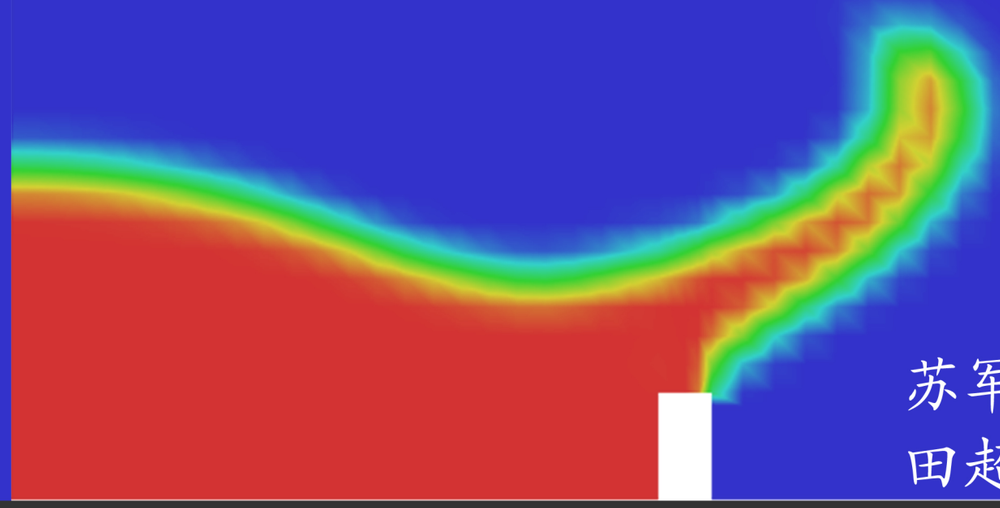
The open source CFD toolbox

“OpenFOAM研究”博文集



苏军伟 著
田超 编纂

“OpenFOAM 研究”博文集

【第一集】

苏军伟 原著

田 超 编纂

2011 年 3 月

说明

本文档内容是根据**苏军伟**博士的“OpenFOAM 研究”博客的博文整理而成，

OpenFOAM 研究网址：http://blog.sina.com.cn/openfoamresearch

OpenFOAM 开源计算群：34757558（国内最大，人数最多的在线 CFD 交流平台）

本博文可以作为初学者关于 OpenFOAM 的 Frequently Asked Questions (FAQ)文件，用于解答学习 OpenFOAM 中的基础问题，请 new Foamers 详细阅读自己感兴趣的部分。文中不免有疏漏和错误，恳请广大 Foamers 批评指正。

田 超

2011 年 3 月于北航

387210626（超人不会飞）

目录

1. 如何做动画

前几天有网友问怎么将计算结果输出成动画，我想可以分以下几个步骤
1）计算结果 case 文件夹中，输入 paraFoam 打开后处理软件 paraview，并作后处理作出你想要的效果。关于 paraview 怎么用？我以前在 QQ 爱好者群 2 中发过那个 guide，看看就行了，不是很难。
2）作出想要的效果后，点击 file 菜单下的 save animation 按钮，将每一针都输出成图片。
3)将所有输出的图片能到 windows 中，用 img2ani 将所有的图片转成动画，或者用 imageReady，后面这个软件功能强大，但是占的内存超大，动不动就几 G，你的图片多了，用前面那个处理方便多了就。

2. OpenFOAM 第 5 次 workshop

于 2010 年 6 月 21－24 日在瑞典 chalmers 科技大学举行。 这次 workshop 的 slide 可以从下面网址下载
http://web.student.chalmers.se/groups/ofw5/Program.htm
这次 workshop 的参加人数较多，涉及到很多领域。有兴趣的不妨去看看

3. OpenFOAM 中不可压缩湍流大涡求解器 oodles 说明

(2009-05-06 05:47:58)转载
标签： openfoam solver 教育 分类： OpenFOAM 求解器说明
本文谈谈 OpenFOAM 中不可压缩湍流流动大涡求解器 oodles,这也许和研究湍流的同志们课题更紧密一些。闲言少叙。
OpenFOAM 中湍流模型架构相似，所以大涡求解器和 RAS 求解器具有很大的相似之处，本文在介绍 oodles 时候和 RAS 求解器 turbFoam 进行比较。如果大家该求解器不熟悉请参看本站博文“OpenFOAM 中的不可压缩湍流流动求解器 turbFoam 的说明”。
（1）求解器位置：applications\solvers\incompressible\oodles
（2）求解器文件夹结构

```
|－Make
|       |-options
|       |-files
|-createFields.H
|-oodles.C
```

（3）求解器功能
任意不可压缩湍流流动，湍流模拟大涡模拟（LES)
（4）文件说明
1.options //编译选项，用于指定编译用到的头文件位置及其动态库

```
//文件内容
#用到的头文件文件夹
EXE_INC = \
#大涡湍流模型头文件
    -I$(LIB_SRC)/turbulenceModels/LES \
#大涡 delta 函数头文件
    -I$(LIB_SRC)/turbulenceModels/LES/LESdeltas/lnInclude \
#传输模型头文件，牛顿流体或者非牛顿流体选择。
    -I$(LIB_SRC)/transportModels \
#有限容积方法头文件
    -I$(LIB_SRC)/finiteVolume/lnInclude
#网格工具头文件
```

-I$(LIB_SRC)/meshTools/lnInclude \
#取样头文件，比如从场中取出模型符合一定条件的点
-I$(LIB_SRC)/sampling/lnInclude

#用到的动态连接库
EXE_LIBS = \
#不可压缩雷诺时均模型库
-lincompressibleLESModels \
#不可压缩传输模型库（牛顿流体传输模型和非牛顿流体传输模型）
-lincompressibleTransportModels \
#有限容积库
-lfiniteVolume \
#网格相关工具库
-lmeshTools
从上面的 options 可以看出，oodles 和 turbFoam 的编译选项仅仅差别在湍流模型的选择上。

2.files //用于指定当前要编译的文件，这里不包含头文件，都是*.C 文件。
//文件内容
oodles.C //主文件
//编译后求解器的名字和存放位置
EXE = $(FOAM_APPBIN)/oodles
3.createFields.H
createFields.H 中 oodles 求解器和 turbFoam 几乎完全相同。 差别在湍流模型的创建上。
turbFoam 创建了 RAS 模型。
autoPtr<incompressible::RASModel> turbulence
(
incompressible::RASModel::New(U, phi, laminarTransport)
);
而 oodles 创建了大涡模型
autoPtr<incompressible::RASModel> turbulence
(
incompressible::RASModel::New(U, phi, laminarTransport)
);
autoPtr 后面会有专门博文介绍这个只能指针，当前你就看成普通指针吧。 createFields.H 其他代码说明，请参看本站博文“OpenFOAM 中的不可压缩湍流流动求解器 turbFoam 的说明”。

4.oodles.C
//有限容积离散相关文件
#include "fvCFD.H"
//单相传输模型。
#include "incompressible/singlePhaseTransportModel/singlePhaseTransportModel.H"
//牛顿非牛顿传输
#include "incompressible/transportModel/transportModel.H"
//大涡模型库
#include "incompressible/LESModel/LESModel.H"
//文件输入流
#include "IFstream.H"
//文件输出流
#include "OFstream.H"
//随机数发生器
#include "Random.H"

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//主程序入口
int main(int argc, char *argv[])
{
    //设置根目录
     #include "setRootCase.H"
     //创建时间对象 runTime
     #include "createTime.H"
     //创建网格
     #include "createMeshNoClear.H"
    //前面刚谈到的那个文件
     #include "createFields.H"
    //连续性误差
    #include "initContinuityErrs.H"
     Info<< "\nStarting time loop\n" << endl;
     //主流程和 turbFoam 大部分一样，不再累述。具体参看本站博文“OpenFOAM 中的不可压缩湍流流动求解器 turbFoam 的说明”，下面主要针对差异的地方说明。
     for (runTime++; !runTime.end(); runTime++)
     {
          Info<< "Time = " << runTime.timeName() << nl << endl;
          #include "readPISOControls.H"
          #include "CourantNo.H"
//turbFoam 的 correct 在后面，大涡在前面，我感觉无所谓了
//就是时候在第一次动量预测的时候考虑湍流模型。
          sgsModel->correct();
          fvVectorMatrix UEqn
          (
               fvm::ddt(U)
             + fvm::div(phi, U)
 //亚格子模型，包括分子扩散项和亚格子应力项的偏分量。
             + sgsModel->divDevBeff(U)
          );
//如果动量预测，则求解动量方程。
          if (momentumPredictor)
          {
               solve(UEqn == -fvc::grad(p));
          }
          // --- PISO loop
          for (int corr=0; corr<nCorr; corr++)
          {
               volScalarField rUA = 1.0/UEqn.A();
               U = rUA*UEqn.H();
               phi = (fvc::interpolate(U) & mesh.Sf())
                     + fvc::ddtPhiCorr(rUA, U, phi);
               adjustPhi(phi, U, p);
               for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
               {
                    fvScalarMatrix pEqn
                    (
                         fvm::laplacian(rUA, p) == fvc::div(phi)
                    );
                    pEqn.setReference(pRefCell, pRefValue);
```

//下面也是和 TurbFoam 不同的地方，在非正交修正循环过程中，前面循环运用的是 fvSolutions 中 p 所指示的求解器，而最后一次迭代采用的是 pFinal 开头的求解器，我看了他们给出的算例，上面这两个求解器本身一样，差别在于残差不一样，pFinal 中的残差较小。这可能是开发者为了降低其计算量吧。
//除了下面的压力求解器选择和上面的亚格子模型选择不同外，oodles 和 turbFoam 没有其他差别，剩下代码说明，参看“OpenFOAM 中的不可压缩湍流流动求解器 turbFoam 的说明”，不再累述。

```
                if (corr == nCorr-1 && nonOrth == nNonOrthCorr)
                {
                    pEqn.solve(mesh.solver(p.name() + "Final"));
                }
                else
                {
                    pEqn.solve(mesh.solver(p.name()));
                }
                if (nonOrth == nNonOrthCorr)
                {
                    phi -= pEqn.flux();
                }
            }
            #include "continuityErrs.H"
            U -= rUA*fvc::grad(p);
            U.correctBoundaryConditions();
        }
        runTime.write();
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
    Info<< "End\n" << endl;
    return(0);
}
```

（5）编译求解器

打开控制台，进入求解器文件夹，输入 wmake

如果是想将前面的编译结构清除掉，输入 wclean

4. OpenFOAM 中的神奇方程定义方式的背后

(2009-05-04 16:50:35)转载

标签： openfoam 教育 分类： OpenFOAM 类解析

没有用过或者刚开始用 OpenFOAM 的人会感觉到 OpenFOAM 中的方程定义紧凑性所惊奇，我开始也是被 OpenFOAM 微分方程定义的精炼性所吸引的，从而放弃了 mfix 而转向 OpenFOAM 的。闲言少叙。

OpenFOAM 中的方程类为 fvMatrix，该类为一个基于有限容积的稀疏矩阵类，更确切的说她是一个类模板。对于定义标量方程 fvMatrix<scalar>，也可以写成 fvScalarMatrix，两者一样，因为 openfoam 运用了 typedef 进行了类型别名定义

```
typedef fvMatrix<scalar> fvScalarMatrix ; //标量
typedef fvMatrix<vector> fvVectorMatrix ; //矢量
```

搞 cfd 的人都知道，微分方程是通过离散从而转化为代数方程组进行求解。对于一个代数方程组主要有两部分：系数矩阵 A，右边值 b。构造成类似 Ax＝b 的代数方程形式。

方程的离散有显式和隐式,隐式离散得到方程稀疏矩阵 A 的，显式离散得到右边的值 b。一旦通过某种离散形式得到 A 和 b 就可以求解线性方程组了。

fvMatrix 中有两个基本变量 A 和 b 初始化为 0，用以存储系数矩阵 A 和右边值 b。

OpenFOAM 的所有的显式离散都在 fvc 空间中，所以以 fvc 开头的都为显式，比如 fvc::div(phi).隐式离散在 fvm 空间中，所有的隐式离散都是以 fvm 开头，比如 fvm::ddt, fvm::div(phi,U)等。所以，以 fvm 开头的项声称 A，以 fvc 开头的项生成 b。下面以的动量方程为例对他们的实现过程进行详述

```
fvVectorMatrix UEqn //定义一个 fvmatrix 对象，向量的方程
(
      fvm::ddt(U)
    + fvm::div(phi, U)
    -fvm::laplacian(nu, U)
    ==
    -fvc::grad(p)
);
```

fvm::ddt(U)隐式离散生成矩阵类，该项会被加到 UEqn 中的 A 上

fvm::div(phi, U)隐式离散矩阵类，该项加到 UEqn 中的 A 上

-fvm::laplacian(nu, U)隐式离散，该项从 UEqn 中的 A 上减下来，因为前面是负号。

==是 c＋＋优先级最小的符号，在 OpenFOAM 中他的功能和“－”相同。

-fvc::grad(p)，显式离散，该项加入到 UEqn 中的 b 上，前面“－”又有一个“＝＝”负负得正

这样稀疏矩阵 A 和右边的 b 在 UEqn 中已全部构建好，也就是得到方程了，求解即可。

solve(UEqn);

or

UEqn.solve();

功能一样，前面是全局函数，后面这个是 UEqn 的成员函数罢了。

也许你会感觉到纳闷，OpenFOAM 怎么知道他该加到 A 上还是 b 上。这点很简单，fvm 返回的是 fvmatrix, 而 fvc 返回的是 GeometryField;类型不一样，重载一下就行了。呵呵。

还有一个问题,如何避免下面的加和，U1 和 U2 属于不同的场

fvm::ddt(U1)+fvm::div(phi,U2)

所以在每次相加之前，他都会 check 一下，看看两个相加或者相减的两者是否是一个对象的操作（比较他们地址是否一样即可），因此 fvMatrix 里面还有一个重要变量 psi_，来记录当前求解变量的地址（实际上里面存有求解场的一个引用）.

现在明朗点了吗？

5. OpenFOAM 中雷诺时均湍流求解器 turbFoam 使用

(2009-05-05 00:17:01)转载

标签： openfoam 教育 分类： OpenFOAM 使用

本站博文“OpenFOAM 中的不可压缩湍流流动求解器 turbFoam 的说明”对 OpenFOAM 中的雷诺时均模型求解器详细介绍后，本文以该求解器下的算例 cavity（顶盖驱动流动）为例，介绍 OpenFOAM 中雷诺时均湍流模型的使用。

在介绍如何使用这个求解器之前，先介绍一下 OpenFOAM 中的湍流模型，OpenFOAM 中的湍流模型主要分为两大类：（1）RAS：雷诺时均模型（包括 k－e，k-omiga,1 方程模型，雷诺应力模型等）,(2)大涡（LES）模型（涡黏性模型，相似理论模型等）。LES 后面介绍 LES 求解器的时候再详细介绍。OpenFOAM 中支持不可压缩流的雷诺模型有

laminar 层流模型。

kEpsilon 标准 k？ ε 模型＋壁面函数

RNGkEpsilon RNG k？ ε 模型＋壁面函数
NonlinearKEShih 非线性 Shih k？ ε 模型＋壁面函数
LienCubicKE Lien 三次 k？ ε 模型＋壁面函数
QZeta q？ ζ 模型
LaunderSharmaKE Launder-Sharma 低雷诺数 k？ ε 模型
LamBremhorstKE Lam-Bremhorst 低雷诺数 k？ ε 模型
LienCubicKELowRE Lien 三次 低雷诺数 k？ ε 模型
LienLeschzinerLowRE Lien-Leschziner 低雷诺数 k？ ε 模型
LRR Launder-Reece-Rodi RSTM 模型＋壁面函数
LaunderGibsonRSTM Launder-Gibson RSTM 模型＋壁面反射项＋壁面函数
SpalartAllmaras Spalart-Allmaras 1 方程模型
上面这些模型，你可以运用 turbFoam 求解器中任意选择。

下面介绍 turbFoam 下的顶盖驱动流动。
（1）位置：
算例位置： /tutorials/turbFoam/cavity
求解器位置: applications/solvers/incompressible/turbFoam
（2）算例文件夹结构

```
|-system
|       |-fvSolution   //代数方程求解器选择文件
|       |-fvSchemes    //离散格式选择文件
|       |-controlDict //计算流程控制文件
|-constant
|       |-transportProperties   //传输参数控制文件，黏性等
|       |-RASProperties          //湍流模型选择文件
|       |-polyMesh                 //网格文件夹
|               |-blockMeshDict //blockMesh 网格设定文件
|               |-boundary       //边界文件，可有可无，blockMeshDict 会将其覆盖
|-0
   |-U   //速度边界条件，初始条件设定文件
   |-R   //雷诺应力边界条件，初始条件，仅仅当选择雷诺应力模型时候需要
   |-p   //压力边界条件，初始条件设定文件
   |-nuTilda //一方程模型中求解的那个变量，仅仅选择 SpalartAllmaras 湍流模型时候有用
   |-k      //湍动能设定文件， k？ ε 中的 k
   |-epsilon //湍流耗散律设定文件， k？ ε 中的 ε
```

(3)文件说明
system 文件夹下的三个文件以及压力文件 p 的说明请参看本站博文“使用 OpenFOAM 的基本流程”;polyMesh 中的文件及其速度文件 U 的说明请参看本站博文“深入解析 icoFoam 下的顶盖驱动流(cavity)”;k,epsilon 和 nuTilda 文件内容和压力文件 p 相似不再累述。
下面主要针对剩下的三个文件 transportProperties，RASProperties，R 进行说明
1.transportProperties

```
//文件头
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
//传输模型，牛顿流体，如果是非牛顿的话，选择下面 CrossPowerLaw 或者 BirdCarreau 黏性模型
transportModel    Newtonian;

//牛顿流体黏性恒定，  设定流体黏性系数。
//关于如何设置并使用参数,参看本站博文"OpenFOAM 中的参数字典使用剖析"
nu                      nu [0 2 -1 0 0 0 0] 1e-05;

//非牛顿流体黏性模型系数
//CrossPowerLaw 模型系数
CrossPowerLawCoeffs
{
      nu0                 nu0 [0 2 -1 0 0 0 0] 1e-06;
      nuInf               nuInf [0 2 -1 0 0 0 0] 1e-06;
      m                   m [0 0 1 0 0 0 0] 1;
      n                   n [0 0 0 0 0 0 0] 1;
}
//BirdCarreau 黏性模型系数
BirdCarreauCoeffs
{
      nu0                 nu0 [0 2 -1 0 0 0 0] 1e-06;
      nuInf               nuInf [0 2 -1 0 0 0 0] 1e-06;
      k                   k [0 0 1 0 0 0 0] 0;
      n                   n [0 0 0 0 0 0 0] 1;
}

2.RASProperties

FoamFile
{
      version       2.0;
      format        ascii;
      class         dictionary;
      object        RASProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
   //运用哪种湍流模型？右边的模型标识为下面的模型系数关键字去掉 Coeffs 即可。
   //比如 kEpsilon 是 kEpsilonCoeffs 去掉 Coeffs 后的词
RASModel                kEpsilon;

//是否运用湍流进行计算，on 为运用湍流模型，off 为不运用湍流模型
turbulence              on;

//在创建湍流模型对象时候是否将下面设定的参数打印到屏幕上，on 打印，off 不打印
printCoeffs             on;

//湍流模型系数，当你选择某一个湍流模型的时候，下面湍流模型的系数会自动调用。
//这些系数通常都是原始文献中模型提出者推荐的参数，通常采用默认值即可。
//究竟是哪种模型的系数请和上面的湍流模型对比。
laminarCoeffs
{
}
```

```
kEpsilonCoeffs
{
    Cmu             0.09;
    C1              1.44;
    C2              1.92;
    alphaEps        0.76923;
}

RNGkEpsilonCoeffs
{
    Cmu             0.0845;
    C1              1.42;
    C2              1.68;
    alphak          1.39;
    alphaEps        1.39;
    eta0            4.38;
    beta            0.012;
}

realizableKECoeffs
{
    Cmu             0.09;
    A0              4.0;
    C2              1.9;
    alphak          1;
    alphaEps        0.833333;
}

kOmegaSSTCoeffs
{
    alphaK1         0.85034;
    alphaK2         1.0;
    alphaOmega1     0.5;
    alphaOmega2     0.85616;
    gamma1          0.5532;
    gamma2          0.4403;
    beta1           0.0750;
    beta2           0.0828;
    betaStar        0.09;
    a1              0.31;
    c1              10;

    Cmu             0.09;
}

NonlinearKEShihCoeffs
{
    Cmu             0.09;
    C1              1.44;
    C2              1.92;
    alphak          1;
    alphaEps        0.76932;
    A1              1.25;
    A2              1000;
```

```
    Ctau1           -4;
    Ctau2           13;
    Ctau3           -2;
    alphaKsi        0.9;
}

LienCubicKECoeffs
{
    C1              1.44;
    C2              1.92;
    alphak          1;
    alphaEps        0.76923;
    A1              1.25;
    A2              1000;
    Ctau1           -4;
    Ctau2           13;
    Ctau3           -2;
    alphaKsi        0.9;
}

QZetaCoeffs
{
    Cmu             0.09;
    C1              1.44;
    C2              1.92;
    alphaZeta       0.76923;
    anisotropic     no;
}

LaunderSharmaKECoeffs
{
    Cmu             0.09;
    C1              1.44;
    C2              1.92;
    alphaEps        0.76923;
}

LamBremhorstKECoeffs
{
    Cmu             0.09;
    C1              1.44;
    C2              1.92;
    alphaEps        0.76923;
}

LienCubicKELowReCoeffs
{
    Cmu             0.09;
    C1              1.44;
    C2              1.92;
    alphak          1;
    alphaEps        0.76923;
    A1              1.25;
    A2              1000;
```

```
    Ctau1           -4;
    Ctau2           13;
    Ctau3           -2;
    alphaKsi        0.9;
    Am              0.016;
    Aepsilon        0.263;
    Amu             0.00222;
}

LienLeschzinerLowReCoeffs
{
    Cmu             0.09;
    C1              1.44;
    C2              1.92;
    alphak          1;
    alphaEps        0.76923;
    Am              0.016;
    Aepsilon        0.263;
    Amu             0.00222;
}

LRRCoeffs
{
    Cmu             0.09;
    Clrr1           1.8;
    Clrr2           0.6;
    C1              1.44;
    C2              1.92;
    Cs              0.25;
    Ceps            0.15;
    alphaEps        0.76923;
}

LaunderGibsonRSTMCoeffs
{
    Cmu             0.09;
    Clg1            1.8;
    Clg2            0.6;
    C1              1.44;
    C2              1.92;
    C1Ref           0.5;
    C2Ref           0.3;
    Cs              0.25;
    Ceps            0.15;
    alphaEps        0.76923;
    alphaR          1.22;
}

SpalartAllmarasCoeffs
{
    alphaNut        1.5;
    Cb1             0.1355;
    Cb2             0.622;
    Cw2             0.3;
```

```
    Cw3                  2;
    Cv1                  7.1;
    Cv2                  5.0;
}
//壁面函数系数
wallFunctionCoeffs
{
    kappa                0.4187;
    E                    9;
}
```

3.文件 R

本文件只有在选择雷诺应力模型（直接对雷诺应力建立方程，并不采用 Boussinesq 假设的 RAS 湍流模型）的时候才有用。

```
//文件头
FoamFile
{
    version         2.0;
    format          ascii;
    class           volSymmTensorField;   //这是一个体心存储的张量场
    object          R;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//单位
dimensions          [0 2 -2 0 0 0 0];
//初始场，张量 全为 0，由于对称张量只有 6 个分量
internalField     uniform (0 0 0 0 0 0);

boundaryField
{
    movingWall   //顶盖，第二类边界
    {
        type                zeroGradient;
    }

    fixedWalls     //墙，第二类边界
    {
        type                zeroGradient;
    }

    frontAndBack       //二维，前后面为 empty
    {
        type                empty;
    }
}
```

（4）程序运行

打开控制台进入/tutorials/turbFoam/cavity

输入: blockMesh 生成网格

输入: turbFoam 运行程序

程序运行结束后,输入 paraFoam 做后处理.

6. pimple 算法简述 (2009-09-30 09:22:33)转载

标签： openfoam 研究 教育 分类： OpenFOAM 求解器说明

终于回国了，事情比较多，blog 更新速度慢了些。 今天一起看看 OpenFOAM 中的 pimple 算法流程。 pimple 算法是 simple 算法和 piso 算法的结合体。

pimple 的基本思想是：将每个时间步长内用 simple 稳态算法求解（也就是将每个时间步内看成稳态流动），时间步长的步进用 piso 算法来完成。

在有限容积离散中，时间项的离散仍采用的差分格式，这样做可以得到某个时间点的流场信息，而非某个时间步长的内的平均值。采用传统的 piso 算法求解变化较快的流动的时候，需要的时间步长较小（因为相邻两个时间点的流场不能差别太大，否则会发散），这样会造成求解的某种流动需要的耗时过长。 pimple 算法将每个时间步长内看成一种稳态的流动（采用亚松驰来解决相邻两个时间段变化大的情况），当按照稳态的求解器求解到一定的时候，则采用标准的 piso 做最后一步求解。下面简单的将 pimpleFoam 流程

```
#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulenceModel.H"
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"
    Info<< "\nStarting time loop\n" << endl;
    while (runTime.run())    //时间步进
    {
        #include "readTimeControls.H"
        #include "readPIMPLEControls.H"    //pimple 控制
        #include "CourantNo.H"      //courant 数
        #include "setDeltaT.H"    //设置时间步长
        runTime++;    //步进时间
        Info<< "Time = " << runTime.timeName() << nl << endl;
        // --- Pressure-velocity PIMPLE corrector loop
        for (int oCorr=0; oCorr<nOuterCorr; oCorr++)      //外循环  simple 修正
        {
            if (nOuterCorr != 1)
            {
                p.storePrevIter();    //每次流程开始存储 p，以便连续性方程亚松弛
            }
            #include "UEqn.H"    //速度方程
            // --- PISO loop
            for (int corr=0; corr<nCorr; corr++)    //piso 修正
            {
                #include "pEqn.H"
            }
            turbulence->correct();    //湍流修正
        }
        runTime.write();    //输出
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "    ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
```

```
    Info<< "End\n" << endl;
    return 0;
}

tmp<fvVectorMatrix> UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  + turbulence->divDevReff(U)
);
if (oCorr == nOuterCorr-1)
{
    UEqn().relax(1);   //最后一步不进行亚松驰，以便采用标准的 piso 流程
}
else
{
    UEqn().relax();
}
volScalarField rUA = 1.0/UEqn().A();
if (momentumPredictor)
{
    //下面的条件语句可以使得最后一次的速度的求解残差和以前修正不一样。
    if (oCorr == nOuterCorr-1)
    {
        solve(UEqn() == -fvc::grad(p), mesh.solver("UFinal"));
    }
    else
    {
        solve(UEqn() == -fvc::grad(p));
    }
}
else
{
    U = rUA*(UEqn().H() - fvc::grad(p));
    U.correctBoundaryConditions();
}
```

压力方程

```
U = rUA*UEqn().H();
if (nCorr <= 1)
{
    UEqn.clear();
}
phi = (fvc::interpolate(U) & mesh.Sf())
    + fvc::ddtPhiCorr(rUA, U, phi);
adjustPhi(phi, U, p);
// Non-orthogonal pressure corrector loop
for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
{
    // Pressure corrector
    fvScalarMatrix pEqn
    (
        fvm::laplacian(rUA, p) == fvc::div(phi)
    );
    pEqn.setReference(pRefCell, pRefValue);
```

```
    if
    (
        oCorr == nOuterCorr-1
     && corr == nCorr-1
     && nonOrth == nNonOrthCorr
    )
    {
        pEqn.solve(mesh.solver("pFinal"));
    }
    else
    {
        pEqn.solve();
    }
    if (nonOrth == nNonOrthCorr)
    {
        phi -= pEqn.flux();
    }
}
#include "continuityErrs.H"
// Explicitly relax pressure for momentum corrector except for last corrector
if (oCorr != nOuterCorr-1)     //最后一次不采用亚松驰，采用标准 piso 流程。
{
    p.relax();
}
U -= rUA*fvc::grad(p);
U.correctBoundaryConditions();
```

上面没有注释的请参看本站其他求解器的说明。

7. 粒子方法讨论版开版

(2009-11-13 18:52:41)转载

标签： openfoam 研究 教育 分类： 其他

在赵伟国、孙中国、王勇和我几个人共同努力下，国内首个关于粒子方法讨论版在蓝色流体论坛正式开版。 欢迎相关领域的研究人员加入。下面是粒子方法的一个介绍。

粒子方法（Particle Method）是一系列新兴方法的总称，它摒弃了传统方法基于计算区域网格离散建立方程的方式，而是基于大量离散粒子的运动以及相互作用来重构介质的宏观力学行为，被誉为下一代数值模拟方法。与传统的基于网格剖分的数值模拟方法相比，粒子方法在处理计算区域存在较大变形的问题时，打破了计算节点间存在固定拓扑结构的限制，成功避免了网格畸变、新旧网格转换及网格质量评估等复杂过程；在处理自由表面捕捉、计算区域破碎和融合问题、非线性材料问题以及奇异性问题时，具有明显的优越性。常见的粒子方法有 Smooth Particle Hydrodynamics (SPH), Moving Particle Semi-implicit Method(MPS), Reproducing Kernel Particle Method (RKPM), Dissipative Particle Dynamics (DPD), Molecular Dynamics Simulation (MD), Discrete Element Method (DEM), Lattice Boltzmann Method (LBM), Monte Carlo Method (MCM)等。这些粒子方法建立在不同的时间空间尺度和物理系统上，并且已经取得了一定的成功。它们的拉格朗日共性，使得这类方法可以建立在统一架构下，并为多物理场及其多尺度之间的“无缝”耦合提供便利。

本版可以讨论各种关于粒子方法的算法理论、应用、软件程序实施过程，以及粒子方法之间或者与传统方法之间的相互耦合问题。希望通过粒子方法版的开版，将国内搞粒子方法的研究者联合起来，促使相互讨论与合作，使粒子方法在国内相关领域得到长足发展。

粒子讨论版网址：http://www.openfluid.cn/forum-70-1.html

8. 面向对象—我的一点理解

OpenFOAM 的 程序架构实在令人折服，最近在做有限容积格子波尔兹曼方法，感受颇深。有些 OpenFOAM fans 可能被 C++面向对象的东西搞的焦头烂额。面向对象说起容易，实现却难。

植入 openfoam 小功能容易，植入大模型比较难。其实大程序的实现， 并不是实现细节而是程序架构。决定程序是否成功，关键不在于写程序的人，而在于设计程序架构的人。软件开发而言，设计程序和写程序差别甚大。说白了，就是程序设计思想的事，软件开发过程中的思想培养才是最重要的。我对面向对象了解也是限于皮毛，现就我理解的面向对象给大家列于此，仅供参考。
下面从面向对象的四个基本特性——封装、继承、虚拟、多态给大家简单的介绍一下，并给出我在程序设计的时候一贯遵循的原则。
1）封装：
正如起表面意思所述，封装就是将一些东西封在一起，只能看到外表，不能看到内部。 对程序而言，就是对外隐藏实现，封装后的东西只需要向外提供一个通用的接口。只要接口不变化，实现变了，不会影响调用的程序。面向对象中经常提到的类 （class）就是为大家提供的一种封装平台，接口部分就是那个 public 里面的东西，而里面具体怎么实现的，类的使用者无需知道，就算里面实现变了， 也不会影响使用者的使用。 个人感觉，“封装”是面向对象程序设计的关键。如何封装，如何设计接口呢。 我一向奉行的原则就是——"发现变化然后封装"，这里就是封装的是变化，这里的变化并不是毫无关系的变化，是相似中的差异。比如，不同的湍流模型，他们是相似的，因为他们都是湍流模型，他们有差异，因为不同的湍流模型，实现是不一样的。这时候就要封装了，设计一个类，湍流模型为父类，不同的湍流模型为子类，变化作为他们的接口。
继承：
继承是子类具有父类的行为。有了他，一些父类的行为，无需在子类进行重新定义。 继承的就是那些相同的东西。如前面说的湍流模型，所有湍流模型的共同的东西，可以写在父类中，子类自动具有这种能力。 但是，在程序设计时候，应尽量少用继承，因为继承子类的访问会减慢程序的执行速度。 一种比较好的程序设计思想就是利用聚集，就是将各个功能分为很多小功能，然后每个小功能组成大功能。可以这样实现：定义大功能类，大功能类中带有小功能类 的一个对象，这样大功能的实现，可以通过不同的小功能协调来完成。 小功能类的封装也要遵循前面的封装原则。
虚拟和多态
虚拟和多态是上面封装继承的实现及其实现后所呈现出来的行为。
前面已经说了，封装的是变化，但是如何让这些变化对于不同的模型表现出不同的实现呢？那就靠虚拟。就是将那些变化的东西写成虚拟的接口，然后不同的模型对这些接口进行实现。这样对于同一个函数调用，对于不同的模型，表现出来的行为就不一样（因为他们的实现不一样）。用面向对象里面的话来说，对于相同的消息，不同对象将得到不 同的相应。

再举个简单的例子
老师说下课了，回家吧。
对于不同的同学，执行的行为肯定不一样（因为所有同学回家的路都不一样）。
变化——不同的同学，回家过程有差异，但他们有相似之处——都是回家
这时候就要封装了。封装什么阿？ ———变化，就是封装的是回家的行为
这样对不同同学给出不同的实现。 这样在老师说“回家吧”的时候，对于不同的同学对于这一消息，才会表现出不同的行为。

回头看看这个例子，封装的是什么？
变化（回家的行为）
什么变化？
不变中的变化（回家这个事）。
总而言之：面向对象设计个人感觉应该遵循一个基本原则：找到相同中的变化，封装之。
我说明白了吗？

9. 如何搞多面体网格

近几年 cfd 商业软件都在鼓吹多面体网格，多面体网格能够克服传统网格的的一些缺点：
1）有更多的临近单元，梯度计算及局部的流动状况更加准确

2）同样的体积，网格数目相对较少，从而减少计算量
3） 多面体网格对几何变形没有四面体敏感。
。。。
目前支持多面体网格的常用 cfd 软件主要有 star ccm+，fluent，OpenFOAM。 然而多面体网格的生成软件相对很少，目前似乎只有 star ccm+， fluent，engrid 和 openfoam 的 polyDualMesh. 下面简单的说一下怎么从这些软件里面做 openfoam 能用的多面体网格。
方法 1： engrid
直接使用 engrid，可以输出 openfoam 格式，该软件刚推出不久，网格质量不好说，没有测试过。
方法 2： Gambit（icemcfd，gridgen 等） -> fluent ->fluent3DToFoam
通过 gambit（icemcfd，gridgen 等)对你的几何做四面体网格（自动），然后输出 fluent 格式，读入的到 fluent 中，并通过 fluent 环境下的 mesh 菜单下的多面体网格对四面体网格进行转化，转换后 write fluent case（注意这里是 case，fluent 不支持网格的输出），该 case 文件中包含网格数据。 将 fluent 的 case 文件 copy 到 openfoam 你要计算的 case 文件夹中，通过 fluent3DToFoam 对网格进行转化。有 2 点需要注意：1）输出 case 的时候不要输出成二进制格式，否则到 openfoam 转化不成功 2）fluent 只会对四面体网格，楔型网格转化，6 面体不变，且只能转化一次。
方法 3：star ccm+ -> openfoam
多面体网格是 star ccm+的看家本领。可以直接在 star ccm+中进行多面体网格的划分，然后将网格输出，并利用 openfoam 的 ccm26ToFoam 进行转换。 应当注意 ccm26ToFoam 默认是编译的，你要手动编译的话，需要到 OpenFOAM-1.7.0\applications\utilities\mesh \conversion\Optional 中手动编译。
方法 4： 四面体网格->polyDualMesh
通过四面体网格生成器生成四面体网格，然后通过 polyDualMesh 将四面体转化为多面体。四面体网格必须是 Delaunay 型的，否则会转换不成功。 同时会生成大量的面，网格质量不好。

推荐采用方法 2 来获取多面体网格，质量比较好。要不要试一试？

10. OpenFOAM-1.6-ext 的安装过程探讨

(2010-11-27 23:51:35)转载
标签： openfoam 研究 分类： OpenFOAM 入门

OpenFOAM-1.6-ext 是以 jasak 为首的 OpenFOAM extension 工程的最新版本，昨天收到他们发布的消息，今天下载下来看了一下，发现里面有不少新的功能。 应当注意，extension 工程的 1.6 和官方的 1.6 并没有严格的对应关系，新发布的 1.6-ext 里面有也包含有官方 1.7 里面的新功能。 笔者发现，1.6-ext 的安装过程和官方的有较大的差异（主要体现在第三方包上）。为了是 fans 的安装过程更加顺利，以便尽早体验新功能，有必要重新探讨一下。

新的安装过程是基于 rpm 包重建的（也就是先将 source 包从网上 down 下来，然后进行 rpm 编译，并安装。使用这种方式可能和 jasak 热衷 OpenSUSE 有关，因为 OpenSUSE 是默认是基于 rpm 包的），对于 ubuntu 的用户来说出来需要以前提到的必须包外还需要 rpm 包，可以通过 sudo apt-get install rpm 来安装。 1.6-ext 和官方版的一个比较大的差别是第三方包不再和 OpenFOAM-version 在同一个文件夹，而是在其子文件夹中。同时，整个安装过程需要你联网，因为很多包都是通过联网获得的，否则你需要较多的手动设置。 安装之前要先 source 一下 etc 下的 bashrc 以更新环境。

ThirdParty 的安装过程分为 5 个等级，你可以根据自己的需要进行设置，虽然这样更加人性话，但是却增加安装难度。下面详细探讨一下 5 个过程
stag0: (文件：AllMake.stage0)
主要用来检查 rpm 相关项
stag1: (文件：AllMake.stage1)
安装 gcc,python, cmake 等
echo ==
echo Starting ThirdParty AllMake: Stage1

```
echo ======================================
echo

# Gcc and companion libraries   //如果你的系统中没有相关项，请去掉前面的#
#( rpm_make gmp-5.0.1          ftp://ftp.gnu.org/gnu/gmp/gmp-5.0.1.tar.gz                    )
#( rpm_make mpfr-3.0.0         http://www.mpfr.org/mpfr-current/mpfr-3.0.0.tar.gz            )
#( rpm_make gcc-4.4.5          ftp://ftp.gnu.org/gnu/gcc/gcc-4.4.5/gcc-4.4.5.tar.gz          )
#
#( rpm_make mpc-0.8.2          http://www.multiprecision.org/mpc/download/mpc-0.8.2.tar.gz )
#( rpm_make gcc-4.5.1          ftp://ftp.gnu.org/gnu/gcc/gcc-4.5.1/gcc-4.5.1.tar.gz          )

# Python //安装 python
#( rpm_make Python-2.7         http://www.python.org/ftp/python/2.7/Python-2.7.tgz           )

# cmake   //安装 cmake，如果系统有自己的 cmake 可以用#注释掉
( rpm_make cmake-2.8.3        http://www.cmake.org/files/v2.8/cmake-2.8.3.tar.gz            )

echo ======================================
echo Done ThirdParty AllMake: Stage1
echo ======================================
echo

stag2:  (文件：AllMake.stage2)
安装 openmpi

echo ======================================
echo Starting ThirdParty AllMake: Stage2
echo ======================================
echo

# MPI
#( rpm_make openmpi-1.4.1
http://www.open-mpi.org/software/ompi/v1.4/downloads/openmpi-1.4.1.tar.gz )
( rpm_make openmpi-1.4.3
http://www.open-mpi.org/software/ompi/v1.4/downloads/openmpi-1.4.3.tar.gz )
#( rpm_make openmpi-1.5
http://www.open-mpi.org/software/ompi/v1.5/downloads/openmpi-1.5.tar.gz     )

echo ======================================
echo Done ThirdParty AllMake: Stage2
echo ======================================
echo

stag3:  (文件：AllMake.stage3)
安装其他相关包
# Metis
( rpm_make metis-5.0pre2
http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/metis-5.0pre2.tar.gz          )

# ParMGridGen
( rpm_make ParMGridGen-1.0
http://www.mgnet.org/mgnet/Codes/parmgridgen/ParMGridGen-1.0.tar.gz           )
```

```
# Libccmio
#( rpm_make libccmio-2.6.1    )

# Mesquite
( rpm_make mesquite-2.1.2
http://software.sandia.gov/~jakraft/mesquite-2.1.2.tar.gz                    )

# Scotch
if [ -d "$OPENMPI_DIR" ]; then
    ( rpm_make scotch-5.1.10b
https://gforge.inria.fr/frs/download.php/27583/scotch-5.1.10b.tar.gz    )
else
    echo "WARNING: The OPENMPI_DIR environment variable is not set."
    echo "WARNING: Please make sure your environment is properly set up for openmpi. This is
necessary for compiling scotch-5.1.10b"
    echo "WARNING: Skipping the compilation of scotch-5.1.10b"
    echo ""
fi

# ParMetis
if [ -d "$OPENMPI_DIR" ]; then
    ( rpm_make ParMetis-3.1.1
http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/ParMetis-3.1.1.tar.gz )
else
    echo "WARNING: The OPENMPI_DIR environment variable is not set."
    echo "WARNING: Please make sure your environment is properly set up for openmpi. This is
necessary for compiling ParMetis-3.1.1"
    echo "WARNING: Skipping the compilation of ParMetis-3.1.1"
    echo ""
fi

echo ========================================
echo Done ThirdParty AllMake: Stage3
echo ========================================
echo
```

这里应当注意，安装过程中使用了 gmake，在 ubuntu 下是没有这个东西的， 直接用下面命令从 make 那做个软连接就行了 sudo ln -s /usr/bin/make /usr/bin/gmake 应该在 AllMake 之前执行。

stag4: (文件：AllMake.stage4)

安装 qt 和 paraview

```
cd ${0%/*} || exit 1

wmakeCheckPwd "$WM_THIRD_PARTY_DIR" || {
    echo "Error: Current directory is not \$WM_THIRD_PARTY_DIR"
    echo "    The environment variables are inconsistent with the installation."
    echo "    Check the OpenFOAM entries in your dot-files and source them."
    exit 1
}
. tools/makeThirdPartyFunctionsForRPM
#------------------------------------------------------------------------------

echo ========================================
```

```
echo Starting ThirdParty AllMake: Stage4
echo ========================================
echo

# qt-everywhere-opensource-src-4.7.0     //如果你没有安装 qt，将前面的＃去掉
#( rpm_make qt-everywhere-opensource-src-4.7.0
http://get.qt.nokia.com/qt/source/qt-everywhere-opensource-src-4.7.0.tar.gz )

# paraview     // 我安装过 qt，所以将所有关于 qt 的 check 去掉。 下面是 paraview 的最新版本，呵呵
#if [ -d "$QT_DIR" -a -r "$QT_DIR"/bin/qmake ]
#then
      ( rpm_make ParaView-3.8.1    http://www.paraview.org/files/v3.8/ParaView-3.8.1.tar.gz #\
    #         -D '_qmakePath          $QT_DIR/bin/qmake'
      )
#else
  #      echo "WARNING: Skipping the installation of ParaView-3.8.1."
  #    echo "WARNING: Please initialize the QT_DIR environment variable to your QT installation directory."
  # echo "WARNING: The command $QT_DIR/bin/qmake needs to be valid"
  #      echo "WARNING: "
#fi

echo ========================================
echo Done ThirdParty AllMake: Stage4
echo ========================================
echo
```

其实上面的 5 个 stage 你最好逐个执行(你也可以通过 AllMake 依次执行)，发现没有错误后再执行 OpenFOAM-1.6-ext 下的 Allwmake，否则你安装过程会很繁琐。 应当注意，安装完上面第三方包之后需要重新 source 一下 etc 下面的 bashrc，以更新环境。 之后进入 OpenFOAM-1.6-ext 并执行./Allwmake 就可以完成安装了。
不妨试一试， God bless you.

11. 多态实现及其子类父类数据传递的方式

(2010-09-02 11:34:43)转载
标签： openfoam 教育 分类： 其他
继承可以实现父类的行为向子类传递，由于父类本身并不知道用户根据哪个子类构造对象，如何让父类如何根据子类的状态调整自己的行为呢？
下面就传 递方法做了个总结，共分五种：
1）父类状态标志方法
顾名思义就是在父类声明 protected 的状态变量，子类根据自己的情况设置状态 变量的值。父类就可以根据该状态调整自己的行为。
2）状态虚构函数查询方法
在父类声明表示状态的虚拟函数，在子类对该虚拟函数进行重 写，并标明各个子类的状态
3）父类行为函数重载法
直接对子类体现不同行为的父类函数直接进行虚构，并在子类中进行重写。
4）函 数参数传递法
直接在父类状态定义在函数接口中，在子类进行重载
5) 模板类参数传递方法
根据参数不同自动调整行为，共性写在模板 类中，差异写在参数类中。 当然也可以专门为模板类某个成员函数针对某个特定参数类进行函数重改。
通过上述 5 种方法任意一种就可以轻松实现类的 多态行为。

12. OpenFOAM 与有限元程序包 deal.II 的无缝耦合方法

(2010-07-26 11:19:02)转载

标签： openfoam 研究 教育 分类： OpenFOAM 使用

OpenFOAM 是一个集成开发环境，理论上可以与任何程序耦合。 本文探讨在 OpenFOAM 环境下使用有限元分析程序 deal.II (http://www.dealii.org)的操作过程，这可能对于想用 fvm 和 fem 进行耦合计算（比如，fluid-structure interaction）的朋友有很大用途。个人认为 deal.II 是目前开源计算领域比较好的基于 fem 的程序包，有兴趣可以去他们主页看看。

操作方法由下面几步来完成

1）直接编译 deal.II, 方法比较简单，直接 ./configure && make all

2）编译成功以 后，他会在 lib 文件夹中生成相应所有的库，包括包括 debug 模式下的（*.g.*）和优化编译的库

3）将库的搜索路径加入到 PATH（搜索 用）中和 LD_LIBRARY_PATH（运行用）。可以手动加入，也可以通过 etc/settings.sh 加入，如 可以在 settings.sh 中加入下面两句话

_foamAddPath /home/sujunwei/OpenFOAM/deal.II/lib //增加搜索路径，需要换成你自己的路径

_foamAddLib /home/sujunwei/OpenFOAM/deal.II/lib //增加运行路径，需要换成你自己的路径

4）创建算例，可以尝试这将 step 改 为 OpenFOAM 编译的程序，实现方法为包括

加入 include 路径：

在 options 里面增加下面几句话

```
EXE_INC = \
-I ../../base/include \
-I../../lac/include \
-I../../deal.II/include \
-I../../contrib/boost/include \
-I../../contrib/boost/include/boost/fusion/include \
-I../../contrib/tbb/tbb22_20090809oss/include \
-I../../contrib/tbb/tbb22_20090809oss/src/rml/include \
-I../../contrib/hsl/include
```

上面为所有的 deal.II 的头文件目录。 想找到所有目录，可以直接在 deal.II 根目录下采用下面命令查询

```
find ./ -name include
```

加入 lib 路 径和加载库（根据你当前使用的库文件，注意应当去掉 lib 和 .so ），通常应该加载 base 和 lac 两个基本库和 deal_II_2d(3d,1d). .g.表明他是 debug 模式下的库

```
EXE_LIBS = \
    -L ../../lib \                          //增加搜索路径，否则通常无法编译
    -ldeal_II_2d.g \
    -lbase.g        \
    -llac.g
```

我已经将 deal.II 的第一个算例在 OpenFOAM 环境下编译成功，该算例可以到”OpenFOAM 开源计算“QQ 群共享去下载。

13. CAD->GAMBIT->CFD 几何

(2010-04-11 11:33:10)转载

标签： openfoam 教育 分类： 其他

工业设备的几何相对比较复杂，通常通过 CAD 来构造几何模型，然后将几何模型导入到网格划分软件中进行网格处理，最后导入到 CFD 软件进行计算。 然而，CAD 软件的模型精度相对较低（以实现较快的模型构造速度），很难满足实际 CFD 计算要求。

本文意在探讨在 CAD 构造过程及其 CFD 前处理网格软件的读入过程中的常见问题及其解决方案。本文以 gambit 为例。 笔者发现 gambit 是对 CAD 模型的输入支持功能是相当强的（起码要强于 pointwise 和 gridgen; icemcfd 没有用过,不是很清楚），他读入的模型有面的信息，体的信息，线的信息。

下面从 CAD 模型构造、导入过程和导入后的线面的处理过程进行探讨。

一、构造 CAD 模型中应当注意以下几点

1） 构造 CAD 模型过程用较小的参差，比如小于 1e-6

2） 在模型 CAD 模型构造过程中尽量运用他们基本模型，比如正方体（最好不要自己从点、线、面到体进行自行构造）

3） 导出的时候尽量采用 solid 几何模型，比如 parasolid，而不要采用基于点或者面的模型，如 igs

4） 如果可能在几何输出之前尽量检查几何的有效性。

5） 多次对同一个几何在 CAD 软件中进行输入输出，以保证几何能够进行重建

二、Gambit 导入 CAD 时候的几个选项

1）Heal Geometry

这个选项很重要，他主要做三件事：1. gambit 在导入几何的时候是采用样条来近似曲线，他负责将样条曲线（面）转化为分析曲线（面），比如球面，圆线（面）等。 2.修补病态拓扑结构。3. 将不封闭的面或者线进行延伸或者切除以便使得几何体封闭。 推荐选上

2）Make Tolerant

因为 CAD 输出的几何容差比较大，而 CFD 用的比较小，从而使得直接导入而造成的几何封闭等问题。 该功能能够通过降低参差使得几何进行封闭。 推荐选上

三、Gambit 对比较比较烂的 CAD 输出的处理

当几何读入后仍不能满足你的需要，则需要使用 Gambit 的修补工具。该工具在 Gambit 右上方最后一个按钮，点击后出现子按钮的倒查第二个按钮。 该按钮下面支持的修补有

1）clean up duplicate faces (volumes) 清除重复几何体（面或者体）

通过指定容差，找到重复的面或者体，清除掉

2） short edge 清除较小的边

指定最小的边长度，将其清除，可以选定某一个或者所有的全部清掉

3）faces with small area: 清除面积比较小的面

注意该小面清除后，这块区域和哪些面组合成一个面，在线面选择。

4） faces with sharp angles: 清除某个角度比较小的面。

注意清除后面的组合。

5） Sliver faces：清除狭长面

6）cracks：清除面缺陷

7） Holes：清除体缺陷

通过上面三个步骤应该可以实现 CAD 到 CFD 几何成功转换。 上游工作做好，可以减少下游的工作量。 要不要试一试？

14. OpenFOAM 中非均匀初始场的设定

(2010-02-04 11:48:59)转载

标签： openfoam 研究 教育 分类： OpenFOAM 使用

采纳网友的建议，这次讨论 OpenFOAM 中非均匀初始场设置问题。 对于 openfoam 非均匀场初始化现成程序有两个: 1)OpenFOAM 自带功能 setFields 2）社区中的 funkySetFields. 下面介绍一下两者的使用。

1 setFields 的使用

使用 setFields 只需要将 system 文件夹中建立 setFieldsDict，然后在其中设置相应的参数，设置完后，在 case 根目录下利用控制台输入 setFields 就可以了。 下面介绍一下 interFoam 算例中的 setFieldsDict.该字典位置在 OpenFOAM-1.6/tutorials/multiphase/interFoam/laminar/damBreak/system/setFieldsDict，文件中的内容为

```
FoamFile    //文件头
{
    version         2.0;
    format          ascii;
    class           dictionary;
    location        "system";
    object          setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues    //用来设定场的默认值
(
    volScalarFieldValue alpha1 0       //  设置场 alpha1 的默认值
);

regions    //设置 alpha1 不为 0  的区域
(
    boxToCell //  方形区域内的 cell
    {
        box (0 0 -1) (0.1461 0.292 1);    //方形区域的边界，两点 xyz 最小点和 xyz 最大点
        fieldValues     //用来指定定值
        (
            volScalarFieldValue alpha1 1 //用来 alpha1 场在上面 box 区域内的值的大小
        );
    }
);
```

setFields 对于规则区域定值使用比较方便，但是很难实现比较复杂内部场的设置，比如满足某种关系式的内部场设置。

2 funkySetFields 的使用

用来设定比较复杂的内部场。 可以指定满足一定条件的关系式的初始场。

2.1 软件的获取

funkySetFields 在官方或者 dev 版本里面都没有该功能，可以通过下面命令获取。运行下面的命令需要你的电脑安装 svn。

svn checkout https://openfoam-extend.svn.sourceforge.net/svnroot/openfoam-extend/trunk/Breeder_1.6/utilities/postProcessing/FunkySetFields/

2.2 软件的安装

该软件安装比较简单，直接进入该文件夹，并运行 Allwmake
./Allwmake

2.3 使用

2.3.1 常用关键字

field //用来指定要修改的场
expression_r //用来指定表达式
condition //用来指定上述表达式应当满足的条件
keepPatches //用来说明是否保持原来边界条件，最好加上，不加的话，funkySetField 会给所有边界为 0 梯度
create //用来说明是否是新建场
valuePatches //用来指定那些定值边界由临近内部节点值给定
dimension //用来指定新建立场的单位
time //用来指定 funkySetField 所指定的时间点
应当指出，上述关键字可以直接在控制台上输，也可以写在名字为 funkySetFieldsDict（类似于 setFieldsDict）中。

2.3.2 使用方法

方法 1 直接在控制台输入
直接进入你要初始话的 case 中，输入类似于下面的命令。 如上面的 setField 也可以通过下面的 funkySetFields 命令来实现

```
funkySetFields -time 0 -keepPatches -field alpha1 -expression_r "1" -condition "pos().x <= 0.1461 && pos().y <= 0.292"
```

注意比较长的式子用单引号或者双引号隔开。 上述关键字没有次序要求。
方法 2 使用 funkySetFieldsDict 字典
方法和上面 setFields 差不多，在 system 文件夹中建立 funkySetFieldsDict 字典文件，对于上面表达式可以通过下面字典文件实现。

```
expression_rs //设置复杂内部边界入口
 (
alpha // 设置 alpha1，名字任意
{
field alpha1; //操作的场
expression_r "1"; //表达式
condition "pos().x <= 0.1461 && pos().y <= 0.292" ;//执行上述表达式的条件
keepPatches true; //是否保持以前边界
}
pressure1 //设置压力，名字任意
{
field p; //指定操作场
expression_r "10.*(0.1-pos().y)"; //执行表达式子，无条件
}
pressure2
{
field p; 指定表达式
expression_r "p+U&U"; //表达式子
condition "pos().x > (max(pos().x)-min(pos().x))/2"; //条件
}
);
```

从上面可以看出，使用 funkySetFieldsDict 可以实现更为复杂内部场，且可以同时对多个场进行操作

2.3.3 表达式中的常用支持

也许你已经注意到，上面例子中 condition 和 expression_r 使用了表达式，funkySetFields 中你可以使用下列常用的操作符或者函数
1） C＋＋基本操作符
+,-,*,/，%， <,>,<=,>=,!=,==， &&,||，？ ：
2）OpenFOAM 定义的向量操作符
&,^
3)圆周率常量
pi
4）标量函数
pow,log,exp,sqr,sqrt,sin,cos,tan
5）OpenFOAM 中的一些函数

mag：求模
grad ：求标量梯度
curl ：求向量旋度
snGrad：表面法向剃度
div ：向量场散度
laplaction ：求一个场的 laplacian 项目
min,max ：标量场的最值
pos :网格中心位置矢量
fpos ：面中心位置矢量
face ：表面法向量场
area：表面面积场
vol ：网格单元体积场
deltaT ：时间步长
time ：当前时间

setFields 和 funkySetField 可以实现比较复杂的内部场初始化，而对于复杂的边界场，你要自定义边界条件了。

要不要试一下？

15. OpenFOAM-1.6 中 sample 的使用

(2010-01-28 19:49:39)转载
标签： openfoam 研究 教育 分类： OpenFOAM 使用

最近忙了一点，很长时间没有更新了。今天我们一起来看看 sample 后处理功能。

OpenFOAM 中 sample 用来从计算结果中取出符合某种要求的点集合，比如：某条线上的点集合或者取某个面上的点集。在 OpenFOAM-1.6 中比 1.5 版本功能有所增强。我们来一起看看 sample 的用法。使用 sample 需要在 system 文件夹中增加 sampleDict 参数字典用于指定你要取得点的限制。在下面路径文件夹中，有个 sampleDict 的例子
OpenFOAM-1.6\applications\utilities\postProcessing\sampling\sample。
下面简单的说一下 sample 的具体功能
1）取某条直线上的点
关键字：

setFormat raw; //用来指定直线上点的输出的格式，该值可以为 xmgr，jplot，gnuplot 分别用来指定软件 xmgr，jplot，gnuplot 能够识别的格式，raw 输出的是文本格式。

interpolationScheme cellPoint; //用来确定取指定点所用到的插值格式，该值可以为 cell（直接利用点 cell 的值），cellPoint（利用 cell 中心和单元节点插值），cellPointFace（利用单元中心，单元节点及其面心进行插值）。

```
    fields
    (
    p
    U
    );      // 用来指定要取的场，也就是从压力和速度场中取值

   sets
    （
        lineX1    //线的名字，可以为任意值
       {
        type            uniform;       //取点类型，该值可以为  uniform（均匀分布点），face（线与网格面的交点），midPoint（线与网格面交点的中点），midPointAndFace（线与网格面的交点及其相邻交点的重点），cloud（用来指定某些点）
        axis            distance;    //输出点值的同时输出的位置相关值信息。该值可以为：x(x 坐标）,y（y 坐标），z（z 坐标），xyz（xyz 坐标），distance（当前点离 start 的距离）。
         start          (0.02 0.051 0.005);      //起始点位置（x y z)
         end            (0.06 0.051 0.005);      //终点位置（x y z)
         nPoints        10;                  //取点的个数
        }
        对于 cloud 可以这样使用
        lineX2
        {
         type      cloud;
         axis      xyz;
         points    ((0.049 0.049 0.005)(0.051 0.049 0.005));      //用来指定要输出的所有点的位置。
       }
    ）;
```

2）取某个面上的点

```
    surfaceFormat vtk;   //输出面上点的格式，可以为 foamFile（像 OpenFOAM 存储网格一样存储点），dx（DX 可以认识的格式），vtk（vtk ascii 格式），raw（直接输出文本格式，点及其对应的值)。
    interpolationScheme     //见上面
    fields   //见上面
    surfaces      //可以取一个平面，一个边界面或者一个某个场等值面上的值的分布
    （
        constantPlane
        {
        type                plane;      //定义一个平面，该面要做三角化（因为有的软件只认识三角形表面网格）
        basePoint           (0.0501 0.0501 0.005);     //面过的点
        normalVector        (0.1 0.1 1);                //经过该点的法向量
        }
        movingWall_constant   // 面名字，可以任意
       {
        type                patch;      //在边界面取点
        patchName           movingWall;   //指定边界名字
        // triangulate      false;      //是否进行三角化，默认不进行
       }
    interpolatedIso      //曲面名字
    {
```

```
        type                isoSurface;     // 再一个等值面上取点，默认做三角化
        isoField            rho;            //等值面场
        isoValue            0.5;            //等值面的值
        interpolate         true;           //是否进行插值
    }
);
```

按照上述格式填写 sampleDict 后，将其放在 system 文件夹中，然后进入控制台进入该 case 的根目录，输入 sample 就可以将所有时刻符合要求的点取出来了。如果对 OpenFOAM 的 case 文件夹结构不熟悉的话，可以看看本站以前的博文。

试一试这个后处理功能？ Good Luck !!

OpenFOAM-1.6 中 sample 的使用 (2010-01-28 19:49:39)转载

标签： openfoam 研究 教育 分类： OpenFOAM 使用

最近忙了一点，很长时间没有更新了。今天我们一起来看看 sample 后处理功能。

OpenFOAM 中 sample 用来从计算结果中取出符合某种要求的点集合，比如：某条线上的点集合或者取某个面上的点集。在 OpenFOAM-1.6 中比 1.5 版本功能有所增强。我们来一起看看 sample 的用法。使用 sample 需要在 system 文件夹中增加 sampleDict 参数字典用于指定你要取得点的限制。在下面路径文件夹中，有个 sampleDict 的例子

OpenFOAM-1.6\applications\utilities\postProcessing\sampling\sample。

下面简单的说一下 sample 的具体功能

1）取某条直线上的点

关键字：

setFormat raw; //用来指定直线上点的输出的格式，该值可以为 xmgr，jplot，gnuplot 分别用来指定软件 xmgr，jplot，gnuplot 能够识别的格式，raw 输出的是文本格式。

interpolationScheme cellPoint; //用来确定取指定点所用到的插值格式，该值可以为 cell（直接利用点 cell 的值），cellPoint（利用 cell 中心和单元节点插值），cellPointFace（利用单元中心，单元节点及其面心进行插值）。

```
fields
(
p
U
);     // 用来指定要取的场，也就是从压力和速度场中取值

sets
(
    lineX1    //线的名字，可以为任意值
    {
        type            uniform;        //取点类型，该值可以为  uniform（均匀分布点），face（线与网格面的交点），midPoint（线与网格面交点的中点），midPointAndFace（线与网格面的交点及其相邻交点的重点），cloud（用来指定某些点）
        axis            distance;    //输出点值的同时输出的位置相关值信息。该值可以为：x(x 坐标）,y（y 坐标），z（z 坐标），xyz（xyz 坐标），distance（当前点离 start 的距离）。
        start           (0.02 0.051 0.005);      //起始点位置（x y z)
        end             (0.06 0.051 0.005);      //终点位置（x y z)
        nPoints         10;                  //取点的个数
    }
    对于 cloud 可以这样使用
    lineX2
    {
        type      cloud;
        axis      xyz;
```

```
        points    ((0.049 0.049 0.005)(0.051 0.049 0.005));      //用来指定要输出的所有点的位置。
    }
);
```

2）取某个面上的点

surfaceFormat vtk; //输出面上点的格式，可以为 foamFile（像 OpenFOAM 存储网格一样存储点），dx（DX 可以认识的格式），vtk（vtk ascii 格式），raw（直接输出文本格式，点及其对应的值)。

interpolationScheme //见上面

fields //见上面

surfaces //可以取一个平面，一个边界面或者一个某个场等值面上的值的分布

```
(
    constantPlane
    {
    type            plane;      //定义一个平面，该面要做三角化（因为有的软件只认识三角形表面网格）
    basePoint       (0.0501 0.0501 0.005);    //面过的点
    normalVector    (0.1 0.1 1);              //经过该点的法向量
    }
    movingWall_constant   // 面名字，可以任意
    {
    type            patch;      //在边界面取点
    patchName       movingWall;   //指定边界名字
    // triangulate  false;      //是否进行三角化，默认不进行
    }
interpolatedIso      //曲面名字
{
    type            isoSurface;      // 再一个等值面上取点，默认做三角化
    isoField        rho;             //等值面场
    isoValue        0.5;             //等值面的值
    interpolate     true;            //是否进行插值
}
);
```

按照上述格式填写 sampleDict 后，将其放在 system 文件夹中，然后进入控制台进入该 case 的根目录，输入 sample 就可以将所有时刻符合要求的点取出来了。如果对 OpenFOAM 的 case 文件夹结构不熟悉的话，可以看看本站以前的博文。

试一试这个后处理功能？ Good Luck !!

16. 利用 pyFOAM 残差的输出

(2009-12-01 16:47:54)转载

标签： openfoam 研究 教育 分类： 其他

蓝色流体社区有兄弟发表关于如何使用 gnuplot 来输出残差图，今天来探讨一下用 pyFoam 来完成残差图的。

pyFoam 是控制 OF 运行的一个 python 程序包，也就是将 openfoam 的运行部分包装了一下。你可以通过 svn 下载最新版本的 pyfoam，可以通过在控制台上输入下面命令。

svn co https://openfoam-extend.svn.sourceforge.net/svnroot/openfoam-extend/trunk/Breeder/other/scripting/PyFoam/

下载了 pyFoam 以后，进入 pyfoam 文件夹，利用下面的命令安装

sudo python setup.py install

能够安装成功的前提是你的系统需要装有 python.同时需要有 gnuplot 的支持。 如果你用的是 ubuntu。可以直接通过下面命令下来 gnuplot 并安装
sudo apt-get install gnuplot。
安装过程比较简单，下面看看如果输出残差图
与 openfoam 残差有关的命令有
pyFoamRunner.py 他是对 of 运行控制命令，如果用这个命令运行 of 的程序，输出结果中就会有残差，但是并不能像 fluent 一下边运行边显示残差图。 但是可以通过 pyFoamPlotWatcher 来查看 of 的残差 log 文件，并显示残差图形。
1）这就产生了第一种方法
pyFoamRunner.py --clear icoFoam -case cavity
运行结束后，用下面命令 plot 残差
pyFoamPlotWatcher.py *.log（*为 case 文件夹中，pyFoamRuner 输出的残差图）
2）如果想实时显示残差，也可以直接通过 pyFoamPlotRunner.py 来完成。
pyFoamPlotRunner.py --clear icoFoam -case cavity
3）当然你也可以通过下面方法显示残差图
到你的 case 文件夹中输入求解器名字，并将残差输出到一个文件中。如到 cavity 中输入下面命令
icoFoam > cavity.log
然后通过 pyFoamPlotWatch.py cavity.log 显示残差。
4）你不想用 gnuplot 画残差图，而是想用 origin 残差图。
可以通过 PyFoamRunner.py 或者 PyFoamPlotRunner.py 来运行程序，运行结果在你的 case 文件夹中就会有所有的变量的残差，时间点和值一一对应，用 origin 画图很方便
也可以直接用普通方式运行求解器，将残差输出到一个文件中。并通过下面的命令对残差进行分析，得到残差值和时间点的对应文件。
pyFoamStandarLogAnalyzer.py + 残差文件名字。

pyFoam 功能很强大，可以试一试。可惜他没有帮助文件，可以通过 命令 --help 来查看简要帮助。
如先看看 pyFoamRunner.py 怎么用，直接
pyFoamRunner.py --help
即可。
试一试？祝好

17. 也来谈谈传值和传址

(2009-11-10 16:31:48)转载
标签： openfoam 研究 校园 分类： 其他
C++中传值和传址是语言的经常遇到的问题。传值和传址是对函数参数调用而言的，先看个例子.我想将编写一个函数，将 a＋b 的和放在 sum 中，差放在 sub 中,这点想必大家都能够实现，如采用如下形式

```
//采用 c＋＋引用传址格式实现
void add(int a, int b, int & sum, int &sub)
 {
     sum=a+b;
     sub=a-b;
}
//采用 c 语言传值格式
void add(int a, int b, int *sum, int *sub);
{
        *sum=a+b;
        *sub=a-b;
}
```

前面两种都是采用传址格式，可以轻松的将里面的数据传输出来。调用方式为
add(a,b,sum,sub);
add(a,b,&sum,&sub); //这里的＆是取地址符号，与前面 c＋＋函数定义前面的＆不一样，那个是引用符。
即，得到他们的和差。
下面的程序能够实现 a,b 的 copy 吗？

```
void copy( int    *a, int *b, int *copya, int *copyb);
{
    copya=a;
    copyb=b;
}
void main ()
{
int *a; int *b;
int c=10,d=20;
a=&c;
b=&d;
int * copya(NULL);
int *copyb(NULL);
copy(a,b,copya,copyb);
cout<<*copya<<"\t"<<*copy<<endl;
}
```

上面的程序能够顺利实现 a，b 的 copy 吗？ 也许你会发现，并不能实现，因为 copya, copyb 是进行的传值而不是传址，为什么呢？
如果按照下面定义呢？（调用格式不变）

```
void copy( int    *a, int *b, int * & copya, int * &copyb);
{
    copya=a;
    copyb=b;
}
```

为什么？

18. 从 pisoFoam 谈谈 OpenFOAM-1.6 湍流模型的结构变化

(2009-11-06 16:29:26)转载
标签： openfoam 教育 分类： OpenFOAM 使用
OpenFOAM-1.6 的一个很重要的改变是将所有的湍流模型结构发生了变化。看 solver 下的不可压缩求解起你会发现，1.5 以前的 oodles,turbFoam 在新版本中已经没有了，其实这两个求解器的功能完全有新版本中的 pisoFoam 包括了。
　　先看一下 pisoFoam 的使用吧。看看 OpenFOAM/OpenFOAM-1.6/tutorials/incompressible/pisoFoam/ras 下的 cavity 算例，不难发现，该算例中 constant 文件夹中比以前 turbFoam 求解器 case 文件多了一个新的文件 TurbulenceProerties，该字典用来指定你选定的湍流模型类型，RANS 还是 les 还是层流。其他完全一样。再看看 OpenFOAM/OpenFOAM-1.6/tutorials/incompressible/pisoFoam/les 下的 pitzDaily，你会发现，和原来的 les 的算例同样增加了这个文件。 其他使用和原来的 turbFoam 和 lesFoam 完全一样。
　再看看这个 pisoFoam 这个 solver，你会发现和以前的 turbFoam 和 oodles 主要在湍流模型的创建上。 TurbFoam 采用了如下形式

```
    autoPtr<incompressible::RASModel> turbulence
     (
          incompressible::RASModel::New(U, phi, laminarTransport)
     );
```

oodles 采用了

```
autoPtr<incompressible::RASModel> turbulence
```

```
 (
 incompressible::RASModel::New(U, phi, laminarTransport)
);
```

而 pisoFoam 采用了

```
 autoPtr<incompressible::turbulenceModel> turbulence
  (
          incompressible::turbulenceModel::New(U, phi, laminarTransport)
 );
```

很显然 pisoFoam 一个声明，将湍流模型 les 和 rans 都包括了，这主要得益于其湍流结构的变化，新版本中将湍流模型增加了一个父类，将 les 和 rans 的通用特性抽象出来了。 湍流模型的提供的接口有

```
//          Access function to velocity field.
const volVectorField &    U () const
//          Access function to flux field.
const surfaceScalarField &          phi () const
//          Access function to incompressible transport model.
transportModel &            transport () const
//          Return the laminar viscosity.
const volScalarField &    nu () const
//          Return the turbulence viscosity.   如果选择的 les 返回亚格子黏性，rans 的话，返回普通雷诺时均黏性
virtual tmp< volScalarField >      nut () const =0
//  有效黏性    nu+nut
virtual tmp< volScalarField >      nuEff () const =0
//  团动能 Return the turbulence kinetic energy.
virtual tmp< volScalarField >      k () const =0
 // Return the turbulence kinetic energy dissipation rate.
virtual tmp< volScalarField >      epsilon () const =0
//文档中说这个是返回的雷诺应力，其实不是 rans 模型返回雷诺应力，les 返回亚格子应力
virtual tmp< volSymmTensorField >          R () const =0
//包括层流应力和湍流应力（亚格子应力或者雷诺应力）
virtual tmp< volSymmTensorField >          devReff () const =0
//有效应力偏分量的散度。也就是那个拉普拉斯项（扩散项）
virtual tmp< fvVectorMatrix >      divDevReff (volVectorField &U) const =0
//重新求解湍流方程，更新湍流黏性
virtual void       correct ()=0
//读取湍流字典，les 和 rans 字典不同
virtual bool       read ()=0
```

关于传输模型和湍流模型的关系，可以参看本站博文 ：OpenFOAM 中 transportModel 与 viscosityModels 关系。OpenFOAM-1.6 没有发生变化。

19. 非惯性旋转系统稳态求解器 simpleSRFFoam 的使用

(2009-11-05 18:28:55)转载

标签： openfoam 研究分类： OpenFOAM 使用

今天探讨一下非惯性单旋转系统求解器 simpleSRFFoam 的使用。该求解器可以用在旋转坐标系统(如泵)流体流动的稳态计算，可采用 rans 湍流模型或者层流模型。 本次说明，以 OpenFOAM-1.6 中的 mixer 算例为例讲解。 该算例是个 3D 算例，一个 1/4 圆柱，流体从上方流入，从下方流出，内墙随着里面的桨旋转，外墙固定，1/4 界面采用周期边界。

求解器位置：

OpenFOAM/OpenFOAM-1.6/tutorials/incompressible/simpleSRFFoam/simpleSRFFoam

算例位置：OpenFOAM/OpenFOAM-1.6/tutorials/incompressible/simpleSRFFoam/mixer
可以看到，本求解器并没有在标准的 application 里面，而是 tutorials 里面的。这点和其他求解器位置不同。 首先看看 mixer 下的 case 结构

```
|--0
|    | —epsilon
|    |-k
|    |-nut
|    |-omega
|    |-p
|    |-Urel
|--constant
|    |-polyMesh
|                          |-blockMeshDict
|                          | -bounary
|    |-RASProperties
|    |-SRFProperties
|    |-transportProperties
|---System
                    |-controlDict
                    |-fvSchemes
                    |-fvSolution
```

从上面文件夹结构来看，该算例和其他算例子无异。各文件的内部的具体内容不再多说，可以参考本站其他关于求解器算例说明。 看看该算例和其他的差异。

差异一：速度边界条件的设定
由于采用了相对坐标系统，在边界条件设定的时候有相对和绝对之分。该求解器求的是相对速度，也就是 case 初始化文件夹 0 里面的 Urel，如果你指定的速度为相对于旋转参考系的速度，则和惯性系里面的指定没有区别，直接给定速度即可。 比如
Urel 文件中内部墙

```
    innerWall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
```

这样指定说明，该速度是相对于参考坐标系的速度，其值为 0 说明，相对参考坐标系为 0，也就是说，该内部墙随坐标系一起旋转。
如果您指定的速度为绝对速度，则采用 SRFVelocity 边界条件。如，入口速度设定为

```
 inlet
    {
        type            SRFVelocity;
        inletValue      uniform (0 0 -10);
        relative        yes;
        value           uniform (0 0 0);
    }
```

relative 务必要 yes，如果 no 的话，就变成了相对速度了。这点似乎和我们想法有点出入，这个 relative 不是说里面的 inletVaue 是相对值，而是，通过绝对速度 inletValue 得到相对值。老外的思维就是和我们有点差异。

差异二：旋转的设定
和其他 case 不一样的地方，是该求解器的算例增加了参考坐标系参数设置字典 SRFProperties，下面是里面的内容

```
//选择旋转类型，OpenFOAM 中就这一种，rpm 就行了。
SRFModel        rpm;
```

```
//-用来指定旋转方向，沿着 z 轴看顺时针，想反过来转的话 就将 1 变为－1，也就是 axis
( 0 0 -1);
axis                ( 0 0 1 );
//rpm 模型系数
rpmCoeffs
{
    rpm                 5000;    //旋转速度，每分钟 5000 转
}
```

差异三：后处理

由于该求解器求解的是速度的相对值，因此，里面有 Uabs 输出，这才是真正的速度。 然而这个真正速度是相对于非惯性坐标系的绝对速度，要将该绝对速度转化到绝对坐标系，也就是需要作一次坐标系旋转得到惯性系下的速度，然而该求解器是稳态求解器，旋转坐标系统没有意义。该求解器比较适合那种转速特别高的情形，这种情形下，流场最终稳定，可以采用稳态来计算。 如果转速比较低，流场会受到旋转几何的影响，采用非稳态似乎更加合适。

20. linux 常用命令集

(2009-08-21 01:23:56)转载

标签： openfoam 研究 教育 分类： linux 基础

学习 OpenFOAM linux 命令是必须的。下面是常用的命令集合

（1）文件命令

ls – directory listing
ls -al – formatted listing with hidden files
cd dir - change directory to dir
cd – change to home
pwd – show current directory
mkdir dir – create a directory dir
rm file – delete file
rm -r dir – delete directory dir
rm -f file – force remove file
rm -rf dir – force remove directory dir *
cp file1 file2 – copy file1 to file2
cp -r dir1 dir2 – copy dir1 to dir2; create dir2 if it doesn't exist
mv file1 file2 – rename or move file1 to file2 if file2 is an existing directory, moves file1 into directory file2
ln -s file link – create symbolic link link to file
touch file – create or update file
cat > file – places standard input into file
more file – output the contents of file
head file – output the first 10 lines of file
tail file – output the last 10 lines of file
tail -f file – output the contents of fle as it grows, starting with the last 10 lines

（2）文件权限控制

chmod octal file – change the permissions of file to octal, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see man chmod
（3）搜索
grep pattern files – search for pattern in files
grep -r pattern dir – search recursively for
pattern in dir
command | grep pattern – search for pattern in
the output of command
locate file – fnd all instances of file
（4）系统信息
date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger user – display information about user
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man command – show the manual for command
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis app – show possible locations of app
which app – show which app will be run by
default
（5）文件压缩与解压缩
tar cf file.tar files – create a tar named
fle.tar containing files
tar xf file.tar – extract the files from file.tar
tar czf file.tar.gz files – create a tar with
Gzip compression
tar xzf file.tar.gz – extract a tar using Gzip
tar cjf file.tar.bz2 – create a tar with Bzip2
compression
tar xjf file.tar.bz2 – extract a tar using Bzip2
gzip file – compresses fle and renames it to
file.gz
gzip -d file.gz – decompresses file.gz back to
fle
(6)快捷键
Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with
fg in the foreground or bg in the background
Ctrl+D – log out of current session, similar to
exit
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

21. 一起看看 OpenFOAM－1.6 中的 pisoFoam

(2009-08-18 18:14:30)转载

标签： openfoam 研究 教育 分类： OpenFOAM 求解器说明

OpenFOAM-1.6 在求解器方面做了较大的调整，很多以前的求解器在新版本都合到了一起，比如湍流求解器 turbFoam 和 oodles 就合成了新的 pisoFoam，并新出了一个 pimpleFoam(该求解器同样能够对层流、雷诺时均流、大涡适应，只是压力速度耦合的算法采用了 simple＋piso 的混合算法，该算法以后在说明)。现在我们一起看看这个 pisoFoam

pisoFoam 的程序架构和以前求解器没有什么区别，这里不再累述。下面通过比较 OpenFOAM－1.5 里面的 turbFoam 和本求解器来说明他们的差别。如果对 turbFoam 不明白，清参看本站博文：“OpenFOAM 中的不可压缩湍流流动求解器 turbFoam 的说明”和“OpenFOAM 中雷诺时均湍流求解器 turbFoam 使用”。 闲言少叙。

差别一

在 createFields.H 中主要差别在湍流模型的创建上，在 turbFoam 中，采用

```
autoPtr<incompressible::RASModel> turbulence
(
        incompressible::RASModel::New(U, phi, laminarTransport)
);
```

在 pisoFoam 中采用

```
autoPtr<incompressible::turbulenceModel> turbulence
(
        incompressible::turbulenceModel::New(U, phi, laminarTransport)
);
```

比较两个可以发现，前面创建的是 RASModel 动态对象指针，而后面创建的是 turbulenceModel 动态对象指针。前面只适合于雷诺时均模型求解流动，而后者适合于雷诺时均，层流，大涡。求解器的功能在 pisoFoam 中扩充了。这主要得益于 OpenFOAM－1.6 对湍流的架构进行了调整。以前版本雷诺时均和大涡他们是两个独立的库，而新版本将两个库合到了一起，采用一共同的父类，也就是你看到的 turbulenceModel 类。这样就可以用统一的父类来在程序书写程序，而通过 New 来动态创建 autoPtr 指向的动态对象。 autoPtr 的使用，请参看本站博文“OpenFOAM 中的智能指针 autoPtr”。

差别二：

求解循环

在 turbFoam 中

```
    for (runTime++; !runTime.end(); runTime++)
```

在 pisoFoam 中

```
while (runTime.loop())
```

前面那个 for 循环，后面采用的是 while 循环。该 while 循环和以前的不太一样，loop 是 openfoam-1.6 的一个新特性。在 loop 中其实做了两件事，一是判定是否在 run，如果 run 的话，将当前的 runTime 推移。 也就是下面两句的合集

```
while(runTime.run())
runTime++;
```

因此在使用 loop 的时候，下面没有 runTime＋＋的。这样可以避免 runTime＋＋而造成的死循环，但是用了 loop 就不要加了。

前面 for 循环不必说了，以前的求解器都说过了。

其他都和 turbFoam 的一样，请参看前面提到的 turbFoam 的相关说明。

22. 一起看看 OpenFOAM－1.6 中的 pisoFoam

(2009-08-18 18:14:30)转载

标签： openfoam 研究 教育 分类： OpenFOAM 求解器说明

OpenFOAM-1.6 在求解器方面做了较大的调整，很多以前的求解器在新版本都合到了一起，比如湍流求解器 turbFoam 和 oodles 就合成了新的 pisoFoam，并新出了一个 pimpleFoam(该求解器同样能够对层流、雷诺时均流、大涡适应，只是压力速度耦合的算法采用了 simple＋piso 的混合算法，该算法以后在说明)。现在我们一起看看这个 pisoFoam
pisoFoam 的程序架构和以前求解器没有什么区别，这里不再累述。下面通过比较 OpenFOAM－1.5 里面的 turbFoam 和本求解器来说明他们的差别。如果对 turbFoam 不明白，清参看本站博文：“OpenFOAM 中的不可压缩湍流流动求解器 turbFoam 的说明”和“OpenFOAM 中雷诺时均湍流求解器 turbFoam 使用”。 闲言少叙。

差别一
在 createFields.H 中主要差别在湍流模型的创建上，在 turbFoam 中，采用

```
autoPtr<incompressible::RASModel> turbulence
(
        incompressible::RASModel::New(U, phi, laminarTransport)
);
```

在 pisoFoam 中采用

```
autoPtr<incompressible::turbulenceModel> turbulence
(
        incompressible::turbulenceModel::New(U, phi, laminarTransport)
);
```

比较两个可以发现，前面创建的是 RASModel 动态对象指针，而后面创建的是 turbulenceModel 动态对象指针。前面只适合于雷诺时均模型求解流动，而后者适合于雷诺时均，层流，大涡。求解器的功能在 pisoFoam 中扩充了。这主要得益于 OpenFOAM－1.6 对湍流的架构进行了调整。以前版本雷诺时均和大涡他们是两个独立的库，而新版本将两个库合到了一起，采用一共同的父类，也就是你看到的 turbulenceModel 类。这样就可以用统一的父类来在程序书写程序，而通过 New 来动态创建 autoPtr 指向的动态对象。 autoPtr 的使用，请参看本站博文“OpenFOAM 中的智能指针 autoPtr”。

差别二：
求解循环
在 turbFoam 中

```
for (runTime++; !runTime.end(); runTime++)
```

在 pisoFoam 中

```
while (runTime.loop())
```

前面那个 for 循环，后面采用的是 while 循环。该 while 循环和以前的不太一样，loop 是 openfoam-1.6 的一个新特性。在 loop 中其实做了两件事，一是判定是否在 run，如果 run 的话，将当前的 runTime 推移。 也就是下面两句的合集

```
while(runTime.run())
runTime++;
```

因此在使用 loop 的时候，下面没有 runTime＋＋的。这样可以避免 runTime＋＋而造成的死循环，但是用了 loop 就不要加了。
前面 for 循环不必说了，以前的求解器都说过了。

其他都和 turbFoam 的一样，请参看前面提到的 turbFoam 的相关说明。

23. 深入解析 OpenFOAM 时间控制参数字典文件 controlDict

(2009-05-08 03:39:04)转载
标签： openfoam 研究 教育 分类： OpenFOAM 使用
在本站博文“使用 OpenFOAM 的基本流程”已经对 controlDict 中的一些基本参数字典关键字进行了简单的讨论。鉴于 controlDict 的在 OpenFOAM 计算的重要性，本文对参数字典文件进行详细探讨。 现在以动态步长的 icoFoam 中算例为例进行说明。至于如何让程序能够自动调节步长,参看本站博文“如何使得 OpenFOAM 的 solver 自动调节时间步长”。

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//计算该算例的应用程序名字。
application     icoFoam

//控制当前计算的开始时间。可为以下三个值
（1）firstTime:时间文件夹中时间最早的时间开始
（2）startTime:从 startTime 指定的时间开始计算
（3）latestTime:从时间文件夹中，时间最晚的时间开始计算。
startFrom       startTime;

//当 startFrom 后指定的为 startTime 的时候，startTime 为指定的计算时间。
startTime       14;

//控制程序什么时候停止计算。可以为以下 4 个值
（1）endTime：当时间到达 endTime 指定的时间停止计算。
（2）writeNow：计算一步，并输出结果，停止计算。
（3）noWriteNow：计算一步，不输出结果，停止计算。
（4）nextWrite：当到达指定的输出时间（由 writeControl 控制），输出结果并停止计算。
stopAt          endTime;

//当指定停止时间为 endTime 时，指定的结束时间。
endTime         30;

//计算的时间步长
deltaT          0.001;

//输出控制，可以 5 个值
（1）timeStep：每 writeInterval 个时间步长写一次。
（2）runTime：每 writeInterval 秒物理时间写一次
（3）adjustableRunTime：每 writeInterval 秒物理时间写一次，但是对于可调节步长的话，会自动调节最后一次的时间步长，以便准确时间输出。
（4）cpuTime：每 writeInteral 秒 cpu 时间写一次。
（5）clockTime：每 writeInterval 秒实际时间写一次。
writeControl    runTime;

//和上面的 writeControl 的值联合使用。意义随该值变化。
writeInterval   0.2;

//写过程是否覆盖，如果 0 则不覆盖，大于 0 为覆盖，比如：2，case 文件家中只有输出文件
 比如 6 和 7，//当算到 8 时候，会覆盖 6，9 会覆盖 8，以此类推
purgeWrite      0;

//写文件的格式，值可谓 2 个
（1）ascii：按照 ascii 格式输出，以便我们可以直接看结果
```

（2）binary：按照二进制格式输出，以便节省空间
writeFormat ascii;

//输出数据精度
writePrecision 6;

//是否对输出文件进行压缩，可谓 2 个值
（1）uncompressed：不压缩
（2）compressed：运用 gzip 压缩
writeCompression uncompressed;

//时间文件夹格式，可为 3 个值
（1）fixed：m.dddddd,其中“d”的个数由 timePrecision 控制
（2）scientific：m.ddddde±xx，“d”的个数有 timePrecision 控制
（3）general：如果指数小于－4 或者大于等于 timePrecision 指定的值，则采用 scientific 方法，其他采用普通小数形式
timeFormat general;

//和 timeFormat 联合使用，具体意义参看 timeFormat 的说明。
timePrecision 6;

//在运行的时候是否允许改变参数。可为 2 值
（1）yes：允许，参数值改变立即反映的当前运行程序
（2）no：即使改变了参数文件的值，也不会反映到当前运行的程序
runTimeModifiable yes;

//是否允许自动调节步长（只适应于那种可自动调节步长的 solver），不能自动调节步长的程序，yes 也没有用，如何使程序可以自动调节步长，参看本站博文“如何使得 OpenFOAM 的 solver 自动调节时间步长”
（1）yes：允许，程序会根据 maxCo 自动调节步长。
（2）no：不允许，程序采用由 deltaT 指定的定步长
adjustTimeStep no;

//最大允许的 Courant 数，如果程序的 Courant 大于指定值，则自动缩小时间不成，调节因子 1.2
maxCo 0.5;

//最大允许的时间步长。
maxDeltaT 1;

24. OpenFOAM 中的智能指针 autoPtr

(2009-07-21 02:35:56)转载
标签： openfoam 研究 教育 分类： OpenFOAM 类解析
OpenFOAM 中有两个常用但很难理解的模板类，智能化指针模板类 autoPtr 和瞬态对象操作模板类 tmp。这两个类几乎无处不在，使用极易出错。OpenFOAM 通过该自动指针来实现对象的多态创建。这使得你可以在不改变程序的情况下，完成多种不同实现。本文意在探讨，这 autoPtr 的使用过程中应当注意的问题及其他们与标准 c++指针的差别。瞬态对象操作模板类 tmp 的使用，后续文章探讨。

C++语言中数据的传递方式有两种：（1）数值传递（2）地址传递。前者是在函数调用中将实参的值复制一份传给形参，形参的变化不会影响到实参。后者在函数调用中传递的实际参数的地址，在函数中，对形参的修改会影响到实参。这点可以参考谭老的 c 语言经典之作“c

语言程序设计”，不再累叙。 就地址传递而言，C 语言采用指针，而 C++中可以采用引用。两者在参数传递的时候效果一样，C++引用相对来说比较简单，但是灵活性较指针差一些。比如：如果想实现，父类声明对象，子类构造该对象，只能通过指针来完成，因为指针可以悬空，而引用则不行。 两者的差别，可以参看相应 C++书籍。 闲言少序。

1.autoPtr
使用：autoPtr<className> objectName;

特性 1：指针自动释放
autoPtr 可实现类 className 的指针操作，但此指针会自动释放，因此无需手动释放。 这是和 C++标准指针最大的差别。C＋＋中指针使用两个函数应该成对出现那就是 new 和 delete。但是 autoPtr 通过某种方式构造后，无需 delete，程序会自动 delete。

特性 2：指向对象的指针唯一（autoPtr 指针所指的对象，只能由唯一指针所指，这点有点难）C＋＋中可以多个指针同时指向一个对象，但是 autoPtr 指向对象不能为多个指针同时指。

autoPtr 使用误区：

（1）将 autoPtr 作为参数
如
//声明一个函数，autoPtr 做参数
function(autoPtr<class1> obj);
//声明一个 autoPtr 对象
autoPtr<class1> obj1;
//函数调用
function(obj1);
上面调用完后，obj1 将不再拥有所指的对象，该对象已经赋给了形参 obj，但 obj 是一个瞬态变量，函数调用完后会自动释放。这是 obj1 指针变为 NULL，无法对采用 obj1 进行任何操作。否则会出现 segment fault。
（2） autoPtr 对象赋值
autoPtr<class1> A=class1:New(); //A 有指对象
autoPtr<class1> B;//B 没有对象可指
B=A;//将 A 指对象给 B，注意是给不是复制。
这时候，B 将指向 A 原来指的对象，A 指针变为空。
（3）将 autoPtr 指针付给其他标准指针
autoPtr<class1> A=class1:New();
class1 *B;
B=A.ptr();
上面操作结束后 A 指针变空，B 标准指针指向 A 原来对象，这时候 B 使用结束后应该释放 B 所指对象，否则会出现内存漏洞。 释放 B 就直接 delete B； 就行了。

并不是说，autoPtr 对象就不能传递数据了，传递数据时候请采用引用。
如： autoPtr<class1> A;
const class1 & B=A(); //注意 A 后面的()为取引用的意思。
这样操作，不会将 A 所指的对象释放掉，对 A 所指对象的操作，就可以通过 B 来完成了。 这点很重要，特别是 openfoam 中很多动态创建对象都是通过 autoPtr 来完成的。 如果在类中声明了 autoPtr，可以通过上述方式将类里面的对象传出来。

25. 如何实现同一用户下的 OpenFOAM 多版本编译

(2009-06-30 19:20:19)转载
标签： openfoam 研究 教育 分类： OpenFOAM 入门

OpenFOAM 目前有多个公司在维护：以 Henry weller 为首的 OpenCFD 的 openfoam 官方公司，还有以 hrvoje Jasak 为首的 Wikki 公司（-dev 版本）。也许你会很苦脑，到底是该用那个版本呢？还有如果我当前工作是基于 1.5 的，如何更新了 1.6，那我以前做得工作能不能在 1.6 下面有条不紊的工作呢？要不要换成 1.6 体验一下新功能呢？ 其实有一种方法可以在同一用户下进行多版本 OpenFOAM 安装，根据你的爱好，可以在不同版本进行随时切换，而不相互影响。

我的笔记本上装了 3 个版本
1.5.x(官方 1.5 的开发版本)，可以用“ git clone http://repo.or.cz/r/OpenFOAM-1.5.x.git”下载
1.4.1（上一个版本）
1.5-dev（hrvoje jsask 为首的 openfoam-extend 工程，说来惭愧，他们也将我列为其中的一个 developer ， 我 并 没 为 此 版 本 做 过 多 少 贡 献 ， 该 版 本 网 址 ： http://sourceforge.net/projects/openfoam-extend/）。

闲言少叙，首先分析一下 openfoam 的运行环境。
（1）openfoam 运行环境配置分析
OpenFOAM 是靠配置当前运行环境来运行的，而起环境配置文件是在 openfoam 根目录下的 etc/bashrc 文件（对于 1.4.1 是在根目录下.OpenFOAM-1.4.1/bashrc，进入 1.4.1 你看不到的，是个隐藏文件夹，注意前面的"."。linux 下以“.”开头的是隐藏文件）。在 openfoam 官方安装指南通常将 bashrc 文件的执行加入到自己根目录下的配置文件".bashrc"中，每次启动控制台的时候，就会自动执行该文件，进而配置 openfoam 环境。 如果我们手动配置 openfoam 环境，这样不就可以实现 openfoam 环境切换了吗？每次切换的时候，只需要更新一下系统中 openfoam 当前版本运行环境即可。
（2）环境配置更新
如果你想对环境进行更新，只需要执行要使用版本下面 etc/bashrc 文件即可。可以采用如下命令
1.4.1 版本
source /home/sujunwei/OpenFOAM/OpenFOAM-1.4.1/.OpenFOAM-1.4.1/bashrc
1.5
source $HOME/OpenFOAM/OpenFOAM-1.5/etc/bashrc
1.5.x
source $HOME/OpenFOAM/OpenFOAM-1.5.x/etc/bashrc
1.5-dev
source $HOME/OpenFOAM/OpenFOAM-1.5-dev/etc/bashrc
但是如果这样每次切换需要执行这么长的命令，很烦，可以将其写成系统可以找到的文件，然后 source 一下那个文件就行了。

（3）环境配置更新文件
只需要将上面的一句话写到一个文件中,附件中有我的系统配置文件（加不上附件，可以到 http://www.openfluid.cn/attachment.php?aid=262&k=9ddb736a1b938b3e4a8f819c72dce8f0&t=1246360743&fid=62&sid=XCNXuN%2B3%2FWvswrbexlGqEgQY1fPq7Q896ae9I6ivUTdr9jW），将起解压缩到$HOME/bin 下就行了（bin 系统可以找到,在系统的$PATH 中）. 你可以仿照着写一下。其实不用改的， 如果你和我安装相同版本的话。文件名字要记住，以便用于版本切换。比如，我只需要在控制台上输入 source OpenFOAM141 就可以使用 1.4.1 了。输入 source openfoam-extend 就可以使用 dev 版本了。是不是很简单？

（4）不同版本的安装
不同版本的安装和单一版本安装没有什么区别，比如你要安装 1.5.x 按照我附件相应配置文件名字为 OpenFOAM15x，直接输入 source OpenFOAM15x 就可以执行相应的安装了。对于不同的版本，需要在新环境下，重新安装即可，所有的都要重新安装，包括第三方包。放心，新的配置环境不会影响你以前安装的版本。 应当注意不要将那个 etc/bashrc 的执行命令，

写在你的.bashrc 下面了，直接写到你的配置文件中就行了，每次使用前 source 一下。

不妨试一试，祝你好运。

如果附件下载有问题，到蓝色流体论坛上 openfoam 社区（www.openfluid.cn）或者流体中文网 OpenFOAM 社区去下 http://www.cfluid.com/bbs/viewthread.php?tid=57155&extra=page%3D1

26. 商业软件划分的网格向 OpenFOAM 转换应注意的问题

(2009-06-17 23:38:43)转载
标签： openfoam 研究 教育 分类： OpenFOAM 使用
openfoam 自己带有网格划分功能 blockMesh 和 snapHexMesh.前者可以生成块结构化网格，后者则基于表面网格文件（stl 格式）自动生成复杂的网格。她还带有商业软件网格转换功能，如 fluentMeshToFoam 等，所有的网格转换器源文件在文件夹 applications/utilities/mesh/conversion 中。
总体而言运用商业软件网格的转换 openfoam 可识别的网格时候，应当注意以下几个问题

（1）几何尺寸
几乎所有的商业软件划分的网格是没有单位的，而 OpenFOAM 的网格是具有单位（其单位 m），因此在划分网格的时候应当完全按照国际单位制划分网格。如果在事先没有注意到这一点，不用着急，后面还有补救方法。

（2）边界条件
几乎所有的商业软件都支持边界条件的设定，然而在这些软件定义的边界条件 openfoam 未必认识，这时候转换器会将其定义为 wall 类型。不要随意定义边界类型，因为一个不同的边界类型可能对网格进行的限制，如你在 gridgen 中定义了周期边界，导出的网格是无法为 openfoam 认识的，一种比较好的就是将起定义为 wall 类型。
如果商业软件允许你对边界条件进行命名的话，一个比较好的名字，可以解决转换后网格边界条件无法识别的问题。但是有的商业软件（gridgen）边界名字是根据边界类型生成的。要是定义多个边界都为 wall 的话并导出 fluent 格式，并转换为 openfoam 时候，相同类型的边界会命名为 wall－1，wall－2，不醒目，你设置类型的时候都不知道那个是那个，我通常采用自定义边界类型，然后输出 fluent 格式，并转换为 openfoam 格式。
当成功的划分网格并转换成 openfoam 可支持的转换格式后，就可以对网格进行转换了。

（3）网格转换
将划分后的 mesh 拷贝到你的 case 文件夹中，然后运用如下命令进行网格转换
转换器名字 网格文件 [-scale scale factor]
比如：fluentMeshToFoam 001.cas -scale 0.01
前面 fluentMeshToFoam 为转换器，001.cas 为网格文件，-scale 用来指定你要划分的网格放大多少倍，用来处理你划分网格的时候没有注意到的单位问题，上面的 0.01 是将网格缩小 100 倍。

（4）修改物理边界名字和边界类型。
所有不识别的边界类型，他们都会认为是 wall 类型。修改边界名字，应当到 case 文件夹中 constant/polyMesh 中的 boundary 文件修改。里面有边界的名字，和边界 type，指定为你要的类型，比如 patch，wall 等，这里指定的是物理边界。

（5）设定数值边界类型
进入初始化文件夹 0 文件中各个场文件进行 boundaryField 设定相应的数值边界条件。 数值边界的名字要和物理边界的名字对应，对顺序没有要求。
如果对 openfoam 中的数值和物理边界不明白，请参看本站博文“OpenFOAM 不可压缩流边

界条件的设定之我见”

现在 check 一下你的 mesh 了。 利用 checkMesh 对网格进行一下 check 或者利用 paraFoam 看看你导入的网格。

27. OpenFOAM 如何定义与时间有关的边界条件

(2009-06-07 23:45:33)转载
标签： openfoam 研究 教育 分类： OpenFOAM 入门
组里一个师兄问我如何定义与时间有关的边界条件，我给他写了一下，顺便贴到 blog 上了。那个师兄是个日本人，只认英文。
In OpenFOAM, there are three basic patch boundary conditions
Dirichlet (first type) boundary condition
Neumann (second type) boundary condition
Mixed Dirichlet and Neumann

For time-dependent boundary condition, if the value on the boundary is time-dependent, this is Dirichlet boundary condition. If the gradient of the value on the boundary is time-dependent, it is a Neumann boundary condition.

In OpenFOAM, there is a boundary condition named oscillatingFixedvalue boundary condition (located in src\finiteVolume\fields\fvPatchFields\derived\oscillatingFixedValue), the boundary values change with time according to
$S=S0(1+A0\sin(2\pi\omega t))$ (1)
S is the variable you apply the boundary to. If you want to implement another boundary similar but with different expression, just doing the following steps
(1) Give your new boundary condition a nice name, tdbc(time-dependent boundary condition) for instance. Do the following operations

```
//make a copy for oscillatingFixedvalue
cp   -r   oscillatingFixedValue   tdbcFixedValue

//go into your new boundary condition dir
cd   tdbcFixedValue

//rename the original files
mv   oscillatingFixedValueFvPatchField.C     tdbcFixedValueFvPatchField.C
mv   oscillatingFixedValueFvPatchField.H     tdbcFixedValueFvPatchField.H
mv   oscillatingFixedValueFvPatchFields.C   tdbcFixedValueFvPatchFields.C
mv   oscillatingFixedValueFvPatchFields.H   tdbcFixedValueFvPatchFields.H
mv   oscillatingFixedValueFvPatchFieldsFwd.H   tdbcFixedValueFvPatchFieldsFwd.H
```

//replace all the “oscillating” with “tdbc” in all the files in this dir. I usually use “kate” editpad in OpenSUSE to replace all the words

(2) Add the new boundary condition to the compile environment
Go to the dir OpenFOAM-1.5\src\finiteVolume\Make using cd
Add the following code to the “files”: line 108 for instance
$(derivedFvPatchFields)/tdbc FixedValue/tdbcFixedValueFvPatchFields.C

(3) Alter the functions
All the function definition is located in the file tdbcFixedValueFvPatchField.C:179 (in function updateCoeffs)
patchField =refValue_*currentScale()

refValue_ is S0 in equation (1), its type (scalar or vector) is determined by the field type this boundary condition applied to. For instance, this boundary is used for pressure, refValue_ should be scalar. If it is used for velocity, refValue_ should be vector. Its definition is defined through class template parameter “type”.
currentScale() is just a function of this class, it is defined at line 38 in this file.
Just write a function similar to currentScalar(), and give its return value to patchField, all the things are OK.
Note:that patchField is a field. If your expression is also dependent on the location, you have to do a loop like this

```
const fvPatch & fvp=this->patch(); //get the geometry of the boundary
const vectorField &fc=fvp.Cf();      //get face center on the boundary.
scalar t=this->db().time().value();    //get the current simulation time.
forAll(patchField,i)
{
    point xyz=fc[i];       //get the face center of face i; x=xyz[0], y=xyz[1], z=xyz[2].
patchField[i]=f(xyz,t); //f is the function of your equation
}
```

The parameter can be added in a similar way to refValue_ or frequence_

(4) Recompile the code
Go to \OpenFOAM-1.5\src\finiteVolume\ using the following line to compile
wmake libso

(5) use the new boundary
If all the things were done correctly, the new boundary condition can be used like the boundary condition in OpenFOAM. Your new boundary condition is “tdbcFixedValue” . you can use like this

```
inlet     //boundary name
      {
            type                tdbcFixedValue;
            para1               value;
            para2               value;
}
```

Para1 and para2 are like parameters in your new boundary condition.

Good Luck.

If there are something not clearly written or encounter some problems in the process of implementation, please don’t hesitate to contact me.

28. OpenFOAM 中 transportModel 与 viscosityModels 关系

(2009-05-22 05:13:00)转载
标签： openfoam 研究 教育 分类： OpenFOAM 类解析
应 openfoamlb 的要求，本文简单的介绍一下 transportModel 和 viscosityModels. transportModel 是个父类，他本身无法构造函数，因为里面有 3 个重要的纯虚函数
virtual tmp<volScalarField> nu() const=0; //返回层流粘性
virtual void correct()=0; //根据不同的粘性模型对层流粘性进行更新
virual bool read()=0; //读如 transportation 属性
该类本身可以说是粘性模型选择器。此类有两个子类 singlePhaseTransportModel 和 twoPhaseMixture 分别用于单相流动粘性和两相混合流动粘性的选择器。其实很简单，单相传输里面有个粘性模型指针（私有变量），该指针会根据你选择的粘性模型对粘性进行更新。更新过程在 correct 里面，可以通过 nu()返回你更新后的粘性。在湍流模型中，不管是 RAS 还是 LES 模型都需要根据 transportModel 的 correct 进行层流粘性的更新。其基本调用过程为

turbulentModel.correct()-->singlePhaseTransportModel.correct()-->viscosityModels.correct().
turbulenceModel.nu()-->singlePhaseTransportmodel.nu()-->viscosityModels.nu()

viscosityModels 是所有的粘性模型父类，该父类和 transportModel 模型一样，有三个纯虚函数，该纯虚函数和 transportModel 中的纯虚函数完全相同
virtual tmp<volScalarField> nu() const=0;　　//返回层流粘性
virtual void correct()=0;　//不同的粘性模型需要重载该函数，该函数就是根据当前的流动情形重新计算粘性的。
virual bool read()=0; //读如 transportation 属性

应当注意上述所有的粘性模型通常是一个场，而非一个变量。对于牛顿流体粘性模型全场恒定，他是根据你在 transportPropertiesDict 里面的那个 nu 对粘性场进行构建的。也就是对牛顿流体全场流体粘性所有值都相同。

通常，大家关心的并不是如何定义一个 transportModel，而是如何定义一个粘性模型 viscosityModels。很简单，子类化 viscosityModels，并重载 nu(), correct(), read()三个纯虚函数即可。 这样你的粘性模型就可以被 transportModels 所调用。 为什么？ 接口相同啰!!

29. OpenFOAM 不可压缩流边界条件的设定之我见

(2009-05-17 01:36:10)转载
标签： openfoam 研究 教育　　分类： OpenFOAM 使用
边界条件是设定正确与否是决定计算成功与否的关键因素。边界条件有物理边界条件和数值边界条件之分。

物理边界条件是根据某种物理现象而对速度和压力必须满足某种数值行为而定义的边界条件，比如 fluent 中的 pressureOutlet or velocityinlet 等。此类边界条件是软件公司为了减少用户的逐个对每个变量进行设定而引起麻烦而定义的一种数值边界条件集合。比如 fluent 中 pressureOutlet 边界条件为该边界压力指定，其他变量梯度为 0，这样设定，给用户带来了很多方便，但同时限制了用户的选择性。比如我想在压力边界上设定某个标量为指定值，这就很难在 fluent 中办到。openfoam 中的 foamX 也为大家定义了一些物理边界条件，比如 wall，atmosphere 等。我很少用 foamX，因为使用起来比较麻烦，不如直接对文件进行编辑。这也是在本站博文中没有对 foamX 进行介绍的一个原因。

数值边界条件：就是我们经常所说的第一类边界条件（定值边界）和第二类边界条件（梯度边界）。在 openfoam 的初始化文件中就需要指定边界条件就是这种数值边界条件。不像 fluent 那样，在用 gambit 画几何时候指定的边界条件可以是 fluent 中的物理边界条件。而 openfoam 中画网格的时候，需要指定相应的几何边界，也就是基本边界类型：patch：基本边界，symmetryPlane 对称板，empty：空边界，用于二维流动，wedge：用于轴对称模拟，cyclic：循环边界，wall：墙边界（主要运用湍流流动的壁面函数），inter-processor:处理器边界。 除了几何上有特殊性的边界，如墙，对称轴，循环边界，空边界，周期边界，wedge 性边界，其他所有的边界都指定为 patch。

边界条件指定：
在 openfoam 的初始化文件中（0 文件夹中的文件），boundaryField 下面的东西用于指定边界条件。
《1》压力－速度
对于不可压缩流动边界而言，知道了压力就可以求出速度，知道了速度就可以求解出压力，在指定的边界条件的时候压力速度必须耦合。也就是说，压力和速度不能同时为第一类或者同时为第二类边界。换句话说是压力指定，速度必须是由程序算出来的，速度指定，压力必须是算出来的。这是针对不可压缩流动而言，速度和压力相互制约，如果由其他变量影响速度和压力，比如浮力驱动流动，压力速度受温度的影响，这种除外。

《2》其他标量
对速度没有影响的其他标量的边界条件，所有的边界不能同时为第二类边界，这样会造成全场的均匀，但压力可以所有都为第二类边界，因为他受速度的影响。

以上只是我个人浅见，不一定正确，仅供参考。　欢迎大家批评指正。

30. OpenFOAM 中气液双欧拉求解器 bubbleColumn 的使用

(2009-05-15 00:21:19)转载
标签：　openfoam 研究　教育　　分类：　OpenFOAM 使用
做气液两相流动模拟的同志们最双欧拉方法可谓再熟悉不过了，今天给大家介绍 OpenFOAM 中专门针对气液两相流动的双欧拉求解器 bubbleColumn 的使用。从求解器的名字上看，他似乎是针对鼓泡床（bubble column）的，不仅仅是这些，他可以适应于任何气液系统的模拟。闲言少叙。
（1）位置
求解器的位置：applications\solvers\multiphase\bubbleFoam
算例的位置：tutorials\bubbleFoam\bubbleColumn
（2）文件夹结构

```
|-0
|   |-alpha     //气相的体积分数，液相的体积分数为 1-alpha
|   |-epsilon //液相的湍流耗散率
|   |-k       //液相的湍动能
|   |-p     //压力，双欧拉里面规定，两相的压力是一样的。
|   |-Ua //气相的速度
|   |-Ub //液相的速度
|
|-constant
|                   |-environmentalProperties //环境属性，比如重力场什么的
|                   |-RASProperties         //雷诺时均湍流模型属性
|                   |-transportProperties    //传输参数相关属性，粘性，密度等
|
|-system
                |-controlDict
                |-fvSchemes
                |-fvSolution
```

其实上面的文件夹结构中还有一个 0.org.这只是另外一个初始化文件，和 0 里的文件除了那个初始场不一样，其他相同的。
（3）文件说明
0 文件夹的内容和 system 文件夹的东西和其他求解器类似，这里不再说明。本文只针对 RASProperties 和 transportProperties 进行说明
1.RASProperties

```
//文件头
FoamFile
{
      version         2.0;
      format          ascii;
      class           dictionary;
      object          RASProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
RASModel                    laminar;        //其实这个关键字一点用途也没有。
```

```
turbulence              off;            //如果用层流就 off，如果用湍流就 on，目前 bubbleColumn 只支持 k－e 湍流
printCoeffs             off;            //是否打印系数
laminarCoeffs
{
}
kEpsilonCoeffs          //k－e 系数
{
    Cmu                 0.09;
    C1                  1.44;
    C2                  1.92;
    alphak              1;
    alphaEps            0.76923;
}
wallFunctionCoeffs      //壁面函数系数
{
    kappa               0.4187;
    E                   9;
}
// ************************************************************************* //
```

2.transportProperties

```
//文件头
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

rhoa                rhoa [1 -3 0 0 0 0 0] 1;        //气相速度

rhob                rhob [1 -3 0 0 0 0 0] 1000;     //液相速度

nua                 nua [0 2 -1 0 0 0 0] 1.6e-05;     //气相的黏性

nub                 nub [0 2 -1 0 0 0 0] 1e-06;       //液相的黏性

da                  da [0 1 0 0 0 0 0] 0.003;         //气相的粒径

db                  db [0 1 0 0 0 0 0] 0.0001;        //液体相的粒径

Cvm                 Cvm [0 0 0 0 0 0 0] 0.5;      //虚拟体积力系数

Cl                  Cl [0 0 0 0 0 0 0] 0;     //升力系数

Ct                  Ct [0 0 0 0 0 0 0] 1;     //湍流响应系数（也就是离散相团动能和连续相湍动能之比值）
```

也许你会感觉到奇怪，为什么液相还有粒径。OpenFoam 中的双偶拉模型和其他软件不太一

样，他是基于 Henrik Rusche 博士论文 中的一种能够计算气液反演过程的一种模型。具体理论，请参看相关章节。

31. OpenFOAM 不可压缩非牛顿流体层流求解器使用说明

(2009-05-13 22:21:58)转载

标签： openfoam 研究 教育 分类： OpenFOAM 使用

上次我在流体中文网 OpenFOAM 社区做了个 OpenFOAM 国内使用情况的调查，本来是将国内 OpenFOAM 的使用现状给 hrv 介绍一下，发现有用 OpenFOAM 搞聚合物或者塑料模拟的，因此，本文将 OpenFOAM 中非牛顿流体求解器 nonNewtonianIcoFoam 的使用进行一下说明。应当注意，包含湍流模型（RAS 或者 LES）的求解器都可直接适应于非牛顿流体，但是层流求解器 icoFoam 不行，因为在该求解器中没有对流体的粘性进行求解，粘性在该求解器中只是一个标量，而非非牛顿流体所需要的场。然而该求解器仍然可以运用牛顿流体的求解，尽管名字说是针对非牛顿流体的。

（1）位置

求解器位置：applications\solvers\incompressible\nonNewtonianIcoFoam

算例位置：tutorials\nonNewtonianIcoFoam\offsetCylinder

（2）文件夹结构

```
|-0
|   |-p   //压力文件
|   |-U   //速度文件
|-constant
|   |-transportProperties   //传输属性文件
|   |-PolyMesh
|             |-blockMeshDict   //生成网格参数文件
|             |-boundary //物理边界定义，生成网格文件时，该文件会被覆盖，可有可无
|-system
        |-fvSchemes   //离散格式选择文件
          |-fvSolution   //代数方程组求解文件
          |-controlDict //流程控制文件
```

（3）文件说明

上面所有的文件和 icoFoam 下的 cavity 近似，请参看本站博文“深入解析 icoFoam 下的顶盖驱动流(cavity)”。本文只对 constant 文件夹下的 transportProperties 进行说明。

“transportProperties”:

```
//文件头
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

/传输模型，也就是粘性模型，如果是非牛顿流体请选择非牛顿流体模型，如果是牛顿流体直接写成： transportModel Newtonian;即可

```
transportModel    CrossPowerLaw;
//如果运用是牛顿流体的话，采用如下的粘性系数
nu                  nu [0 2 -1 0 0 0 0]       1;

//下面是支持的粘性模型。去掉后面的 Coeffs 之后加到 transportModel 后面。

CrossPowerLawCoeffs
{
      nu0               nu0 [0 2 -1 0 0 0 0] 0.01;
      nuInf             nuInf [0 2 -1 0 0 0 0] 10;
      m                 m [0 0 1 0 0 0 0] 0.4;
      n                 n [0 0 0 0 0 0 0] 3;
}

BirdCarreauCoeffs
{
      nu0               nu0 [0 2 -1 0 0 0 0] 1e-06;
      nuInf             nuInf [0 2 -1 0 0 0 0] 1e-06;
      k                 k [0 0 1 0 0 0 0] 0;
      n                 n [0 0 0 0 0 0 0] 1;
}
```

事实上 OpenFOAM 中还有两个粘性模型：HerschelBulkley 粘性模型和 PowerLaw 粘性模型可用，可惜的是，在本算例中并没有将这两种模型给出。

```
HerschelBulkleyCoeffs
{
      tao0              tao0 [0 2 -1 0 0 0 0]       1e-06;
      nu0               nu0 [0 2 -1 0 0 0 0]        1e-06;
      k                 k [0 0 1 0 0 0 0]           0;
      n                 n [0 0 0 0 0 0 0]           1;
}

powerLawCoeffs
{
      nuMax             nuMax [0 2 -1 0 0 0 0]   <参考值>
      nuMin             nu0 [0 2 -1 0 0 0 0]     <参考值>
      k                 k [0 0 1 0 0 0 0]          0;
      n                 n [0 0 0 0 0 0 0]          1;
}
```

上面的模型的参考值请参阅相关文件，由于我不做非牛顿流体，所以对这些模型的参数不是很明白。

关于如何自己定义一个粘性模型，以后在相关博文中介绍。

32. OpenFOAM 中不可压缩稳态求解器 simpleFoam 的使用

(2009-05-10 21:29:47)转载

标签： openfoam 研究 教育 分类： OpenFOAM 使用

simpleFoam 为 OpenFOAM 中稳态不可压缩流动（层流或者 RAS 湍流）求解器，压力速度耦合采用的 simple 算法。从以往博文阅读量来看，读者对求解器的说明的阅读远远小于求解器使用的阅读量。因此以后的本站帖子尽量介绍与 OpenFOAM 应用相关的东西。

simpleFoam 和湍流求解器 turbFoam 有以下两点区别

（1）simpleFoam 为稳态的，turbFoam 为非稳态的

（2）simpleFoam 压力速度耦合用的 simple 算法，trubFoam 则采用 PISO 算法

本文详细介绍一下，simpleFoam 求解器下的算例 pitzDaily。本文的介绍主要和 turbFoam 进行对比，以说明稳态求解器和非稳态求解器在使用上的差异。关于如何使用 turbFoam，请参看本站博文“OpenFOAM 中雷诺时均湍流求解器 turbFoam 使用”

（1)位置
求解器位置：applications\solvers\incompressible\simpleFoam
算例位置：tutorials\simpleFoam\pitzDaily
（2）求解器文件夹结构
2）算例文件夹结构
|-system
|　　|-fvSolution　//代数方程求解器选择文件
|　　|-fvSchemes　　//离散格式选择文件
|　　|-controlDict //计算流程控制文件
|-constant
|　　|-transportProperties　//传输参数控制文件，黏性等
|　　|-RASProperties　　//湍流模型选择文件
|　　|-polyMesh　　//网格文件夹
|　　|-blockMeshDict //blockMesh 网格设定文件
|　　|-boundary　　//边界文件，可有可无，blockMeshDict 会将其覆盖
|-0
　|-U　//速度边界条件，初始条件设定文件
　|-R　//雷诺应力边界条件，初始条件，仅仅当选择雷诺应力模型时候需要
　|-p　//压力边界条件，初始条件设定文件
　|-nuTilda //一方程模型中求解的那个变量，仅仅选择 SpalartAllmaras 湍流模型时候有用
　|-k　　//湍动能设定文件，　k？ ε 中的 k
　|-epsilon //湍流耗散律设定文件，　k？ ε 中的 ε

从上面的文件夹结构来看，该算例和 turbFoam 下的顶盖驱动流动算例完全相同。当采用该求解器与 turbFoam 使用在有 3 个文件有差别，他们为 system 下的三个文件 fvSolution，fvScheme，controlDict
（1）fvSolutions
　　差别一：SIMPLE 子字典的收敛判据
　SIMPLE
　{
　　nNonOrthogonalCorrectors 0;
　}
　其实该字典中还有一个有效关键字 convergence，遗憾的是在算例并没有给出，该参数的默认值为 0，也就是上一步迭代结果带入方程后得到残差小于 convergence（=0 默认）时候，方程认为收敛。你可以自己设定该参数。如
 SIMPLE
　{
　　nNonOrthogonalCorrectors 0;
　　convergence　　1e-6;
　}
 当残差小于 1e-6 时，认为方程收敛。

　　差别二：亚松弛因子
relaxationFactors
{
　　p　　0.3;
　　U　　0.7;

```
    k               0.7;
    epsilon         0.7;
    R               0.7;
    nuTilda         0.7;
}
```

由于方程的初始场通常和收敛场差距很大，一步迭代的话极其容易产生求解的代数方程发散，因此通常采用亚松驰以改善求解的稳定性。而在非稳态求解器中通常采用较小的时间步长来改善两次迭代值之间的差异，以防止方程发散。

（2）fvSchemes

对于稳态求解器中 ddt 都会写成稳态形式 steadyState。应当注意，稳态还是非稳态，并不是有下面的设定来控制，他是求解器本身的特性。

```
ddtSchemes
{
    default steadyState;
}
```

（3）controlDict

```
application simpleFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
//这里的 endTime 不再是求解结束时间，而是最大迭代次数有关的量（迭代次数=（endTime-startTime）/deltaT)。如果在设定的最大迭代次数内方程收敛，程序也会自动退出计算。
endTime         1000;
//这里不再是时间步长，而是一个月迭代次数有关的量，该量与最大迭代次数的关系，上面已经说明。
//其他量的说明，参看本站博文“深入解析 OpenFOAM 时间控制参数字典文件 controlDict”
deltaT          1;
writeControl    timeStep;
writeInterval   50;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression uncompressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
```

除了上述三个文件与 turbFoam 下的 cavity 有差异，其他近似，请参看本站博文“OpenFOAM 中雷诺时均湍流求解器 turbFoam 使用”

33. 深入解析 OpenFOAM 离散格式参数字典文件 fvSchemes

(2009-05-09 00:50:02)转载

标签： openfoam 研究 教育 分类： OpenFOAM 使用

在本站博文“使用 OpenFOAM 的基本流程”已经对 fvSchemde 中的一些基本参数字典关键字进行了简单的谈论，本文对该参数字典进行详细探讨。

在该字典文件中可能出现的关键字有

interpolationSchemes 点对点插值格式

snGradSchemes 面梯度发方向分量

gradSchemes 梯度格式 ?
divSchemes 散度格式 ? ?
laplacianSchemes 拉普拉斯项格式 ?2
timeScheme 时间的一阶二阶微分格式 ?/?t, ? 2 /? 2 t
fluxRequired 需要计算流率的场
由于该部分内容较多，本文只对前 3 中格式进行探讨，后面几种格式在后续文章中说明。
（1）interpolationSchemes 插值格式。
OpenFoam 中所有的插值格式有

1.中心格式
linear 线性插值（中心差分）
cubicCorrection 三次格式
midPoint 线性插值，带有对称加权

2.迎风差分
upwind 迎风差分
linearUpwind 线性迎风差分
skewLinear 带有偏度修正的线性格式
QUICK Quick 格式

3.TVD 格式
limitedLinear 限制型线性差分
vanLeer van Leer 格式
MUSCL MUSCL 格式
limitedCubic 三次限制性格式

4.NVD 格式
SFCD Self-?ltered 中心差分
Gamma ψ Gamma 差分（Jasak 提出的一种格式）
上面的插值格式可分为两类：（1）普通差分格式（中心格式），（2）带有对流项的差分格式（后三种）。这两种插值格式用法也不一样。

普通差分格式： 关键字 ＋ 差分格式
如： default linear; //默认插值格式为中心差分

带有对流项的差分格式：关键字＋差分格式＋表面流率场（速度场的表面插值场）。
如：default QUICK phi; //默认格式为基于表面流率场 phi 的 QUICK 格式

有一些 TVD/NVD 格式需要一个系数 ψ， 0 ≤ ψ ≤ 1， ψ＝0 对应于该格式的最好精度，ψ＝1 对应于该格式的最好稳定性，这种插值格式采用如下方式指定：关键字＋差分格式＋ψ＋phi
如：default limitedLinear 1.0 phi;

对于一些标量场的插值，有时会需要对该标量场进行限制（比如，插值结果需要在［－2，3］之间），这时候在指定插值格式的时候，需要在格式关键字前面加上 limited 如
default limitedVanLeer -2.0 3.0 ;
当限制值在［0，1］内的时候，可采用他的一个特殊版本
default vanLeer01;
适合这种情况的有 limitedLinear 格式， vanLeer, Gamma, LimitedCubic, MUSCL 和 SuperBee
对于向量场的插值在 limited 时候，采用一般名字加上 V，加 V 之后的版本为气修正版本。如 limitedLinearV，vanLeerV, GammaV, limitedCubicV,SFCDV;

（2）snGradSchemes 面梯度发方向分量
支持的表面梯度向量面发方向分量的格式有
corrected　　显式的带有非正交修正
uncorrected　不带有非正交修正
limited　ψ　　限制性非正交修正
bounded　　对正标量有阶修正
fourth　　四阶格式
一般的用法为 关键字＋上面的值，
如:snGrad(p)　　corrected ;
当用 limited 这种格式时候需要加上关键字 ψ，即
snGrad（p） limited　ψ;
其中 ψ 的取值为
?
?0　　对应于上面的 uncorrected
?
?
?0.333　非正交修正 ≤ 0.5 × 正交的部分,
ψ=?
?0.5　　非正交修正≤ 正交修正部分
?
?
?1　　对应于上面的 corrected.
关于非正交修正的相关理论，请参看 jasak 的博士论文 Error analysis and estimation for the finite volume method with applications to fluid flow;

(3) 梯度格式 gradSchemes
OpenFOAM 中的所有梯度格式有
Gauss <interpolationScheme> 二阶，高斯积分
leastSquares　　二阶，最小二乘
fourth　　四届，最小二乘
limited <gradScheme>　　以上种格式的 limited 版本
对于采用第 2，3 种的梯度格式. 直接使用即可。
grad(p)　　fourth;
对于第 1 种梯度格式,需要加上 gauss 和插值格式（因为运用高斯理论，需要利用中心点到面的插值）
grad(p) Gauss linear ;
对于第四种梯度格式，
grad(p) limited ＋前三种任意格式。
如： grad(p) limited Gauss linear ;

（4）拉普拉斯(laplacianSchemes)?? (∨ ?U)
通常采用的形式为 Gauss <interpolationScheme> <snGradScheme>
如：laplacian(nu,U) Guss linear corrected ;
通常，<interpolationScheme> 为 linear.
不同的<snGradScheme>,laplacianSchemes 的行为不同，如下表所示
corrected　　无界，二阶，守恒格式
uncorrected　　有界，一阶，不守恒格式
limited　ψ　　一种 correct 和 uncorrected 混合格式
bounded　　一阶有阶格式
fourth　　无界，四阶，守恒格式

（5）散度格式 divSchemes
通常采用的形式为 Gauss <interpolationScheme>, 应当注意，对于对流相关的插值格式在 divSchemes 后面的插值格式没有必要指定表面流率，因为表面流率在 divSchemes 格式内部已经指定了。
如 div(phi,U) Gauss upwind;
下面为不同的插值格式，divSchemes 表现出来的行为。

linear	二阶，无界
skewLinear	二阶，更加无界，带有偏斜度修正
cubicCorrected	四阶，无界
upwind	一阶，有界
linearUpwind	一阶或二阶 有界
QUICK	一阶/二阶，有界
TVD schemes	一阶/二阶，有界
SFCD	二阶格式，有界
NVD schemes	一阶/二阶，有界

（6）时间格式 ddtSchemes 和 d2dt2Shemes
有效的时间离散格式为

Euler	一阶隐式，有界格式
CrankNicholson ψ	二阶有界隐式格式
backward	二阶隐式
steadyState	不求解时间离散项

通常采用格式 非稳态项 + 离散格式
如 ddt(U) Euler.
对于 CrankNicholson 中 ψ=0 时，为 Euler 格式， ψ=1 为 CrankNicholson 格式。
有效的 d2dt2Shemes 格式只有 Euler 格式。

（7）fluxRequired 那些场需要计算流率。
比如

```
fluxRequired
{
p;
}
```

34. 如何使得 OpenFOAM 的 solver 自动调节时间步长

(2009-05-07 18:34:45)转载
标签： openfoam 研究 教育 分类： OpenFOAM 使用
OpenFOAM 很多求解器时间步长都是恒定的，这使得我们在 CFD 计算的时候不得不一个一个的试。求解器时间步长的自动调整会节省我们很多时间。下面以 icoFoam 为例谈谈如何动态设定时间步长。其实 OpenFOAM 已经为我们定制了自动调节时间步长的功能，如果要使用这个功能，需要以下几个头文件。

```
//读入动态设定步长相关参数
#    include "readTimeControls.H"

//计算 CourantNo
#    include "CourantNo.H"

//初始化设定时间步长
#    include "setInitialDeltaT.H"
```

//根据 CourantNo 重新设定时间步长
#　　include "setDeltaT.H"

要让程序自动调节时间步长，主要分 4 个步骤，下面在 icoFoam 中实例

```
#include "fvCFD.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{

#    include "setRootCase.H"

#    include "createTime.H"
#    include "createMesh.H"
#    include "createFields.H"
#    include "initContinuityErrs.H"

//步骤 1：将前 3 个头文件加到这里。

//读入动态设定步长相关参数
#    include "readTimeControls.H"

//计算 CourantNo
#    include "CourantNo.H"

//初始化设定时间步长
#    include "setInitialDeltaT.H"


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;
//步骤 2 将 for 循环改成 while 循环

//      for (runTime++; !runTime.end(); runTime++)

    while(runTime.run())
    {

//          Info<< "Time = " << runTime.timeName() << nl << endl;

#        include "readPISOControls.H"
#        include "CourantNo.H"

//步骤 3：将重新设定步长的头文件写到这，及其推移时间，将上面的 Info 挪下来。
#        include "setDeltaT.H"
         runTime++;
         Info<< "Time = " << runTime.timeName() << nl << endl;
```

```
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
        );

        solve(UEqn == -fvc::grad(p));

        // --- PISO loop

        for (int corr=0; corr<nCorr; corr++)
        {
            volScalarField rUA = 1.0/UEqn.A();

            U = rUA*UEqn.H();
            phi = (fvc::interpolate(U) & mesh.Sf())
                + fvc::ddtPhiCorr(rUA, U, phi);

            adjustPhi(phi, U, p);

            for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
            {
                fvScalarMatrix pEqn
                (
                    fvm::laplacian(rUA, p) == fvc::div(phi)
                );

                pEqn.setReference(pRefCell, pRefValue);
                pEqn.solve();

                if (nonOrth == nNonOrthCorr)
                {
                    phi -= pEqn.flux();
                }
            }

#           include "continuityErrs.H"

            U -= rUA*fvc::grad(p);
            U.correctBoundaryConditions();
        }

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "    ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return(0);
}
```

//步骤 4： 进入控制台重新编译程序
打开控制台，进入/applications/solvers/incompressible/icoFoam/
wmake

如何使用自动调节步长功能呢？

打开 case 文件夹下 system/controDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         2;

deltaT          0.1;

writeControl    runTime;

writeInterval   0.1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

//加入下面的就行了

adjustTimeStep  yes;    //想变成静态步长，no 即可

maxCo           0.5;    //如果 Co 数大于 0.5 则减小步长

maxDeltaT       1;      //最大时间步长
```

运行程序测试一下。

如何让程序自动调节步长的介绍已经完毕，是不是很简单？

35. OpenFOAM 中不可压缩流大涡求解器 oodles 的使用

(2009-05-06 20:50:41)转载
标签： openfoam 研究 教育 分类： OpenFOAM 使用

本站博文“OpenFOAM 中不可压缩湍流大涡求解器 oodles 说明”介绍了 OpenFOAM 中大涡求解器 oodles 的实现细节，以该求解器下的算例 pitzDaily 为例。

在介绍本算例之前，现将 OpenFOAM 中支持的大涡模型进行简单的介绍。OpenFOAM 对不可压缩流动支持的大涡模型有

Smagorinsky	Smagorinsky 模型
Smagorinsky2	3 维滤波的 Smagorinsky 模型
dynSmagorinsky	动态 Smagorinsky 模型
scaleSimilarity	尺度相似模型
mixedSmagorinsky	Smagorinsky 和尺度相似混合模型
dynMixedSmagorinsky	动态 Smagorinsky 和尺度相似混合模型
oneEqEddy	k 方程涡粘性模型
dynOneEqEddy	动态 k 方程涡粘性模型
locDynOneEqEddy	局部动态 k 方程涡粘性模型
spectEddyVisc	波谱涡粘性模型
LRDDiffStress	LRR 差分应力模型
DeardorffDiffStress	Deardorff 差分应力模型
SpalartAllmaras	Spalart-Allmaras 模型

大涡滤波函数

laplaceFilter	Laplace 滤波器
simpleFilter	Simple 滤波器
anisotropicFilter	Anisotropic 滤波器

大涡 deltas 函数 LESdeltas

PrandtlDelta	Prandtl delta
cubeRootVolDelta	单元体积的三次根号 delta
smoothDelta	Smoothing of delta

以上这些模型能够在 OpenFOAM 的 oodles 中任意使用。下面对如何使用这些模型进行说明。

（1）位置

求解器位置：applications\solvers\incompressible\oodles

算例位置：tutorials\oodles\pitzDaily

（2）pitzDaily 文件结构

```
|-system
|       |-fvSolution    //代数方程求解器选择文件
|       |-fvSchemes     //离散格式选择文件
|       |-controlDict //计算流程控制文件
|-constant
|       |-transportProperties   //传输参数控制文件，黏性等
|       |-LESProperties         //湍流模型选择文件
|       |-polyMesh              //网格文件夹
|               |-blockMeshDict //blockMesh 网格设定文件
```

```
|                    |-boundary        //边界文件，可有可无，blockMeshDict 会将其覆盖
|-0
    |-U   //速度边界条件，初始条件设定文件
    |-B   //亚格子应力边界条件，初始条件，仅仅当选择亚直接求解亚格子应力模型时候需
要
    |-p   //压力边界条件，初始条件设定文件
    |-nuTilda //一方程模型中求解的那个变量，仅仅选择 SpalartAllmaras 湍流模型时候有用
    |-k        //湍动能设定文件，上面的 k 方程模型
    |-nuSgs //亚格子粘性，仅对亚格子粘性项进行求解方程的模型
```

(3)文件说明

system 文件夹下的三个文件以及压力文件 p 的说明参看本站博文“使用 OpenFOAM 的基本流程”;polyMesh 中的文件及其速度文件 U 的说明，请参看本站博文“深入解析 icoFoam 下的顶盖驱动流(cavity)”；k,nuSgs 和 nuTilda 文件内容和压力文件 p 相似不再累述，亚格子应力 B 和 turbFoam 中的雷诺应力 R 相似，transportProperties 传输模型参考文件，请参看本站博文“OpenFOAM 中雷诺时均湍流求解器 turbFoam 使用”；下面主要针对剩下的文件。下面主要针对 LESProperties 进行说明。

文件 LESProperties

```
//文件头
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      LESProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//当前湍流计算采用哪种大涡模型，大涡模型的具体写法为下面模型系数字典中去掉 Coeffs，
比如 oneEqEddy 为 oneEqEddyCoeffs 去掉 Coeffs 的字符串。

LESModel            oneEqEddy;
//delta 函数采用哪种形式：
delta               cubeRootVol;
//是否在建立模型的时候在屏幕上打印下面的系数。
printCoeffs         on;
//层流系数
laminarCoeffs
{
}
oneEqEddyCoeffs     //k 方程涡粘性模型系数
{
    ck              0.07;
    ce              1.05;
}
dynOneEqEddyCoeffs //动态 k 方程涡粘性模型系数
{
    ce              1.05;
    filter          simple; //滤波函数
}
locDynOneEqEddyCoeffs    //局部动态 k 方程涡粘性模型系数
{
    ce              1.05;
```

```
    filter          simple;
}
SmagorinskyCoeffs   //Smagorinsky 模型系数
{
    ce              1.05;
    ck              0.07;
}
Smagorinsky2Coeffs //3 维滤波的 Smagorinsky 模型系数
{
    ce              1.05;
    ck              0.07;
    cD2             0.02;
}
spectEddyViscCoeffs   // 波谱涡粘性模型系数
{
    ce              1.05;
    cB              8.22;
    cK1             0.83;
    cK2             1.03;
    cK3             4.75;
    cK4             2.55;
}
dynSmagorinskyCoeffs   //动态 Smagorinsky 系数
{
    ce              1.05;
    filter          simple;
}
mixedSmagorinskyCoeffs   //Smagorinsky 和尺度相似混合模型系数
{
    ce              1.05;
    ck              0.07;
    filter          simple;
}
dynMixedSmagorinskyCoeffs //动态 Smagorinsky 和尺度相似混合模型系数
{
    ce              1.05;
    filter          simple;
}
LRRDiffStressCoeffs   //LRR 差分应力模型系数
{
    ce              1.05;
    ck              0.09;
    c1              1.8;
    c2              0.6;
}
DeardorffDiffStressCoeffs //Deardorff 差分应力模型系数
{
    ce              1.05;
    ck              0.09;
    cm              4.13;
}
SpalartAllmarasCoeffs   //Spalart-Allmaras 模型系数
{
```

```
    alphaNut        1.5;
    Cb1             0.1355;
    Cb2             0.622;
    Cw2             0.3;
    Cw3             2;
    Cv1             7.1;
    Cv2             5.0;
    CDES            0.65;
    ck              0.07;
}
```

以上为 LES 模型参数，下面为 delta 函数系数。

```
cubeRootVolCoeffs
{
    deltaCoeff      1;
}
PrandtlCoeffs
{
    delta           cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }
    smoothCoeffs
    {
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }
        maxDeltaRatio   1.1;
    }
    Cdelta          0.158;
}
vanDriestCoeffs
{
    delta           cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }
    smoothCoeffs
    {
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }
        maxDeltaRatio   1.1;
    }
    Aplus           26;
    Cdelta          0.158;
}
smoothCoeffs
{
```

```
    delta               cubeRootVol;
    cubeRootVolCoeffs
    {
        deltaCoeff          1;
    }
    maxDeltaRatio     1.1;
}
kappa               0.4187;
//壁面函数系数
wallFunctionCoeffs
{
    E                   9;
}
```

（4）程序运行

打开控制台，进入 tutorials\oodles\pitzDaily
输入：blockMesh 生成网格
输入：oodles 运行程序
程序运行结束后，输入 paraFoam 做后处理。

36. OpenFOAM 中的不可压缩湍流流动求解器 turbFoam 的说明

(2009-05-04 21:28:43)转载
标签： openfoam solver 教育 分类： OpenFOAM 求解器说明

本文谈谈 OpenFOAM 不可压缩湍流流动求解器 turbFoam 的实现细节。

(1) 求解器位置：applications/solvers/incompressible/turbFoam

(2) 求解器文件夹结构

```
|－Make
|       |-options
|       |-files
|-createFields.H
|-turbFoam.C
```

(3) 求解器功能

任意不可压缩湍流流动，湍流模拟采用雷诺时均方法

(4) 文件说明

1.options //编译选项，用于指定编译用到的头文件位置及其动态库

```
//文件内容
#用到的头文件文件夹
EXE_INC = \
#雷诺是均湍流模型头文件
    -I$(LIB_SRC)/turbulenceModels/RAS \
#传输模型头文件，牛顿流体或者非牛顿流体选择。
    -I$(LIB_SRC)/transportModels \
#有限容积方法头文件
    -I$(LIB_SRC)/finiteVolume/lnInclude

#用到的动态连接库
EXE_LIBS = \
#不可压缩雷诺时均模型库
    -lincompressibleRASModels \
#不可压缩传输模型库（牛顿流体传输模型和非牛顿流体传输模型）
    -lincompressibleTransportModels \
```

```
#有限容积库
    -lfiniteVolume \
#网格相关工具库
    -lmeshTools

    2.files   //用于指定当前要编译的文件，这里不包含头文件，都是*.C 文件。
//文件内容
turbFoam.C    //主文件
//编译后求解器的名字和存放位置
EXE = $(FOAM_APPBIN)/turbFoam

    3.createFields.H //创建场
    // 提 示 ， 并 创 建 压 力 场 ， 各 项 意 义 ， 参 看 本 站 博 文 “OpenFOAM>>solver>>basic>>potentialFoam 的说明”
    Info<< "Reading field p\n" << endl;
    volScalarField p
    (
        IOobject
        (
            "p",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
//提示，并创建速度场，各项意义，参看本站博文“OpenFOAM>>solver>>basic>>potentialFoam 的说明”
    Info<< "Reading field U\n" << endl;
    volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
//创建表面流律场，面心存储，用于将体积分转化面积分，文件位于
//src ? finiteVolume ? cfdTools ? incompressible
#   include "createPhi.H"

//设定压力参考值和参考 cell
    label pRefCell = 0;
    scalar pRefValue = 0.0;
    setRefCell(p, mesh.solutionDict().subDict("PISO"), pRefCell, pRefValue);

//建立传输模型，这里 laminarTransport 并不是说本 solver 为层流模型。
    singlePhaseTransportModel laminarTransport(U, phi);
```

```
//创建湍流模型，返回指向 incompressible::RASModel 的 autoPtr 指针。
//autoPtr 指针有点复杂，后面将会有专门博文探讨该指针，目前你就将其当作普通指针就行了。
    autoPtr<incompressible::RASModel> turbulence
    (
        incompressible::RASModel::New(U, phi, laminarTransport)
    );
```

4.turbFoam.C

```
//头文件
//有限容积库相关的所有头文件
#include "fvCFD.H"
//单相传输模型
#include "incompressible/singlePhaseTransportModel/singlePhaseTransportModel.H"
//雷诺时均模型
#include "incompressible/RASModel/RASModel.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//主程序入口
int main(int argc, char *argv[])
{
//设置 case 文件夹结构，程序运行时候输入的参数
#   include "setRootCase.H"
//创建时间对象 runTime 用于控制程序运行
#   include "createTime.H"
//创建网格对象 mesh
#   include "createMesh.H"
//创建场，参看上面的说明
#   include "createFields.H"
//初始化连续误差。
#   include "initContinuityErrs.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;
//程序主循环
    for (runTime++; !runTime.end(); runTime++)
    {
//显示物理时间
        Info<< "Time = " << runTime.timeName() << nl << endl;
//读入 piso 控制参数
#       include "readPISOControls.H"
//计算 courant 参数
#       include "CourantNo.H"

        // Pressure-velocity PISO corrector
        {
            // Momentum predictor
//创建动量方程。关于下面书写紧凑性的背后问题，清参看本站博文“OpenFOAM 中的神奇方程定义方式的背后”
            fvVectorMatrix UEqn
            (
                fvm::ddt(U)
```

```
            + fvm::div(phi, U)
        //该项包含两部分：1.分子扩散项 2.雷诺应力的偏分量的散度。
        //雷诺应力主分量的散度归结到了压力项中，这是大多数雷诺时均和大涡模型实现的一贯做法。
        //因此压力比层流模型中多了一项,那就是雷诺应力的主分量，通常被称为湍动压力。
            + turbulence->divDevReff(U)
        );
//如果需要动量预测，则求解速度方程。
        if (momentumPredictor)
        {
            solve(UEqn == -fvc::grad(p));
        }

        // --- PISO loop
//PISO 流程和 icoFoam 完全一样，请参看本站博文“OpenFOAM>>solver>>incompressible>>icoFoam 的说明”
        for (int corr=0; corr<nCorr; corr++)
        {

            volScalarField rUA = 1.0/UEqn.A();

            U = rUA*UEqn.H();
            phi = (fvc::interpolate(U) & mesh.Sf())
                + fvc::ddtPhiCorr(rUA, U, phi);

            adjustPhi(phi, U, p);

            // Non-orthogonal pressure corrector loop
            for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
            {
                // Pressure corrector

                fvScalarMatrix pEqn
                (
                    fvm::laplacian(rUA, p) == fvc::div(phi)
                );

                pEqn.setReference(pRefCell, pRefValue);
                pEqn.solve();

                if (nonOrth == nNonOrthCorr)
                {
                    phi -= pEqn.flux();
                }
            }

#           include "continuityErrs.H"

            U -= rUA*fvc::grad(p);
            U.correctBoundaryConditions();
        }
    }
```

```
        //重新计算湍动黏性，也就是对 turbulence->divDevReff(U)需要的量进行更新。
        //比如 k－e 模型中下面函数包括求解 k－e 方程和计算有效黏性系数。
          turbulence->correct();
        //输出计算结果到文件中
          runTime.write();
        //显示当前执行时间和 cpu 时间。
          Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
              << "    ClockTime = " << runTime.elapsedClockTime() << " s"
              << nl << endl;
    }
    //显示程序结束
    Info<< "End\n" << endl;
    //返回 0
    return(0);
}
```

（5）编译求解器

打开控制台，进入求解器文件夹，输入 wmake

37. 深入解析 icoFoam 下的顶盖驱动流(cavity)

(2009-05-02 17:14:52)转载

标签： openfoam 教育 分类： OpenFOAM 使用

icoFoam 为 OpenFOAM 中单相不可压缩流求解器,运用 PISO 算法求解压力-速度耦合.顶盖驱动流动为不可压流动经典算例,通常用来验证编写程序的正确性和有效性.

求解器 icoFoam 位置: /applications/solvers/incompressible/icoFoam

顶盖驱动流算例位置: /tutorials/icoFoam/cavity

顶盖驱动流算例下文件夹结构

```
|--0-              //初始条件和边界条件
|       |--p       //压力
|       |--U       //速度
|
|-constant-     //参数和网格
|                 |-transportProperties    //物理参数设定文件
|                 |-polyMesh                    //网格文件夹
|                             |-blockMeshDict //blockMesh 网格生成文件
|                             |-boundary          //边界文件,运用 blockMesh 可生成该文件,可删除
|-system-
            |-controlDict //程序运行控制文件
            |-fvSchemes     //离散格式设定文件
            |-fvSolution    //代数方程求解器设定文件
```

上面文件 p,controlDict, fvSchemes, fvSolution 已经在本站博文"使用 OpenFOAM 的基本流程"中详述

下面对其他文件进行说明

文件 U

```
//文件头
FoamFile
{
    version         2.0;
```

```
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//单位,关于物理单位配置,在本站博文"OpenFOAM>>solver>>basic>>potentialFoam 的说明"

dimensions      [0 1 -1 0 0 0 0];
//初始条件,全场速度为 0
internalField   uniform (0 0 0);
//边界条件
boundaryField
{
    movingWall //顶盖速度
    {
        type            fixedValue;     //恒定速度
        value           uniform (1 0 0); //速度大小 Ux=1 m/s
    }

    fixedWalls //其他 3 面为墙,速度无滑移
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBack //前面和后面为 empty,因为是二维的
    {
        type            empty;
    }
}

文件 transportProperties
//文件头
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

//流体黏性系数,关于如何设置并使用参数,参看本站博文"OpenFOAM 中的参数字典使用剖析"

nu              nu [0 2 -1 0 0 0 0] 0.01;

文件 blockMeshDict
//文件头
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
```

```
    object          blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;    //单位转换,也就是说下面所有点都要乘以 0.1 才是真值

vertices //网格中的点,点的位置没有顺序要求,点的序号从 0 开始
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);

blocks    //块结构化网格中的一个块
(
    hex (0 1 2 3 4 5 6 7) //块中的 8 个边界点,方的是前面定义点的序号
    (20 20 1)                    //x y z 3 个方向的网格的个数
    simpleGrading (1 1 1) //网格均匀划分
);
//块的 8 个点的顺序
(1)每个块都有一个局部坐标,坐标的原点为 hex 中的第 0 个点,也是点序列的第 0 个点
(2)hex 中的第 0 个点和第 1 个点构成 X 方向
(3)hex 中的第 1 个点和第 2 个点构成 Y 方向
(4)hex 中的第 0 1 2 3 个点构成平面 Z=0
(5)hex 中第 0 个点和第 4 个点构成方向 Z
(6)第 5 6 7 个点的寻找方法和点 1 2 3 相同.
应当注意,上面的那个例子中 hex 中点的序号和点在 vertices 的序号恰好一致.

//边的定义,对于曲线边才会定义,直边不定义,本例子中所有边是直边
edges
(
);

patches //边界
(
    wall movingWall    //上面墙
    (
        (3 7 6 2)
    )
    wall fixedWalls //其他三面墙
    (
        (0 4 7 3)
        (2 6 5 1)
        (1 5 4 0)
    )
    empty frontAndBack //前面和后面两个面
    (
        (0 3 2 1)
```

```
            (4 5 6 7)
        )
);
```

//上面边界中点有一定顺序,满足右手定则,是边界面的法向量指向外,也就是从外面向里看,逆时针方向.

```
mergePatchPairs       //需要和在一起的 patch 对
(
);
```

文件 boundary 中的内容不用管他,运用 blockMesh 会将其覆盖.

程序运行命令
运用控制台进入/tutorials/icoFoam/cavity
输入: blockMesh 生成网格
输入: icoFoam 运行程序
程序运行结束后,输入 paraFoam 做后处理.

38. OpenFOAM 中的参数字典使用剖析

(2009-05-02 06:52:32)转载
标签： openfoam 教育 分类： OpenFOAM 使用

对于模型参数 OpenFOAM 采用了参数字典 dictionary 来由用户指定，参数字典的使用，对参数设置及其新模型的 OpenFOAM 扩充至关重要。本文从参数字典类对象构建，字典文件设置，字典查询对参数字典类的使用进行详解。

（1）字典构建
对字典类对象构建通常采用如下形式

```
    IOdictionary ObjectName
    (
        IOobject
        (
            "dictionaryFileName",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
```

其中：ObjectName 和“dictionaryFileName”换成你要构建对象名字和字典文件名字。

（2）字典文件设置
典型参数类型设置方法：

scalar，label，word 类型：
关键字 值;
如参数 C1 的值为 0.1，则可设置为
C1 0.1;

带有单位的类型 dimensionedScalar（带有单位的标量）
关键字 创建的参数名字 参数单位 值;
如 黏性设置

```
nu              nu [0 2 -1 0 0 0 0] 5.952e-06;
```
加速度矢量
```
g               g [0 1 -2 0 0 0 0] (0 0 0);
```

字典可以分组，成为子字典（subDict)
如大涡一方程模型参数设置
```
oneEqEddyCoeffs//子字典名字
{
    ck              0.07; //参数
    ce              1.05; //参数
}
```
子字典还可以有子字典以此类推。

（3）字典的查询
通过字典文件设定参数以便在程序中使用。
label,scalar 类型查询
比如查询 ObjectName 字典中参数 C1 的值，将其值付给变量 Vc1,;
```
scalar Vc1(readScalar(ObjectName.lookup("C1"));
```

查询带单位的量，比如查询黏性
```
dimensionedScalar nu(ObjectName.lookup("nu"));
```
查询加速度
```
dimensionedVector g(ObjectName.lookup("g"));
```
注意对于基本类型 label 或者 scalar 需要使用 readLabel 或者 readScalar 两个函数，而对于带单位的量直接查询就可以了。

子字典查询
如查询 oneEqEddyCoeffs 下的 ck
```
dimensionedScalar ck(readScalar(ObjectName.subDict("oneEqEddyCoeffs").lookup("ck")));
```
如果子字典还有子字典，则连续使用 subDict。

39. OpenFOAM>>solver>>incompressible>>icoFoam 的说明

(2009-05-02 00:42:06)转载
标签： openfoam solver 教育 分类： OpenFOAM 求解器说明
```
//createFields.H
//读入属性提示
Info<< "Reading transportProperties\n" << endl;
//声明属性字典类对象，该对象由 constant 文件夹下的“transportProperties”初始化创建。
    IOdictionary transportProperties
    (
        IOobject
        (
            "transportProperties",//字典文件名字
            runTime.constant(), //位置
            mesh,                   //网格对象
            IOobject::MUST_READ, //必须读，由文件创建
            IOobject::NO_WRITE     //不对该文件进行写
        )
    );
//字典查询黏性一遍初始化带有单位的标量
    dimensionedScalar nu
    (
```

```
            transportProperties.lookup("nu")
        );
//提示创建压力场
        Info<< "Reading field p\n" << endl;
//创建压力场
        volScalarField p
        (
            IOobject
            (
                "p",
                runTime.timeName(),
                mesh,
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            mesh
        );

//提示并创建速度场
        Info<< "Reading field U\n" << endl;
        volVectorField U
        (
            IOobject
            (
                "U",
                runTime.timeName(),
                mesh,
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            mesh
        );

//创建界面流律
#       include "createPhi.H"

//设定压力参考 cell 的 index 和参考值
        label pRefCell = 0;
        scalar pRefValue = 0.0;
//通过查询 system/fvSolution 下的 PISO 进行更新压力参考 cell 更新并设定参考值。
        setRefCell(p, mesh.solutionDict().subDict("PISO"), pRefCell, pRefValue);

//icoFoam.C
int main(int argc, char *argv[])
{

//下面是很多 solver 都包括的，前面 solver 介绍时候已经说明。
#       include "setRootCase.H"
#       include "createTime.H"
#       include "createMesh.H"
#       include "createFields.H"
#       include "initContinuityErrs.H"
```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

// 用 runTime 对象控制时间推移
    for (runTime++; !runTime.end(); runTime++)
    {
 //显示当前物理时间
        Info<< "Time = " << runTime.timeName() << nl << endl;
 //读入 piso 控制参数，写在 runTime 循环中，以便运行过程中的修改生效
#   include "readPISOControls.H"
 //计算 courant 数
#   include "CourantNo.H"
      //构造速度方程
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
        );

//应当指出上面方程中并没有包含 grad(p),动量方程中所有的项要么在压力方程中要么在动量方程中，而不同时在两个方程中。
//求解动量方程，下面方程中的 p 由上一时刻 p 计算，下面求解称为速度预测。
        solve(UEqn == -fvc::grad(p));
        // --- PISO loop---- 速度修正步
        for (int corr=0; corr<nCorr; corr++)
        {
        //根据预测的速度值求解系数矩阵对角元素的倒数。
            volScalarField rUA = 1.0/UEqn.A();

// 更新速度， See Hrv Jasak's thesis eqn. 3.137 and Henrik Rusche's thesis, eqn. 2.43

            U = rUA*UEqn.H();

// 计算表面流律
// The ddtPhiCorr 用来考虑差值速度场计算的流律和实际流律的差别
            phi = (fvc::interpolate(U) & mesh.Sf())
                + fvc::ddtPhiCorr(rUA, U, phi);
 //调整边值，以便满足连续性条件
            adjustPhi(phi, U, p);
//非正交循环
            for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
            {
  //设定压力方程并进行求解
                fvScalarMatrix pEqn
                (
                    fvm::laplacian(rUA, p) == fvc::div(phi)
                );
                  pEqn.setReference(pRefCell, pRefValue);
                  pEqn.solve();
    //根据新求解的压力，更新表面流律
                if (nonOrth == nNonOrthCorr)
```

```
                {
                    phi -= pEqn.flux();
                }
            }   // end of non-orthogonality looping
 //计算连续性方程误差
#               include "continuityErrs.H"

//根据新的压力，修正速度

            U -= rUA*fvc::grad(p);
//修正速度边界(主要针对第二类边界条件）
            U.correctBoundaryConditions();

        } // end of the PISO loop
 //输出计算结果
        runTime.write();
 //显示执行时间，CPU 时间
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "    ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;

    } // end of the time step loop
 //提示运行结束。
    Info<< "End\n" << endl;
 //返回 0
    return(0);
}
```

40. OpenFOAM 安装详解

(2009-04-30 17:54:01)转载

标签： openfoam 安装 教育 分类： OpenFOAM 入门

OpenFOAM 的安装并没有想象的那么复杂，感觉安装比较困难主要是由于对 linux 和 OpenFOAM 的配置不熟造成的。下面对 OpenSUSE（一种 linux 系统，OpenFOAM 就是基于该平台开发的）下 OpenFOAM 的安装过程详细的介绍一下。

（1）系统的选择。本人推荐选择 OpenSUSE，他比较容易用，图形界面和 windows 接近，且安装 OpenFOAM 时候需要的包，在系统安装的时候可以一并选择，省去了以后重新安装包的麻烦。当然你可以选择其他 linux 系统，可能安装比较麻烦一些。

（2）OpenSUSE 的安装，比较简单，你腾出一个 windows 盘（最好大于 30g），然后在网上下载一个 openSUSE（最好刻成盘）直接用图形界面安装安装到腾出的盘中。网上 OpenSUSE 的安装说明很多，可以按照一步一步来，很简单。在软件安装选择的时候记着将下面包一起安装：gcc(编译器)，python（一种语言，paraview 需要），cmake（编译 paraview 需要）Qt(paraview 需要)。忘记选择也不要紧，可以在系统安装后用 yast 安装。

（3）下载 OpenFOAM 包，到 openfoam 网站上去，选择 linux 平台，32 位或者 64 位，这个是你的 opensuse 是 32 为或者 64 位的，不是指的硬件。
网址是
http://www.opencfd.co.uk/openfoam/linux.html

该网页上说的 source pack 是指只有源代码的包，下载该包需要编译，binary pack 是编译好的包，下载下来直接用就行了，不用安装的。如果你只是像用 fluent 或 cfx 一样用 openfoam，不搞开发，直接用 binary 包就行了。你要是想对 OpenFOAM 包进行扩充的话，下载 source parck，在本机上用 debug 模式重新编译一下。

（4） 包的解压缩

在你的根目录下创建一个名字为 OpenFOAM 的文件夹，将上面下载的包放在里面。在控制台下进入 OpenFOAM 文件夹

cd ~/OpenFOAM

并输入命令

tar xzf OpenFOAM-1.5.General.gtgz

tar xzf ThirdParty.General.gtgz

对你下载的包进行当前文件夹解压缩

（5）环境配置

将下面一句话加入到根目录下的.bashrc 中，注意文件前面有一个“.”，说明该文件为隐藏文件

只需要在控制台上输入

kate $HOME/.bashrc

就会代开.bashrc，将下面一句话加入到文件中

. $HOME/OpenFOAM/OpenFOAM-<version>/etc/bashrc

其中，<version>用版本代替，比如你安装了 1.5，则

. $HOME/OpenFOAM/OpenFOAM-1.5/etc/bashrc

注意前面的"."，不能去掉，linux 下的点表示对某个文件的执行。

如果你想在 debug 模式下编译更改文件

/OpenFOAM/OpenFOAM-1.5/etc/bashrc 中

将${WM_COMPILE_OPTION:=Opt};..前面加上＃，将${WM_COMPILE_OPTION:=Debug}前的＃去掉。

#: ${WM_COMPILE_OPTION:=Opt}; export WM_COMPILE_OPTION

: ${WM_COMPILE_OPTION:=Debug}; export WM_COMPILE_OPTION

如果你要使用你系统的编译器（通常都需要改，ThirdParty 里面没有 gcc），将

/OpenFOAM/OpenFOAM-1.5/etc/settings.sh 中的

compilerInstall=OpenFOAM 前面加＃，并将 compilerInstall=System 前＃去掉

#compilerInstall=OpenFOAM

compilerInstall=System

（6）更新环境

在控制台上输入命令

source $HOME/.bashrc

或者

关掉控制台，在重新打开

（7）编译第三方包

进入 OpenFOAM/ThirdParty

运用命令 cd $HOME/OpenFOAM/ThirdParty

输入 ./Allwmake

（8）编译 OpenFOAM
进入/OpenFOAM/OpenFOAM-1.5
cd $HOME/OpenFOAM/OpenFOAM-1.5
输入./Allwmake

(9)编译 paraview
cd $FOAM_INST_DIR/ThirdParty
rm -rf ParaView3.3-cvs/platforms
buildParaView3.3-cvs

现在安装过程已经说完了，祝你好运

41. OpenFOAM>>solver>>basic>>scalarTransportFoam 的说明

(2009-04-30 03:23:05)转载
标签： openfoam solver 教育 分类： OpenFOAM 求解器说明
createFields.H

```
//提示读入温度场
Info<< "Reading field T\n" << endl;
//温度场创建，标量场，需要初始化文件，下面各项具体含义，参看以前 solver 的说明
volScalarField T
    (
        IOobject
        (
            "T",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
//提示读入速度场
    Info<< "Reading field U\n" << endl;
//速度场创建，各项意义，前面 solver 说明中已经给出
volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
//读入参数提示
    Info<< "Reading transportProperties\n" << endl;
//根据名字为 transportProperties 的参数文件构建参数字典，以便查找。
IOdictionary transportProperties
```

```
    (
        IOobject
        (
            "transportProperties",//参数字典文件名字
            runTime.constant(),//参数字典文件位置
            mesh,                    //网格对象
            IOobject::MUST_READ,//需要读入文件
            IOobject::NO_WRITE //不对文件进行重写
        )
    );
//查询参数字典提示
    Info<< "Reading diffusivity D\n" << endl;
//参数字典查询，初始化带单位标量 DT（温度扩散率）
dimensionedScalar DT
    (
        transportProperties.lookup("DT")
    );
//创建表面流率场，该文件位于
//src ? finiteVolume ? cfdTools ? incompressible
#   include "createPhi.H"

scalarTransportFoam.C
#include "fvCFD.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//主程序入口
int main(int argc, char *argv[])
{
//设置 case 目录相关，位于 src ? OpenFOAM ? include
#   include "setRootCase.H"

//创建 time 对象 runTime，位于 src ? OpenFOAM ? include
#   include "createTime.H"

//创建网格对象 mesh，位于 src ? OpenFOAM ? include
#   include "createMesh.H"
//创建场对象，前面已经详述
#   include "createFields.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//提示计算标量传输方程
    Info<< "\nCalculating scalar transport\n" << endl;
//显示当前 courant 数，位于 src ? finiteVolume ? cfdTools ? incompressible
#   include "CourantNo.H"

//计算主流程
    for (runTime++; !runTime.end(); runTime++)
    {
//显示当前时间（物理时间，非 cpu 耗时）
        Info<< "Time = " << runTime.timeName() << nl << endl;
//读入 simple 算法控制参数，位于 src ? finiteVolume ? cfdTools ? general ? include
#       include "readSIMPLEControls.H"
//网格非正交循环
        for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
```

```
        {
//构造并求解方程
            solve
            (
                fvm::ddt(T)   //非稳态项，隐式离散
              + fvm::div(phi, T) //对流项，隐式离散
              - fvm::laplacian(DT, T) //扩散项，隐式离散
            );
        }
        runTime.write(); //求解结果输出，由于采用了注册机制，所有 AUTO_WRITE 声明的
变量，
//都会输出
    }
    Info<< "End\n" << endl; //提示程序结束
    return(0); //返回 0
}
```

42. OpenFOAM>>solver>>basic>>potentialFoam 的说明

(2009-04-29 21:25:27)转载
标签： openfoam solver 教育 分类： OpenFOAM 求解器说明

```
//creatField.H
//提示创建压力场
Info<< "Reading field p\n" << endl;
//压力场为标量场，网格中心存储变量（OpenFOAM 用的是非结构化同位网格），下面为创建
标量场压力，两个参数 IOobject 对象和网格对象 mesh，IOobject 主要从事输入输出控制
    volScalarField p
    (
        IOobject
        (
            "p",            //压力场初始文件的名字。
            runTime.timeName(), //位置，该位置由 case 中的 system/controlDict 中的
startTime 控制
            mesh,                   //网格对象，主要从事对象注册，以便由 runTime.write()
控制输出
            IOobject::MUST_READ,//该对象由文件读出创建，因此，需要 READ
            IOobject::NO_WRITE //不输出压力场
        ),
        mesh   //压力场所用的网格对象，在 createMesh.H 创建
    );
//压力场初始化为 0，单位为上面文件中的单位。dimensionedScalar 为带单位的标量，初始
化三个
//参数，名字，单位，数值。也可采用如下形式
//dimensionedScalar("zero",dimensionSet(1,-1,-2,0,0 ,0, 0),0.0);
//应当注意，OpenFOAM 中的大部分 case 对动量的求解都是求解的速度场，压力单位初始化
时候，
//一般为除去密度后的值.
//dimensionSet 中有 7 个参数，他们依次为质量、长度、时间、温度、摩尔数、安培、坎德
拉。

p = dimensionedScalar("zero", p.dimensions(), 0.0);
```

```
//提示读入速度场
Info<< "Reading field U\n" << endl;
    //创建速度场，向量场，体心存储变量。
volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE   //自动写，根据 runTime.write()或者 U.write();
        ),
        mesh
    );

//初始化速度场。这里初始化速度为 0 ，如果初始化 Ux=1，Uy=2，Uz=3 （单位为国际单位）可采用
//U = dimensionedVector("0", U.dimensions(), vector(1,2,3));
U = dimensionedVector("0", U.dimensions(), vector::zero);

//表面场，phi，界面流率，存储在体之间的交接面上。表面场(surface...)不能和体积场(vol...)
//直接计算，因为他们存储在不同地方，大小不同。
//可以将体积场转化为表面场（运用 fvc::interpolate())
//或者由表面场转化为体积场（运用 fvc::reconstruct())进行计算。
surfaceScalarField phi
    (
        IOobject
        (
            "phi",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
    //U 是体积场，运用插值转为表面场和表面积场进行相乘来初始化流率
//mesh.Sf()返回网格交接面面积矢量。
    fvc::interpolate(U) & mesh.Sf()
);
//压力参考 cell
    label pRefCell = 0;
//压力参考值
    scalar pRefValue = 0.0;
    setRefCell(p, mesh.solutionDict().subDict("SIMPLE"), pRefCell, pRefValue);
   //只有求解区域所有的压力边界都为第二类边界条件是，上面的值才会用到。如果有第一类边界条件，
//压力参考值为这点边界值。对于不可压缩流动压力值为相对值，上面的参考值的大小对结果无影响。


potentialFoam.C
#include "fvCFD.H"
```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//主程序入口
int main(int argc, char *argv[])
{
//增加新的有效输入参数。
argList::validOptions.insert("writep", "");
//设置 case 目录
#    include "setRootCase.H"
//创建时间，对计算流程进行控制，主要是名为 runTime 的对象
#    include "createTime.H"
//创建网格对象 mesh
#    include "createMesh.H"
//创建场变量，前面已经说过
#    include "createFields.H"
//读 simple 控制参数
#    include "readSIMPLEControls.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//计算势流提示
    Info<< nl << "Calculating potential flow" << endl;
//对流率进行调整，以保证方程的连续性
    adjustPhi(phi, U, p);
//网格非正交性循环，如果网格是正交的，可以设定 nNonOrthCorr=1
  for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
//创建压力方程，该方程为标量稀疏矩阵类
        fvScalarMatrix pEqn
        (
            fvm::laplacian   //拉普拉斯离散，隐式
            (
                dimensionedScalar
                (
                    "1",
                    dimTime/p.dimensions()*dimensionSet(0, 2, -2, 0, 0),
                    1
                ),
                p
            )
          ==
            fvc::div(phi)//速度离散，显示。因为是压力方程，其他变量只能显示
        );
//设定压力参考值
        pEqn.setReference(pRefCell, pRefValue);
//求解压力方程，调用的 fvMatrix 成员函数
        pEqn.solve();
//流率修正，应当注意 OpenFOAM 中对压力本身求解，而非压力变化值。
//关于 simple 算法和 PISO 算法的 OpenFOAM 实现，会在 simpleFoam 及其 icoFoam 中详细说
明。
if (nonOrth == nNonOrthCorr)
        {
            phi -= pEqn.flux();
        }
    }
//提示连续性方程求解误差
```

```
    Info<< "continuity error = "
        << mag(fvc::div(phi))().weightedAverage(mesh.V()).value()
        << endl;
//根据表面场重建速度场
    U = fvc::reconstruct(phi);
//对速度场边界进行修正，主要针对第二类边界条件下的边界场
    U.correctBoundaryConditions();
    Info<< "Interpolated U error = "
        << (sqrt(sum(sqr((fvc::interpolate(U) & mesh.Sf()) - phi)))
           /sum(mesh.magSf())).value()
        << endl;
    // Force the write
//直接对速度进行输出
    U.write();
//界面流率输出。注意当前场存储在界面上，phi 的大小（个数）和 U 的大小（个数）不相同的
    phi.write();
//如果用户计算是，让输出 p，即输入了 writep，则输出 p
    if (args.options().found("writep"))
    {
        p.write();
    }
//提示执行时间，cpu 时间
    Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
        << "  ClockTime = " << runTime.elapsedClockTime() << " s"
        << nl << endl;
//提示程序结束
    Info<< "End\n" << endl;
//返回 0
return(0);
}
```

43. OpenFOAM>>solver>>basic>>laplacianFoam 的说明

(2009-04-29 18:01:22)转载
标签： openfoam solver 分类： OpenFOAM 求解器说明

```
//createFields.H

//一屏幕提示。Info 等价于 C++中 std::cout,Info 间接调用 cout
    Info<< "Reading field T\n" << endl;

//一声明一个标量场,网格中心存储变量。该场是通过读入文件(IOobject and MUST_READ)进行设置,并根据 controlDict 中的设置进行自动 write,由 write.H 中的 runTime.write()来执行 write();。

    volScalarField T
    (
        IOobject
        (
            "T",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
```

```
        mesh
    );

     //- 提示读入参数控制文件
    Info<< "Reading transportProperties\n" << endl;
    //- 参数控制文件声明过文件形式读入
    IOdictionary transportProperties
    (
        IOobject
        (
            "transportProperties", //文件名字
            runTime.constant(), //文件位置，case 文件夹中 constant 子文件夹
            mesh,
            IOobject::MUST_READ,//通过 read 一个文件，初始化
            IOobject::NO_WRITE   //并不根据时间对文件进行写
        )
    );

    //－提出读入扩散律
    Info<< "Reading diffusivity DT\n" << endl;

    //－通过查询参数控制文件，初始化带有单位的标量，lookup 中的“DT”为关键字
    dimensionedScalar DT
    (
        transportProperties.lookup("DT")
    );


//laplacianFoam.C


#include "fvCFD.H"   //-cfd 头文件，包括大多数 cfd 计算需要的头文件，在
src ? finiteVolume ? cfdTools ? general ? include


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//主程序入口
int main(int argc, char *argv[])
{
//设置 rootcase，根据输入参数 argc  和  argv 对
#    include "setRootCase.H"

//－创建时间，下面的 runTime 控制
#    include "createTime.H"

//创建网格，根据 constant 文件中 polyMesh 文件夹中的网格数据，创建网格对象 mesh，位
于
src ? OpenFOAM ? include
#    include "createMesh.H"

//创建场对象，在前面已经说明
#    include "createFields.H"
```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//提示计算温度分布
    Info<< "\nCalculating temperature distribution\n" << endl;
//计算主控制流程
    for (runTime++; !runTime.end(); runTime++)
    {
 //提示当前计算时间
        Info<< "Time = " << runTime.timeName() << nl << endl;
//读入 simple 算法参数，位于
src ? finiteVolume ? cfdTools ? general ? include

#       include "readSIMPLEControls.H"
//对于网格非正交循环修正。
        for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
        {
//求解拉普拉斯方程,这里的 solve 是 Foam 空间的全局函数，内参数为 fvMatrix，fvm 表示隐
式离散，返回有限容积稀疏矩阵类 fvMatrix 对象，具体对象中内容，以后说明
            solve
            (
                fvm::ddt(T) - fvm::laplacian(DT, T)
            );
        }
//对求解变量进行写
#       include "write.H"
//提示执行时间及 CPU 耗时
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "    ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
//提示程序结束
    Info<< "End\n" << endl;

    return(0);
}
```