

Giskard's CFD Learning Tricks

CFD and Scientific Computing

2016-06-25 • OPENFOAM

OpenFOAM 中的热物理类之 hashTable 的创建

这一篇来看一下热物理类是如何编译并创建储存了可选模型的 hashTable 的。

根据以前的经验，编译和构建 hashTable 肯定是跟这个源文件有关：

src/thermophysicalModels/basic/rhoThermo/rhoThermos。

这个文件里，全部都是在调用宏函数。而且，从宏函数的参数来看，似乎就是个排列组合的游戏，把所有可用的组合都写了一遍。

```
1  makeThermo
2  (
3      rhoThermo,
4      heRhoThermo,
5      pureMixture,
6      constTransport,
7      sensibleInternalEnergy,
8      hConstThermo,
9      perfectGas,
10     specie
11 );
12 .....
```

为了弄清这部分内容，需要先理解 `makeThermo` 这个宏函数的定义，见 `src/thermophysicalModels/basic/fluidThermo/makeThermo.H`：

```

1  #define makeThermoTypedefs(BaseThermo,Cthermo,Mixture,Transport,
2
3  typedef
4      Transport
5      <
6          species::thermo
7      <
8          Thermo
9      <
10         EqnOfState
11
12         <
13             Specie
14         >
15     >,
16     Type
17 > Transport##Type##Thermo##EqnOfState##Specie;
18
19 typedef
20     Cthermo
21     <
22         BaseThermo,
23         Mixture<Transport##Type##Thermo##EqnOfState##Specie>
24     > Cthermo##Mixture##Transport##Type##Thermo##EqnOfState##Specie
25
26 defineTemplateNameAndDebugWithName
27 (
28     Cthermo##Mixture##Transport##Type##Thermo##EqnOfState##Specie
29     (
30         #Cthermo"<"#Mixture"<"
31         + Transport##Type##Thermo##EqnOfState##Specie::typeName()
32         + ">>"
33     ).c_str(),
34     0

```

```
35 );
36
37
38 #define makeThermo(BaseThermo,Cthermo,Mixture,Transport,Type,Ther
39
40 makeThermoTypedefs
41 (
42     BaseThermo,
43     Cthermo,
44     Mixture,
45     Transport,
46     Type,
47     Thermo,
48     EqnOfState,
49
50     Specie
51 )
52 addToRunTimeSelectionTable
53 (
54     basicThermo,
55     Cthermo##Mixture##Transport##Type##Thermo##EqnOfState##Specie
56     fvMesh
57 );
58
59 addToRunTimeSelectionTable
60 (
61     fluidThermo,
62     Cthermo##Mixture##Transport##Type##Thermo##EqnOfState##Specie
63     fvMesh
64 );
65
66 addToRunTimeSelectionTable
67 (
68     BaseThermo,
69     Cthermo##Mixture##Transport##Type##Thermo##EqnOfState##Specie
70     fvMesh
71 );
```

可见，在 `makeThermo` 这个宏函数里，先调用了 `makeThermoTypedefs` 宏函数，然后调用 `addToRunTimeSelectionTable` 函数。按照之前对 RTS 机制的理解，调用 `addToRunTimeSelectionTable` 函数的作用是往 `hashTable` 里插入元素，细节不需再赘述。这里主要来看看 `makeThermoTypedefs` 函数的功能，以上文列举的这个实例为例。先来看第一个 typedef：将实例中的参数代入后，

```
1  typedef
2      constTransport<species::thermo<hConstThermo<perfectGas<specie:
3      constTransportsensibleEnthalpyhConstThermoperfectGasspecie;
```

第二个 typedef

```
1  typedef
2      heRhoThermo
3      <
4      rhoThermo,
5      pureMixture<constTransportsensibleEnthalpyhConstThermoper
6      > heRhoThermopureMixtureconstTransportsensibleEnthalpyhConstTI
```

除了这两个 typedef，还调用了 `defineTemplateNameAndDebugWithName` 宏函数，这个函数的定义在 `src/OpenFOAM/db/typeInfo/className.H`：

```
1  #define defineTemplateNameAndDebugWithName(Type, Name, DebugS
2      defineTemplateNameWithName(Type, Name);
3      defineTemplateDebugSwitchWithName(Type, Name, DebugSwitch)
4
5  #define defineTemplateNameWithName(Type, Name)
6      defineTypeNameWithName(Type, Name)
7
8  #define defineTypeNameWithName(Type, Name)
9      const ::Foam::word Type::typeName(Name)
```

很显然，这个宏函数的作用是修改类对应的 `typeName` 和 `debug` 选项。在 OpenFOAM 中，很多类中都会调用 `TypeName("typename")`，这里的 `TypeName` 也是一个宏函数，定义在 `src/OpenFOAM/db/typeInfo/className.H`：

```

1  #define TypeName(TypeNameString)
2      ClassName(TypeNameString);
3      virtual const word& type() const { return typeName; }
4
5  #define ClassName(TypeNameString)
6      ClassNameNoDebug(TypeNameString);
7
8      static int debug
9
10 #define ClassNameNoDebug(TypeNameString)
11     static const char* typeName_() { return TypeNameString; }
12     static const ::Foam::word typeName

```

可见，`TypeName` 这个宏函数，声明了一个类静态变量 `typeName`，定义了一个函数 `type` 用于返回 `typeName` 的值，并定义了一个静态变量 `debug` 用于存储 `debug` 选项。这里与 RTS 机制有关的是 `typeName` 这个变量。

绕了半圈，回到 `defineTemplateNameAndDebugWithName` 函数。了解了 `typeName` 这个变量的定义，很容易就能看出来，`defineTemplateNameAndDebugWithName` 这个函数其实就是在对类的静态变量 `typeName` 进行赋值。根据上文的实例提供的参数，宏函数 `defineTemplateNameAndDebugWithName` 可以理解为：对 `heRhoThermopureMixtureconstTransportsensibleEnthalpyhConstThermoperfectGas` 对应的类的静态变量 `typeName` 进行赋值，赋值结果为：

```

heRhoThermo<pureMixture< +
constTransport<species::thermo<hConstThermo<perfectGas<specie>>,sensibleI
+ >>

```

。这里调用了 `constTransport` 类的成员函数 `typeName()`。

经过一番冗长的函数调用，得到的最终结果是：将

`heRhoThermopureMixtureconstTransportsensibleEnthalpyhConstThermoperfectGas` 对应的类的静态变量 `typeName` 赋值为：

```

heRhoThermo<pureMixture<const<hConst<perfectGas<specie>>,sensible

```

至此，经过一番宏函数的调用，得到了 `addToRunTimeSelectionTable` 宏函数的参数。前面 RTS 机制部分讲过，这个函数的作用就是对 hashTable 增加元素，以

```

1  addToRunTimeSelectionTable
2  (
3      BaseThermo,
4      Cthermo##Mixture##Transport##Type##Thermo##EqnOfState##Specie
5      fvMesh
6  );

```

为例，第一个参数，表示元素将增加到 `BaseThermo` 类（这里是 `rhoThermo`）中声明的 hashTable，第二个参数，表示将要添加的类，添加成功以后，这个类的 `typeName` 将是 hashTable 的 key，而返回这个类的对象的一个函数，将是 hashTable 的 value。第三个参数对应着 hashTable 对象的名字，`fvMesh` 对应的 hashTable 对象名为 `fvMeshConstructorTable`，这与在 `rhoThermo` 中声明的名字是对应的。

最后总结如下：

宏函数

```

1  makeThermo
2  (
3      rhoThermo,
4      heRhoThermo,
5      pureMixture,
6      constTransport,
7      sensibleInternalEnergy,
8      hConstThermo,
9      perfectGas,
10     specie
11 );

```

调用以后，向 `rhoThermo` 类中声明的 hashTable 中增加了一组元素，其 key 为 `heRhoThermo<pureMixture<const<hConst<perfectGas<specie>>,sensibleInternalEnergy>>>`，value 对应的函数返回的是类

```

1  heRhoThermo
2  <
3      rhoThermo,
4      pureMixture
5  <
6      constTransport<species::thermo<hConstThermo<perfectGas<sp
7  >
8  >

```

的对象。

每调用一次 `makeThermo` 函数，就增加了一个新组元素，也即增加了一个可选的模型。不同的参数，其实对应的是不同的模板实例。

至此，就知道了在 `twoPhaseEulerFoam` 的 `phaseModel` 中定义的热物理类接口 `thermo_` 最终指向的是 `heRhoThermo` 类的对象。虽然代入的模板数很复杂，但整个架构仍然是基于 RTS 机制的。

接下来，要想理解能量方程，理解温度，粘度，压力等这些热物理相关的量是怎么计算更新的，就需要仔细看一下 `heRhoThermo` 类的继承派生关系了。

#Code Explained #thermophysicalModels

 Share

NEWER

OpenFOAM 中的热物理类之继承派生关系

OLDER

OpenFOAM 中的热物理类之接口

CATEGORIES

[C++](#) (2)
[DEM](#) (1)
[OpenFOAM](#) (44)
[Paraview](#) (5)
[swak4Foam](#) (1)
[test](#) (2)
[vim](#) (1)

TAGS

[Boundary conditions](#) (6)
[C++](#) (2)
[CentOS](#) (1)
[Code Explained](#) (29)
[LES](#) (1)
[LIGGGHTS](#) (1)
[ODE](#) (1)
[OpenFOAM](#) (20)
[Postprocessing](#) (9)
[Preprocessing](#) (2)
[RTS](#) (3)
[TIL](#) (1)
[Windows](#) (1)
[fvOptions](#) (2)
[groovyBC](#) (1)
[paraview](#) (1)
[test](#) (2)
[thermophysicalModels](#) (5)
[turbulence model](#) (7)
[vim](#) (1)
[wall functions](#) (4)

TAG CLOUD

[Boundary conditions](#) [C++](#) [CentOS](#) [Code Explained](#) [LES](#) [LIGGGHTS](#) [ODE](#) [OpenFOAM](#) [Postprocessing](#) [Preprocessing](#) [RTS](#) [TIL](#) [Windows](#) [fvOptions](#) [groovyBC](#) [paraview](#) [test](#) [thermophysicalModels](#) [turbulence model](#) [vim](#) [wall functions](#)

ARCHIVES

[五月 2017](#) (1)
[八月 2016](#) (8)
[六月 2016](#) (5)
[五月 2016](#) (3)
[四月 2016](#) (12)
[三月 2016](#) (6)
[十二月 2015](#) (1)
[十一月 2015](#) (2)
[十月 2015](#) (1)
[九月 2015](#) (6)
[八月 2015](#) (1)
[六月 2015](#) (2)
[五月 2015](#) (6)
[四月 2015](#) (2)

RECENTS

[多说评论系统将停止提供服务](#)
[Paraview 脚本一例](#)
[Paraview 中有关 Camera 的操作两例](#)
[Paraview 中创建 Custom Filter](#)
[在 Paraview 中画截面上的流线](#)

© 2017 Giskard Q.

Powered by [Hexo](#)