三种数组传递方式

Fortran 中，调用函数或子程序时，默认将实参的地址传递给形参，称为地址传递或引用传递。究其原因，是因为 Fortran 主要针对数值计算，参数多为大型数组（二维数组称矩阵），如果采用值传递，会复制实参的一个拷贝给形参，占用时间和内存，而地址传递则仅仅将实参数组的首地址传递给形参，没有时间和内存冗余。

这里介绍 3 种常见的数组传递方式。看下面的代码：

```
program test
  implicit none
  real a(2,3)
  interface
    subroutine fun3(a)
    real a(:,:)
    end subroutine
  end interface

  call fun1(a)
  write(*,'(6f3.0)') a
  call fun2(a,2,3)
  write(*,'(6f3.0)') a
  call fun3(a)
  write(*,'(6f3.0)') a
end program
```

```
! 方法 1
subroutine fun1(a)
  real a(*)
  a(1:6)=1
end subroutine
! 方法 2
subroutine fun2(a,m,n)
  integer m,n
  real a(m,n)
  a=2
end subroutine
! 方法 3
subroutine fun3(a)
  real a(:,:)
  a=3
end subroutine
```

执行结果：

三种方法的形参数组，第一种称假定大小数组(assumed-size arrays)，第二种称自动数组(auto arrays)，第三种称假定形状数组(assumed-shape arrays)。

第一种：可对数组元素或数组片段进行操作（如 a(3)=2, a(2:4)=3 ），但不能直接对数组名进行操作(如 a=1)。如果对整个数组进行操作，需指明数组边界（如代码所示）；

第二种：最常见，也最常用，不解释；

第三种：可用 size 函数获取数组每一维的元素个数：m = size(a,1)，n = size(a,2)，操作与第二种一致。注意：这种方式需要显式接口，可用 interface 指定接口，或将子程序写入 module 中使用。

在某些老代码中，可能会见到第四种写法，其与第一种类似。

```
! 方法 4
subroutine fun4(a)
  real a(1)
  a(1:6)=4   !全部 6 个元素赋值为 4
  a = 0      !第一个元素赋值 0，其余不变
end subroutine
```

总结：

第一种将高维数组变形为 1 维数组，丢失了数组的维度信息，实参和形参元素的位置对应关系不确定。对于 2×3 的数组，其在内存中的排列顺序有两种：1、按列存储 a11,a21,a12,a22,a13,a23；2、按行存储 a11,a12,a13,a21,a22,a23. Fortran 标准并未对此做规定，其存储方式取决于编译器本身，因此不建议使用。

第二种最常用，但需要传递额外的参数来指定数组大小。

第三种很灵活，能实现第二种的所有功能，而且减少了参数个数，但需要显式接口。

第四种则坚决反对使用。