目录

你已经读了 0 %

OpenFOAM 场（field）的操作和运算

2019-04-01 | code analysis | 字数总计: 1.9k | 阅读时长: 9 分钟

Field **O**peration **A**nd **M**anipulation

field 相关类的结构

几个常见的类：

```c++
volScalarField
volVectorField
surfaceScalarField
surfaceVectorField
```

其实它们都是别名，定义如下：

```c++
typedef GeometricField<scalar, fvPatchField, volMesh> volScalarField;//src\finiteVolume\fields\volFields\volFieldsFwd.H
typedef GeometricField<vector, fvPatchField, volMesh> volVectorField;
typedef GeometricField<scalar, fvsPatchField, surfaceMesh> surfaceScalarField;//src\finiteVolume\fields\surfaceFields\surfaceFieldsFwd.H
typedef GeometricField<vector, fvsPatchField, surfaceMesh> surfaceVectorField;
```

发现，它们实际上都是 `GeometricField`，不过是提供的模板不同。

下面给出 `GeometricField` 类的 UML 类图。首先我们看到它需要三个模板：`<Type,PatchField,GeoMesh>`。结合 `volScalarField` 和 `surfaceScalarField` 的定义，我们不难发现：第一个 `Type` 可以是 `scalar` `vector` 等，第二个 `PatchField` 可以是 `fvPatchField` `fvsPatchField`（这里的 s 表示 surface）等，第三个 `GeoMesh` 可以是 `volMesh` `surfaceMesh` 等。

`GeometricField` 类定义于 `src\OpenFOAM\fields\GeometricFields\GeometricField`。

- `GeometricField` – 场+网格，包含内部场及其边界场，边界场是场的场（FieldField）
- `DimensionedField` – 带单位的场，只有内部场，没有边界场
- `Field` – 数组（即 List）+代数操作

field 相关类的用法

构造

```c++
volScalarField A
(
    IOobject
    (
        "A",
        mesh.time().timeName(),
        mesh, //字典注册对象
        IOobject::MUST_READ, // MUST_READ_IF_MODIFIED  NO_READ  READ_IF_PRESENT
        IOobject::AUTO_WRITE, // NO_WRITE
        true //默认是注册
```

切换站点概览

目录

你已经读了 0 %

1. field 相关类的结构
2. field 相关类的用法
3. 场的操作和运算(Field Operation And Manipulation)

```
    ),
    mesh
)
```

c++

```
volScalarField B
(
    IOobject
    (
        "B",
        mesh.time().timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionedScalar(dimensionSet(1, -3, -1, 0, 0, 0, 0), 0.)
    // 量纲还可以：dimless，或者 dimMass/dimVolume
)
```

上述两种构造函数对应 GeometricField 源代码中的：

c++

```
GeometricField(const IOobject&,const Mesh&);
GeometricField(const IOobject&,const Mesh&,const dimensioned<Type>&,const word&
patchFieldType=PatchField<Type>::calculatedType());
```

第一种是从文件中读取，内部场和边界场的初始值都由文件给定，边界条件也由文件给定。

第二种不需要从文件中读取，内部场和边界场的初始值都是这里给定的 0，边界条件默认是 calculated 。也可以指定边界条件类型，如：

c++

```
volScalarField C
(
    IOobject
    (
        "C",
        mesh.time().timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    mesh,
    dimensionedScalar(dimensionSet(1, -3, -1, 0, 0, 0, 0), 0.),
    zeroGradientFvPatchScalarField::typeName
)
```

下面介绍第三种构造方式，以一个量为蓝本，构造第二个

c++

```
volScalarField D
(
    IOobject
    (
        "D",
        mesh.time().timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    C
)
```

目录

你已经读了 0 %

这个用法很有意思，当你提供了 `D` 时，它就会从文件读取；没提供时，它就会从 `C` 复制。

本人亲测。但是这个用法背后对应的代码有待挖掘。

它使用的是这个构造函数：

```c++

template<class Type, template<class> class PatchField, class GeoMesh>
Foam::GeometricField<Type, PatchField, GeoMesh>::GeometricField
(
    const IOobject& io,
    const GeometricField<Type, PatchField, GeoMesh>& gf
)
:
    Internal(io, gf),
    timeIndex_(gf.timeIndex()),
    field0Ptr_(nullptr),
    fieldPrevIterPtr_(nullptr),
    boundaryField_(*this, gf.boundaryField_)
{
    if (debug)
    {
        InfoInFunction
            << "Constructing as copy resetting IO params"
            << endl << this->info() << endl;
    }

    if (!readIfPresent() && gf.field0Ptr_)
    {
        field0Ptr_ = new GeometricField<Type, PatchField, GeoMesh>
        (
            io.name() + "_0",
            *gf.field0Ptr_
        );
    }
}
```

其中 `Internal` 的是 `DimensionedField` 的别名，调用的这个构造函数：

```c++
template<class Type, class GeoMesh>
DimensionedField<Type, GeoMesh>::DimensionedField
(
    const IOobject& io,
    const DimensionedField<Type, GeoMesh>& df
)
:
    regIOobject(io),
    Field<Type>(df),
    mesh_(df.mesh_),
    dimensions_(df.dimensions_)
{}
```

除了常规的 `IOobject` 构造，还有复制构造：

```c++
tmp<volScalarField> tmagGradP = mag(fvc::grad(p));
volScalarField normalisedGradP
(
    "normalisedGradP",//如果不指定名字，默认是什么?
    tmagGradP()/max(tmagGradP())
);
```

获取

- const

- `Internal& internalField()`，有量纲的内部场
- `Internal::FieldType& primitiveField()`，无量纲的内部场
- `Boundary& boundaryField()`，无量纲的边界场

-non-const

* `ref()`，有量纲的内部场
* `primitiveFieldRef()`，无量纲的内部场
* `boundaryFieldRef()`，无量纲的边界场

用 `forAll` 来遍历的时候，遍历的其实就是那个 `field<Type>` 。

所以总体是有量纲的，用 `forAll` 对每个 cell 或 face 遍历的时候，就失去量纲了。

使用举例：

```c++
    Info<<"T============"<<thermo.T()<<endl; // 有量纲的 内部+边界
    Info<<"T.internalField()============"<<thermo.T().internalField()<<endl; // 有量纲的内部+边界
    Info<<"T.primitiveField()============"<<thermo.T().primitiveField()<<endl; // 无量纲的 内部
    Info<<"T.boundaryField()============"<<thermo.T().boundaryField()<<endl; // 无量纲的边界
    Info<<"T.ref()============"<<thermo.T().ref()<<endl; // 有量纲的 内部+边界
    Info<<"T.primitiveFieldRef()============"<<thermo.T().primitiveFieldRef()<<endl; // 无量纲的 内部
    Info<<"T.boundaryFieldRef()============"<<thermo.T().boundaryFieldRef()<<endl; //无量纲的 边界
```

T文件：

```c++
    dimensions [0 0 0 1 0 0 0];

    internalField uniform 900;

    boundaryField
    {
        walls
        {
            type fixedValue;
            value uniform 1800;
        }
    }
```

输出：

```c++
    T============dimensions      [0 0 0 1 0 0 0];

    internalField   uniform 900;

    boundaryField
    {
        walls
        {
            type            fixedValue;
            value           uniform 1800;
        }
    }

    T.internalField()============dimensions      [0 0 0 1 0 0 0];

    internalField   uniform 900;

```

切换站点概览

目录

你已经读了 0 %

1. field 相关类的结构
2. field 相关类的用法
3. 场的操作和运算(Field Operation And Manipulation)

```
boundaryField
{
    walls
    {
        type            fixedValue;
        value           uniform 1800;
    }
}

T.primitiveField()============8{900}
T.boundaryField()============
1
(
    type            fixedValue;
    value           uniform 1800;

)

T.ref()============dimensions      [0 0 0 1 0 0 0];

internalField   uniform 900;

boundaryField
{
    walls
    {
        type            fixedValue;
        value           uniform 1800;
    }
}

T.primitiveFieldRef()============8{900}
T.boundaryFieldRef()============
1
(
    type            fixedValue;
    value           uniform 1800;

)
```

这里有一个奇怪的现象，即 ref 指的是内部场，但这里输出的确实内部+边界。不要被这里迷惑了，其它地方仍指的是内部场：

c++

```
dimensionedScalar A(dimTemperature, 100.);
thermo.T().ref() = A;
Info<<"T============"<<thermo.T()<<endl;
```

输出：

c++

```
T============dimensions      [0 0 0 1 0 0 0];

internalField   uniform 100;

boundaryField
{
    walls
    {
        type            fixedValue;
        value           uniform 1800;
    }
}
```

场的操作和运算(Field Operation And Manipulation)

切换站点概览

目录

你已经读了 0 %

1. field 相关类的结构
2. field 相关类的用法
3. 场的操作和运算(Field Operation And Manipulation)

IOobject 中定义的函数

- writeOpt

  使用举例：

```c++
mu_.writeOpt() = IOobject::AUTO_WRITE;
```

DimensionedField 中定义的函数

- 平均

```c++
p.average() = gAverage(p)
```

- 加权平均

```c++
p.weightedAverage(weightField) = gSum(weightField*p)/gSum(weightField)
```

使用举例：

```c++
Info<<"average p is "<<p.weightedAverage(mesh.V()).value()<<endl;
//全场压力的体积加权平均值
```

- dimensions

  使用举例：

```c++
Z_.dimensions().reset(dimless);
```

GeometricField 中定义的函数

- max, min

  取最大值，包括内部场和边界场。并行计算时，Info输出的是master的还是全局的？未测试

  使用举例：

```c++
Info<<"T=== "<<max(thermo.T())<<endl;
```

- writeMinMax

  输出最小最大值，只包括内部场。

```c++
template<class Type, template<class> class PatchField, class GeoMesh>
void Foam::GeometricField<Type, PatchField, GeoMesh>::writeMinMax
(
    Ostream& os
) const
{
    os  << "min/max(" << this->name() << ") = "
        << Foam::min(*this).value() << ", "
        << Foam::max(*this).value()
        << endl;
}
```

切换站点概览

目录

你已经读了 0 %

1. field 相关类的结构
2. field 相关类的用法
3. 场的操作和运算(Field Operation And Manipulation)

使用举例：

```c++
Z_.writeMinMax(Info);
```

下面是详细的测试：

T文件：

```c++
dimensions [0 0 0 1 0 0 0];

internalField uniform 900;

boundaryField
{
    walls
    {
        type fixedValue;
        value uniform 1800;
    }
}
```

使用举例：

```c++
Info<<"max(T)=== "<<max(thermo.T())<<endl; //内部+边界
Info<<"min(T)=== "<<min(thermo.T())<<endl; //内部+边界
Info<<"max(T.internalField)=== "<<max(thermo.T().internalField())<<endl; // 内部
Info<<"min(T.internalField)=== "<<min(thermo.T().internalField())<<endl; // 内部
Info<<"max(T.ref)=== "<<max(thermo.T().ref())<<endl; // 内部
Info<<"min(T.ref)=== "<<min(thermo.T().ref())<<endl; // 内部
Info<<"max(T.boundaryField)=== "<<max(thermo.T().boundaryField())<<endl; // 边界
Info<<"min(T.boundaryField)=== "<<min(thermo.T().boundaryField())<<endl; // 边界
thermo.T().writeMinMax(Info); // 内部
```

输出：

```c++
max(T)=== max(T) [0 0 0 1 0 0 0] 1800
min(T)=== min(T) [0 0 0 1 0 0 0] 900
max(T.internalField)=== max(T) [0 0 0 1 0 0 0] 900
min(T.internalField)=== min(T) [0 0 0 1 0 0 0] 900
max(T.ref)=== max(T) [0 0 0 1 0 0 0] 900
min(T.ref)=== min(T) [0 0 0 1 0 0 0] 900
max(T.boundaryField)=== 1800
min(T.boundaryField)=== 1800
min/max(T) = 900, 900
```

- gMax

  max的并行版，但是返回的是无量纲的值?

FieldFunctions

一些底层的函数，一般用不到

DimensionedFieldFunctions

```c++
pow
sqr
magSqr
mag
```

切换站点概览

目录

你已经读了 0 %

1. field 相关类的结构
2. field 相关类的用法
3. 场的操作和运算(Field Operation And Manipulation)

```
cmptAv
gMax
gMin
gSum
gSumMag
gAverage
+
-
*外积
/
^叉乘
&点乘，内积
&& 双内积
```

GeometricFieldFunctions

c++

```
pow
sqr
magSqr
mag
cmptAv
gMax返回的是无量纲的?
gMin
gSum
gSumMag
gAverage
+
-
*外积
/
^叉乘
&点乘，内积
&& 双内积
```

fvc 中的操作

volumeIntegrate

volumeIntegrate 的功能是对每个网格的某个物理量，乘以其网格体积，然后形成一个新的场。

c++

```
volumeIntegrate(df) = df.mesh().V()*df.field();
```

domainIntegrate

domainIntegrate 的功能是对某个物理量，乘以其网格体积，然后对所有网格求和。

c++

```
domainIntegrate(df) = gSum(fvc::volumeIntegrate(df));
```

这里的 gSum 是对全场求和。

其它 fvc 中的操作

c++

```
surfaceIntegrate
surfaceSum
Su
Sp
SuSp
snGrad
reconstruct
laplacian
```

```
9   grad
10  flux
11  div
12  DDt
13  curl
14  average
15  cellReduce
```


field

您的肯定会给我更大动力~

fvMesh 简单分析（待补充）

OpenFOAM 拉格朗日库深度解析

0 条评论 未登录用户

说点什么

支持 Markdown 语法 使用 GitHub 登录 预览

来做第一个留言的人吧!