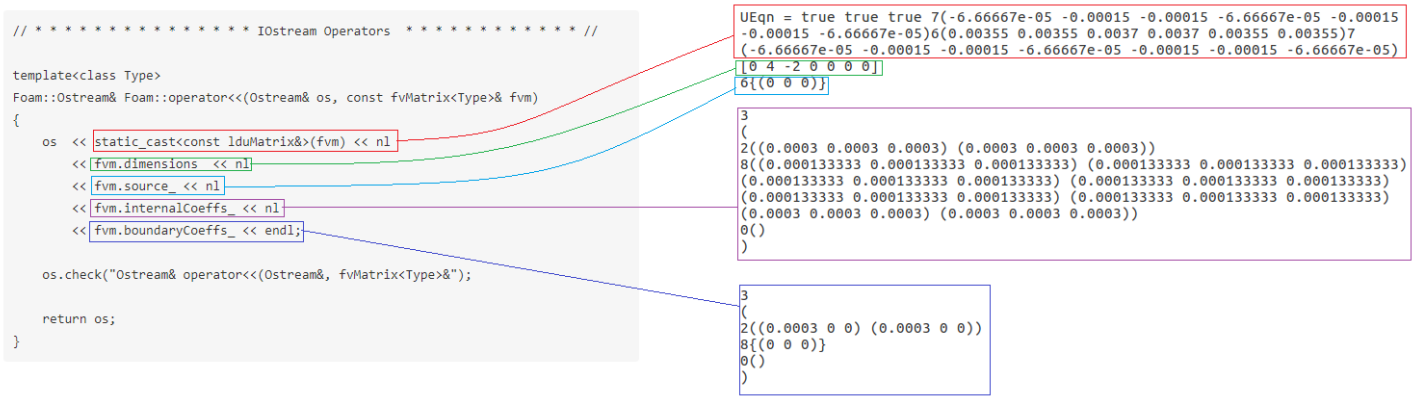
知乎 首发于 有所思



OpenFOAM中的动态多态（Info,solve）

陈与论
答案一直很简单

关注他

19 人赞同了该文章

我到目前为止，只碰到两个函数在doxygen里找不到，一个是Info，一个是solve。都是FOAM类下面直接调用的全局函数，doxygen里找不到。他们都是C++中的动态多态。以后可能还会遇到更多的动态多态，所以这里总结一下他们是怎么实现的。

（动态多态这个词我其实以前不太清楚，我知道OpenFOAM里是怎么操作Info和solve的，但是不太清楚专业的计算机领域怎么称呼这种操作。以前有人说这叫“delegate委托”，昨天评论区里有人说这叫做“动态多态”，我google了一下，感觉“动态多态”更符合我的理解。）

首先，什么是“动态多态”呢?

以我目前的理解，可以以税收制度打个比方，不同的人交的税不一样。面对这种情况，如果你是税务官，你怎么去收税？假如你自己手里拿着张收税规则，挨门挨户问对方有多少收入，然后自己啪啪敲一遍算盘，告诉他要交多少钱，这就不是多态，而是普通的函数式编程；多态就省事多了，税务官直接在操场上摆个桌子：“收税了！每个人自己算算要交多少！算好了直接过来给我！”然后坐着喝茶收钱就行了。

通过上面的例子可以看出，多态的特点是让每个对象自己忙活，税务官这个function其实手里没有一收税规则，他只是喊了一嗓子“收税了！”而已，真正背后每个对象交多少税这个数学形式，是分散分布在每个纳税人心里的，假如你非要统计一下其中的数学形式，就要跑到每个人那里，挨个

▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ...

知乎 首发于 有所思

水的，他交1/20的收入；没有问题。但假如人多呢？比如1000个纳税人，你没有精力搞出来1000种交税细则，而且可能有些人即砍柴又挑水，那你怎么办？这时候你就需要设置一些抽象的不存在的虚拟人物“砍柴人”，“挑水人”，告诉民众：假如你是“砍柴人”，你就交收入的1/10，假如你是“挑水人”，你就交收入的1/20，假如你两个都是，就交双份，这就是动态多态。

如果还不太清楚，可以看看这篇文章中的代码例子：

C++中的静态多态和动态多态 - lizhenghn - 博客园
www.cnblogs.com

Info函数：

Info是个也是个动态多态。他实际上没有任何function，所有function都写在他调用的class里，调用fvMatrix就用fvMatrix里的function，调用volVectorField就用volVectorField里的function。

在OpenFOAM里，我们经常看到这样的定义：

```
Info << UEqn;
```

对于一个1*2*3网格的例子，输出结果是这样的：

```
UEqn = true true true 7(-6.66667e-05 -0.00015 -0.00015 -6.66667e-05 -0.00015 -0.00015
[0 4 -2 0 0 0 0]
6{(0 0 0)}

3
(
2((0.0003 0.0003 0.0003) (0.0003 0.0003 0.0003))
8((0.000133333 0.000133333 0.000133333) (0.000133333 0.000133333 0.000133333) (0.00013
0()
)
```

▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ···

知乎 首发于 有所思

```
2((0.0003 0 0) (0.0003 0 0))
8{(0 0 0)}
0()
)
```

我们知道Info<<会输出这个类的信息，但是输出什么信息，是如何在代码中定义的呢?

Info是一个动态多态，他本身没有任何对UEqn进行处理的代码，当这行语句执行时，是执行<<UEqn, 此时你可以把<<看作是一个函数名称，那么<<这个函数在UEqn中是怎么定义的呢?

UEqn是一个fvVectorMatrix类:

```
typedef fvMatrix<vector> fvVectorMatrix
```

fvVectorMatrix继承自fvMatrix类;

fvMatrix.C:

```
// * * * * * * * * * * * * * * * * * * IOstream Operators  * * * * * * * * * * * * * * * * * //

template<class Type>
Foam::Ostream& Foam::operator<<(Ostream& os, const fvMatrix<Type>& fvm)
{
    os  << static_cast<const lduMatrix&>(fvm) << nl
        << fvm.dimensions_ << nl
        << fvm.source_ << nl
        << fvm.internalCoeffs_ << nl
        << fvm.boundaryCoeffs_ << endl;

    os.check("Ostream& operator<<(Ostream&, fvMatrix<Type>&");

    return os;
}
```

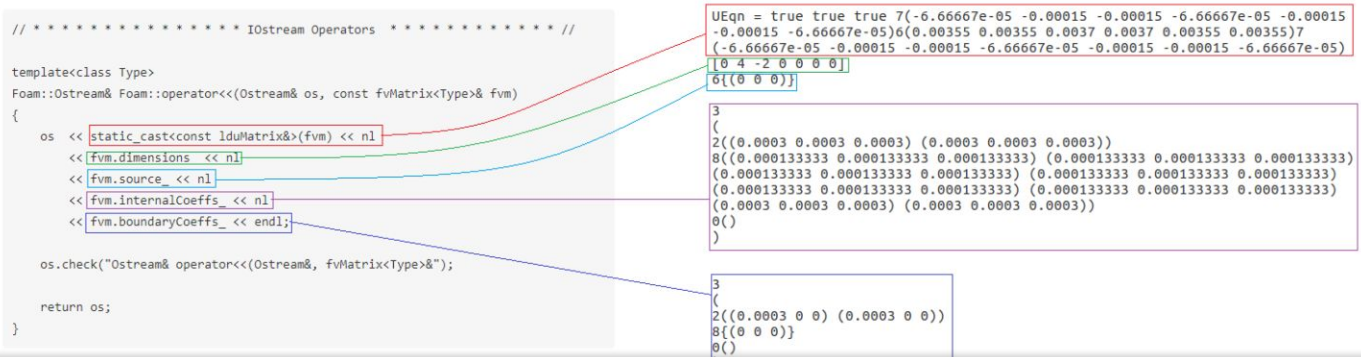
fvMatrix继承自lduMatrix类。可以看出fvMatrix中的<<函数先调用了lduMatrix中的<<函数。static_cast是一种强制类型转换的命令，通过这个命令将fvMatrix类型的UEqn转换成lduMatrix类型的UEqn，然后输出lduMatrix类型中对<<函数的定义。

▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ...

知乎 首发于 有所思

```
    Switch hasLow = ldum.hasLower();
    Switch hasDiag = ldum.hasDiag();
    Switch hasUp = ldum.hasUpper();

    os  << hasLow << token::SPACE << hasDiag << token::SPACE
        << hasUp << token::SPACE;

    if (hasLow)
    {
        os  << ldum.lower();
    }

    if (hasDiag)
    {
        os  << ldum.diag();
    }

    if (hasUp)
    {
        os  << ldum.upper();
    }

    os.check("Ostream& operator<<(Ostream&, const lduMatrix&");

    return os;
}
```

可以看出，这里先输出了是否有low/diag/upper数组，然后再分别输出这几个数组。跟上面的输出结果也能对的上。接着输出了fvMatrix类中的三个子变量，分别是dimensions_，source_，internalCoeffs_，boundaryCoeffs_。这就是Info工作的全过程。下图左边是例子的输出结果，右边是UEqn所在的fvVectorMatrix类对输出什么的定义：



▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ...

知乎 首发于 有所思

对应：

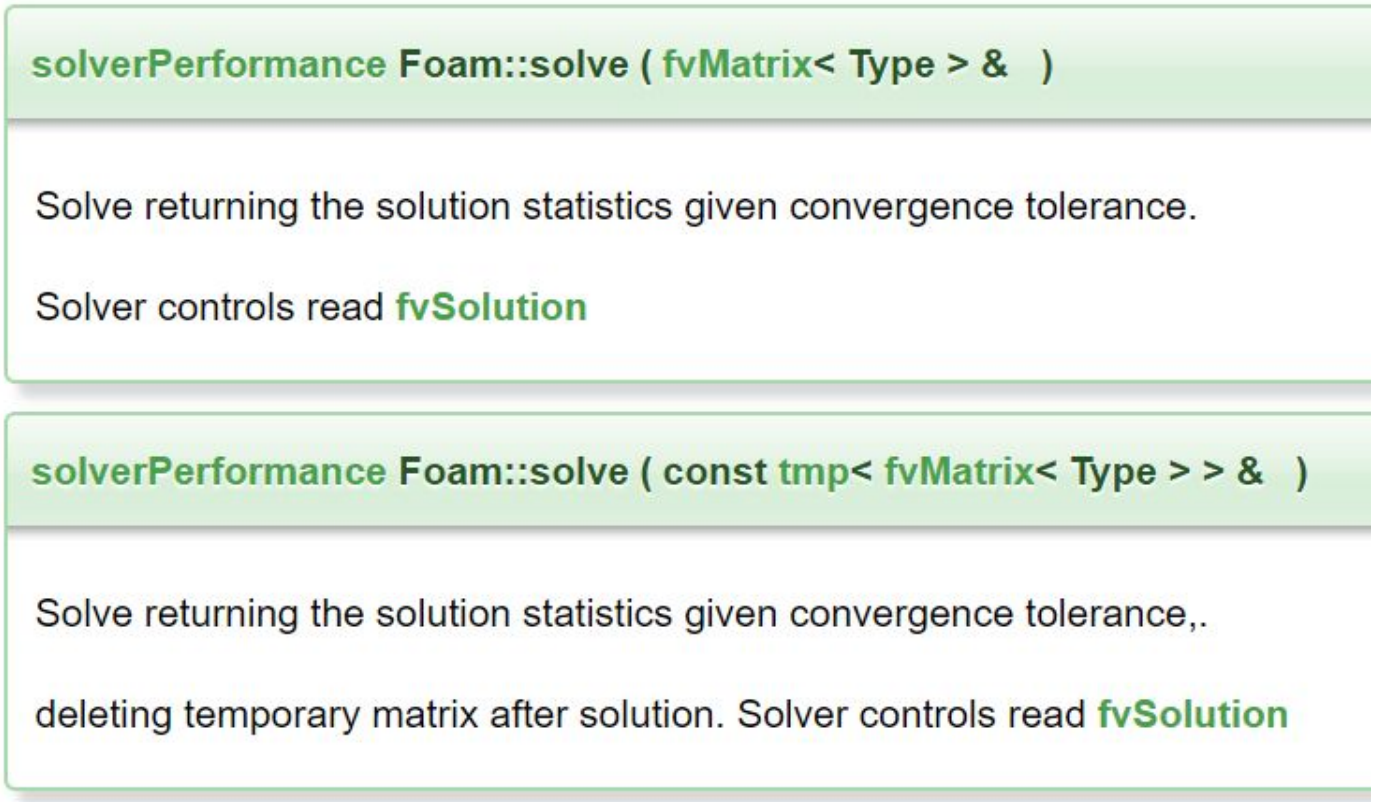
```
template<class Type>
Foam::Ostream& Foam::operator<<(Ostream& os, const fvMatrix<Type>& fvm)
```

可以看出，UEqn对应fvMatrix<Type>类，<<对应Foam::operator<<，Info对应Foam::Ostream&。OpenFoam里的<<符号一定要配合Info使用，随便搞一个cout是不行的，因为cout不是Ostream类。

---

solve函数：

如果你直接在doxygen上找，你是很难找到solve函数的链接的，你会找到下图的描述：



然后就没有链接了！

这个页面上的所有链接，都没有说这个定义是在源代码的那个文件里给出来的，非常不负责任

▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ···

知乎 首发于 有所思

src/finiteVolume/fvMatrices/fvMatrix/fvMatrix.C

```
template<class Type>
Foam::solverPerformance Foam::solve
(
    fvMatrix<Type>& fvm,
    const dictionary& solverControls
)
{
    return fvm.solve(solverControls);
}
```

也就是说，这个语句里定义了：凡是对我fxMatrix类施加的Foam::solve函数，都要按我这里的定义来操作。这样Foam::solve函数其实啥都没干了，都是fvm.solve(solverControls)这个函数在操作。而fvm又属于fvMatrix，很明显这就是“自己的税自己算”的多态方法。

输入参数里：fvm是待解的方程组信息，solverControls应该需要到system/fvSolution文件里找，内容就是用什么求解器，设置多少tolerance之类的。这个东西我倒是从来没注意过，应该就是不同的求解线性矩阵的算法吧，我还很少碰到需要改这里的情况，暑假里做一个case的时候经常发散，一个老司机给了我建议，修改了fvSolution里的东西，收敛性就好了些，但也没有完全解决问题，这都是我未曾探索的领域，深了。比如Krylov算法解线形方程什么的，应该就是放在这个solverControls里。

返回值是fvm.solve(solverControls)。然而在fvMatrix.C文件里没有solve这个函数。

去doxygen上查看，却显示fvMatrix里有solve这个函数：

```
solverPerformance solve (const dictionary &)
                  Solve segregated or coupled returning the solution statistics. More...
```

连忙追踪过去一看，原来fvMatrix.C文件还不是全部内容，fvMatrix.C文件最后还把另一部分内容写到了fvMatrixSolve.C这个文件里，solve就是在那个文件里定义的：

```
#include "fvMatrixSolve.C"
```

fvMatrixSolve.C

▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ···

知乎 首发于 有所思

```
59 )
60 {
61     if (debug)
62     {
63         Info.masterStream(this->mesh().comm())
64             << "fvMatrix<Type>::solve(const dictionary& solverControls) : "
65                "solving fvMatrix<Type>"
66             << endl;
67     }
68
69     label maxIter = -1;
70     if (solverControls.readIfPresent("maxIter", maxIter))
71     {
72         if (maxIter == 0)
73         {
74             return solverPerformance();
75         }
76     }
77
78     word type(solverControls.lookupOrDefault<word>("type", "segregated"));
79
80     if (type == "segregated")
81     {
82         return solveSegregated(solverControls);
83     }
84     else if (type == "coupled")
85     {
86         return solveCoupled(solverControls);
87     }
88     else
89     {
90         FatalIOErrorIn
91         (
92             "fvMatrix<Type>::solve(const dictionary& solverControls)",
93             solverControls
94         )   << "Unknown type " << type
95             << "; currently supported solver types are segregated and coupled"
96             << exit(FatalIOError);
97
98         return solverPerformance();
99     }
```

▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ...

知乎 首发于 有所思

到这里这篇博客就介绍完了。

编辑于 2018-01-09

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

计算流体力学（CFD） openfoam

文章被以下专栏收录

有所思
与PhD们交流读博期间的所思所感，呼吁社会关爱

关注专栏

推荐阅读

【原创】还在苦恼于求解多相流的龟速吗，快跟小编看看这个...

本文原创作者：Exercise团队 林志鹏 硕士小编在做多相流模拟时碰到一个网格规模为600k+的球头圆柱空化案例。该案例的一次完整模拟耗时三个多小时，而小编大概要做几十次模拟才能做完一个实...

神秘色彩

OpenFOAM中

本文内容源自：
usage -- CFD O
OpenFOAM 2.4
也许有改动。#0
界层网格（30&
对数律区）应采用

薛铖

▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ...

知乎 首发于 **有所思**

写下你的评论...

程迪 2018-01-09
所有的动态多态都是这么干的好吧。
赞

Cloud 2018-03-09
原来这叫作动态多态
赞

胡万鹏 2019-08-19
c++的虚函数也是实现动态多态的，Info的实现方法挺像虚函数。C++里的模板就是静态多态了，OpenFOAM也大量使用。
赞

倒霉蛋 04-19
Info不是运算符重载吗，solve不是一个重载的模板函数吗
赞

Micro 10-25
功在当代，利在千秋
赞

▲ 赞同 19 ▼ 5 条评论 分享 喜欢 收藏 申请转载 ...