CGNS 用户指南

1 引言

本用户指南的编写是为了帮助用户执行 CGNS（CFD 通用符号系统）。其基本原则是：轻内容，重实例、建议和指导方针。对其他详细情况感兴趣的读者请参看参考文献中所列的其他文章，也可以通过 www.cgns.org 网站查询。

1.1 什么是 CGNS?

CGNS（CFD 通用符号系统）是由波音公司与 NASA 于 1994 年共同合作创建的，并且现已发展到有世界上其他多家有影响的机构参与。该系统努力使 CFD 的输入和输出标准化，包括网格（结构网格与非结构网格）、流动解、连接性、BC 以及辅助信息。CGNS 也很容易扩展，并允许给文件打上标记和用户插入注释。它采用 ADF（高级数据格式）系统，该系统创建了二进制文件，这种文件便于跨越计算机平台。CGNS 还包括第二层软件，称为中级程序库，或 API（应用程序接口），它使得更容易执行进入现有 CFD 程序的 CGNS。

1999 年，CGNS 的管理完全移交给一个公共组织，叫做 CGNS 管理委员会。它是由各国政府和私有工业集团派出的国际代表组成，所有 CGNS 软件都全面对每个人免费和公开（开放式资源）。CGNS 标准也是使流体动力学数据实现 ISO 标准化的目的，2000 年中尽早交付。

1.2 为什么需要 CGNS?

CGNS 最终将会消除目前工作于机器之间和 CFD 程序之间时所需的大部分翻译程序。而且，最终可允许从一个程序出来的结果，很容易地利用另一个程序重新开始。这就有望节约大量的时间和资金。特别是，希望未来的网格生成软件生成的网格具有完全连接性和所包含的 BC 信息，作为 CGNS 数据库的一部分，从而节约时间，并避免在事后组合这种信息的过程中可能造成的昂贵的误差。

1.3 什么是 CGNS 文件?

CGNS 文件是一个整体，编成（在文件自身内部）一组呈树状结构的“节点”，其方式与在 UNIX 环境下编制指南大致相同[1]。最顶端节点称为“根节点”。根节点下的每一个节点都由名称和标示定义，可能含有也可能不含有信息或数据。每一个节点也可能是一个“父”节点，含有一个或几个“子”节点。一个节点也可以作为一个子节点，与文档中其他地方的节点链接或与单独的 CGNS 文档中的节点链接。这些链接对用户是透明的：用户可以“看见”链接的子节点，就好象它们真实存在于现在的树中。CGNS 的树状结构图如图 1 所示。

为了使任一用户都可以理解 CGNS 文件，其节点必须根据专门的规则汇编。例如，图 2 列出了根据我们大多数人都很熟悉的规则而列举的动物种类的一个简单的树状结构图。（注意，该

[1] 严格地讲，由于链接可能用来存储多个文件中的信息，因此，没有 CGNS 文件的概念，而只有在一个或多个文件中执行的 CGNS 数据库的概念。但是，在本文中，这两种表达法可以互换使用。

图与图 1 不同之处在于它没有使用“标示”或“数据”，只有“名称”。）在该结构图中，越往下走其类别范围也越来越窄。类别范围最宽的为“动物”，慢慢变窄到具体的狗的类别（有两只“Fido”、一只“Spot”和一只“Ginger”）。如果提前了解该树是如何构造的，就可以快速并轻松地从树结构中找到你可能感兴趣的具体某项信息。如果另有人根据完全不同的规则，构造出完全不同形式的同样动物界的图的话，若不花费大量时间查找和研究该树的结构，就很难获得所需要的信息。

《标准接口数据结构（SIDS）》中介绍了编制用于气动数据的 CGNS 文件的专门规则，该规则可以使用户轻松获取所需信息。由于 CGNS 是二进制文件，用户用 UNIX ASCII 编辑 工具无法看到。创造出 Adfedit，使用户可以轻松获取 CGNS 文件。具体情况参见附录 A。

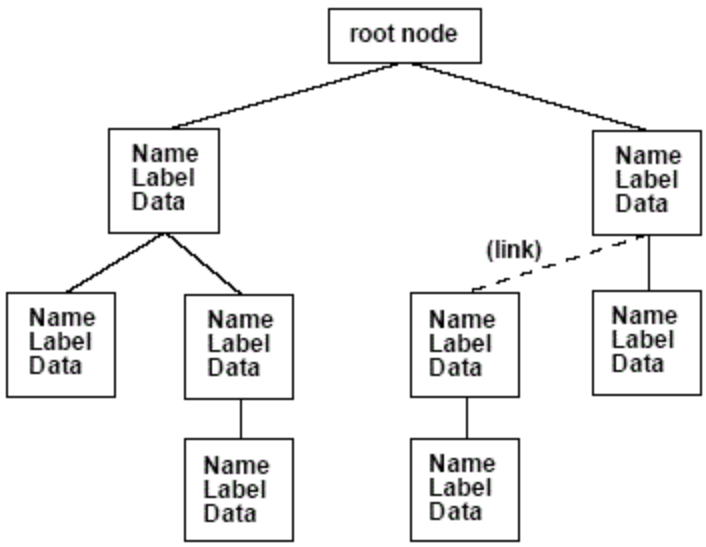


图 1　CGNS 树状结构示例

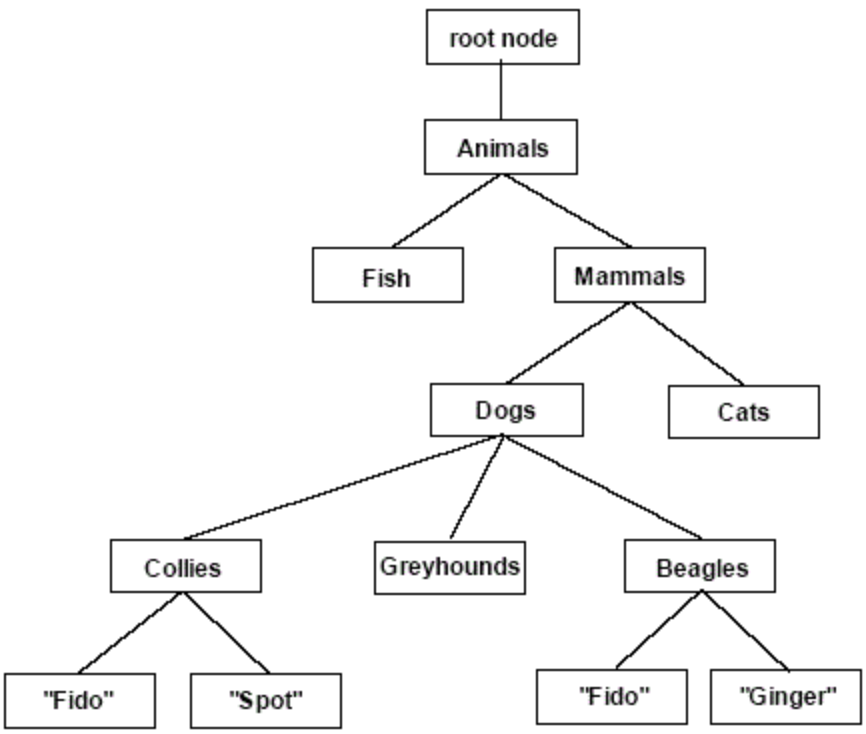


图 2　动物界种类的简易树状结构图

1.4 用户指南的组织结构

本指南的主要内容在第 2 节，列举了几个结构网格和非结构网格的简单示例。该节内容包含了大多数用户为开始运用 CGNS 而想要或需要学习的基本内容。即使用户只关心非结构网格的应用，但作为一个整体，笔者也建议读者首先阅读关于结构网格的内容。本指南的另外一些内容在第 3 节介绍；虽然觉得该部分内容很重要，但并不象第 2 节所介绍的基本内容那么至关重要。最后，第 4、5 节分别简要介绍了查找故障，以及常常问到的一些问题。

需要注意的是，本文所给出的所有程序和程序段都可以从 CGNS 的 www.cgns.org 网站获取。这些程序的名称及其功能参见附录 B。还要注意的是，本文并不包括所有的 CGNS 的功能，只是起到一个简要指南的作用。

2 开始使用

指导在 CGNS 文件中如何构成节点的规则和习惯，包括它们的名称和标示，在 SIDS 文档中作了详细说明，其他内容见参考文献[2，3]。这些文档还详述了 CFD 信息是如何以标准化方式存储在节点中的，便于用户容易进入并读取。如果 CGNS 文件严格遵循 SIDS 所给出的规则，则称为“SIDS 相容的”。CGNS 文件必须是相容 SIDS 的，以便其他用户可以正确理解。附录 C 简要介绍了最通用的 SIDS。

但是，为开始使用 CGNS，用户不必完全理解 SIDS 文件或附录 C。已创建了中级程序库或 API 调用，用以帮助用户编写和阅读相容 SIDS 的 CGNS 文件[2]。利用 API，大多数用户感兴趣的大多数 CFD 数据可以容易地写入 CGNS 文件或从中读出，只需对 SIDS 有初步了解。

在接下来的部分中，我们详细介绍有关如何创建典型的 CGNS 文件或一部分文件的细则。这些细则以简单例子的形式给出。这要使用中级 API 调用，尽管本文并不包括所有的 API 调用（完整的可利用的 API 调用可见参考文献[5]）。笔者建议用户按顺序通读本节中所列出的示例，因为后面几节中的一些信息建立在对前文所给出信息的了解上。所希望的是，用户应该能够将这些简单示例推广到他们的应用中。第 3 介绍了另外一些应用情况。对于那些已经熟悉用于 CFD 数据的 PLOT3D 格式的用户，我们在附录 D 中详细介绍了在 CGNS 文件中如何读和写 PLOT3D 类型的变量。

还需要注意的是，我们推后了对单位和非量纲化的讨论，把它放在了第 3 节。目前，所有的示例都只是简单的存储和检索单纯的数，假设用户知道每个变量的量纲值或无量纲值。

2.1 结构网格

本节给出几个结构网格的示例，第 2.2 节则给出非结构网格的示例。不过，即使用户只关心非结构网格的应用，但作为一个整体，我们也建议先阅读一下第 2.1 节中的内容。这是因为对于两种网格类型，CGNS 文件的大部分构造都是相同的，因此文章后面部分假设用户已经熟悉前文中提及的信息。

2.1.1 单块结构网格

第一个示例是一个非常简单的 3-D 笛卡尔网格，其大小为 21×17×9。网格点本身采用下列 FORTRAN 程序段建立。

---

[2] 目前有两个能利用 CGNS 的编程级别，最低的级别包括 ADF 级调用。这些调用完成大部分基本功能，比如创建子节点、写数据、读数据，等等。这些功能总是用字符“ADF”开始。但是，这些低级别的调用完全不了解 SIDS，因此，用户负责将数据放入正确的位置，以使 CGNS 文件相容 SIDS。中级程序库，或 API 调用，用 SIDS 的经验来编写，它总是用字符“cg_”开始。所以，当用 API 调用编写 CGNS 文件时，更容易遵循 SIDS 标准，并且当存取 CGNS 文件时，对相容 SIDS 情况的某些检测也通过 API 调用来完成（并不保证相容 SIDS，但 API 调用大大有助于简化它）。API 调用还大大缩短了调用序列，它是实现创建和阅读 CGNS 文件所需要的许多功能所必需的。

```
do k=1,nk
   do j=1,nj
      do i=1,ni
         x(i,j,k)=float(i-1)
         y(i,j,k)=float(j-1)
         z(i,j,k)=float(k-1)
      enddo
   enddo
enddo
```

其中，ni=21，nj=17，nk=9。网格图如图 3 所示。

下面给出完整的 FORTRAN 程序，该程序建立了上述网格并用 API 调用将它写入称为 grid.cgns 的 CGNS 文件（注意，FORTRAN 行的继续用一个“+”表示）。这个（以及所有后面的）编程示例都可以通过 CGNS 网站 www.cgns.org 查到（在“用户指南”部分）。可参见附录 B 。

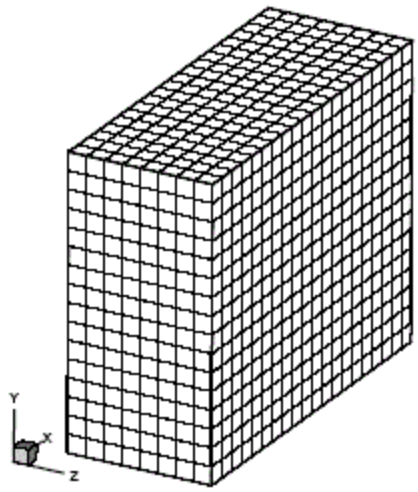


图 3　简单的笛卡尔结构网格

```
      program write_grid_str
c
c Creates simple 3-D structured grid and writes it to a
c CGNS file.
c
c This program uses the fortran convention that all
c variables beginning with the letters i-n are integers,
c by default, and all others are real
c
c UNIX compilation (IRIX 5.3 or higher with mips4_64 option)
c for this program is:
c f90 -r8 -64 write_grid_str.f CGNSLib/lib/libcgns.mips4_64.a
```

```
c (CGNSLib/lib/ is the location where the compiled
c library libcgns.mips4_64.a is located)
c (Note it is compiled double precision because RealDouble
c is used below)
c
c must include path to cgnslib_f.h file:
   include 'CGNSLib/cgnslib_f.h'
c dimension statements (note that tri-dimensional arrays
c x,y,z must be dimensioned exactly as (21,17,N) (N>=9)
c for this particular case or else they will be written to
c the CGNS file incorrectly!  Other options are to use 1-D
c arrays, use dynamic memory, or pass index values to a
c subroutine and dimension exactly there):
   dimension x(21,17,9),y(21,17,9),z(21,17,9)
   dimension isize(3,3)
   character basename*32,zonename*32
c
c create gridpoints for simple example:
   ni=21
   nj=17
   nk=9
   do k=1,nk
      do j=1,nj
         do i=1,ni
            x(i,j,k)=float(i-1)
            y(i,j,k)=float(j-1)
            z(i,j,k)=float(k-1)

         enddo
      enddo
   enddo
   write(6,'('' created simple 3-D grid points'')')
c
c WRITE X, Y, Z GRID POINTS TO CGNS FILE
c open CGNS file for write
   call cg_open_f('grid.cgns',MODE_WRITE,index_file,ier)
c create base (user can give any name)
   basename='Base'
   icelldim=3
   iphysdim=3
   call cg_base_write_f(index_file,basename,icelldim,iphysdim,
+    index_base,ier)
c define zone name (user can give any name)
   zonename = 'Zone  1'
c vertex size
   isize(1,1)=21
   isize(2,1)=17
   isize(3,1)=9
c cell size
   isize(1,2)=isize(1,1)-1
```

```
      isize(2,2)=isize(2,1)-1
      isize(3,2)=isize(3,1)-1
c boundary vertex size (always zero for structured grids)
      isize(1,3)=0
      isize(2,3)=0
      isize(3,3)=0
c create zone
      call cg_zone_write_f(index_file,index_base,zonename,isize,
+       Structured,index_zone,ier)
c write grid coordinates (user must use SIDS-standard names here)
      call cg_coord_write_f(index_file,index_base,index_zone,RealDouble,
+       'CoordinateX',x,index_coord,ier)
      call cg_coord_write_f(index_file,index_base,index_zone,RealDouble,
+       'CoordinateY',y,index_coord,ier)
      call cg_coord_write_f(index_file,index_base,index_zone,RealDouble,
+       'CoordinateZ',z,index_coord,ier)
c close CGNS file
      call cg_close_f(index_file,ier)
      write(6,'('' Successfully wrote grid to file grid.cgns'')')
      stop
      end
```

关于该程序，有几方面需要注意。每当采用 API 建立一个新实体时，都要返回一个整数索引。该索引用于后面的 API 调用中以参照实体。比如，上述打开 grid.cgns 文件的调用 cg_open_f，分配索引 index-file 给该实体。同样的 index_file 用来确定随后的调用中的实体。与此相似，cg_base_write_f 分配索引 index_base 给库，cg_zone_write_f 分配索引 index_zone 给块，cg_coord_write_f 分配索引 index_coord 给每一个坐标。

对于 FORTRAN 程序，必须列出指向 cgnslib_f.h 的包含语句。(cgnslib_f.h 文件包含在 CGNS 软件内。) 另外，x、y、z 的大小必须精确定为 (21, 17, N)，在该例中 N≥9（要么定为至少是 21*17*9 的一维数组)；这是因为，cg_coord_write_f 程序写入包括在数组中的第一个 21*17*9 值，与它保存在内存中一样。如果 x、y、z 是三维数组，而且前两个索引分别大于 21 和 17，那么在 CGNS 文件中就会置入不正确的值。在实际工作程序中，我们可能会：(a) 使用一维数组；(b) 为 x、y、z 动态分配适当的内存；(c) 传递索引值给子程序，并通过一个大小恰当的工作数组写入。

在该例中，单元维数为 3（因为网格由体积单元组成)，实际维数为 3（因为 3 个坐标确定 3 维)。(更详细情况参看附录 C。) isize 数组包含每个索引方向的顶点数组的大小、单元数组的大小和边界数组的大小。对于 3 维结构网格，索引维数总是与单元维数相同，这就意味着有 3 个顶点数组的大小，3 个单元数组的大小，3 个边界顶点数组的大小 (i, j, k 方向各一个)。对于结构网格，单元数组的大小总是比相应的顶点数组的大小少 1，而边界顶点数组的大小往往没有意义，总为零。当写网格坐标时，用户必须使用 SIDS-标准名称。比如，x、y、z 坐标必须分别命名为 CoordinateX、CoordinateY、CoordinateZ。对于其他可能的选择，有其他的标准名称 (见参考文献[1])。最后，basename 和 zonename 必须表示为字符串，必须恰当确定整数数组的大小。

网格坐标数组可以单倍或双倍的精度写入。采用关键字 RealSingle 或 RealDouble，将想要的数据类型传递给 API。用户必须确保传送到 API 的数据类型与表示坐标数组所采用的类型一致。在编译时，程序也必须链接到编译的 CGNS 程序库 libcgns.xxx.a 上，其中 xxx 是根据要在其上执行程序的计算机系统设定的。包含在 CGNS 软件内的 README 文件中，介绍了编译

CGNS 程序库的说明，或者，适用于给定系统的预先编译的 libcgns.xxx.a 程序库也可从 CGNS 网站下载。

这里给出了用 C 语言编写的完整程序，它完成了同样的任务，即创建网格坐标并将它们写入 CGNS 文件中。

```
/*
Creates simple 3-D structured grid and writes it to a
CGNS file.

UNIX compilation (IRIX 5.3 or higher with mips4_64 option)
for this program is:
cc -r8 -64 write_grid_str.c CGNSLib/lib/libcgns.mips4_64.a
(CGNSLib/lib/ is the location where the compiled
library libcgns.mips4_64.a is located)
(Note it is compiled double precision because RealDouble
is used below)
*/

#include <stdio.h>
/* must include path to cgnslib.h file:  */
#include "CGNSLib/cgnslib.h"

main()
{
/*
   dimension statements (note that tri-dimensional arrays
   x,y,z must be dimensioned exactly as [N][17][21] (N>=9)
   for this particular case or else they will be written to
   the CGNS file incorrectly!  Other options are to use 1-D
   arrays, use dynamic memory, or pass index values to a
   subroutine and dimension exactly there):
*/
   double x[9][17][21],y[9][17][21],z[9][17][21];
   int isize[3][3];
   int ni,nj,nk,i,j,k;
   int index_file,icelldim,iphysdim,index_base;
   int index_zone,index_coord;
   char *basename,*zonename;

/* create gridpoints for simple example:  */
   ni=21;
   nj=17;
   nk=9;
   for (k=0; k < nk; ++k)
   {
      for (j=0; j < nj; ++j)
      {
         for (i=0; i < ni; ++i)
         {
```

```
            x[k][j][i]=i;
            y[k][j][i]=j;
            z[k][j][i]=k;

         }
      }
   }
   printf("\ncreated simple 3-D grid points");
/* WRITE X, Y, Z GRID POINTS TO CGNS FILE */
/* open CGNS file for write */
   cg_open("grid_c.cgns",MODE_WRITE,&index_file);
/* create base (user can give any name) */
   basename="Base";
   icelldim=3;
   iphysdim=3;
   cg_base_write(index_file,basename,icelldim,iphysdim,&index_base);
/* define zone name (user can give any name) */
   zonename="Zone 1";
/* vertex size */
   isize[0][0]=21;
   isize[0][1]=17;
   isize[0][2]=9;
/* cell size */
   isize[1][0]=isize[0][0]-1;
   isize[1][1]=isize[0][1]-1;
   isize[1][2]=isize[0][2]-1;
/* boundary vertex size (always zero for structured grids) */
   isize[2][0]=0;
   isize[2][1]=0;
   isize[2][2]=0;
/* create zone */
   cg_zone_write(index_file,index_base,zonename,*isize,Structured,
      &index_zone);
/* write grid coordinates (user must use SIDS-standard names here) */
   cg_coord_write(index_file,index_base,index_zone,RealDouble,"CoordinateX",
      x,&index_coord);
   cg_coord_write(index_file,index_base,index_zone,RealDouble,"CoordinateY",
      y,&index_coord);
   cg_coord_write(index_file,index_base,index_zone,RealDouble,"CoordinateZ",
      z,&index_coord);
/* close CGNS file */
   cg_close(index_file);
   printf("\nSuccessfully wrote grid to file grid_c.cgns\n");
}
```

注意，在 C 语言中，必须包含“.h”的文件称为 cgnslib.h。从现在起，所有的程序都将只用 FORTRAN 语言编写。与 C 语言相当的调用是相似的，见上面所述。另外，从现在起，为节省空间，不再列出完整的程序，而只是列出程序段。不过，用户可以从 CGNS 网站 www.cgns.org 获得完整的程序（在“用户指南”一节）。

由上述程序创建的 CGNS 文件 grid.cgns 是一个二进制程序，在其内部拥有一个树状结构，如图 4 所示。如引言中所提到的，每个节点都有一个名称和一个标示，可能包含与可能不包含

数据。在图 4 所示的例子中，除 GridCoordinates 节点外，所有节点都包含有数据，MT 表示没有数据。

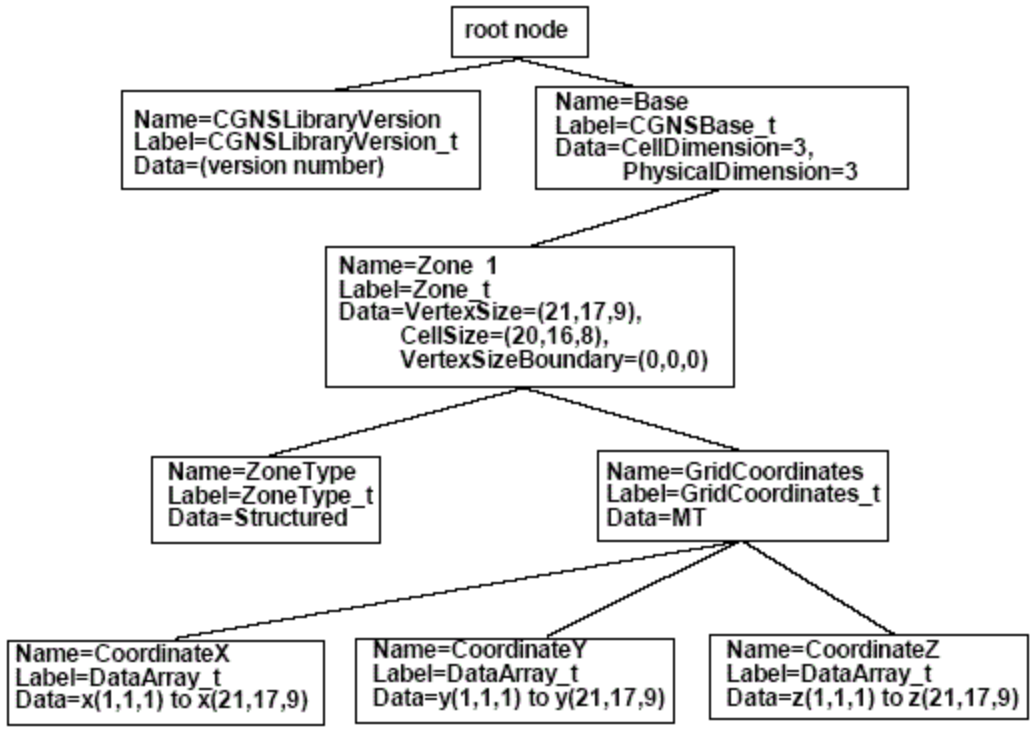


图 4　用于简单的笛卡尔网格坐标的 CGNS 文件结构图

但是，用户实际上无需了解本例中树状结构的所有细节。API 已自动生成了相容 SIDS 的 CGNS 文件！现在，用户可以用 API 轻松地阅读 CGNS 文件。我们刚才创建的用于阅读 CGNS 文件 grid.cgns 的 FORTRAN 程序段如下：

```
c READ X, Y, Z GRID POINTS FROM CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for read
   call cg_open_f('grid.cgns',MODE_READ,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1

c we know there is only one zone (real working code would check!)
   index_zone=1
c get zone size (and name - although not needed here)
   call cg_zone_read_f(index_file,index_base,index_zone,zonename,
+    isize,ier)
c lower range index
   irmin(1)=1
   irmin(2)=1
   irmin(3)=1
c upper range index of vertices
   irmax(1)=isize(1,1)
   irmax(2)=isize(2,1)
   irmax(3)=isize(3,1)
```

```
c read grid coordinates
   call cg_coord_read_f(index_file,index_base,index_zone,
+    'CoordinateX',RealSingle,irmin,irmax,x,ier)
   call cg_coord_read_f(index_file,index_base,index_zone,
+    'CoordinateY',RealSingle,irmin,irmax,y,ier)
   call cg_coord_read_f(index_file,index_base,index_zone,
+    'CoordinateZ',RealSingle,irmin,irmax,z,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

注意，该 FORTRAN 程序段还非常不成熟。假设我们知道只有一个库和一个块。在实际的工作程序中，应该检查文件中的数量，要么允许有多个库或块存在的可能性，要么就明确禁止。另外，该程序段还不明显地假设，grid.cgns 文件是一个 3 维结构网格（单元维数=实际维数=3）。在实际工作程序中，应该进行检查以确保这是真实的，或者允许其他的可能性存在。如果希望这样，还应该检查以确保块类型是结构型的（Structured）。

如前所述，在本例中，必须正确确定 x、y、z 数组的大小：对于 3 维数组，为（21, 17, N），其中 N≥9。（在实际工作程序中，我们可能会：（a）使用一维数组；（b）在阅读 isize 之后，为 x、y、z 动态分配适当的内存；（c）传递 isize 值给子程序，并在阅读之前恰当确定工作数组的大小。）还要注意的是，不管网格坐标写入 CGNS 文件的精度如何（单倍或双倍），人们都可以任何一种方式阅读：API 自动完成翻译。（上述程序中的数组 x、y、z，如果采用 RealSingle，就必须表示为单精度的，如果采用 RealeDouble，就必须表示为双精度的。）最后，应该恰当确定 isize 的大小，zonename 应该表示为字符变量，应该恰当确定 irmin 和 irmax 的大小。

2.1.2 单块结构网格和流动解

在本节中，我们写入与 2.1.1 节的网格有关的流动解。我们假设有两个可利用的流动解数组：静密度和静压。为了阐述三个重要方案，我们将说明（a）在顶点，（b）在单元中心，（c）在单元中心加外层单元这三种不同情况下如何写流动解。

（a）顶点处的流动解

第一个方案以图 5 中的 2-D 情况进行说明。简单地说，顶点处的流动解位于与网格点相同的位置。假设网格点已写入 CGNS 文件，下面的 FORTRAN 程序段增加了顶点处的流动解。

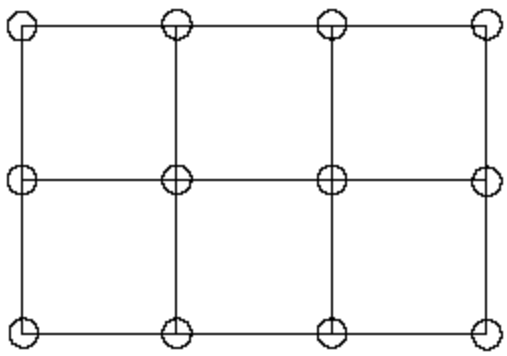


图 5　与网格有关的顶点流动解的位置（圆圈）示意图

```
c WRITE FLOW SOLUTION TO EXISTING CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c we know there is only one zone (real working code would check!)
   index_zone=1
c define flow solution node name (user can give any name)
   solname = 'FlowSolution'
c create flow solution node
   call cg_sol_write_f(index_file,index_base,index_zone,solname,
+    Vertex,index_flow,ier)
c write flow solution (user must use SIDS-standard names here)
   call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+    RealDouble,'Density',r,index_field,ier)
   call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+    RealDouble,'Pressure',p,index_field,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

在本程序中，对此特殊情况，必须正确确定密度（r）和压力（p）变量的大小：对于 3 维数组，为（21, 17, N），其中 N≥9（见 2.1.1 节）。注意，已知流动解类型为 Vertex，API 自动写出正确的索引范围，与块的网格索引范围相对应。还要注意的是，我们打开现有的 CGNS 文件并修改它（MODE_MODIFY）——我们之前就知道只有一个库和一个块；实际工作程序要进行适当地检测。最后 solname 应该表示为字符变量，当采用 RealDouble 时，r 和 p 必须表示为双精度变量。

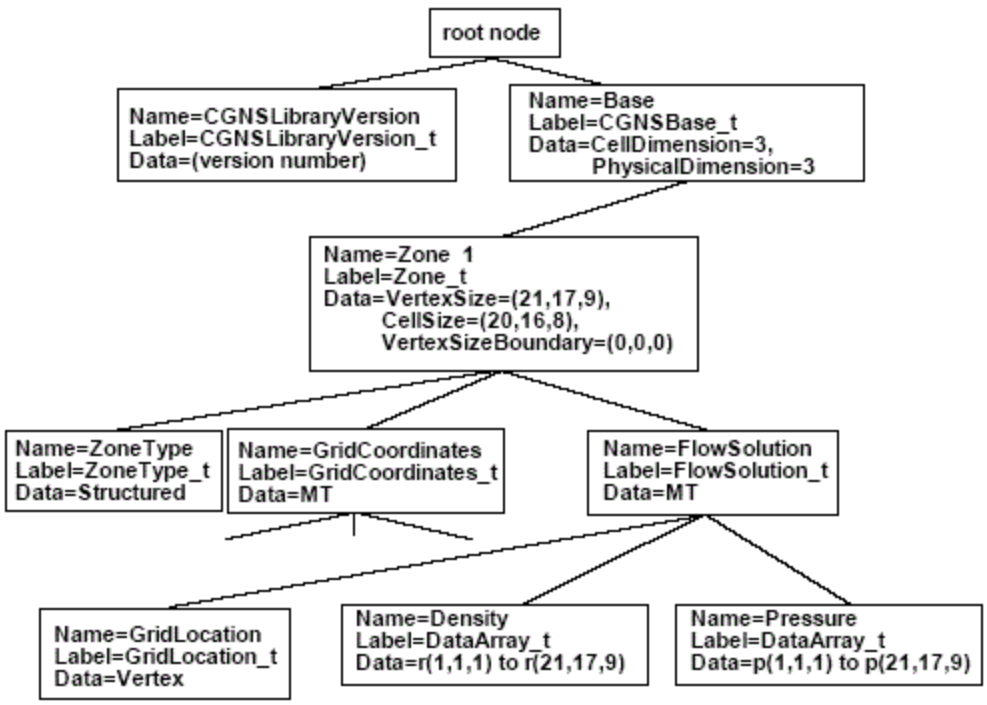


图 6　顶点处流动解的简单笛卡尔结构网格的 CGNS 文件结构图(注意：由于 GridLocation =Vertex 是缺省值，就不必详细说明。事实上，最后的 API 软件不包括在文件的此节点内)

图 6 示出了顶点处流动解的 CGNS 文件结构图。为节省空间，图中去掉了在 GridCoordinates_t 下的三个节点，但当用 3 个不连续的行表示时，它们是存在的。

通过利用下列 FORTRAN 程序段，可以阅读顶点处的流动解（可以单精度或双精度阅读——见 2.1.1 节中的讨论）。

```
c READ FLOW SOLUTION FROM CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for read
   call cg_open_f('grid.cgns',MODE_READ,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c we know there is only one zone (real working code would check!)
   index_zone=1
c we know there is only one FlowSolution_t (real working code would check!)
   index_flow=1
c get zone size (and name - although not needed here)
   call cg_zone_read_f(index_file,index_base,index_zone,zonename,
+    isize,ier)
c lower range index
   irmin(1)=1
   irmin(2)=1
   irmin(3)=1
c upper range index - use vertex dimensions
c checking GridLocation first (real working code would check
c to make sure there are no Rind cells also!):
   call cg_sol_info_f(index_file,index_base,index_zone,index_flow,
+    solname,loc,ier)
   if (loc .ne.  Vertex) then
      write(6,'('' Error, GridLocation must be Vertex!'')')
      stop
   end if
   irmax(1)=isize(1,1)
   irmax(2)=isize(2,1)
   irmax(3)=isize(3,1)
c read flow solution
   call cg_field_read_f(index_file,index_base,index_zone,index_flow,
+    'Density',RealSingle,irmin,irmax,r,ier)
   call cg_field_read_f(index_file,index_base,index_zone,index_flow,
+    'Pressure',RealSingle,irmin,irmax,p,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

注意，该程序段假设已知流动解不包含外层数据（下面给予详细介绍）。如果的确存在外层数据，而用户又不计算它的话，那么，流动解信息将会读取错误。因此，实际工作程序将会检测外层数据，并恰当调整大小和索引范围。这里还要给出其他类似的警告，比如早先关于确定变量大小以及工作程序检测的提醒等等。从此以后，不会总是重复这类警告。

（b）单元中心处的流动解

在单元中心输出流动解的方案如图 7 所示。以 2-D 图解形式绘出。在通过 4 个环绕的网格点确定的单元中心，确定流动解。在 3-D 情况下，单元中心通过 8 个环绕的网格点确定。写入单元中心的程序段与上述顶点处给出的程序段相同，只是调用 cg_sol_write_f 被下列程序段代替：

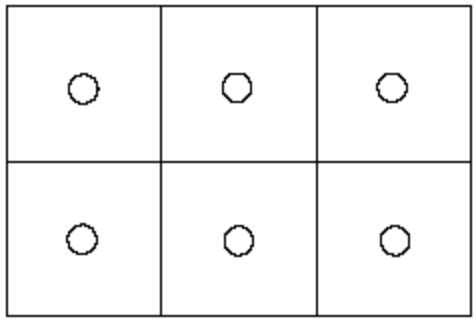


图 7　与网格有关的单元中心流动解的位置（圆圈）示意图

```
c create flow solution node (NOTE USE OF CellCenter HERE)
   call cg_sol_write_f(index_file,index_base,index_zone,solname,CellCenter,
+    index_flow,ier)
```

另外，对此特殊情况，现在必须正确确定密度（r）和压力（p）变量的大小：对于 3 维数组，为（20, 16, N），其中 N≥8（即，在每个索引维数内比网格自身少 1）。此外，已知流动解类型为 CellCeter，API 自动写出正确的索引范围，与块的网格索引范围在每个索引方向上减去 1 相对应。

单元中心处流动解的 CGNS 文件结构图如图 8 所示（仅在 FlowSolution_t 节点下）。注意，在其上写流动解的顶点现在是从（1, 1, 1）到（20, 16, 8）（与图 6 中的 FlowSolution 部分对照）。

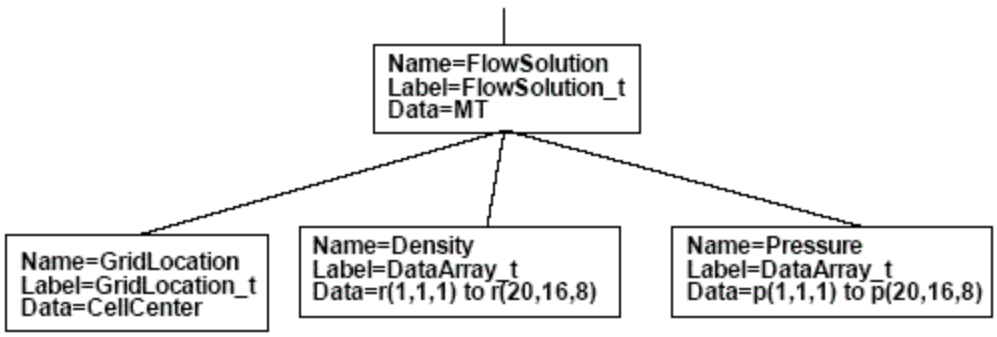


图 8　单元中心处流动解的简单笛卡尔结构网格的 CGNS 文件（在 FlowSolution_t 节点下）的结构图

读取单元中心处流动解的 FORTRAN 程序段与上述顶点处给出的程序段相同，只是定义 irmax 的部分被下列程序代替：

```
c upper range index - use cell dimensions
c checking GridLocation first (real working code would check
c to make sure there are no Rind cells also!):
```

```
      call cg_sol_info_f(index_file,index_base,index_zone,index_flow,
     +   solname,loc,ier)
      if (loc .ne.  CellCenter) then
         write(6,'('' Error, GridLocation must be CellCenter!'')')
         stop
      end if
      irmax(1)=isize(1,2)
      irmax(2)=isize(2,2)
      irmax(3)=isize(3,2)
```

并且，仍然必须正确确定 r 和 p 数组的 大小。

（c）附加有外层数据的单元中心处的流动解

外层数据是在结构网格外面附加的流动解数据，它在“虚假单元”位置处。外层数据可与除单元中心(CellCenter)外的其他网格位置(GridLocation)值有关，尽管我们在此处只用单元中心(CellCenter)给出了示例。输出附加有外层数据的单元中心流动解的方案如图 9 中 2-D 情况所示。在图中，我们在网格自身下面的行中给出一行外层单元数据。在网格的其他面上也可能有外层数据，或在给定面上可能有不只一行外层数据。

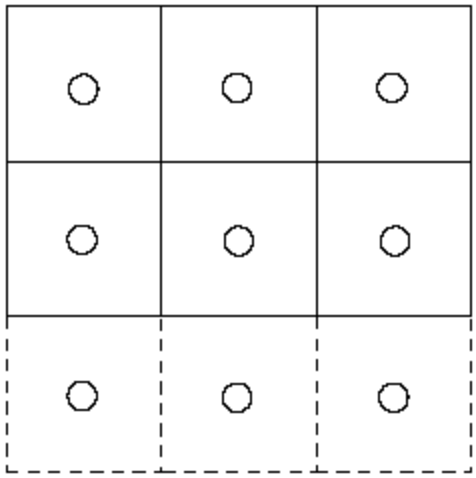


图 9　单元中心流动解的位置（圆圈）示意图，包括与网格有关的外层单元

在 CGNS 中，外层单元处的流动解并不存储为单独的实体，而是将流动解的范围延伸到包括外层单元。比如，在图 9 的 2-D 情况中，压力的索引范围 p（3, 2）不是存储在单元中心，而是流动解现在有一个索引范围 p（3, 0∶2）或 p（3, 3）。详见参考文献[1]。

对于 3-D 例子，我们假设在块的 4 个面（ilo, ihi, jlo, jhi，它们分别代表 i 和 j 方向的低端和高端）上有一行外层数据，在 klo 或 khi 上没有外层单元。编写流动解和外层数据的程序段如下：

```
c WRITE FLOW SOLUTION TO EXISTING CGNS FILE
      include 'cgnslib_f.h'
c open CGNS file for modify
      call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
      index_base=1
```

```
c we know there is only one zone (real working code would check!)
   index_zone=1
c define flow solution node name (user can give any name)
   solname = 'FlowSolution'
c create flow solution node
   call cg_sol_write_f(index_file,index_base,index_zone,solname,CellCenter,
+    index_flow,ier)
c go to position within tree at FlowSolution_t node
   call cg_goto_f(index_file,index_base,ier,'Zone_t',index_zone,
+    'FlowSolution_t',index_flow,'end')
c write rind information under FlowSolution_t node (ilo,ihi,jlo,jhi,klo,khi)
   irinddata(1)=1
   irinddata(2)=1
   irinddata(3)=1
   irinddata(4)=1
   irinddata(5)=0
   irinddata(6)=0
   call cg_rind_write_f(irinddata,ier)
c write flow solution (user must use SIDS-standard names here)
   call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+    RealDouble,'Density',r,index_field,ier)
   call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+    RealDouble,'Pressure',p,index_field,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

注意，在外层数据中，用户必须利用 cg_goto_f 调用恰当确定 Rind_t 节点的位置。在此情况下，Rind_t 节点隶属于 FlowSolution_t 节点之下。

对于具有外层数据的单元中心流动解的情况，用下列索引范围将密度（r）和压力（p）写入 CGNS 文件：从 i=0 到 i=20+1=21（或者 i 的总长度为 22）；从 j=0 到 j=16+1=17（或者 j 的总长度为 18）；从 k=0 到 k=8。为反映这些通过外层值修改的索引范围，必须恰当确定变量 r 和 p 的大小。

图 10 给出了本示例（仅在 FlowSolution-t 节点下）的 CGNS 文件结构形式图。将此图与图 6 和图 8 进行比较。

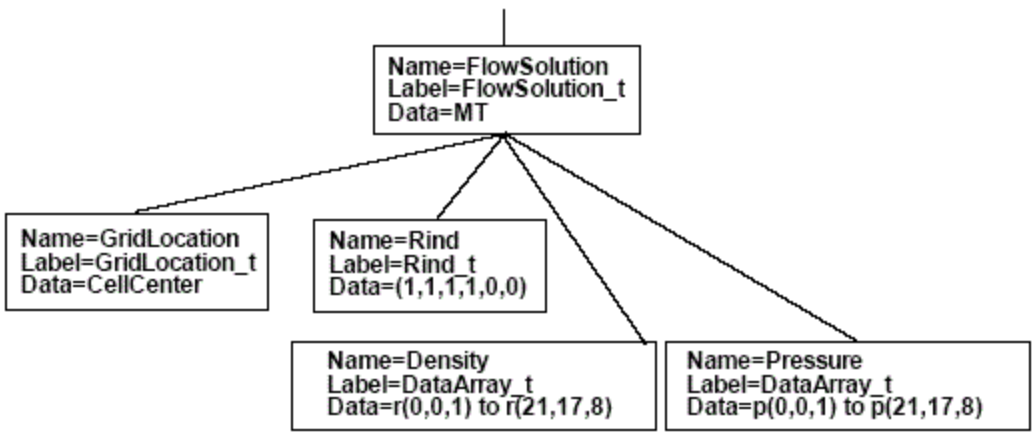


图 10　单元中心流动解加外层数据的简单笛卡尔结构网格的 CGNS 文件（在 FlowSolution_t 节点下）的结构图

读取本示例流动解的 FORTRAN 程序段如下。

```
c READ FLOW SOLUTION FROM CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for read
   call cg_open_f('grid.cgns',MODE_READ,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c we know there is only one zone (real working code would check!)
   index_zone=1
c we know there is only one FlowSolution_t (real working code would check!)
   index_flow=1
c get zone size (and name - although not needed here)
   call cg_zone_read_f(index_file,index_base,index_zone,zonename,isize,ier)
c go to position within tree at FlowSolution_t node
   call cg_goto_f(index_file,index_base,ier,'Zone_t',index_zone,
+    'FlowSolution_t',index_flow,'end')
c read rind data
   call cg_rind_read_f(irinddata,ier)
c lower range index
   irmin(1)=1
   irmin(2)=1
   irmin(3)=1
c upper range index - use cell dimensions and rind info
c checking GridLocation first:
   call cg_sol_info_f(index_file,index_base,index_zone,index_flow,
+    + solname,loc,ier)
   if (loc .ne.  CellCenter) then
      write(6,'('' Error, GridLocation must be CellCenter!'')')
      stop
   end if
   irmax(1)=isize(1,2)+irinddata(1)+irinddata(2)
   irmax(2)=isize(2,2)+irinddata(3)+irinddata(4)
   irmax(3)=isize(3,2)+irinddata(5)+irinddata(6)
c read flow solution
   call cg_field_read_f(index_file,index_base,index_zone,index_flow,
+    'Density',RealSingle,irmin,irmax,r,ier)
   call cg_field_read_f(index_file,index_base,index_zone,index_flow,
+    'Pressure',RealSingle,irmin,irmax,p,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

2.1.3 具有边界条件的单块结构网格

为了说明边界条件的应用，我们再一次利用 2.1.1 节中同样的单块笛卡尔网格。回顾图 3，我们希望应用以下内容：

ilo — BCTunnlInflow
ihi — BCExtrapolate
jlo — BCWallInviscid
jhi — etc.
klo — etc.

khi — etc.

其中 BCTunnlInflow、BCExtrapolate、WallInviscid 都是针对边界条件的数据名称标识符。有关完整内容见参考文献[1]。在本例中，我们采用允许执行最低级别 BC 的方法——见图 24 和附录 C。

在本节中，我们给出了两种不同的方法，来确定要在其上表现每个边界条件的区域。第一种方法采用 PointRange 类型，这意味着我们在一个定义逻辑矩形区的面上确定最大点和最小点（这种方法只能用于能够用以这种方式定义的面）。第二种方法采用 PointList 类型，它给出应用边界条件的所有点的列表。后一种方法通常用于非结构块，或者用于不是逻辑矩形区的任何块。

（a）规定了范围的边界条件

将 PointRange 类型的边界条件信息写入从 2.1.1 节或 2.1.2 节得出的现有 CGNS 文件的 FORTRAN 程序段如下：

```
c WRITE BOUNDARY CONDITIONS TO EXISTING CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c we know there is only one zone (real working code would check!)
   index_zone=1
c get zone size (and name - although not needed here)
   call cg_zone_read_f(index_file,index_base,index_zone,zonename,
+    isize,ier)
   ilo=1
   ihi=isize(1,1)
   jlo=1
   jhi=isize(2,1)
   klo=1
   khi=isize(3,1)
c write boundary conditions for ilo face, defining range first
c (user can give any name)
c lower point of range
   ipnts(1,1)=ilo
   ipnts(2,1)=jlo
   ipnts(3,1)=klo
c upper point of range
   ipnts(1,2)=ilo
   ipnts(2,2)=jhi
   ipnts(3,2)=khi
   call cg_boco_write_f(index_file,index_base,index_zone,'Ilo',
+    BCTunnelInflow,PointRange,2,ipnts,index_bc,ier)
c write boundary conditions for ihi face, defining range first
c (user can give any name)
c lower point of range
   ipnts(1,1)=ihi
   ipnts(2,1)=jlo
   ipnts(3,1)=klo
c upper point of range
```

```
      ipnts(1,2)=ihi
      ipnts(2,2)=jhi

      ipnts(3,2)=khi
      call cg_boco_write_f(index_file,index_base,index_zone,'Ihi',
     +    BCExtrapolate,PointRange,2,ipnts,index_bc,ier)
     c write boundary conditions for jlo face, defining range first
     c (user can give any name)
     c lower point of range
      ipnts(1,1)=ilo
      ipnts(2,1)=jlo
      ipnts(3,1)=klo
     c upper point of range
      ipnts(1,2)=ihi
      ipnts(2,2)=jlo
      ipnts(3,2)=khi
      call cg_boco_write_f(index_file,index_base,index_zone,'Jlo',
     +    BCWallInviscid,PointRange,2,ipnts,index_bc,ier)
        ...  etc...
     c close CGNS file
      call cg_close_f(index_file,ier)
```

块名称（如 Ilo）是任意的。注意，变量 zonename 必须表示为字符变量，必须恰当确定 isize 和 ipnts 的大小。

用于本示例的 CGNS 文件结构形式如图 11 所示。为简便起见，去掉了 4 个子节点 ZoneBC-t。

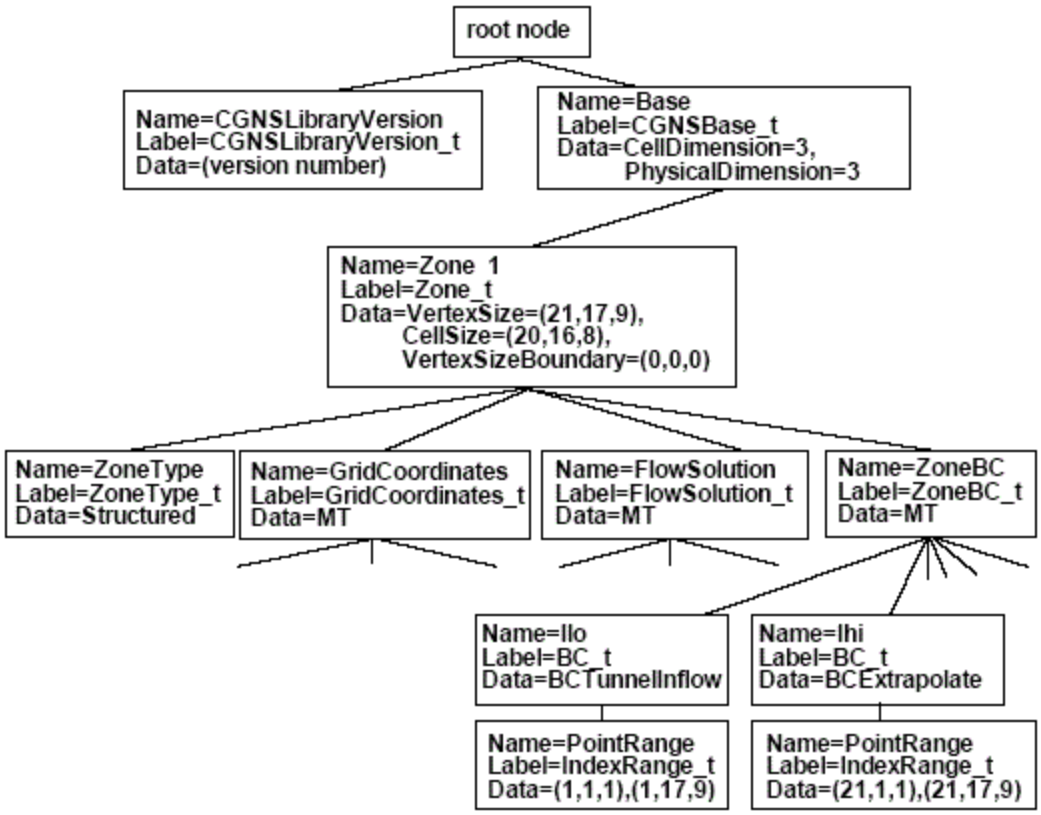


图 11　采用 PointRange 的流动解和边界条件的简单笛卡尔结构网格的 CGNS 文件结构图

通过使用 API 调用，很容易读取边界条件，但我们在此处不再举例。因为在 ZoneBC-t 节点

下有多个 BC-t 子节点，所以，用户必须首先读存在的子节点数，然后循环，并分别检索来自每一个的信息。

（b）规定了点的边界条件

采用 PointList 写边界条件的 FORTRAN 程序段与 PointRange 的情况相同，只是如下程序段被后面一个程序段代替：

```
c write boundary conditions for ilo face, defining range first
c (user can give any name)
   ipnts(1,1)=ilo
   ipnts(2,1)=jlo
   ipnts(3,1)=klo
   ipnts(1,2)=ilo
   ipnts(2,2)=jhi
   ipnts(3,2)=khi
   call cg_boco_write_f(index_file,index_base,index_zone,'Ilo',
+    BCTunnelInflow,PointRange,2,ipnts,index_bc,ier)
```

```
c write boundary conditions for ilo face, defining pointlist first
c (user can give any name)
   icount=0
   do j=jlo,jhi
      do k=klo,khi
         icount=icount+1
         ipnts(1,icount)=ilo
         ipnts(2,icount)=j
         ipnts(3,icount)=k
      enddo
   enddo
   call cg_boco_write_f(index_file,index_base,index_zone,'Ilo',
+    BCTunnelInflow,PointList,icount,ipnts,index_bc,ier)
```

除了 PointRange(IndexRange_t)变成 PointList(IndexArray_t)外且 PointList 节点中有 icount 数据外，本例中 CGNS 文件的结构形式图与图 11 的相同。

2.1.4 具有 1 对 1 连接性的多块结构网格

对于多块结构网格，每个块以与前面几节的例子相同的方式单独处理。但是，多块网格还需要有关将各块互相连接的更多信息。不同类型的块与块之间连接的讨论可参见附录 C。在本节的示例中，我们只列举一个简单的 1 对 1 连接。我们假设有一个 2 块网格，每个都与图 3 所示的相同（21×17×9），只是块 2 偏离 x 方向 20 个单位。这样，块 2 的 ilo 面就与块 1 的 ihi 面毗邻，而且两个块的每一个毗邻点都可以接触到相邻块的一个点。网格图形如图 12 所示。

在这里，没有给出这种 2 个块 CGNS 文件的整体结构图。它与早先给出的类似，只是现在有 2 个块而不是 1 个。其他例子见附录 C。

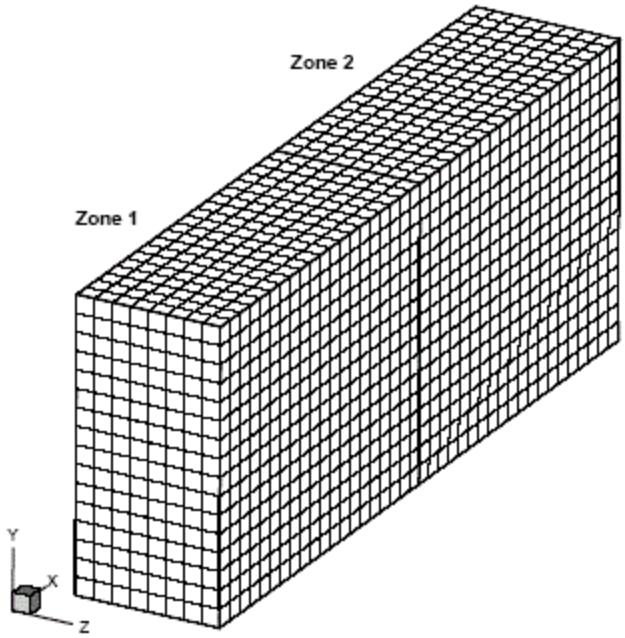


图 12　1 对 1 连接性的 2 块笛卡尔结构网格

现在，1 对 1 连接性的信息必须写入每个块。有两种方式用来记录这种 1 对 1 信息。第一种（具体的）方法仅对 1 对 1 接口有效，而且此区域必须是逻辑矩形的区（因为它们通过 PointRange 和 PointRangeDonor 节点记录，对于这种情况，只有两个点确定整个区域）。第二种方法则更普遍。它采用将 PointList 节点与 PointListDonor 相结合的方法。（第三种方法，用来描述不是点匹配的接口比如失配块或重叠块的接口，它利用 CellListDonor 和 InterpolationsDonor）。有关描述连接性的各种不同方法的详细情况参见 SIDS 文档。

（a）采用特定 1 对 1 方法的连接性

对于本示例，1 对 1 连接性信息可以通过以下 FORTRAN 程序段写入 CGNS 文件（假设已经写入所有网格信息）。

```
c WRITE 1-TO-1 CONNECTIVITY INFORMATION TO EXISTING CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c get number of zones (should be 2 for our case)
   call cg_nzones_f(index_file,index_base,nzone,ier)
c loop over zones to get zone sizes and names
   do index_zone=1,nzone
      call cg_zone_read_f(index_file,index_base,index_zone,
   +    zonename(index_zone),isize,ier)
      ilo(index_zone)=1
      ihi(index_zone)=isize(1,1)
      jlo(index_zone)=1
      jhi(index_zone)=isize(2,1)
      klo(index_zone)=1
      khi(index_zone)=isize(3,1)
   enddo
```

```
c loop over zones again
   do index_zone=1,nzone
c set up index ranges
      if (index_zone .eq.  1) then
         donorname=zonename(2)
c lower point of receiver range
         ipnts(1,1)=ihi(1)
         ipnts(2,1)=jlo(1)
         ipnts(3,1)=klo(1)
c upper point of receiver range
         ipnts(1,2)=ihi(1)
         ipnts(2,2)=jhi(1)
         ipnts(3,2)=khi(1)
c lower point of donor range
         ipntsdonor(1,1)=ilo(2)
         ipntsdonor(2,1)=jlo(2)
         ipntsdonor(3,1)=klo(2)
c upper point of donor range
         ipntsdonor(1,2)=ilo(2)
         ipntsdonor(2,2)=jhi(2)
         ipntsdonor(3,2)=khi(2)
      else
         donorname=zonename(1)
c lower point of receiver range
         ipnts(1,1)=ilo(2)
         ipnts(2,1)=jlo(2)
         ipnts(3,1)=klo(2)
c upper point of receiver range
         ipnts(1,2)=ilo(2)
         ipnts(2,2)=jhi(2)
         ipnts(3,2)=khi(2)
c lower point of donor range
         ipntsdonor(1,1)=ihi(1)
         ipntsdonor(2,1)=jlo(1)
         ipntsdonor(3,1)=klo(1)
c upper point of donor range
         ipntsdonor(1,2)=ihi(1)
         ipntsdonor(2,2)=jhi(1)
         ipntsdonor(3,2)=khi(1)
      end if
 c set up Transform
       itranfrm(1)=1
       itranfrm(2)=2
       itranfrm(3)=3
 c write 1-to-1 info (user can give any name)
       call cg_1to1_write_f(index_file,index_base,index_zone,
    +    'Interface',donorname,ipnts,ipntsdonor,itranfrm,
    +    index_conn,ier)
    enddo
 c close CGNS file
    call cg_close_f(index_file,ier)
```

注意，本程序段特别适合我们的 2 个块情况，即它依赖于我们对这种特殊情况的了解。Transform 确定相邻块的 i、j、k 顶点的相对取向。这里不给出 Transform 值的详细情况，具体可参见文献[1]。但要注意的是，Transform 值（1, 2, 3）表示两个块的 i, j, k 轴都在同一个方向。利用 API 调用也可容易地阅读连接性信息，但这里我们不给出任何示例。最后，我们也没有给出与连接性有关的节点的结构图。感兴趣的用户可参见附录 C 的例图。

（b）采用普通方法的连接性

采用更普通的方法列出了每个连接对（而不是范围）。本示例的连接性信息可通过以下 FORTRAN 程序段写入 CGNS 文件：

```
c WRITE GENERAL CONNECTIVITY INFORMATION TO EXISTING CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c get number of zones (should be 2 for our case)
   call cg_nzones_f(index_file,index_base,nzone,ier)
c loop over zones to get zone sizes and names
   do index_zone=1,nzone
      call cg_zone_read_f(index_file,index_base,index_zone,
   +    zonename(index_zone),isize,ier)
      ilo(index_zone)=1
      ihi(index_zone)=isize(1,1)
      jlo(index_zone)=1

      jhi(index_zone)=isize(2,1)
      klo(index_zone)=1
      khi(index_zone)=isize(3,1)
   enddo
c loop over zones again
   do index_zone=1,nzone
c set up point lists
      if (index_zone .eq.  1) then
         icount=0
         do j=jlo(index_zone),jhi(index_zone)
            do k=klo(index_zone),khi(index_zone)
               icount=icount+1
               ipnts(1,icount)=ihi(1)
               ipnts(2,icount)=j
               ipnts(3,icount)=k
               ipntsdonor(1,icount)=ilo(2)
               ipntsdonor(2,icount)=j
               ipntsdonor(3,icount)=k
            enddo
         enddo
         donorname=zonename(2)
      else
         icount=0
```

```
          do j=jlo(index_zone),jhi(index_zone)
             do k=klo(index_zone),khi(index_zone)
                icount=icount+1
                ipnts(1,icount)=ilo(2)
                ipnts(2,icount)=j
                ipnts(3,icount)=k
                ipntsdonor(1,icount)=ihi(1)
                ipntsdonor(2,icount)=j
                ipntsdonor(3,icount)=k
             enddo
          enddo
          donorname=zonename(1)
       end if
c write integer connectivity info (user can give any name)
       call cg_conn_write_f(index_file,index_base,index_zone,
     +    'GenInterface',Vertex,Abutting1to1,PointList,icount,ipnts,
     +    donorname,Structured,PointListDonor,Integer,icount,
     +    ipntsdonor,index_conn,ier)
    enddo
c close CGNS file
    call cg_close_f(index_file,ier)
```

我们在本文中没有讲述记录不匹配或重叠连接性信息的方法；详细情况用户可参见文献[1]。但要注意，在这种情况下使用 CellListDonor（以及 InterpolantsDonor）意味着要详细说明 donor 面上的单元中心顶点（它们对应于非结构块内的单元数）。InterpolantsDonor 信息由实际价值的 interpolant 组成。

2.2 非结构网格

本小节给出几个非结构网格的例子。用户应该已经熟悉 2.1 小节所介绍的内容，在 2.1 小节中介绍了结构网格的例子。由于 CGNS 文件的许多程序结构对这两种网格形式是等同的，因此，结构网格章节中所涉及的概念在此就不再重述了。

2.2.1 单块非结构网格

此例子采用前面章节中图 3 所示的严格相同的网格。但此时是作为一种非结构网格来编写的，它由一系列 6 边单元（本算例为立方体）组成。在此给出了一个 FORTRAN 程序段，该程序段用 API 调用将此网格写入名为 grid.cgns 的 CGNS 文件中。（注意，非结构块内节点的规则怎样并没有影响，但在本例子中为了叙述方面，这些节点是按顺序排列的）：

```
c WRITE X, Y, Z GRID POINTS TO CGNS FILE
      include 'cgnslib_f.h'
c open CGNS file for write
      call cg_open_f('grid.cgns',MODE_WRITE,index_file,ier)
c create base (user can give any name)
      basename='Base'
      icelldim=3
      iphysdim=3
      call cg_base_write_f(index_file,basename,icelldim,iphysdim,index_base,ier)
c define zone name (user can give any name)
      zonename = 'Zone  1'

c We use the same grid as for the structured example with ni=21,
c nj=17, nk=9.  The variables ni, nj, and nk are still used later,
c for convenience when numbering the unstructured grid elements.
      ni=21
      nj=17
      nk=9
c vertex size (21*17*9 = 3213)
      isize(1,1)=3213
c cell size (20*16*8 = 2560)
      isize(1,2)=2560
c boundary vertex size (zero if elements not sorted)
      isize(1,3)=0
c create zone
      call cg_zone_write_f(index_file,index_base,zonename,isize,
+       Unstructured,index_zone,ier)
c write grid coordinates (user must use SIDS-standard names here)
      call cg_coord_write_f(index_file,index_base,index_zone,RealDouble,
```

```
+    'CoordinateX',x,index_coord,ier)
   call cg_coord_write_f(index_file,index_base,index_zone,RealDouble,
+    'CoordinateY',y,index_coord,ier)
   call cg_coord_write_f(index_file,index_base,index_zone,RealDouble,
+    'CoordinateZ',z,index_coord,ier)
c set element connectivity:
c do all the HEXA_8 elements (this part is mandatory):
c maintain SIDS-standard ordering
   ielem_no=0
c index no of first element
   nelem_start=1
   do k=1,nk-1
      do j=1,nj-1
         do i=1,ni-1
            ielem_no=ielem_no+1
c in this example, due to the order in the node numbering, the
c hexahedral elements can be reconstructed using the following
c relationships:
            ifirstnode=i+(j-1)*ni+(k-1)*ni*nj
            ielem(1,ielem_no)=ifirstnode
            ielem(2,ielem_no)=ifirstnode+1
            ielem(3,ielem_no)=ifirstnode+1+ni
            ielem(4,ielem_no)=ifirstnode+ni
            ielem(5,ielem_no)=ifirstnode+ni*nj
            ielem(6,ielem_no)=ifirstnode+ni*nj+1
            ielem(7,ielem_no)=ifirstnode+ni*nj+1+ni
            ielem(8,ielem_no)=ifirstnode+ni*nj+ni
         enddo
      enddo
   enddo
c index no of last element (=2560)
   nelem_end=ielem_no

c unsorted boundary elements
   nbdyelem=0
c write HEXA_8 element connectivity (user can give any name)
   call cg_section_write_f(index_file,index_base,index_zone,
+    'Elem',HEXA_8,nelem_start,nelem_end,nbdyelem,ielem,
+    index_section,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

注意，对于非结构块，索引数大小始终是 1（因为要确定网格中的一个位置只需要一个索引值），所以，isize 数组包括该块的总顶点数组的大小、单元数组的大小和边界层顶点数组的大小。本例子中，ielem 数组的维数必须精确地定为（8, N），其中，N

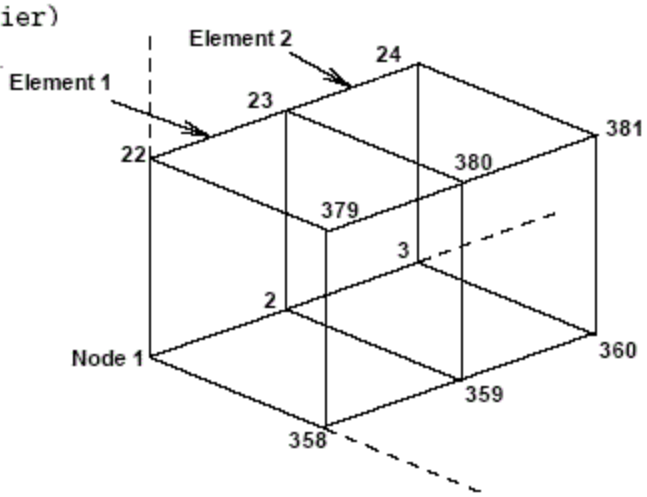


图 13　非结构网格的节点和单元的示意图

大于或等于总单元数。图 3 中左下角的节点在图 13 中示意性地给出了两个单元。比如，在此图中可见，节点编号 1，2，23，22，358，359，380 和 379 组成了单元 1。

由上述程序段生成的 CGNS 文件的总体结构形式示于图 14。由于没有多余的空间，放弃了 y 和 z 的节点。请将该图与用于图 4 中该网格的构成方案的结构形式进行比较。

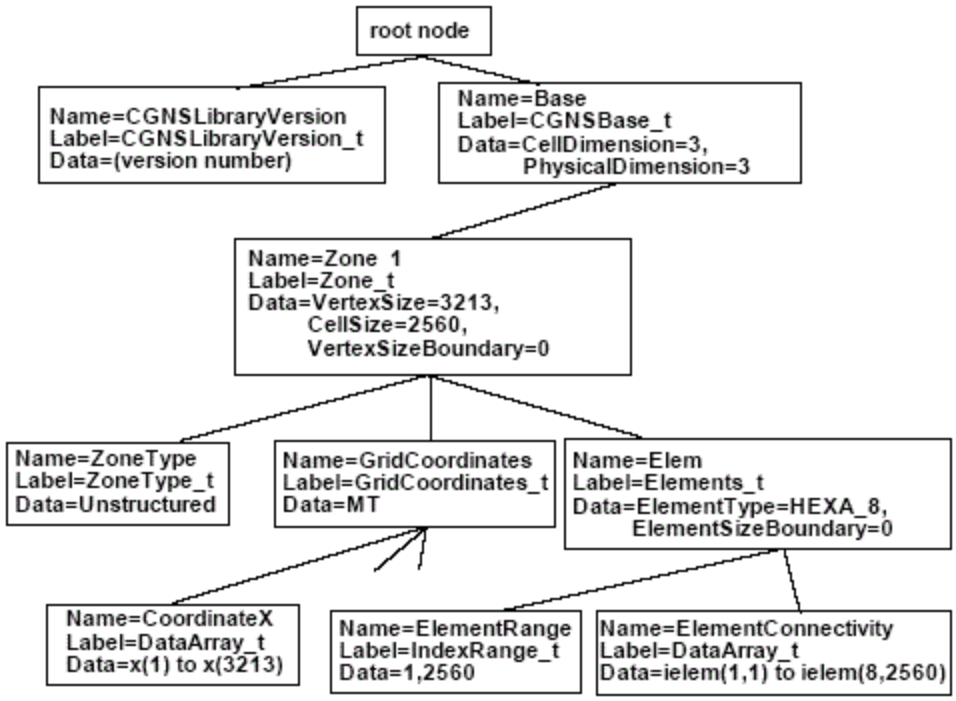


图 14　非结构网格的 CGNS 文件的结构图

对非结构块，用户还希望在 CGNS 文件中分别列出边界单元。这可能对边界条件赋值有用，就如我们将在下面的 2.2.3 节所要阐述的那样。在本例子中，假设用户要对 3 种不同类型的边界条件赋值：在一端流入，在另一端流出，在中间的 4 个面上的侧壁。为了实现此目标，在 CGNS 文件中有 3 个附加的 Elements_t 节点会很有帮助，其中每个节点列出相应的面作为单元（本例子中为 QUAD_4 ）。

在此给出完成了一部分的一个 FORTRAN 程序段。该程序段可以是定义网格和 HEXA_8 连接性的相同程序（见上）的一部分。

```
c do boundary (QUAD) elements (this part is optional,
c but you must do it if you eventually want to define BCs
c at element faces rather than at nodes):
c INFLOW:
   ielem_no=0
c index no of first element
   nelem_start=nelem_end+1
```

```
      i=1
      do k=1,nk-1
         do j=1,nj-1
            ielem_no=ielem_no+1
            ifirstnode=i+(j-1)*ni+(k-1)*ni*nj
            jelem(1,ielem_no)=ifirstnode
            jelem(2,ielem_no)=ifirstnode+ni*nj
            jelem(3,ielem_no)=ifirstnode+ni*nj+ni
            jelem(4,ielem_no)=ifirstnode+ni
         enddo
      enddo
c index no of last element
      nelem_end=nelem_start+ielem_no-1
c write QUAD element connectivity for inflow face (user can give any name)
      call cg_section_write_f(index_file,index_base,index_zone,
+       'InflowElem',QUAD_4,nelem_start,nelem_end,nbdyelem,
+       jelem,index_section,ier)
c OUTFLOW:
         ...  etc...
```

本例子中，jelem 数组的维数必须精确地定为（4, N），其中，N 应大于或等于单元总数。注意，nelem_start 和 nelem_end 范围是在该块内已经定义了任何其它单元（即 HEXA_8 单元）的范围之后才定义的。换句话讲，给定块内的所有单元必须有不同的数值。

本例子中 CGNS 文件的结构形式完全与图 14 所示的相同，只是在 Zone_t 下有 3 个附加的 Elements_t 节点。图 15 分别示出了这 3 个节点。

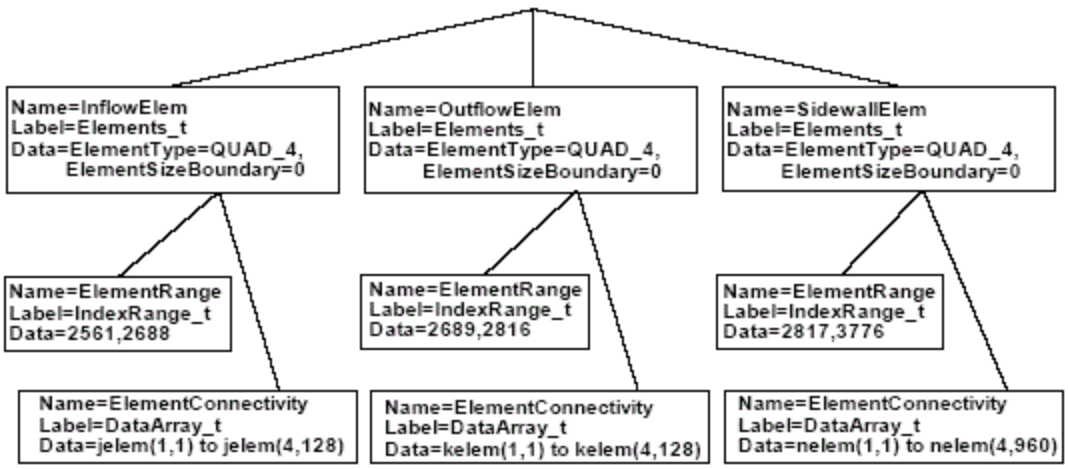


图 15　附加的 Elements_t 边界面节点的结构形式

2.2.2 单块非结构网格和流动解

添加非结构块的流动解，此过程与结构块的相同。但是，外层单元对非结构块无效，因而不应该采用。也就是说，对于顶点和单元中心流动解（（a）和（b）小节），2.1.2 节中所给出的程序段对非结构块也是有效的，但采用外层单元（（c）小节）的程序段是无效的。对顶点和单元中心的例子，非结构块的唯一不同是所有的数组是一维的（只有一个索引），这与 3 维结构数组的 3 个索引不同。顶点解显示，在顶点或节点存储流动解。在上述例子中，每个解变量将列出 3213 个数据数组项。单元中心解表明，在每个单元中心存储流动解。上面的例子中，每个解

变量将列出 2560 个数据数组项。

CGNS 文件的总体结构形式与图 14 所示的相同，只是在块 1 下面也会有一个 FlowSolution_t 节点，并且，这个节点会有许多子节点 GridLocation、Density 和 Pressure。

2.2.3 具有边界条件的单块非结构网格

当向一个非结构块的 CGNS 文件写边界条件时，编程人员要按照 2.1.3 节中所描述的结构块的相同的一般过程来编写。也就是说，边界条件是对点的范围或单个的点所规定的，在此，点看作网格的节点（顶点）。编制程序实际上会与 2.1.3 节中所描述的一样，只是点和/或范围在此是一维的（只有一个索引），与 3 维结构数组的 3 个索引不同。

然而，对非结构块来讲，编程人员还有其它的方案。例如，如果编程人员想要在面中心而不是顶点处应用边界条件，那么，他们就可生成附加的 Elements_t 节点，这些节点确定边界面单元，然后指向这些单元而不是指向节点。通过缺省，假设在顶点（节点）处应用边界条件。但是，当 GridLocation 是不同于 Vertex 的某种东西时，非结构块的边界条件就不再看作节点，而是单元。

由于这种方案与早先的结构块的处理方法完全不同，我们要用一个例子来说明。在 2.2.1 节的末尾，我们阐述过如何生成确定边界面的附加的 Elements_t 节点。现在，可采用下面的程序段编写面-中心的边界条件。

```
c WRITE BOUNDARY CONDITIONS TO EXISTING CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
```

```
c we know there is only one base (real working code would check!)
   index_base=1
c we know there is only one zone (real working code would check!)
   index_zone=1
c we know that for the unstructured zone, the following face elements
c have been defined as inflow (real working code would check!):
   nelem_start=2561
   nelem_end=2688
   icount=0
   do n=nelem_start,nelem_end
      icount=icount+1
      ipnts(icount)=n
   enddo
c write boundary conditions for ilo face
   call cg_boco_write_f(index_file,index_base,index_zone,'Ilo',
+    BCTunnelInflow,PointList,icount,ipnts,index_bc,ier)
c we know that for the unstructured zone, the following face elements
c have been defined as outflow (real working code would check!):
   nelem_start=2689
   nelem_end=2816
   icount=0
   do n=nelem_start,nelem_end
      icount=icount+1
      ipnts(icount)=n
   enddo
c write boundary conditions for ihi face
   call cg_boco_write_f(index_file,index_base,index_zone,'Ihi',
+    BCExtrapolate,PointList,icount,ipnts,index_bc,ier)
c we know that for the unstructured zone, the following face elements
c have been defined as walls (real working code would check!):
   nelem_start=2817
   nelem_end=3776
   icount=0
   do n=nelem_start,nelem_end
      icount=icount+1
      ipnts(icount)=n
   enddo
c write boundary conditions for wall faces
   call cg_boco_write_f(index_file,index_base,index_zone,'Walls',
+    BCWallInviscid,PointList,icount,ipnts,index_bc,ier)
c
c the above are all face-center locations for the BCs - must indicate this,
c otherwise Vertices will be assumed!
   do ibc=1,index_bc
c (the following call positions you in BC_t - it assumes there
c is only one Zone_t and one ZoneBC_t - real working code would check!)
      call cg_goto_f(index_file,index_base,ier,'Zone_t',1,
+       'ZoneBC_t',1,'BC_t',ibc,'end')
      call cg_gridlocation_write_f(FaceCenter,ier)
   enddo
c close CGNS file
   call cg_close_f(index_file,ier)
```

注意，我们在此假设事先知道与每个边界相关的单元数。我们把这些单元数写作 PointList，但是，由于它们是有顺序的，因此我们很容易用 PointRange 替换。在此情况下，只需要 2 个 ipnts 值，相当于 nelem_start 和 nelem_end，icount 为 2。最后要注意的是，必须按照 cg_gridlocation_write_f，利用 API 调用 cg_goto_f（确定在树中的位置）来写 BC_t 下的 GridLocation_t 节点。

图 16 示出了 ZoneBC_t 节点和其子节点的 CGNS 文件结构的一部分。ZoneBC_t 节点位于 Zone_t 的正下方。图 14、15 和 16 三个图合在一起，便组成了完整的文件结构。

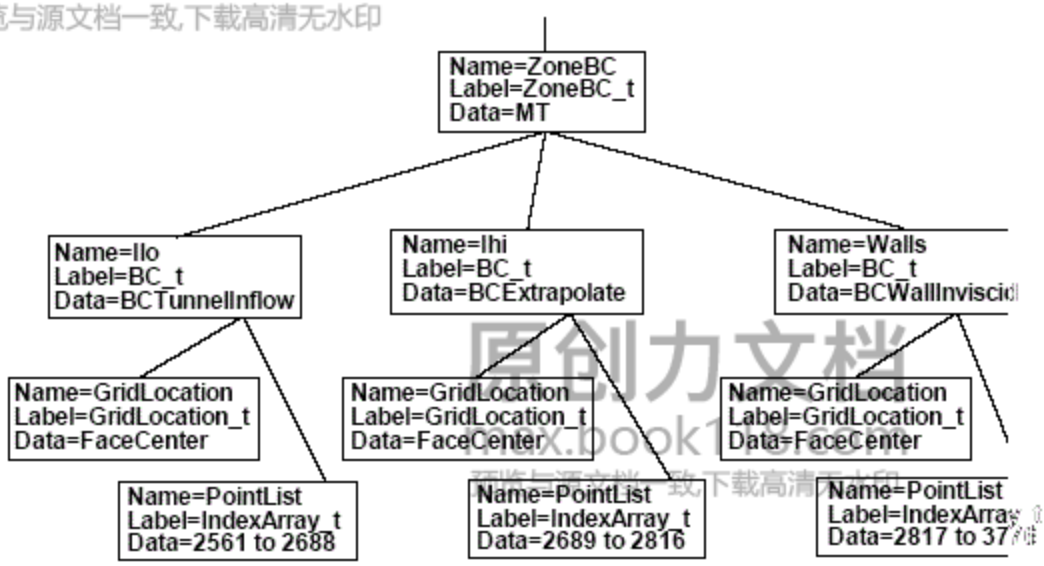


图 16　非结构块 CGNS 文件的部分结构，具有面-中心单元确定的边界条件

3 附加信息

本章节介绍 CGNS 中几个附加的数据形式。在启动程序时，这些附加项没有包括进去的必要，但是很有可能，多数用户将来终会在 CGNS 文件中对某些点执行其中一些附加数据形式。在本章节的末尾会讨论链接的使用。

3.1 收敛历程

ConvergenceHistory_t 节点可用于存储与 CFD 解的收敛相关的数据。例如，编程人员可能想存储总升力系数作为迭代次数的函数。在这种情况下，此变量应该存储在 CGNS 文件的 CGNSBase_t 级。通过在下面的 FORTRAN 程序段中采用 API 来实现这一目标：

```
c WRITE CONVERGENCE HISTORY INFORMATION TO EXISTING CGNS FILE
   include 'cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c go to base node
   call cg_goto_f(index_file,index_base,ier,'end')
c create history node (SIDS names it GlobalConvergenceHistory at base level)
c ntt is the number of recorded iterations
   call cg_convergence_write_f(ntt,'',ier)
c go to new history node
   call cg_goto_f(index_file,index_base,ier,'ConvergenceHistory_t',
+    1,'end')
c write lift coefficient array (user must use SIDS-standard name here)
   call cg_array_write_f('CoefLift',RealDouble,1,ntt,cl,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

在此例子中，数组 cl 必须表示为 ntt 或更大的一组数组。相同大小的附加数组也可写在 ConvergenceHistory_t 节点下。注意，在此情况下，调用 cg_convergnce_write_f 包含一个空白串，因为我们没有记录标准定义。

3.2 描述符节点

描述符节点记录字符串并几乎能插入 CGNS 文件中的任何地方，它有许多可能出现的应用。用户可插入注释或描述符来帮助说明 CGNS 文件中的某些数据的内容。在附录 C 中，我们讲到描述符节点的一种可能的用途，描述用户定义的（UserDefined）数据。描述符节点另一个可能的用途是保存来自 CFD 应用程序的整个输入文件的拷贝。由于描述符节点可包含回车，因此整个 ASCII 文件可被“吞没”到 CGNS 文件中。这样，未来的用户就能看到并检索 CFD 程序所使用的准确输入文件，从而生成 CGNS 文件中包含的数据。唯一不确切的可能就是，从那时起 CFD 程序自身是否发生了变化；但是，如果 CFD 程序具有严格的版本控制的话，那么完全可恢

复性应该是可能的。

在此给出了一个在 CGNSBase 级写描述符节点的例子：

```
c WRITE DESCRIPTOR NODE AT BASE LEVEL
   include 'cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c go to base node
   call cg_goto_f(index_file,index_base,ier,'end')
c write descriptor node (user can give any name)
   text1='Supersonic vehicle with landing gear'
   text2='M=4.6, Re=6 million'
   textstring=text1//char(10)//text2
   call cg_descriptor_write_f('Information',textstring,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

在此例子中，Descriptor_t 节点被命名为 Information，并在此写出了字符串 textstring（由中间具有换行字符 char（10）的 text 1 和 text 2 组成）。所有字符串都必须恰当地表示。

3.3 量纲数据

节点 DataClass_t 表示数据种类。当数据有量纲时，DataClass_t =Dimensional。DataClass_t 节点可出现在 CGNS 层次的许多级上；优先规则规定，层次中较低的 DataClass_t 替代任何更高层的层次。

对于量纲数据，编程人员通常希望通过利用 DataClass_t、DimensionalUnits_t 和 DimensionalExponents_t 来表示每个独立变量的量纲。下面的程序段给出了这种例子，在该程序段中，给 2.1.1 和 2.1.2 节中的结构网格和单元中心流动解加入了单位。

```
c WRITE DIMENSIONAL INFO FOR GRID AND FLOW SOLN
   include 'CGNSLib/cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c we know there is only one zone (real working code would check!)
   index_zone=1
c we know there is only one FlowSolution_t (real working code would check!)
   index_flow=1
c we know there is only one GridCoordinates_t (real working code would check!)
   index_grid=1
c put DataClass and DimensionalUnits under Base
   call cg_goto_f(index_file,index_base,ier,'end')
   call cg_dataclass_write_f(Dimensional,ier)
   call cg_units_write_f(Kilogram,Meter,Second,Kelvin,Degree,ier)
c read fields
   call cg_nfields_f(index_file,index_base,index_zone,index_flow,
+    nfields,ier)
   do if=1,nfields
      call cg_field_info_f(index_file,index_base,index_zone,
+       index_flow,if,idatatype,fieldname,ier)
      if (fieldname .eq.  'Density') then
         exponents(1)=1.
         exponents(2)=-3.
         exponents(3)=0.
         exponents(4)=0.
         exponents(5)=0.
      else if (fieldname .eq.  'Pressure') then
         exponents(1)=1.
         exponents(2)=-1.
         exponents(3)=-2.
         exponents(4)=0.
         exponents(5)=0.
      else
         write(6,'('' Error!  this fieldname not expected:  '',a32)')
+          fieldname
         stop
      end if
c write DimensionalExponents
      call cg_goto_f(index_file,index_base,ier,'Zone_t',1,
+       'FlowSolution_t',1,'DataArray_t',if,'end')
      call cg_exponents_write_f(RealSingle,exponents,ier)
   enddo
c read grid
   call cg_ncoords_f(index_file,index_base,index_zone,ncoords,ier)
   exponents(1)=0.
   exponents(2)=1.
```

```
      exponents(3)=0.
      exponents(4)=0.
      exponents(5)=0.
      do ic=1,ncoords
c write DimensionalExponents
         call cg_goto_f(index_file,index_base,ier,'Zone_t',1,
+          'GridCoordinates_t',1,'DataArray_t',ic,'end')
         call cg_exponents_write_f(RealSingle,exponents,ier)
      enddo
c close CGNS file
      call cg_close_f(index_file,ier)
```

在此例子中需要注意的是，DataClass_t 节点和 DimensionalUnits_t 节点放置在顶层附近，位于 CGNSBase_t 下。DataClass_t 规定为 Dimensional，DimensionalUnits_t 规定为（Kilogram, Meter, Second, Kelvin, Degree）。一般来讲，这就说明了整个数据库是用 MKS 单位表示的有量纲的数据（任何无量纲的或非 MKS 单位的数据都可在更低级上被替代）。然后，对每个局部变量，编程人员只需规定 DimensionalExponents，其中，为每个单位确定一个指数。

图 17 给出了从上面例子得到的最终 CGNS 文件的部分结构。密度的单位是千克/米 ^{3}，压力单位是千克/（米-秒 ^{2}）。网格坐标（图中没有示出）的单位是米。

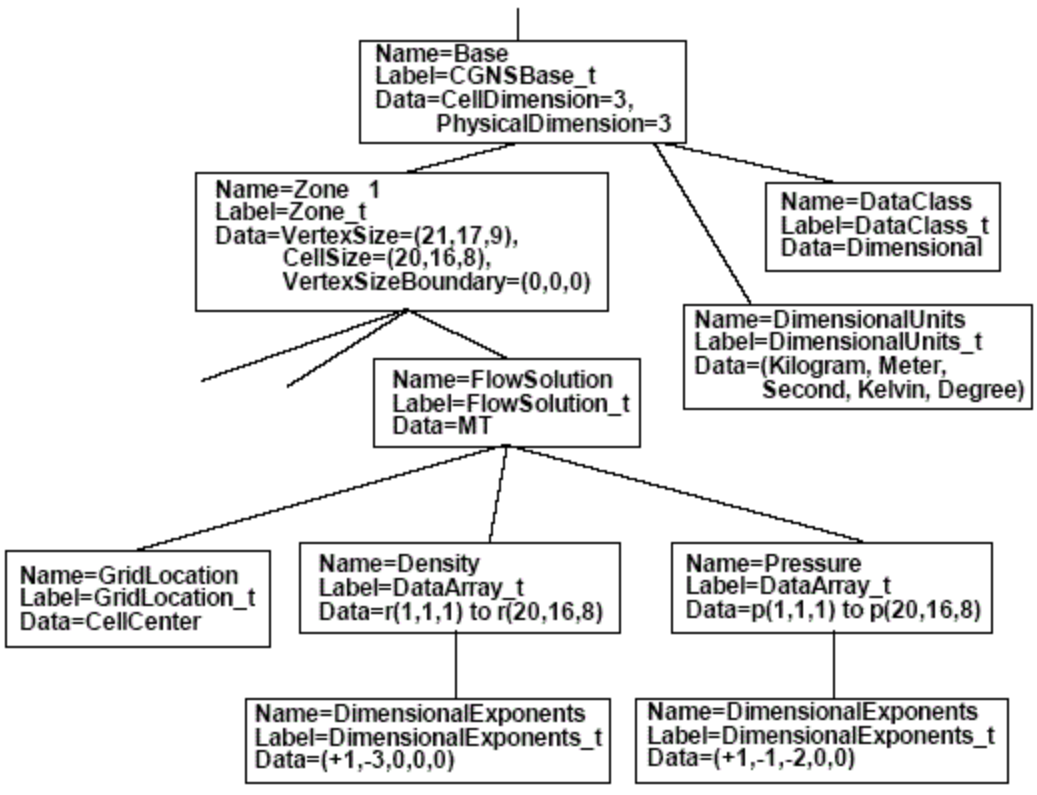


图 17 在单元中心采用量纲数据的流动解的 CGNS 文件的部分结构

3.4 无量纲数据

本例子是有关完全无量纲 CFD 数据的共有问题，参考状态是任意的（未知的）。这种类型

称为 NormalizedByUnknownDimensional。另一种无量纲类型 NormalizedByDimensional 不在此讨论，对于这种类型，数据是无量纲的，但参考状态是明确已知的。

对于 NormalizedByUnknownDimensional 数据库，DataClass 是这样记录的，但 ReferenceState 也是确定所采用的无量纲化数据所必须的。（ReferenceState 节点可为任何数据组所采用，以便表示总参考状态（比如自由流），以及参考马赫数和雷诺数之类的量。ReferenceState_t 节点不包含在 3.3 节中，但它可能已经存在。）

对于本例子，我们并不详细讨论有关项目选择的问题，这些项目应该为 ReferenceState 数据库提供参考状态。我们在例子中简单地给出了一些通常包含在内的典型的选择。在 SIDS 文件中，详细讨论了 ReferenceState_t 中的数据是怎样确定无量纲化数据的 [1]。

```
c WRITE NONDIMENSIONAL INFO
   include 'CGNSLib/cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c put DataClass under Base
   call cg_goto_f(index_file,index_base,ier,'end')
   call cg_dataclass_write_f(NormalizedByUnknownDimensional,ier)
c put ReferenceState under Base
   call cg_state_write_f('ReferenceQuantities',ier)
c Go to ReferenceState node, write Mach array and its dataclass
   call cg_goto_f(index_file,index_base,ier,'ReferenceState_t',1,
+    'end')
   call cg_array_write_f('Mach',RealSingle,1,1,xmach,ier)
   call cg_goto_f(index_file,index_base,ier,'ReferenceState_t',1,
+    'DataArray_t',1,'end')
   call cg_dataclass_write_f(NondimensionalParameter,ier)
c Go to ReferenceState node, write Reynolds array and its dataclass
   call cg_goto_f(index_file,index_base,ier,'ReferenceState_t',1,
+    'end')
   call cg_array_write_f('Reynolds',RealSingle,1,1,reue,ier)
   call cg_goto_f(index_file,index_base,ier,'ReferenceState_t',1,
+    'DataArray_t',2,'end')
   call cg_dataclass_write_f(NondimensionalParameter,ier)
c Go to ReferenceState node to write reference quantities:
   call cg_goto_f(index_file,index_base,ier,'ReferenceState_t',1,
+    'end')
c First, write reference quantities that make up Mach and Reynolds:
c Mach_Velocity
   call cg_array_write_f('Mach_Velocity',RealSingle,1,1,xmv,ier)
c Mach_VelocitySound
   call cg_array_write_f('Mach_VelocitySound',RealSingle,
+    1,1,xmc,ier)
c Reynolds_Velocity
   call cg_array_write_f('Reynolds_Velocity',RealSingle,
+    1,1,rev,ier)
c Reynolds_Length
   call cg_array_write_f('Reynolds_Length',RealSingle,
+    1,1,rel,ier)
c Reynolds_ViscosityKinematic
   call cg_array_write_f('Reynolds_ViscosityKinematic',RealSingle,
```

```
+    1,1,renu,ier)
c
c Next, write flow field reference quantities:
c Density
   call cg_array_write_f('Density',RealSingle,1,1,rho0,ier)
c Pressure
   call cg_array_write_f('Pressure',RealSingle,1,1,p0,ier)
c VelocitySound
   call cg_array_write_f('VelocitySound',RealSingle,1,1,c0,ier)
c ViscosityMolecular
   call cg_array_write_f('ViscosityMolecular',RealSingle,
+    1,1,vm0,ier)
c LengthReference
   call cg_array_write_f('LengthReference',RealSingle,
+    1,1,xlength0,ier)
c VelocityX
   call cg_array_write_f('VelocityX',RealSingle,1,1,vx,ier)
c VelocityY
   call cg_array_write_f('VelocityY',RealSingle,1,1,vy,ier)
c VelocityZ
   call cg_array_write_f('VelocityZ',RealSingle,1,1,vz,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

在这种情况下，添加到 CGNS 文件的唯一信息在 CGNSBase_t 级。注意，Mach 和 Reynolds（存储在 ReferenceState 下）是称为“NondimensionalParameter”的变量，因此，它们每一个都必须包含一个表示这一方面的 DataClass 子节点（当地 DataClass 节点取代整个 NormalizedByUnknownDimensional 数据类型，这种数据类型适用于所有其它情况）。

从上面例子得到的最终 CGNS 文件的相关部分结构图示于图 18。为了节省空间，图中省略了那些出现在 ReferenceState_t 下的许多参考量。

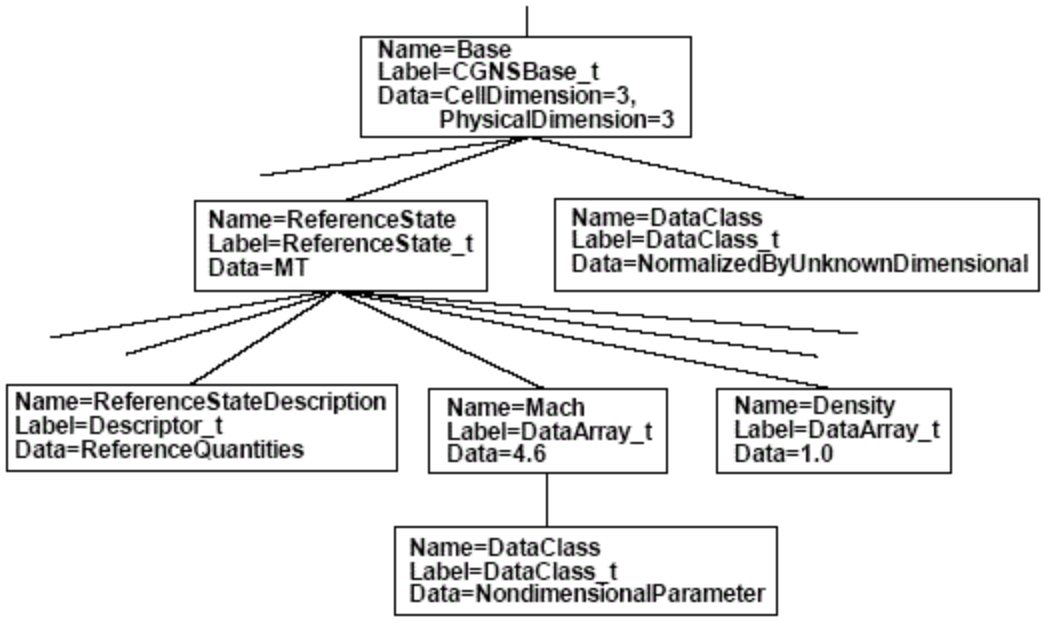


图 18　采用完全无量纲数据的 CGNS 文件的部分结构（参考状态未知）

3.5 流动方程组

FlowEquationSet_t 节点对描述流动解是如何生成的很有用。这是 CGNS 文件的有效自描述特性之一，可以提高 CFD 数据库的有效性和使用寿命。例如，在此节点下，可以记录下列各项信息：通过解薄层 N-S 方程（只在 j 坐标方向上有扩散）获得流场；采用 Spalart-Allmaras 湍流模型，假设γ=1.4 的理想气体。

根据我们早先在 2.1 节介绍的单块结构网格的例子，下面的 FORTRAN 程序段写出了 Zone_t 节点下上述例子流动方程组的某些信息。（注意，FlowEquationSet_t 节点可以置于 CGNSBase_t 节点下的一个更高的级上。采用通常的优先规则）。

```
c WRITE FLOW EQUATION SET INFO
   include 'CGNSLib/cgnslib_f.h'
c open CGNS file for modify
   call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
   index_base=1
c we know there is only one zone (real working code would check!)
   index_zone=1
c existing file must be 3D structured (real working code would check!)
```

```
c Create 'FlowEquationSet' node under 'Zone_t'
   call cg_goto_f(index_file,index_base,ier,'Zone_t',index_zone,
+    'end')
c equation dimension = 3
   ieq_dim=3
   call cg_equationset_write_f(ieq_dim,ier)
c
c Create 'GoverningEquations' node under 'FlowEquationSet'
   call cg_goto_f(index_file,index_base,ier,'Zone_t',index_zone,
+    'FlowEquationSet_t',1,'end')
   call cg_governing_write_f(NSTurbulent,ier)
c Create 'DiffusionModel' node under 'GoverningEquations'
   call cg_goto_f(index_file,index_base,ier,'Zone_t',index_zone,
+    'FlowEquationSet_t',1,'GoverningEquations_t',1,'end')
   idata(1)=0
   idata(2)=1
   idata(3)=0
   idata(4)=0
   idata(5)=0
   idata(6)=0
   call cg_diffusion_write_f(idata,ier)
c
c Create 'GasModel' under 'FlowEquationSet'
   call cg_goto_f(index_file,index_base,ier,'Zone_t',index_zone,
+    'FlowEquationSet_t',1,'end')
   call cg_model_write_f('GasModel_t',Ideal,ier)
c Create 'SpecificHeatRatio' under GasModel
   call cg_goto_f(index_file,index_base,ier,'Zone_t',index_zone,
+    'FlowEquationSet_t',1,'GasModel_t',1,'end')
   call cg_array_write_f('SpecificHeatRatio',RealSingle,1,1,
+    gamma,ier)
c Create 'DataClass' under 'SpecificHeatRatio'
   call cg_goto_f(index_file,index_base,ier,'Zone_t',index_zone,
+    'FlowEquationSet_t',1,'GasModel_t',1,'DataArray_t',
+    1,'end')
   call cg_dataclass_write_f(NondimensionalParameter,ier)
c close CGNS file
   call cg_close_f(index_file,ier)
```

这个特别的例子是专门为 3 维结构块选定的。在非结构块，采用 DiffusionModel 是无效的。图 19 示出的是根据上述例子得到的最终 CGNS 文件相关部分的结构形式。

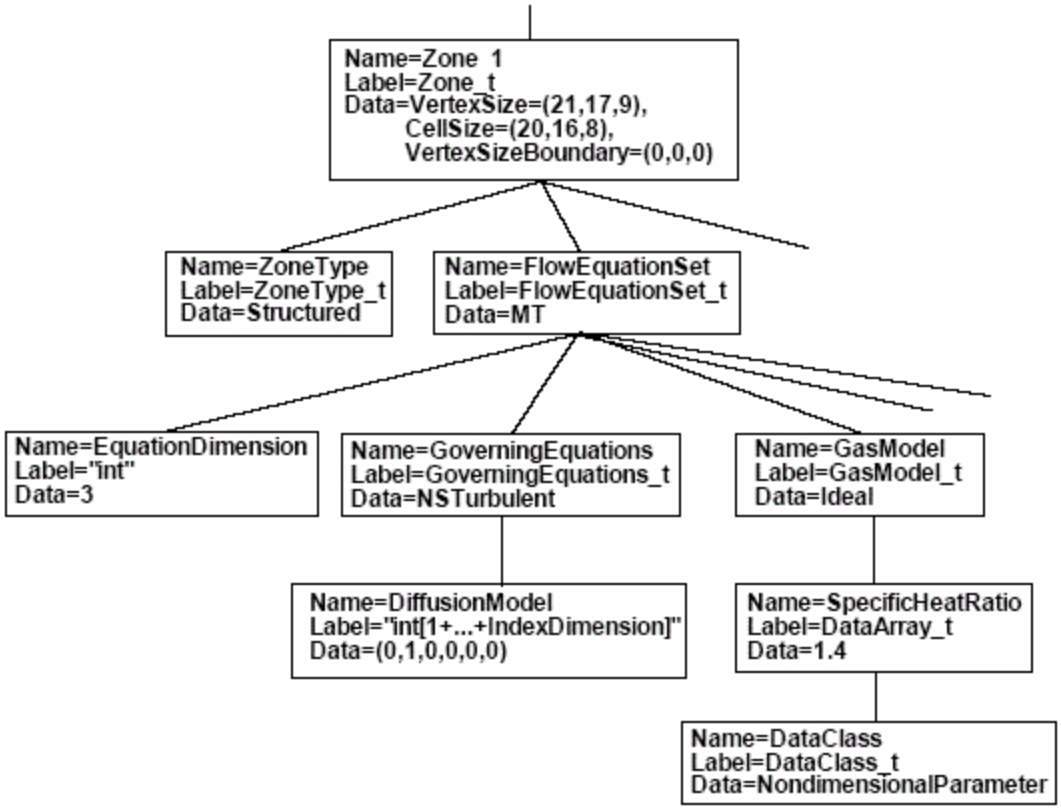


图 19　有流动方程组信息的 CGNS 文件的部分结构

3.6 时间相关数据

时间相关数据（数据具有多个流动解）也可存储于 CGNS 文件中。不同的情况可以产生具有多种流动解的数据：比如：

（1）非移动网格

（2）刚性移动网格

（3）变形或变化网格

上面每一种既可以是时间精确运行的结果，也可以是简单的非时间精确运行的多个快速生成结果，只要它的迭代是收敛的。

本节只给出第 1 种类型的例子。对其它 2 种类型感兴趣的读者可参阅 SIDS 文献[1]。就非移动网格而言，存储多个流动解的方法比较简单：多个 FlowSolution_t 节点，每一个都有不同的名称，置于每个 Zone_t 节点下。但是，还需要成为一种机制，以便将每个 FlowSolution_t 节点与特定的时间和/或迭代联系起来。这要通过使用 BaseIterativeData_t（CGNSBase_t 下）和 ZoneIterativeData_t（每个 Zone_t 下）来实现。BaseIterativeData_t 包含 NumeberOfSteps、所存储的时间和/或迭代的数值，以及这些参数的值。ZoneIterativeData_t 包含 FlowSolutionPointers 作为字符数据组。确定 FlowSolutionPointers 的大小为 NumberOfSteps，并包含本块内与各自的时间和/或迭代对应的 FlowSolution_t 节点的名称。最后，将 SimulationType 节点置于 CGNSBase_t 下，以表示生成数据的模拟类型（比如，TimeAccurate, NonTimeAccurate）。（注意：为了供时间相关数据使用，SimulationType_t 节点不受限制；任何 CGNS 数据组都能利用它！）

下面的 FORTRAN 程序段写出了上面的一些信息，采用的是我们早先在 2.1 节中单块结构网格的例子。对这个例子来说，假设从时间精确模拟得到 3 个流动解，作为时间的函数输出到 CGNS 文件。变量 r1 和 p1 代表时间 1 时的密度和压力，r2 和 p2 则是时间 2 时的值，r3 和 p3

是时间 3 时的值。

```
c WRITE FLOW SOLUTION TO EXISTING CGNS FILE
      include 'CGNSLib/cgnslib_f.h'
c open CGNS file for modify
      call cg_open_f('grid.cgns',MODE_MODIFY,index_file,ier)
c we know there is only one base (real working code would check!)
      index_base=1
c we know there is only one zone (real working code would check!)
      index_zone=1
c set up the times corresponding to the 3 solutions to be
c stored:
      time(1)=10.
      time(2)=20.
      time(3)=50.
c define 3 different solution names (user can give any names)
      solname(1) = 'FlowSolution1'
      solname(2) = 'FlowSolution2'
      solname(3) = 'FlowSolution3'
c do loop for the 3 solutions:
      do n=1,3
c create flow solution node
      call cg_sol_write_f(index_file,index_base,index_zone,solname(n),
+       Vertex,index_flow,ier)
c write flow solution (user must use SIDS-standard names here)
      if (n .eq.  1) then
      call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+       RealDouble,'Density',r1,index_field,ier)
      call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+       RealDouble,'Pressure',p1,index_field,ier)
      else if (n .eq.  2) then
      call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+       RealDouble,'Density',r2,index_field,ier)
      call cg_field_write_f(index_file,index_base,index_zone,index_flow,
```

```
+     RealDouble,'Pressure',p2,index_field,ier)
      else
      call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+     RealDouble,'Density',r3,index_field,ier)
      call cg_field_write_f(index_file,index_base,index_zone,index_flow,
+     RealDouble,'Pressure',p3,index_field,ier)
      end if
      enddo
c create BaseIterativeData
      nsteps=3
      call cg_biter_write_f(index_file,index_base,'TimeIterValues',
+     nsteps,ier)
c go to BaseIterativeData level and write time values
      call cg_goto_f(index_file,index_base,ier,'BaseIterativeData_t',
+     1,'end')
      call cg_array_write_f('TimeValues',RealDouble,1,3,time,ier)
c create ZoneIterativeData
      call cg_ziter_write_f(index_file,index_base,index_zone,
+     'ZoneIterativeData',ier)
c go to ZoneIterativeData level and give info telling which
c flow solution corresponds with which time (solname(1) corresponds
c with time(1), solname(2) with time(2), and solname(3) with time(3))
      call cg_goto_f(index_file,index_base,ier,'Zone_t',
+     index_zone,'ZoneIterativeData_t',1,'end')
      idata(1)=32
      idata(2)=3
      call cg_array_write_f('FlowSolutionPointers',Character,2,idata,
+     solname,ier)
c add SimulationType
      call cg_simulation_type_write_f(index_file,index_base,
+     TimeAccurate,ier)
c close CGNS file
      call cg_close_f(index_file,ier)
```

如早先编程摘录中所提示的，必须对所有变量设定恰当的量纲。变量 time（在本算例中是一个用大小 3 计算的数组）包含存储在 BaseIterativeData_t 下的时间值。图 20 给出了从上面例子得到的最终 CGNS 文件的结构形式。将此图与图 6 比较。为了节省空间，GridCoordinates_t、ZoneType_t 和在 FlowSolution_t 下的所有节点都被省略了。

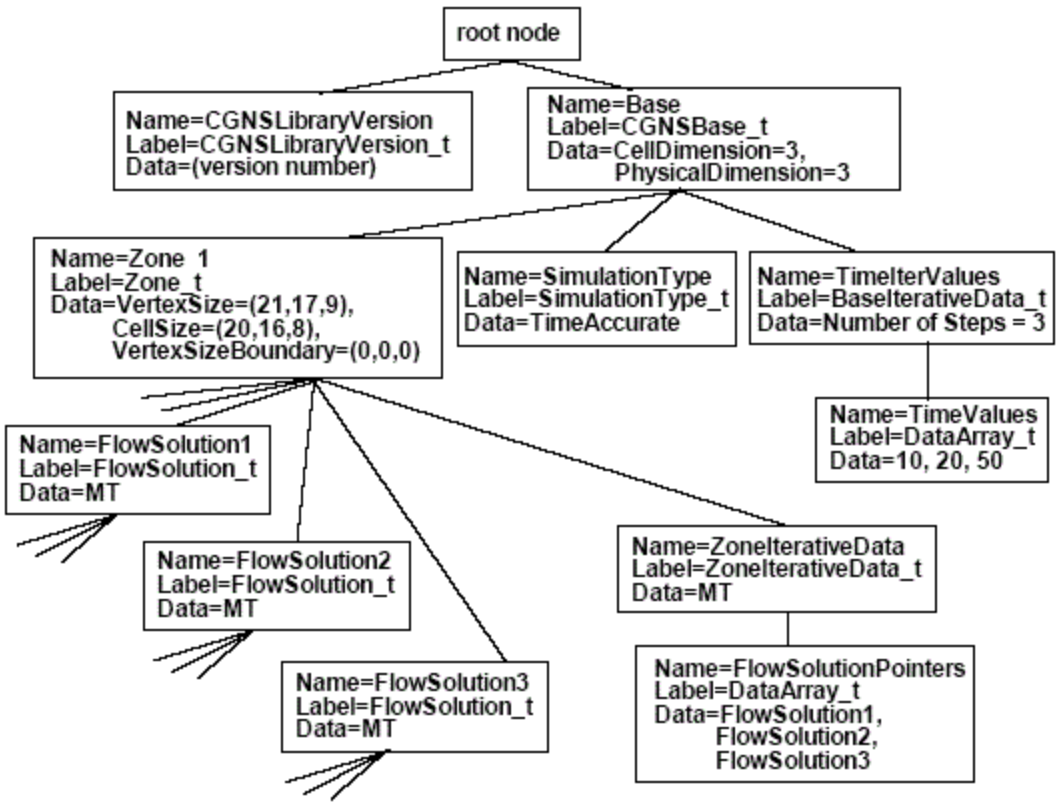


图 20　有多个时间精确流动解（非移动网格坐标）的简单笛卡尔结构网格的 CGNS 文件的结构形式

3.7 采用链接

链接将 CGNS 树内的一个节点与另一个节点联系起来，或者将一个单独的 CGNS 文件内的一个节点与另一个节点全部联系起来。当有重复数据时，这很有用：不用多次写相同的数据，链接可指向只写一次的数据。

许多用户都可能需要链接，它的一个非常重要的用途是指向网格坐标。这种用途源自下列方式。假设用户计划将一个特殊网格用于多种情况。有几种存储数据的方案。它们是：

（1）在单独的 CGNS 文件内保存具有每个流动解的网格的副本。

（2）只保存一个具有网格和多个 FlowSolution_t 节点的 CGNS 文件；每一个 FlowSolution_t 节点对应一种不同的情况。

（3）只保存一个具有多个 CGNSBase_t 节点的 CGNS 文件。网格和一个流动解将存储在一个库下。其它的库将各自包含一个独立的流动解，向第一个库中的网格坐标加入一个链接。

（4）保存一个具有确定的网格坐标的 CGNS 文件，并采用指向网格坐标的链接，在单独的 CGNS 文件内存储每种情况的流动解。

第 1 种方案在概念上最直接，并且肯定是通常推荐的方法（这是本文中到目前为止描述所有举例的 CGNS 文件的方法）。但是，如果网格非常大，那么该方法会产生大量的存储空间，这对多次存储相同的网格点是不必要的。第 2 种方案可能是也可能不是一种可行的方案。如果用户采用描述相关状态的 ReferenceState 和 FlowEquationSet，力求使 CGNS 文件具有完全自我描述性的话，那么，只要 ReferenceState 或 FlowEquationSet 在两种情况之间是不同的（比如，不同的马赫数、雷诺数或攻角），就不能使用这种方法。第 3 种方案消除了这种约束。它在同

一个文件中采用指向网格坐标的链接。第 4 种方案与第 3 种方案类似，只是网格坐标和每个流动解都全部存储在单独的文件中。

图 21 给出了第 4 种方案的一个例子，结构图上显示了两个单独的 CGNS 文件的相关部分。注意，对于多块网格，在此例子中，FILE 1 中的每个块都将有一个单独的链接指向 FILE 2 中适当块的网格坐标。

现在，CGNS API 就有了指定链接的能力，并能询问 CGNS 文件中的链接信息。在以前，这只有通过采用 ADF 核心程序库的软件才有可能。然而，当为编写打开 CGNS 文件，并且链接创建调用命令发出时，链接信息只是被记录，实际的链接创建延迟到写完文件关闭之后。因此，在为编写打开 CGNS 文件时，任何试图进入到链接的文件中的特定位置的操作都是徒劳的。当为修改打开 CGNS 文件时，这种问题是不存在的，因为链接创建是快速的。

对 API 来讲，阅读链接的 CGNS 文件是不成问题的，因为链接是“透明”的。只要任何单独的链接文件保持它们的名称不变，并相对于母文件保持在同一位置（在 Unix-目录内），打开母文件就会自动地存取链接文件。

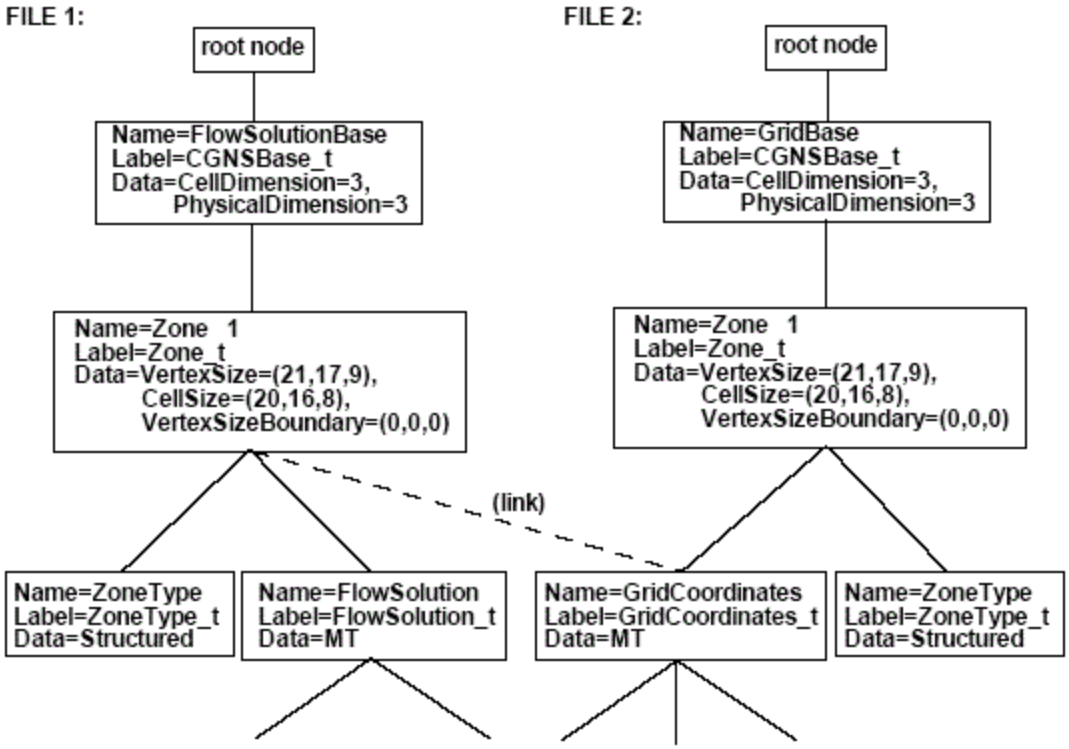


图 21　从一个文件链接到另一个文件的网格坐标的两个 CGNS 文件的部分结构

4 查找故障

4.1 处理错误

API 拥有大量的检查错误的手段，既包括 ADF 的非法使用，也包括不相容 SIDS 的错误。但是，这并不能保证在到达核心级之前 API 会找出所有问题。在此没有列出那些可能在 ADF 核心程序本身内出现的错误列表；它们可能在“Error strings”下的 ADF_interface.c 文件中发现，以及在 ADF 用户指南[2]中找到。

如果出现了错误，由 ADF 或 API 程序给出的信息会帮助描述错误并指出错误所在。

4.2 已知问题

可能发生的一个已知问题与链接有关，这与其说是一个问题，不如说是一个约束。假如用户做了从一个 CGNS 文件到另一个文件的链接，那么，只要用户想以 MODE_MODIFY 或 MODE_WRITE 方式打开链接文件，链接的文件就必须允许写入。也就是说，以 MODE_MODIFY 或 MODE_WRITE 方式打开 CGNS 文件，意味着整个 CGNS 文件层次，包括链接（因为它们是透明的），都可以那种方式进入。

5 通常要询问的问题

问：CGNS 是否支持 SIDS 中没有列出的解的数组名？

答：你可以使用任何你想为解数组用的数据名。但是，如果你创建了一个 SIDS 中没有列出的新名称，它可能不被其他读你文件的应用软件认识。

问：CGNS 文件中的族是什么？

答：族是用来链接网格与几何结构的。数据结构的 Family_t 是可选择的，并可用来通过参照 CAD 实体来确定边界补块的几何形状。反过来，网格补块能参照族，因此，我们得出：网格->族->几何形状。

问：DiscreteData_t 和 IntegralData_t 有何用途？

答：DiscreteData_t 可用于存储场数据，这种数据并不是流动解 FlowSolution_t 要特别考虑的部分。IntegralData_t 用于存储一般的整体数据或集成数据（IntegralData_t 下的每个 DataArray_t 节点内，允许有单个整数或浮点数）。

问：在我的程序中执行 CGNS 时，有什么好的编程惯例能帮助我避免出现问题？

答：通常良好的编程标准应用：大量采用注释，采用逻辑空格以使程序更易读，等等。此外，API 会从每次调用中返回一个错误代码；当错误代码信息不为零时，有规律且适当地检查附有错误信息的程序出口不失为一种好的办法。在 FORTRAN 程序中，你可采用：

```
if （ier .ne. o） call cg_error_exit_f
```

问：我怎样才能看到 CGNS 文件中的内容？

答：发布本公告时，应用程序 adfedit 是查看 CGNS 文件的最佳方法。此应用程序可使你观察整个树结构，或查看单个的数据节点。附录 A 简要介绍了 adfedit。

问：假如我创建了一个真实的相容 SIDS 的 CGNS 文件，我应如何告知？

答：目前还很难确切地讲有用户已经创建了一个其他读者能读懂且相容 SIDS 的 CGNS 文件。但由于 API（中级程序库）对非相容标准有许多检测，与你应用 ADF（核心级）调用相比，当使用 API 时，你出错的几率要少得多。

附录 A　adfedit 应用程序

应用程序 adfedit 可观察（和编辑）CGNS 文件。目前 adfedit 包含 CGNS 软件，虽然将来它可能被更高级的应用所替代。

下面列出的是 adfedit 应用最基本能力的一个简要描述。一段典型的 adfedit 应用程序可能是这样的：

```
adfedit
ADFmain> br              go to “browse”
ADFbrowse> o cgns.file   open file cgns.file
ADFbrowse> t             go to “tools”
ADFtools> pt             print out entire directory tree structure
ADFtools> pt -l          same, but include node label info
ADFtools> br             go to “browse”
ADFbrowse> ls            list children (sub-nodes) of current node
ADFbrowse> cd Base       go down one level to the node named “Base”
ADFbrowse> dd            give description of node data
ADFbrowse> pd            print node data
ADFbrowse> cd ..         go up one level in the tree
ADFbrowse> cd            go to the top of the tree (to root node)
ADFbrowse> ?             help
ADFbrowse> quit          exits program
```

附录 B 计算机程序例子

下面的计算机程序是在本用户指南文本中提到的（加上一些没提到的）完整的、可使用的程序版本。这些可从 CGNS 网站 www.cgns.org（见用户指南）上获取.。这些程序对非常简单的 CGNS 文件实例进行了读写，目的是帮助用户理解 CGNS 概念和 API 调用的使用。在 UNIX 系统上编译这些程序的说明包含在每个程序的注释行内。前提是假设用户已经获得了 CGNS 程序库（从同一网站），并将其作为子目录/CGNSLib 就近放置。下面的程序大多用 FORTRAN 编写。

值得注意的是，这些程序非常简单，主要是为了具有易读性。实际的工作程序会写得更通用，具有更多的检查功能，而且对特殊情况不会难以应用。下面列出相应章节的程序。

结构网格

Section 2.1.1:

write_grid_str.f	writes grid
write_grid_str.c	writes grid (C-program example)
read_grid_str.f	reads grid

Section 2.1.2:

write_flowvert_str.f	writes vertex-based flow solution
read_flowvert_str.f	reads vertex-based flow solution
write_flowcent_str.f	writes cell centered flow solution
read_flowcent_str.f	reads cell centered flow solution
write_flowcentrind_str.f	writes cell centered flow solution with rind cells
read_flowcentrind_str.f	reads cell centered flow solution with rind cells

Section 2.1.3:

write_bc_str.f	writes PointRange boundary condition patches
read_bc_str.f	reads PointRange boundary condition patches
write_bcpnts_str.f	writes PointList boundary condition patches
read_bcpnts_str.f	reads PointList boundary condition patches

Section 2.1.4:

write_grid2zn_str.f	writes 2-zone grid
read_grid2zn_str.f	reads 2-zone grid
write_con2zn_str.f	writes 1-to-1 connectivity for 2-zone example
read_con2zn_str.f	reads 1-to-1 connectivity for 2-zone example
write_con2zn_genrl_str.f	writes general 1-to-1 connectivity for 2-zone example
read_con2zn_genrl_str.f	reads general 1-to-1 connectivity for 2-zone example

非结构网格

Section 2.2.1:

write_grid_unst.f	writes grid
read_grid_unst.f	reads grid

Section 2.2.2:

write_flowvert_unst.f	writes vertex-based flow solution
read_flowvert_unst.f	reads vertex-based flow solution

Section 2.2.3:

write_bcpnts_unst.f	writes PointList boundary condition patches (FaceCenter)
read_bcpnts_unst.f	reads PointList boundary condition patches (FaceCenter)

通用程序

Section 3.1:

write_convergence.f	writes convergence history
read_convergence.f	reads convergence history

Section 3.2:

write_descriptor.f	writes descriptor node under CGNSBase_t
read_descriptor.f	reads descriptor node under CGNSBase_t

Section 3.3:

write_dimensional.f	writes dimensional data to an existing grid + flow solution
read_dimensional.f	reads dimensional data from an existing grid + flow solution

Section 3.4:

write_nondimensional.f	writes nondimensional data to an existing CGNS file
read_nondimensional.f	reads nondimensional data from an existing CGNS file

Section 3.5:

write_floweqn_str.f	writes flow equation information for structured example
read_floweqn_str.f	reads flow equation information for structured example

Section 3.6:

write_timevert_str.f	writes time-dependent flow soln (as Vertex) for structured example
read_timevert_str.f	reads time-dependent flow soln (as Vertex) for structured example

附录 C　SIDS 综述

C. 1 大体情况

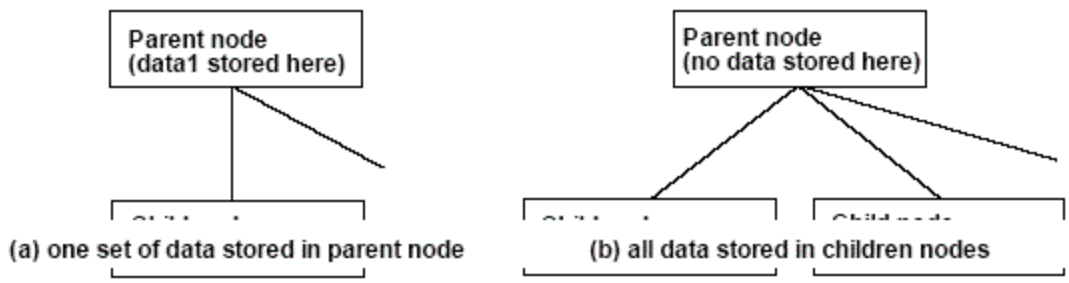
正如引言中提到的那样，CGNS 文件由一组树状结构中的节点组成，跟 UNIX 环境中的目录构成方式很相似。每个节点通过标示和名称进行识别。大多数节点的标示由以“-t”结尾的一串字符组成。在 CGNS 文件中，确定标示的习惯通常有非常严格的规则。节点的名称有时由用户确定，但有时也必须按照严格的惯例命名。标示决定节点的“类型”。例如，“Zone_ t”确定了块类型的节点，“DataArray_t”确定了包含数组的节点类型。名称确定了特定节点类型的特例。例如，“Density”是“DataArray_t”类型节点的名称，包含了一组密度数据。

当你对 CGNS 文件的构成形式越来越熟悉的时候，你就会注意到，一般来说，你在 CGNS 中的级别越高，标示就越重要（名称往往由用户定义）；而级别越低，名称就越重要。这种情况的出现是由于在较高的级别里，建立了更广泛的种类，并用来在级别中确定“往哪儿去”；在较低的级别里，类别变得不那么重要，因为这是你搜寻特殊项目的区域。

在这一节的剩余部分，我们主要谈的是用标示表示的节点，因为我们的焦点在“大体情况”上。在后面几节里，在研究具体实例时，名称和标示都会被提及。

SIDS 文档以“父”和“子”这样的术语对 CGNS 文件的结构进行详细说明，注意到这一点是很重要的。但是，当一条给定的信息列在一个节点“之下”时，实际上存在两种可能：信息可能存储为本节点“之中”的数据，也可能存储在一个单独的子节点“之中”或“之下”。这种差别如图 22 所示。SIDS 到 ADF 的映象文档[3]确定在哪一种情况下使用哪一种可能，并且必须始终与 SIDS 文档一起考虑。在附录的余下部分，根据 SIDS 到 ADF 的映象文档，信息的位置（不管是作为数据还是作为单独的子节点）总是会得到明确的描述。

图 22　用处于父节点“之下”的数据组来对待此节点的两种可能的处理方式



本附录的剩余部分打算概述 SIDS 最重要的和最通用的部分。它并没有涵盖所有可能的节点或情况，而只打算进行一个概述。随着将来 SIDS 的扩展，也可能存在增加比我们现在介绍的更多能力的情况。

CGNS 文件的顶级，或者说入口级，总是被称为“根节点”的部分。直接处于这个节点下的子节点是 CGNSLibraryVersion_t 节点和一个或多个 CGNSBase_t 节点。就像它的数据一样，CGNSLibraryVersion_t 节点具有程序库版本（发行）数。CGNSBase_t 节点代表给定的数据库或者“事例”的顶级。尽管 SIDS 为了保持可扩展性以及允许在一个文件中有超过一个“事例”的可能而允许有任何数值，但大多数 CGNS 文件只有一个 CGNSBase_t 节点。这里，“事例”的定义是开放的。对附录的余下部分，我们假定在一个给定的 CGNS 文件里只有一个 CGNSBase_t

节点。

CGNSBase_t 节点可以包括以下节点（像其子节点一样）：Zone_t, ConvergenceHistory_t, BaseIterativeData_t, SimulationType_t, Family_t, IntegralData_t, DataClass_t, FlowEquationSet_t, DimensionalUnits_t, ReferenceState_t, 和 Descriptor_t。

Zone_t 节点给出关于网格的特定块的信息：CGNS 文件的大多数数据通常处于这一节点下。这一级允许有任意个 Zone_t 节点。它的子节点将在下面得到更详尽的描述。ConvergenceHistory_t 包含一般由许多 CFD 程序输出的解的历史信息，比如：截断误差，升力，阻力等，它们是迭代次数的函数。习惯上它取名为 GlobalConvergenceHistory。ConvergenceHistory_t 节点也可以存在于 Zone_t 节点之下，但是，按照习惯它在那儿被称为 ZoneConvergenceHistory。对于多次存储流动解和/或网格的数据库，BaseIterativeData_t 存储与时间和/或迭代次数有关的信息。SimulationType_t 用于描述被存储的模拟类型（即 TimeAccurate 或 NonTimeAcuurate）。Family_t 通常用来将网格与几何形状的 CAD 数据连接在一起，或者将某些实体连接在一起成为一个共同的部分（例如，“机翼”、“支杆”，等等）。可以有任意个 Family_t 节点。

允许处于 CGNSBase_t 之下的剩余节点稍微更普通，可以存在于除此之外的层次中的其他级。这里对它们进行一些简单的描述。IntegralData_T 是一个“回收器”节点，用来存储一些任何想要的普通数据。这一级可以有任意个 IntegralData_t 节点。DataClass_t（习惯上取名为 DataClass）给出 CGNSBase_t 中数据的存储形式，例如：Dimensional，NormalizedByDimensional 或 NormalizedByUnknownDimensional。FlowEquationSet_t（习惯上取名为 FlowEquationSet）确定 CFD 模拟中用到的方程。DimensionalUnits_t（习惯上取名为 DimensionalUnits）确定所用的量纲单位（如果有的话）。ReferenceState_t（习惯上取名为 ReferenceState）确定参考状态。这个节点用来存储确定流场条件和/或无量纲化的量，如 Re 数、M 数和其他参考量。最后，Descriptor_t 用来存储描述符字符串。这一级可以有任意个 Descriptor_t 节点。

存储在 CGNSBase_t 节点本身之中的数据是 CellDimension 和 PhysicalDimension。CellDimension 是网格单元的维数（例如，体积元是 3，面元是 2）。PhysicalDimension 是确定节点位置所需的坐标的数值（例如，1-D 是 1，2-D 是 2，3-D 是 3）。索引维数是查询节点所需的不同索引的数值（例如，1=i，2=i,j，3=i,j,k），它不存储，但对每个块，它可以根据其类型来确定（Structured 或 Unstructured）。如果是 Structured，索引维数和 CellDimension 一样。如果是 Unstructured，索引维数是 1。

Zone_t 下可以存储许多信息。因为这是一个概述，在这儿我们不讲述所有信息，而只突出那些大多数用户可能会用到的特性。ZoneType_t（习惯上取名为 ZoneType）存储名称 Structured 或 Unstructured。GridCoordinates_t 是网格坐标数组的父节点，例如 CoordinateX, CoordinateY 和 CoordinateZ。这一级允许有任意个 GridCoordinates_t 节点（为了处理变形网格的情况）。习惯上原始网格坐标取名为 GridCoordinates。FlowSolution_t 下存储包含流动解的节点：例如，Density、VelocityX、 VelocityY、 VelocityZ 和 Pressure。它也给出流动解存储的位置（例如 CellCenter、Vertex），并且包括这样的可能性，即包含外层网格（虚假单元）信息。这一级允许有任意个 FlowSolution_t 节点。Elements_t 数据结构保存着非结构元素数据，比如连通性，邻居等等。这一级允许有任意个 Elements_t 节点。ZoneIterativeData_t 保存着多次存储流场解的数据库必需的信息。Zone_t 下面其他重要的节点有 ZoneBC_t（习惯上取名为 ZoneBC）和 ZoneGridConnectivity_t（习惯上取名为 ZoneGridConnectivity）。它们分别存储边界条件和网格连接信息。这些节点将在以后介绍。

Zone_t 节点自身存储的数据有 VertexSize，CellSize 和 VertexSizeBoundary。它们的维数由索引维数给出，并且分别给出了顶点数、单元数和边界顶点数（只用于非结构块中分类的单元）。

值得注意的重要一点是，API 在读取 Zone_t 节点时是根据它们的名称对其进行字母数字分类的。之所以认为这很必要，是因为目前大多数 CFD 程序都是按照一定的顺序对多块网格的各

个块完成操作。为了复制现有的非 CGNS 应用软件，有必要保证各个块能按照期望的顺序读取。（ADF 没有必要按照存储的顺序来找回数据，因此为各块建立了 API 阅读器来完成。）因此，当给块命名时，用户应当保证它们是按字符数字命名的（如果需要按顺序的话）。

举例说明，按照命名习惯，ZoneN 中的 N 是块号，它是个字符数字名，最多只到 Zone9。Zone10 到 Zone19 在 Zone1 和 Zone2 之间进行分类，依此类推。名称中允许有空格，因此 Zone N 有 2 个空格，（例如，Zone 1，Zone 2，…，Zone 99, Zone100，…）是字符数字的，最多到 Zone999。其他的块的命名习惯当然也是可能的，而且完全该由用户来恰当地定义它。

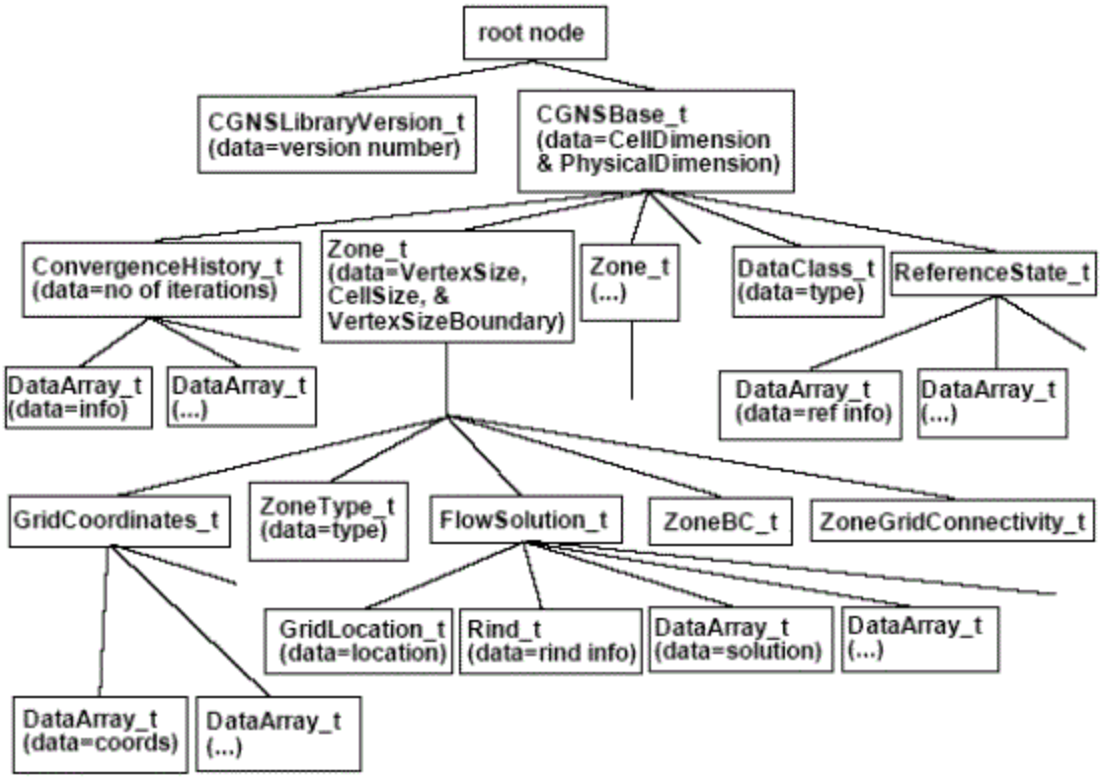


图 23　典型的 CGNS 文件的层级结构（结构网格类）

典型 CGNS 文件的整体结构形式如图 23 所示。这个图说明了层级数据结构，以及各节点的相对位置。它也指出了（非正式地）每个节点中存储什么样的数据（如果有的话）。注意，这里没有将所有可能的节点包括进去。特别要注意的是，Elements_t 节点没有出现在 Zone_t 下面，只有对非结构块 Elements_t 节点才存在。还要注意到，ZoneBC_t 和 ZoneGridConnectivity_t 之下的节点被省略掉了；在下面部分将会讲到它们。可选节点 SimulationType_t（在 CGNSBase_t 之下）没有包括进来。最后，还要注意，GridCoordinates_t 和 FlowSolution_t 节点是可以多个的，但图中每种只给出了一个。多个节点 FlowSolution_t 通常只用于多次存储时间相关数据的情形，多个节点 GridCoordinates_t 则用于变形网格。

C. 2 层级中较低级的执行

大多数实际的数据处于 CGNS 层级中的较低层。这里我们不会涉及很多细节；本文档正文部分的例子对此进行了说明。但是，几个与数据存储有关的要点可以在这里适当地提及。

SIDS 中详细说明了许多与 CFD 数据有关的项目、变量和条件。这些必须用的标准化名称，是为了让其他用户明白你的 CGNS 文件里是些什么。比如，密度必须称为 Density。任何其他名

称都可能不被其他用户认识。实际上，如果另一个应用软件需要的是“Density”，而你取的名称是“density”（小写字母“d”），就会出现其他程序搜索失败的可能。

自然，SIDS 中列的项目不可能包括用户可能需要的所有项目。因此，SIDS 允许对于没有包括进去的任何特殊类型使用 UserDefined 类型。比如，SIDS 中现在只有有限数量的对湍流模型的定义名称（如 OneEquation_SpalartAllmaras）。众所周知，有大量的湍流模型和湍流模型变量存在，所以 SIDS 不可能希望为所有这些定义标准名称。UserDefined 类型就涵盖了这种情况。

但是当运用 UserDefined 类型时，用户冒着其他人不能读懂 CGNS 文件的危险。因此，我们建议，每当使用 UserDefined 类型不可避免的时候，用户也可以同时包括进一个伴随的 Descriptor_t 节点来详细说明做了些什么。

可能的情况是，如果随着时间的推移，发现某些项目用起来更吃力，将来就可能会创造出标准化名称并添加入 SIDS 中。

C. 3 边界条件

CGNS 中边界条件的层次结构一开始可能显得有点让人感到胆怯。因为 CGNS 研究小组决定尽可能使得边界条件信息是描述性的，并且易于延伸到复杂情况。层级中可能包括很多层，而且使用规则变得很复杂。

然而，SIDS 允许使用 ZoneBC_t 节点的简化版，它更容易理解和应用。实质上，简化版跟完全版的 SIDS 边界条件描述相比“切去”了层级中的较高层。这意味着使用简化版的应用程序必须在没有 CGNS 文件帮助时解释出每一个特殊的边界条件类型的含义。

例如，BCFarfield 这个边界条件类型表示用于远场边界的边界条件。大多数 CFD 程序都具有这种类型，它根据当地流场是流入还是流出，是亚声速还是超声速来执行不同的功能。完全版的 SIDS 对 BCFarfield 的描述尝试详细地描述边界条件涉及的研究方法。但是，如果用户选择使用最小的“切割”版，那么，关于存储在 CGNS 文件中的边界条件功能的唯一信息就是名称 BCFarfield。应用程序必须只根据它的名称确定其含义。

图 24 是对 ZoneBC_t 节点最简化执行和完全执行的层级结构实例。（这些层级采用 IndexRange_t 节点。也有可能采用 IndexArray_t 节点，它给出了一组完整的边界索引或单元，而不是一个范围。）注意，还允许有中间结构，在此结构中，给出了 BCDataSet_t 和 BCTypeSimple_t，但是 DirichletData 和 NeumannData 没有给出。

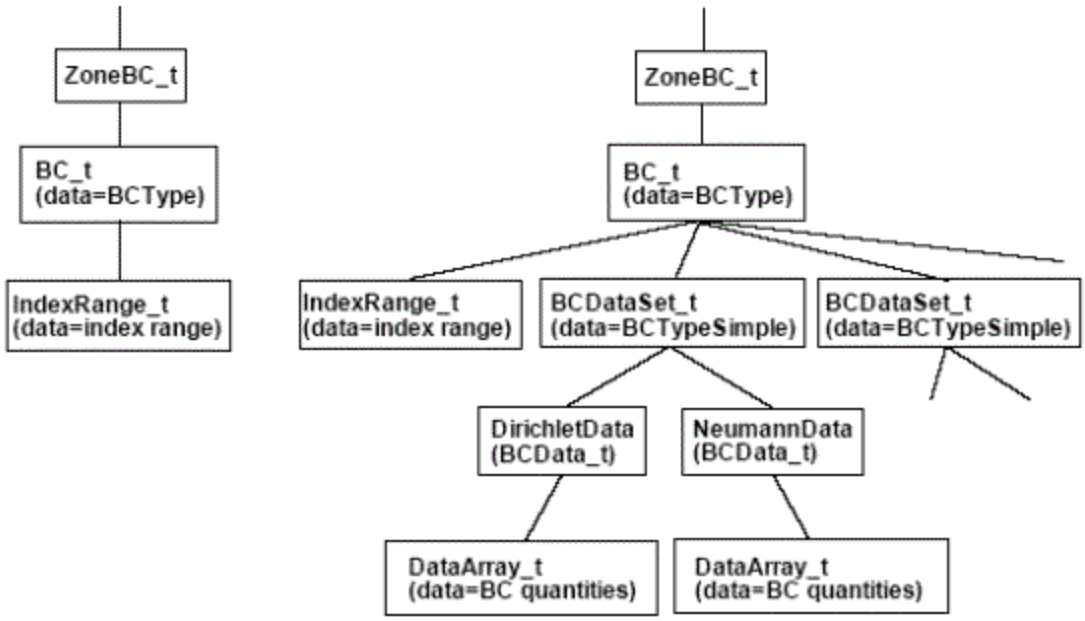


图 24　ZoneBC_t 的一般层级结构

现在 SIDS 中定义了许多边界条件类型，但绝不包括所有可能的边界条件。UserDefined 类型可以用于没有包括进去的任何特殊类型，对于这种类型，用户发现不可能用现存的 SIDS 来描述它。当采用 UserDefined 类型时，伴随的描述符节点有助于描述做了些什么。

C. 4 块的连接性

当一个块的局部与另一个块的局部或它本身相连时，通常希望详细说明块的连接性信息。连接性信息指出了块是怎样接合在一起的，或一个块是如何扭曲与它自身接合的：大多数 CFD 流动解算器都需要这种信息。

可能出现 3 种连接形式：逐点型，补块型和重叠型。逐点型，或者说 1 对 1 的类型，当块的边界毗邻，并且其中一块的网格顶点精确地对应另一块的网格顶点，没有不能配对的孤点时才会出现。当块的边界毗邻，但是点之间没有相互的对应关系，或者这些点与其他点不成对，则出现补块型。当块互相重叠（或一个块覆盖其自身）时出现重叠型。

SIDS 允许在 ZoneGridConnectivity_t 节点下对块的每一种连接类型进行详细说明。所有 3 种类型可以通过普通的 GridConnectivity_t 子节点来实现（重叠型还需要利用 OversetHoles_t 节点）。但在某些情况下，1 对 1 型也可以用更加特殊的 GridConnectivity1to1_t 子节点。

图 25 为采用 GridConnectivity1to1_t 子节点的 1 对 1 型的接口在 ZoneGridConnectivity_t 节点处开始的抽样层级结构。注意，在该图中，我们现在列出了每个节点内的名称、标示和数据。对这个结构，底层的命名习惯尤其重要，实际上比标示更具描述作用。事实上，Transform 节点的标示非常奇怪，甚至没有遵循通常的“_t”的惯例。正如图中所见到的，ZoneGridConnectivity_t 节点以下允许有若干个节点。它们可以是 GridConnectivity1to1_t，OversetHoles_t 或 Descriptor_t 节点的任意组合。

图 26 给出的是采用 GridConnectivity_t 子节点的重叠型接口的抽样层级（也在

ZoneGridConnectivity_t 节点处开始）。补块型接口的情况看上去也是同样的，只是没有 OversetHoles_t 节点或子节点，以及 GridConnectivityType 是 Abutting。注意，CellListDonor 和 InterpolantsDonor 是用于补块型或重叠型接口的。（如果接口是 1 对 1 的，PointListDonor 可以用于它们的位置。详细情况见参考文献[1,3]。）

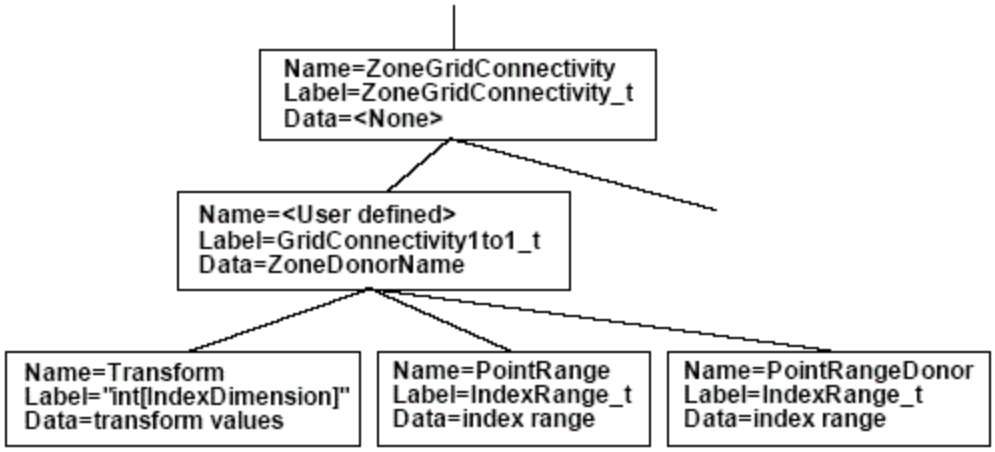


图 25　1 对 1 接口的 ZoneGridConnectivity_t 层级结构

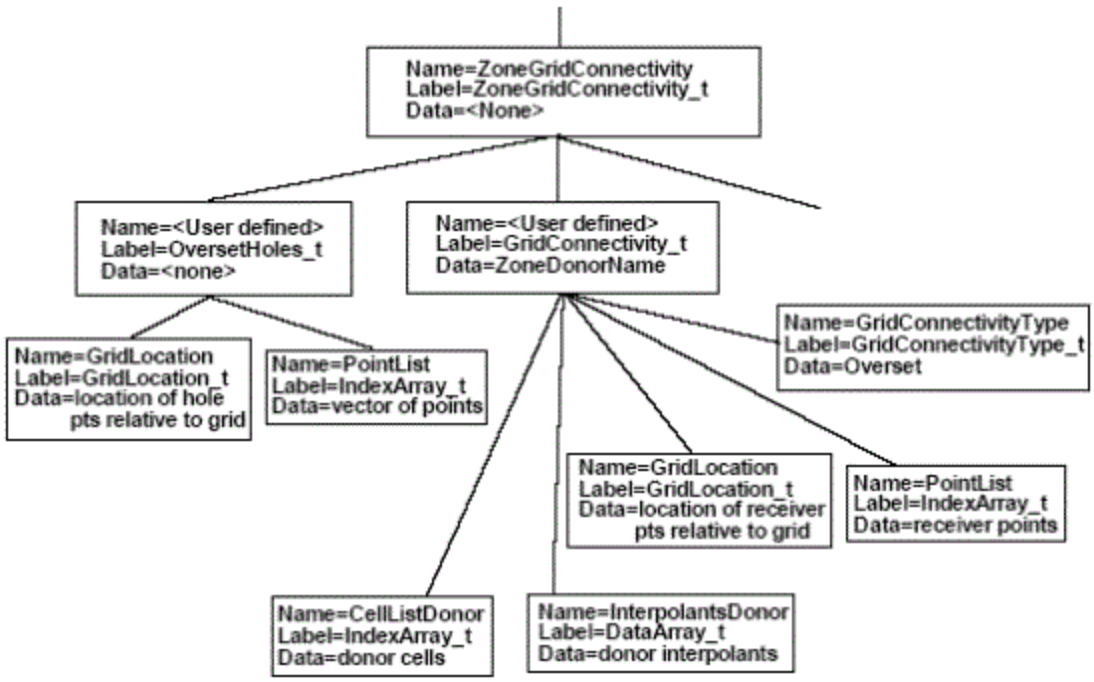


图 26　重叠接口的 ZoneGridConnectivity_t 层级结构

C. 5 结构块的例子

下面是一个结构网格的例子。它对应着 SIDS 文档中 8-A 的例子[1]。它是一个 3 维 2 块的情况，两个块在其每个面处以 1 对 1 方式连接在一起。块 1 为 9×17×11，块 2 为 9×17×21。块 1 的 *k*-max 面邻接块 2 的 *k*-min 面。

图 27～30 中给出层级结构图。为了更清楚，图中只给出了层级中直接相关的部分。例如，DataClass_t 、ReferenceState_t、ConvergenceHistory_t、 FlowEquationSet_t 和 ZoneBC_t 都省略掉了。但是，并不要求这些（和其他）项目，图形仍然代表有效的符合 SIDS 的 CGNS 文件。

注意，MT 的数据类型说明节点里没有存储数据。

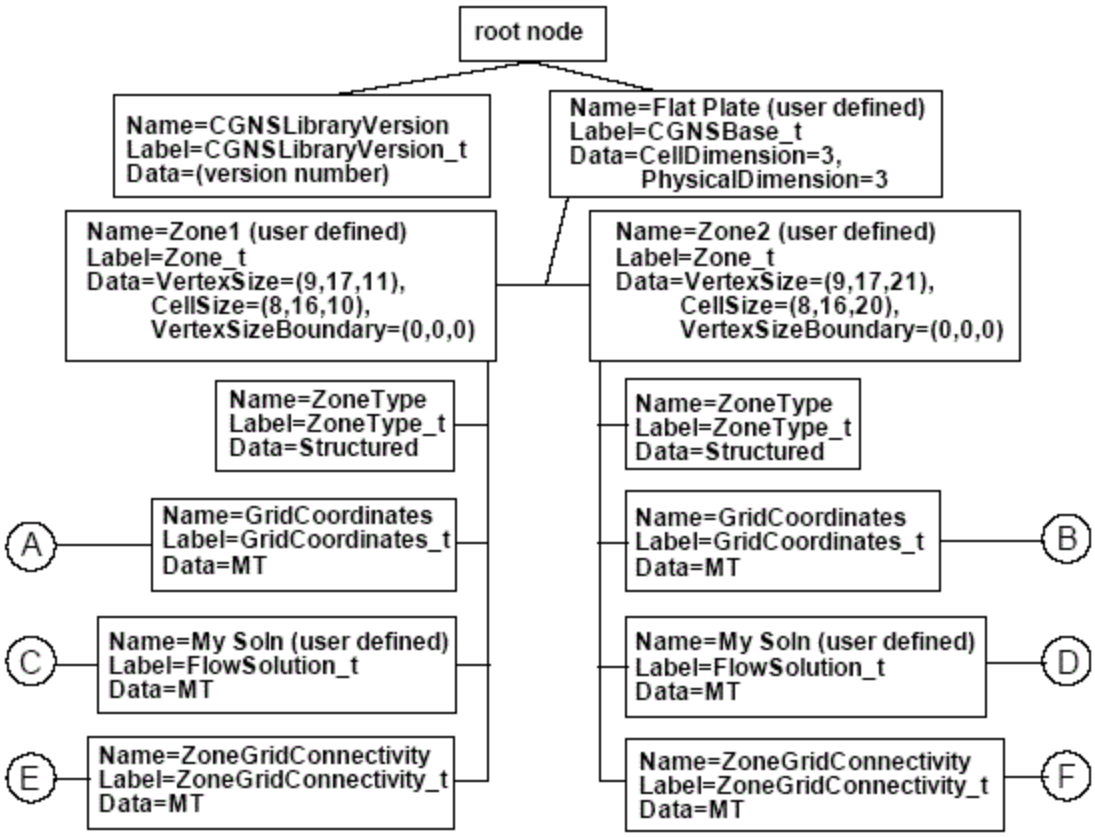


图 27　包括 2 个结构块的情况的 CGNS 顶层

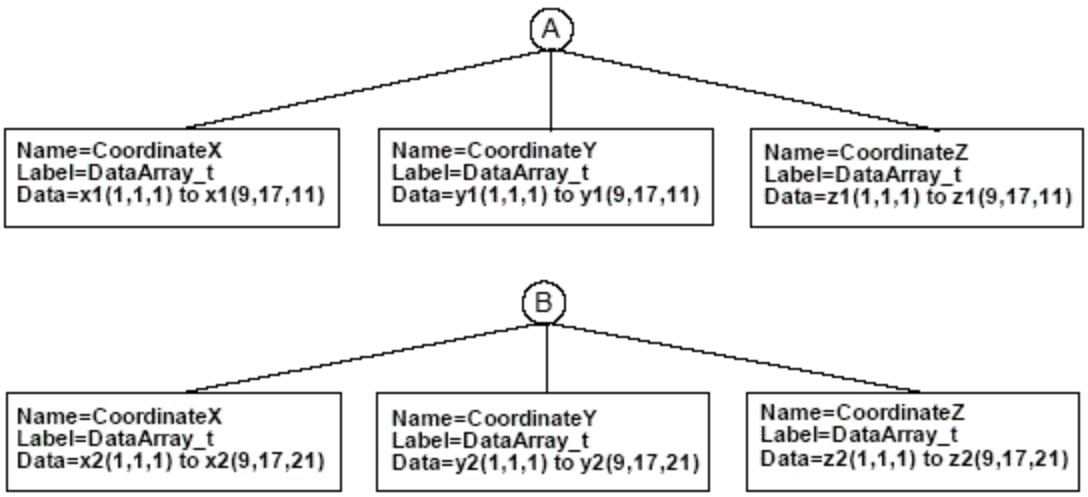


图 28　结构块实例的 GridCoordinates_t 节点

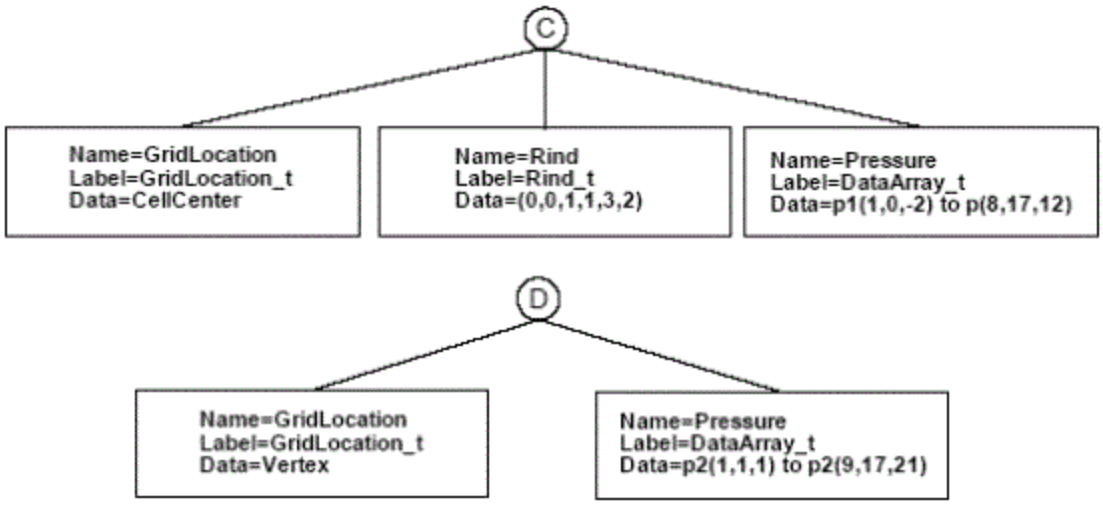


图 29　结构块实例的 FlowSolution_t 节点

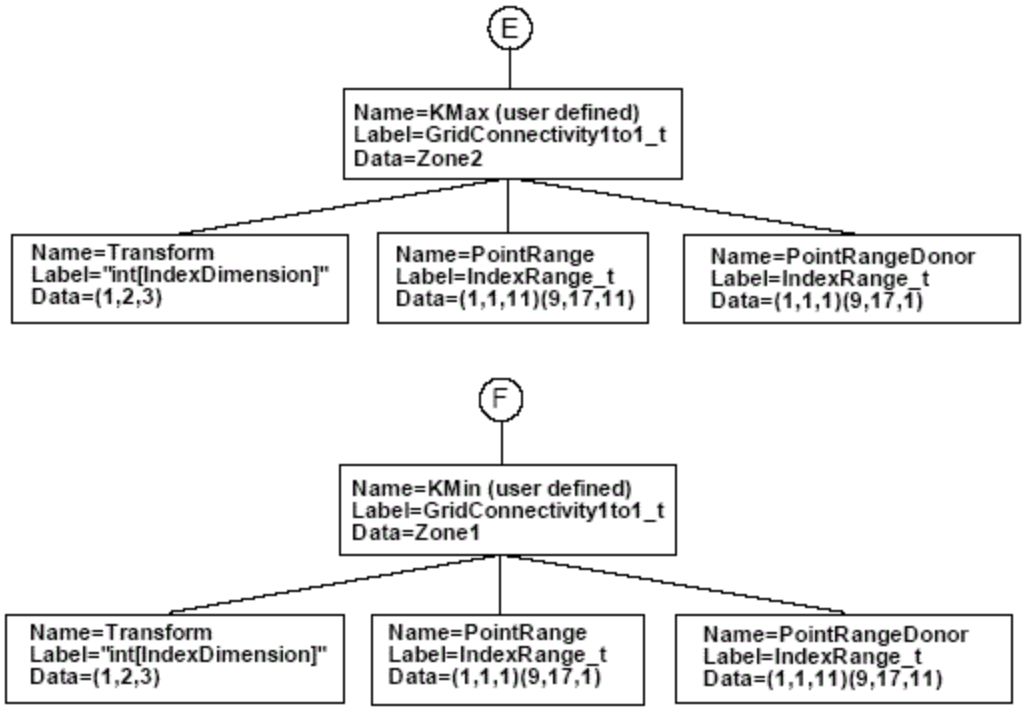


图 30　结构块实例的 ZoneGridConnectivity_t 节点

在这个例子中，在单元格中心给出块 1 的流动解，而在顶点给出块 2 的流动解（见图 29）。换句话说，块 1 的解的点与网格点并不是一一对应的（跟块 2 的不一样）。它们限定在被网格点所包围的体积里面。为便于说明，这个例子以这种方式建立，但它不是常见的；比较典型的是，人们对整个文件一般只会采用一个流动解数据位置。

这个例子也说明了 Rind_t 节点的采用，以及它是如何影响在 FlowSolution_t 下的数组的。FlowSolution_t 下的一个外层节点用来指出流动解正在输出多余的外层或“虚假”数据，这些数据在块的一个或多个边界之外。(外层节点也可以在 GridCoordinates_t 和 DiscreteData_t 下采用。)它只可以用于结构块。更完全的描述见 SIDS 文档[1]。在本例的块 1 中，*i* 方向上没有额外的虚假单元数据，*j*-min 和 *j*-max 附近各有一个虚假单元，*k*-min 附近有 3 个，*k*-max 附近有 2 个。(不

可否认，这个例子很大程度上是人为的——大部分实际应用在外层单元的使用上会更一致。）由于有外层单元，块 1 中所有流动解数据组在 *i*，*j*，和 *k* 方向的范围会得到适当的拓展。

对用户来说，认识到将外层单元包括进去会影响到数据在 DataArray_t 中的存储方式是很重要的。换句话说，当读 CGNS 文件时，如果存在 Rind_t 节点，人们不可能忽略掉它们；试图利用未更改的 VertexSize 或 CellSize 大小读取 DataArray_t，会导致无意义数据的恢复。

注意，SIDS 详细叙述了许多默认值。例如，Transform 默认值是（1，2，3），GridLocation 的默认值是 Vertex。因此，本例中包含这些特殊值的节点不是严格需要的。API 有时省略掉默认的信息。

本例还说明了另一个重要的事实。当一种（给定标示的）节点类型的名称由用户定义时，如果它们有同样的父节点，那么它们的名称必须是不同的。比如，本例中的两个 Zone_t 节点必须有不同的名称（请回忆一下前面关于块命名的讨论）。但是，当它们处于层级的不同位置时，有同样标示的两个节点可以有相同的名称。比如，本例中处于两个不同块的两个 FlowSolution_t 节点，都取了同样的用户定义的名称“My Soln”。

最后，虽然本例中没有包括 ZoneBC_t 节点，但请注意，如果将它们包括进去，那么，它们应该对除了块 1 的 *k*-max 面和块 2 的 *k*-min 面之外的所有边界面上的边界条件进行描述。边界条件中没有包括这两个面，是因为它们已经被定义为连接性接口。

附录 D　PLOT3D 变量应用指南

CGNS 留有很宽的余地，允许用户基本上将任何东西放进 CGNS 文件里。虽然这一点从文件的可扩展性的角度来看是很管用的，却使得在没有一系列细致的校对和翻译的情况下读别人的 CGNS 文件变得更困难。这种情况确实存在，不仅因为变量的输出有许多选择，而且因为 CGNS 中允许有很多形式的有量纲和无量纲的数据存在。

目前，CFD 界有许多人以 PLOT3D 格式输出流场数据[7]，尤其是用于后置处理的可视化程序中。从某种意义上，它已经成为了共享 CFD 数据的 de facto 标准。由于这种格式运用得非常普遍，在这个附录中我们给出了在 CGNS 文件中输出和读入这类数据的一个指南。如果你遵循这个指南，很可能其他用户就能够更容易地读懂和解释你的 CGNS 文件。

PLOT3D 标准网格变量是 x、y 和 z（在 3-D 中）。这些坐标可能是有量纲的或无量纲的。为了遵循这个指南，必须给出 3 个坐标 CoordinateX、CoordinateY 和 CoordinateZ（有量纲的或是无量纲的）。可能也会有“iblank”信息存在，它和重叠网格有关。如果要采用它，就将重叠洞的序列存储在 OversetHoles_t 节点下（见 SIDS 文档[1]）。本附录不包括网格坐标各种不同的量纲化和无量纲化的选择方案。一般说来，从便捷的角度来看，网格坐标的单位和/或无量纲化的问题不像“Q”变量的这类问题那么至关重要，下面将详述这类问题。但是，还是应该遵循 SIDS 标准，正确定义在 CGNS 文件中采用的网格单位或无量纲化方式。

PLOT3D 标准“Q”变量是（在 3-D 中）：

ρ / ρ_{ref} = 无量纲密度

$\rho u /(\rho_{ref} a_{ref})$ = 无量纲 x 方向动量

$\rho v /(\rho_{ref} a_{ref})$ = 无量纲 y 方向动量

$\rho w /(\rho_{ref} a_{ref})$ = 无量纲 z 方向动量

$\rho e_0 /(\rho_{ref} a_{ref}^2)$ = 无量纲单位体积总能量

式中，a 是声速，ref 表示参考状态。标准的 PLOT3D Q 文件还规定了参考 M 数、Re 数、攻角和时间值。为便于讨论，将不涉及时间值。如果需要的话，CGNS 的确有存储时间精确数据的能力（见 3.6 节），但这种数据不包括在这个 PLOT3D 指南中。下面我们会谈到 CGNS 存储 M 数、Re 数和攻角（间接地）的习惯。

上面的 5 个流场变量都有一个标准名称，在 SIDS 中进行了定义，它们分别是：Density，MomentumX，MomentumY，MomentumZ 和 EnergyStagnationDensity。为了遵循这个指南，这 5 个变量应该输出到你的 CGNS 文件中（在 3-D 情况下），并且你期望从别人的 CGNS 文件中读出的，如果它们也遵循这个指南的话。

CGNS 中允许有多个库，但是，为了进一步加强类似 PLOT3D 的数据组的可移植性，本指南只推荐 1 个 CGNSBase_t 节点。换言之，多个算例（例如不同攻角情况）应该存储在只有 1 个库的几个单独的 CGNS 文件下，而不是存储在有多个库的单个文件中。

在 CGNS 文件中可以输出的 3 个最通用的数据类型是：

DataClass = Dimensional

DataClass = NormalizedByDimensional

DataClass = NormalizedByUnknownDimensional

第一种类型表示数据有量纲单位；第二种类型表示数据已被已知的参考值无量纲化，它们在 CGNS 文件中是规定了的；第三种类型表示数据是无量纲的，但参考值没有规定或是未知的。因为 CGNS 处理每一种数据的方式稍有不同，因此，我们将分节给出这 3 种类型的指导方针。

D. 1 Dimensional 数据

输出量纲数据：

（1）在 CGNSBase_t 下，设置 DataClass = Dimensional；

（2）在 CGNSBase_t 下，设置 ReferenceState；在 ReferenceState 下，设置 Density 和 VelocitySound 的量纲参考值。在本指南中，这些值的单位必须相互一致，且与 FlowSolution 中给出的 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity 单位一致（比如，全部是米-千克-秒（MKS）单位制）。在 ReferenceState 下还要设置 Mach，Reynolds，VelocityX，VelocityY 和 VelocityZ。

（3）在 FlowSolution 下，设置量纲变量 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity。在本指南中，这 5 个变量的单位必须相互一致，且与 ReferenceState 中 Density 和 VelocitySound 的单位一致。

读入量纲数据（即，在 CGNSBase_t 下，如果 DataClass = Dimensional）：

（1）在 ReferenceState 下（直接在 CGNSBase_t 下），读 Density、VelocitySound、Mach 和 Reynolds。如果需要参考状态下的攻角，还要读入 VelocityX、VelocityY 和 VelocityZ。

（2）在 FlowSolution 下，读 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity。

（3）要获得 PLOT3D Q 变量，按以下步骤：

ρ / ρ_{ref} = Density / Density(ref)

$\rho u /(\rho_{ref} a_{ref})$ = MomentumX / (Density(ref) * VelocitySound(ref))

$\rho v /(\rho_{ref} a_{ref})$ = MomentumY / (Density(ref) * VelocitySound(ref))

$\rho w /(\rho_{ref} a_{ref})$ = MomentumZ / (Density(ref) * VelocitySound(ref))

$\rho e_0 /(\rho_{ref} a_{ref}^2)$ = EnergyStagnationDensity / (Density(ref) * VelocitySound(ref)2)

D. 2 NormalizedByDimensional 数据

输出具有已知归一化量的无量纲数据：

（1）在 GNSBase_t 下，设置 DataClass = NormalizedByDimensional。

（2）在 CGNSBase_t 下，设置 ReferenceState；在 ReferenceState 下，设置 Mach，Reynolds，VelocityX，VelocityY 和 VelocityZ。然后设置下面任意一个：

• Density 和 VelocitySound 的量纲参考值。在本指南中，这些值的单位必须相互一致，且与 FlowSolution 中给出的原始（量纲）数据 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity 的单位一致，优先于归一化量。

• Density 和 VelocitySound 的无量纲参考值，以及它们对应的 ConversionScale 和 ConversionOffset 值。在本指南中，原始（量纲）数据 Density 和 VelocitySound 的单位必须相互一致，优先于利用 ConversionScale 和 ConversionOffset 进行归一化的量，且与 FlowSolution 中给出的原始（量纲）数据 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity 的单位一致，优先于归一化量。

（3）在 FlowSolution 下，设置无量纲变量 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity，以及它们对应的 ConversionScale 和 ConversionOffset 值。在本指南中，原始（有量纲）变量的单位必须互相一致，优先于利用 ConversionScale 和 ConversionOffset 进行归一化的量，且与 ReferenceState 中的原始（量纲）变量 Density 和 VelocitySound 的单位一致。

阅读具有已知归一化量的无量纲数据（即，在 CGNSBase_t 下，如果 DataClass = NormalizedByDimensional ）：

（1）在 ReferenceState 下（直接在 CGNSBase_t 下），读 Density 和 VelocitySound。还要读它们的 ConversionScale 和 ConversionOffset 值，如果存在的话。另外，读 Mach 和 Reynolds 数。如果需要参考状态下的攻角，还要读 VelocityX、VelocityY 和 VelocityZ。

（2）在 FlowSolution 下，读 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity。还要读每一个 ConversionScale 和 ConversionOffset 值。

（3）要获得 PLOT3D Q 变量，按以下步骤。首先，只有当它们作为无量纲量给出时（下面用“′”表示），通过下面的方式恢复 Density 和 VelocitySound 的原始（量纲）参考值：

Density(ref) = Density'(ref)*ConversionScale + ConversionOffset

VelocitySound(ref) = VelocitySound'(ref)*ConversionScale + ConversionOffset

然后按以下方式做：

ρ / ρ_{ref} = (Density*ConversionScale + ConversionOffset) / Density(ref)

$\rho u /(\rho_{ref} a_{ref})$ = (MomentumX*ConversionScale+ConversionOffset)/(Density(ref)

* VelocitySound(ref))

$\rho v /(\rho_{ref} a_{ref})$ = (MomentumY*ConversionScale + ConversionOffset)/(Density(ref)

* VelocitySound(ref))

$\rho w /(\rho_{ref} a_{ref})$ = (MomentumZ*ConversionScale+ConversionOffset)/(Density(ref)

* VelocitySound(ref))

$\rho e_0 /(\rho_{ref} a_{ref}^2)$ = (EnergyStagnationDensity*ConversionScale+ConversionOffset)

/ (Density(ref) * VelocitySound(ref)²)

注意，PLOT3D Q 变量的换算比例和偏移量可能跟参考条件是对应的。这意味着变量可以不需要上面的换算而直接输出。但是，CGNS 允许变量通过与参考条件无关的性质进行归一化，因此，推荐上面的程序以避免模棱两可。

D. 3 NormalizedByUnknownDimensional 数据

输出具有未知归一化量的无量纲数据：

（1）在 CGNSBase_t 下，设置 DataClass = NormalizedByUnknownDimensional.。

（2）在 CGNSBase_t 下，设置 ReferenceState；在 ReferenceState 下，设置 Density = 1 和 VelocitySound = 1。还要设置 Mach、Reynolds、VelocityX、VelocityY 和 VelocityZ。

（3）在 FlowSolution 下，设置无量纲变量 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity。这些变量必须用以下方式无量纲化：

ρ/ρ_{ref}， $\rho u/(\rho_{ref}a_{ref})$， $\rho v/(\rho_{ref}a_{ref})$， $\rho w/(\rho_{ref}a_{ref})$， $\rho e_0/(\rho_{ref}a_{ref}^2)$。

（在 ReferenceState 中设置 Density = 1 和 VelocitySound = 1，确定了特殊的无量纲化量，该量是上面给 PLOT3D 变量规定的；详细情况及其他例子见 SIDS 文档[1]。）

阅读具有未知归一化量的无量纲数据（即，在 CGNSBase_t 下，如果 DataClass = NormalizedByUnknownDimensional）：

（1）在 eferenceState 下（直接在 CGNSBase_t 下）进行检验，以保证 Density= 1 及 VelocitySound = 1。然后，读 Mach 和 Reynolds。如果需要参考状态下的攻角，还要读 VelocityX、VelocityY 和 VelocityZ。

（2）在 FlowSolution 下，读 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity。

这种情况下要获得 PLOT3D Q 变量什么也无须做，它们已经是正确的形式。

D. 4 注意事项

（1）除了 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity 这几个流场变量以外（在 FlowSolution 下），如果需要，你还可以输出其他变量，但必须保证有这 5 个。

（2）可能还需要在 ReferenceState 下放置其他参考值（比如，可能需要 LengthReference 来确定跟网格坐标有关的参考长度），但是采用 Density 和 VelocitySound 来确定 PLOT3D Q 变量的无量纲化量就足够了。

（3）在 ReferenceState 下的量 Mach、Reynolds、VelocityX、VelocityY、VelocityZ、Density 和 VelocitySound（及其他任何量）都必须代表相同的流动参考状态。对外流空气动力学而言，这个状态通常被当作自由流，但也不一定必须是这样。

（4）从 PLOT3D 方面来说，速度分量只是用来提供参考状态下流场的攻角。在 3-D 里，攻角本身的定义不是唯一的，因此，SIDS 中也没有它的标准。比如，有一套可能的角度定义方法，它假定 z 方向是“向上的”，并使得：

$u = V\cos\beta\cos\alpha$

$v = -V\sin\beta$

$w = V\cos\beta\sin\alpha$

其中，$V = \sqrt{u^2+v^2+w^2}$，α是攻角，β是侧滑角。这样，攻角可以利用 $\alpha = \tan^{-1}(w/u)$ 得到，其中 u = VelocityX，w = VelocityZ。

（5）读别人的 CGNS 文件时，正确解读和/或使用它的低级别的方法如下。首先，查看是否只有 1 个 CGNSBase_t 节点。（正如前面讨论的，一般情况下允许有多个库，但是在这个指南里只允许有 1 个存在。）第二，保证在每一个块的 GridCoordinates_t 节点下变量 CoordinateX、CoordinateY 和 CoordinateZ 都存在，在每一个块的 FlowSolution_t 节点下变量 Density、MomentumX、MomentumY、MomentumZ 和 EnergyStagnationDensity 都存在。（注意：对时间精确数据组，每个块下可能有多个 GridCoordinates_t 和 FlowSolution_t 节点（见第 3.6 节），但本 PLOT3D 指南没有包括这种情形。）然后，在文件中搜索下面的特征参数：

- 如果 DataClass = Dimensional，那么 ReferenceState（直接在 CGNSBase_t 下）必须包含 Density、VelocitySound、Mach 和 Reynolds。只有当需要参考攻角时，在 ReferenceState 下才需要 VelocityX、VelocityY 和 VelocityZ。
- 如果 DataClass = NormalizedByDimensional，那么 ReferenceState（直接在 CGNSBase_t 下）必须包含 Density、VelocitySound、Mach 和 Reynolds。只有当需要参考攻角时，在 ReferenceState 下才需要 VelocityX、VelocityY 和 VelocityZ。而且，对于在 FlowSolution 下的 5 个流场变量的每一个，都必须有 ConversionScale 和 ConversionOffset。对于在 ReferenceState 下的变量，可以有也可以没有 ConversionScale 和 ConversionOffset。
- 如果 DataClass = NormalizedByUnknownDimensional，那么 ReferenceState（直接在 CGNSBase_t 下）必须包含 Density = 1、VelocitySound = 1 以及 Mach 和 Reynolds。只有当需要参考攻角时，在 ReferenceState 下才需要 VelocityX，VelocityY 和 VelocityZ。

如果遇到这些情况，那么一个低级别的读者就可以遵循上面小节里列出的指导方针，利用上面给出的过程可以轻易地获得 PLOT3D 变量。更高级别的读者将可以检测大小和换算比例的一致性，以确保遵守指南的原则。

参 考 文 献

[1] CGNS Project Group, \The CFD General Notation System Standard Interface Data Structures," Version 2.0 beta 2, February 2001; http://www.grc.nasa.gov/www/cgns/sids/index.html

[2] CGNS Project Group, \The CFD General Notation System Advanced Data Format (ADF) User's Guide," April 2001; http://www.grc.nasa.gov/www/cgns/adf/index.html

[3] CGNS Project Group, \The CFD General Notation System SIDS-to- ADF File Mapping Manual," Version 1.2 revision 8, February 2001; http://www.grc.nasa.gov/www/cgns/_lemap/index.html

[4] Poirier, D. M. A., Allmaras, S., McCarthy, D. R., Smith, M., and Enomoto, F., \The CGNS System," AIAA Paper 98-3007, June 1998.

[5] CGNS Project Group, \The CFD General Notation System Mid-Level Library," July 2001; http://www.grc.nasa.gov/www/cgns/midlevel/index.html

[6] Poirier, D. M. A., Bush, R. H., Cosner, R. R., Rumsey, C. L., and McCarthy, D. R., \Advances in the CGNS Database Standard for Aerodynamics and CFD," AIAA Paper 2000-0681, January 2000.

[7] Walatka, P. P., Buning, P. G., Pierce, L., Elson, P. A., \PLOT3D User's Guide," NASA TM 101067, March 1990.