板的实现机制,揭示了函数模板与重载函数间的关系;运用 Fortran 90 泛型接口块结合外部例程,对 C ++ 重载函数进行了模拟;然后,在 C ++ 环境中添加包装子以实例化函数模板,并将函数模板实例转化成接口一致的"重载"例程,进而在 Fortran 90 环境中使用 C ++ 函数模板. 辅以示例程序,对相关的处理方法给予详细说明,也为其它语言借用 C ++ 函数模板提供了借鉴.

关键词: 科学与计算工程;泛型编程;函数模板;泛型接口块;混合编译

中图分类号: TP311 **文献标志码**: A doi:10.3969/j.issn.1671 - 6833.2013.00.017

0 引言

在计算机科学与技术领域,泛型编程(Generic Programming)具有广泛的意义. 用泛型编程先驱(Alexander Stepanov)的话来说:泛型编程是对算法、数据结构进行抽象和分类,其目标是递增式构造实用、高效、抽象的算法、数据结构的系统目录结构或框架[1]. 简言之,泛型编程是将算法、数据结构由具体的实例提升到一般、抽象的形式,使之可以操作不同的数据类型.

C ++ 提供了模板(包括类模板和函数模板),并逐步积累有相对完善的标准模板库 STL[2],对泛型编程给予了很好的支持. Fortran 从 77 到 90[3]、2003[4]对泛型编程的支持不断加强,但直至 2008 也没能提供模板工具[5]. 假如 Fortran 90 借用 C ++ 函数模板能够获得成果,那么无疑会极大地拓展 C ++ 的应用空间,给科学与工程计算增添新的活力. 笔者就此展开探讨,示例程序测试环境:C ++ 为 VC 6.0,Fortran 90 为 Compaq Visual Fortran 6.6.

1 C ++ 特殊的重载函数簇 — 函数模板

C ++ 支持函数重载,允许在参数表不同的前提下于同一编译单元定义几个同名函数,调用时依据参数表最佳匹配的原则自动选择合适的函数. 比如在编程计算中,1/2 整数除结果为 0,1.0/2.0实数除结果为 0.5,笔者用两个重载函数予以验证(当中的参数采取引用传递,和 Fortran 90 的参数传递保持一致):

```
int divid(int &a, int &b){return a / b;}
float divid (float &a, float &b) {return a / b;}.
```

测试上列重载函数的主函数为:

```
void main(void){
  int a = 1, b = 2;float x = 1.0, y = 2.0;
  cout << "1/2 =" << divid(a,b) << endl;//整数除
  cout << "1.0/2.0 =" << divid(x,y) << endl;}//实数除.
```

程序运行结果为:

```
1/2 = 0
1.0/2.0 = 0.5.
```

观察上列重载函数,不难发现两个特点:①是接口类同,惟有函数结果、参数的数据类型不同;②是算法相同. 在这种情况下,将上列重载函数抽象为一个函数模板、用一个泛型 T 代替函数结果、参数的数据类型:

```
template <typename T>// 亦可用 class 代替 typename 声明函数模板中的泛型
T divid(T &a, T &b) {return a / b;}// divid,a, b 的类型均为泛型 T
```

同样的测试主函数,当调用 divid(a, b)函数

收稿日期:2013 - 01 - 20;**修订日期**:2013 - 02 - 20

基金项目:国家自然科学基金资助项目(41101095)

作者简介:毕苏萍(1965 -),女,河南郑州人,郑州大学高级工程师,研究方向为结构工程及计算机应用.

时构造的是 divid < int > 函数模板实例,而当调用 divid(1.0, 2.0)函数时则构造的是 divid < float > 函数模板实例,分别与整型和实型重载函数 divid 相当,所以测试结果与上列重载函数的相同.说明上列重载函数与函数模板的效果完全相同,从而证明,可以将函数模板看成是一特殊的重载函数簇.

2　Fortran 90 模拟 C ++ 函数重载

要模拟 C ++ 函数重载,有必要先回顾一下 Fortran 90 接口块的引入. Fortran 90 共有 4 种程序单元:主程序、外部例程(子程序和函数统称为例程)、模块和数据块,当被调程序为外部例程时,为使编译器产生正确调用,Fortran 90 要求在调用程序中建立被调外部例程的接口块,以明确其接口信息:例程名、例程实现机制(函数,或者子程序)、函数类型、参数的类型、属性及传递方式.当被调外部例程接口简单时,是否在调用程序中建立其接口块是可选的;当接口复杂时,建立其接口块就成为必须的.比如:外部函数返回数组或变长字符串,参数中有可选参数,有假定形状数组、指针或目标属性参数,有例程参数(即例程作参数,类似于 C 语言中的函数指针作参数)等.接口块的构造形式为:

```
Interface
  Function/ Subroutine 例程名 (形参表) ! 接口
    形参声明(包括函数结果类型声明)
  End Function/ Subroutine
End Interface.
```

Fortran 90 不直接支持例程重载,不允许定义同名的外部例程,但允许将几个外部例程接口置于同一接口块内,并给接口块命名、以接口块名作为各个外部例程的统称,调用时依据接口匹配的原则自动选择相对应的外部例程,从而推出了支持泛型编程的接口块(姑且称为泛型接口块).

```
Interface 泛型接口块名
  接口体
End Interface.
```

其中,接口体由几个外部例程或者模块例程接口构成.

下面用 Fortran 90 实现前述 C ++ 函数重载示例.首先,用外部例程(函数)div_int 和 div_real 分别实现 C ++ 整数除和实数除重载函数,其实现代码只比各自的接口多一行.

```
  div_int = x/y 或 div_real = x/y
```

包含其泛型接口块(divid)的主程序为:

```
PROGRAM test_overloading
  Implicit None
  Interface divid! 泛型接口块
    Integer Function div_int(x, y) ! 外部例程接口
      Integer, Intent(IN) :: x, y
    End Function
    Real Function div_real(x, y) ! 外部例程接口
      Real, Intent(IN) :: x, y
    End Function
  End Interface
  WRITE ( * , * )'1/2 = ', divid(1, 2) ! 整数除
  WRITE ( * , * )'1.0/2.0 = ', divid(1.0, 2.0) ! 实数除
END PROGRAM.
```

程序运行结果为:

```
1/2 = 0
1.0/2.0 = 0.500 000 0.
```

调用程序使用了统一的泛型接口块名 divid,而真正调用的是与接口匹配的 div_int、div_real 外部例程或称为"重载"例程;C ++ 尽管重载函数名称相同,但由于编译时增加的特殊修饰其目标函数名并不相同,这样才有可能依据不同的参数表调用与之匹配的重载函数.可见:这里的外部例程加泛型接口块与 C ++ 重载函数的效果是相同的.

3　C ++ 函数模板实例化为"重载"例程

无论是 C ++ 的重载函数还是 C ++ 的函数模板,都只有在 C ++ 环境中才能直接调用或实例化,即便在其子集 C 语言中也无法直接使用.推想背后的道理,可能是编译器的功能所致. C ++ 编译器能够添加特殊的命名修饰,据此可以判明对应的重载函数或构造不同的函数模板实例;C 编译器无此功能,所以它不支持函数重载或函数模板,C ++ 的重载函数或函数模板也禁止使用 C 链接(其作用是消除 C ++ 编译器的特殊命名修饰).

前面笔者已经探讨过:Fortran 90 在泛型接口块的支持下,可以将普通外部例程当作是 C ++ 的重载函数,进而也可以看成是 C ++ 函数模板实例.这样一来,如果设法在 C ++ 环境中将函数模

板实例化为 Fortran 90“重载”例程,就可采取 C ++ 与 Fortran 的混合编译[6],从而在 Fortran 90 环境中使用 C ++ 函数模板.循这一思路,在前述 C ++ 函数模板示例代码下面增加包装子

```
extern "C"{
    int __stdcall DIV_INT(int &a, int &b) {return divid(a,b);}
    float __stdcall DIV_REAL(float &a, float &b) {return divid(a,b);}}
```

为使接口与 Fortran 90 的“重载”例程接口保持一致,上列设置采取 C 链接、__stdcall 调用约定、大写命名约定及引用参数传递方式.此处的包装子有两个作用:对内,实例化函数模板;对外,承担 Fortran 90“重载”例程.

将前述 C ++ 函数模板和包装子单独保存为一个文件(.cpp),并与 Fortran 90 主程序文件(.f90)置于同一项目.程序运行结果,与模拟 C ++ 函数重载示例的结果相同.

4 结论

将 C ++ 函数模板看成接口相似、算法相同的特殊重载函数簇,在泛型接口块支持下,将 Fortran 90 外部例程模拟成 C ++ 重载函数,然后在 C ++ 环境中添加包装子,将函数模板实例化成 Fortran 90“重载”例程,进而在 Fortran 90 环境中以正常方式使用 C ++ 函数模板.像 C 等其它语言要借用 C ++ 函数模板,也可采取同样的思路.

参考文献:

[1] ALEXANDER A. STEPANOV. Generic programming [EB/OL]. http://www.stepanovpapers.com/,2012.5.22.

[2] DAVID V, NICOLAI M J. C ++ Templates: The Complete Guide[M]. Addison Wesley,2003.

[3] 周振红,郭恒亮,张君静,等. Fortran 90/95 高级程序设计[M]. 郑州:黄河水利出版社,2005.

[4] Fortran 2003 standard[EB/OL]. http://www.j3-fortran.org/doc/year/04/04-007.pdf,2012.5.22.

[5] CHIVERS S. Introduction to programming with fortran with coverage of fortran 90, 95, 2003, 2008 and 77 [M]. Springer, 2012.

[6] 任慧,周振红,张成才. Fortran 与 C/C ++ 的混合编译[J]. 计算机工程与设计,2007,28(17):4096 - 4098、4111.

Fortran 90 Using Function Templates from C ++

BI Su-ping[1], ZHOU Zhen-hong[2], HE Xiao-hui[2]

(1. School of Civil Engineering, Zhengzhou University, Zhengzhou 450001, China; 2. School of Water Conservancy and Environment, Zhengzhou University, Zhengzhou 450001, China)

Abstract: The function template is a kind of generic programming, which possesses important practical significance in scientific and engineering computing. First, the implementing mechanism of C ++ function template is explored, and the relationship between function templates and overloaded functions is disclosed. C ++ overloaded functions are modeled with the generic interface block plus external procedures of Fortran 90. Then, a wrapper of C ++ function templates is appended in order to instantiate function templates, convert function template instances into "overloaded" procedures, and make use of C ++ function templates in Fortran 90. The above-mentioned methods are particularly demonstrated with a demo. The research will also be helpful in applying C ++ function templates in other languages.

Key words: scientific and engineering computing; generic programming; function template; generic interface block; mixed compiling