切换站点概览

目录

你已经读了 95 %

1. OF-7
2. 求解器中创建对象
3. RTS 选择热物理模型
4. RTS 库的生成
5. OF-8

OpenFOAM 热物理库深度解析

2018-03-03 | code analysis | 字数总计: 2.5k | 阅读时长: 10 分钟

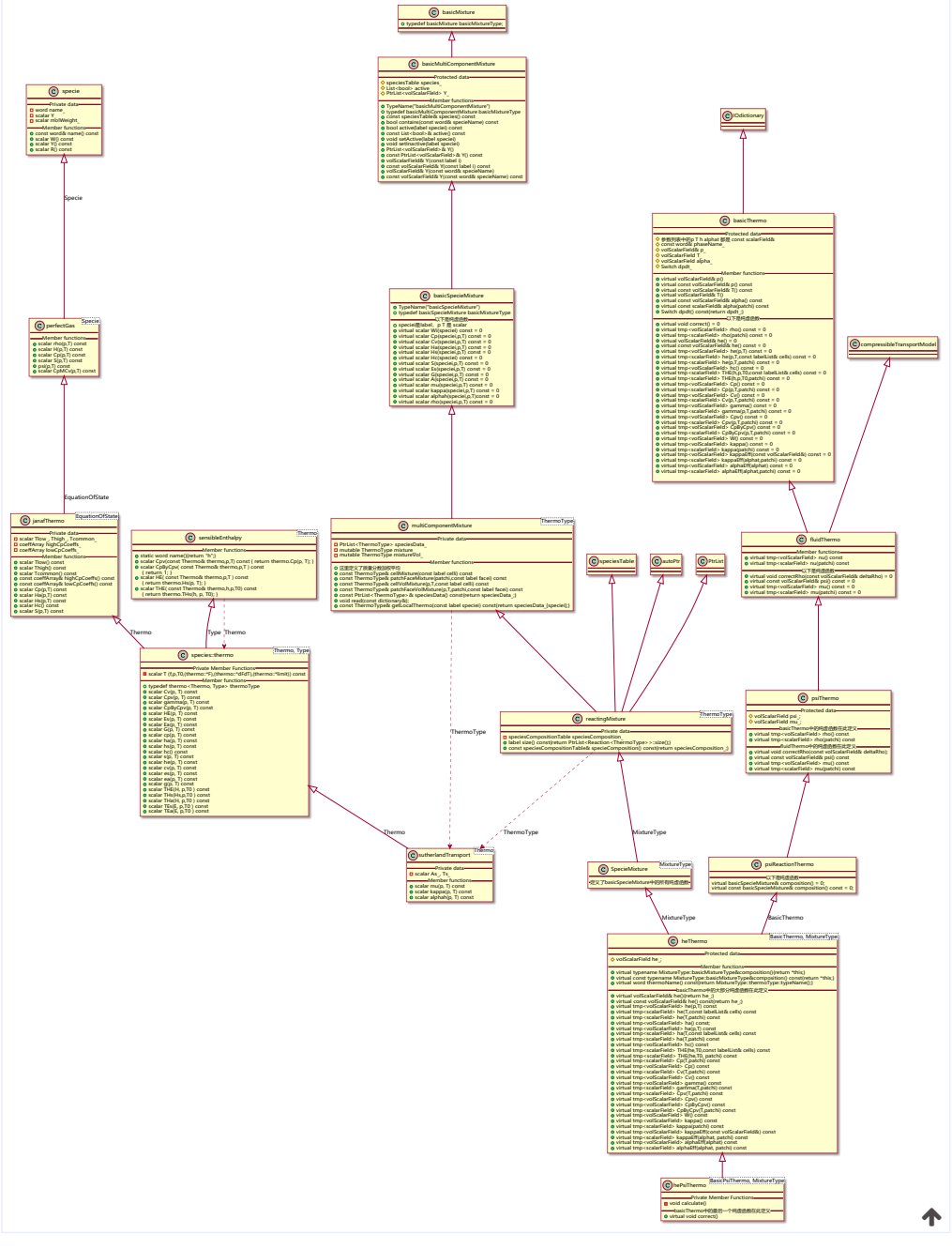
本帖描述 OpenFOAM 热物理库的框架结构，希望可以帮到你。

OF-7

UML 类图

下图是热物理库部分类的 UML 类图。欲获取更佳阅读体验，可以右键在新标签页打开该图。

▼ OF-7 UML



目录

solver 中的 creatFields.H 中：

```c++
autoPtr<psiReactionThermo> pThermo(psiReactionThermo::New(mesh));
psiReactionThermo& thermo = pThermo();
```

创建 thermo，在 reactingFoam 自带算例中，创建的是 hePsiThermo 的对象，但是这里 thermo 是基类 psiReactionThermo 的引用，即只能使用 psiReactionThermo 这一继承链的方法。

```c++
basicSpecieMixture& composition = thermo.composition();
```

创建 composition，是指系统中的混合物。可以调用继承链中 mixture 这一分支的方法。

燃烧类的求解器中，一般都没有对单个组分进行操作，这些操作都在热物理库里边进行。

multiComponentMixture 类中的私有数据 speciesData_ 的类型是 PtrList<sutherlandTransport<species::thermo<janafThermo<perfectGas<specie>>,sensibleEnthalpy>>>，它是组分的 list，可以用它获取组分的所有信息。solver 中可以通过 multiComponentMixture（solver 中 composition 的类型是 basicSpecieMixture，需要强制转换成 multiComponentMixture 才行）的 speciesData() 或者 getLocalThermo(speciei) 这两个函数来调用。如：

```c++
dynamic_cast<const multiComponentMixture<gasHThermoPhysics>&>
(composition).speciesData()[speciei].rho(p,T)
```

也可以通过 multiComponentMixture 的派生类 reactingMixture 来调用。

```c++
dynamic_cast<const reactingMixture<gasHThermoPhysics>&>(composition).speciesData()
[speciei].rho(p,T)
```

也可以通过派生类的派生类 SpecieMixture 来调用。

```c++
dynamic_cast<const SpecieMixture<reactingMixture<gasHThermoPhysics>>&>
(composition).speciesData()[speciei].rho(p,T)
```

上述操作说明可以使用组分列表调用单个组分的函数（即 sutherlandTransport<species::thermo<janafThermo<perfectGas<specie>>,sensibleEnthalpy>> 继承链中的任何函数），如 speciesData_[i].rho(p, T)，这里的 rho(p,T) 来自于 perfectGas 类。

也可以在求解器中重新构造组分列表：

```c++
autoPtr<psiReactionThermo> pThermo(psiReactionThermo::New(mesh));
psiReactionThermo& thermo = pThermo();

basicSpecieMixture& composition = thermo.composition();
```

```
8   forAll(specieData, i)
9   {
10  //对于每个组分，通过 thermo 调用 Lib 中的 speciesData_ 来构造 solver 中的 specieData
11      specieData.set
12      (
13          i,
14          new gasHThermoPhysics
15          (
16              dynamic_cast<const reactingMixture<gasHThermoPhysics>&>
17                  (thermo).speciesData()[i]
18          )
19      );
20  }
21
22  // 每个网格构造一个 mixture,这个 mixture 相当于 multiComponentMixture 中的私有数据
mixture_。
23  // 它是混合物等效成的一个组分，和单个组分同一类型，即 `gasHThermoPhysics`。
24  forAll(T, celli)
25  {
26      gasHThermoPhysics mixture(0.*specieData[0]);
27      forAll(Y, speciei)
28      {
29          mixture += Y[speciei][celli]*specieData[speciei];
30      }
31  }
```

gasHThermoPhysics 是 sutherlandTransport<species::thermo<janafThermo<perfectGas<specie>>,sensibleEnthalpy>> 的别名，它是单个组分的类型。

事实上， multiComponentMixture 类中的函数 cellMixture 函数可以用于获取 mixture_ 。

c++

```
1   template<class ThermoType>
2   const ThermoType& Foam::multiComponentMixture<ThermoType>::cellMixture
3   (
4       const label celli
5   ) const
6   {
7       mixture_ = Y_[0][celli]*speciesData_[0];
8
9       for (label n=1; n<Y_.size(); n++)
10      {
11          mixture_ += Y_[n][celli]*speciesData_[n];
12      }
13
14      return mixture_;
15  }
```

使用方法参考 hePsiThermo.C

c++

```
1   const scalarField& hCells = this->he_;
2   const scalarField& pCells = this->p_;
3
4   scalarField& TCells = this->T_.primitiveFieldRef();
5   scalarField& psiCells = this->psi_.primitiveFieldRef();
6   scalarField& muCells = this->mu_.primitiveFieldRef();
7   scalarField& alphaCells = this->alpha_.primitiveFieldRef();
```

```
    const typename MixtureType::thermoType& mixture_ =
        this->cellMixture(celli);

    TCells[celli] = mixture_.THE
    (
        hCells[celli],
        pCells[celli],
        TCells[celli]
    );

    psiCells[celli] = mixture_.psi(pCells[celli], TCells[celli]);

    muCells[celli] = mixture_.mu(pCells[celli], TCells[celli]);
    alphaCells[celli] = mixture_.alphah(pCells[celli], TCells[celli]);
}
```

RTS 选择热物理模型

从字典文件的设置开始

下面介绍我们是如何从 `thermophysicalProperties` 字典文件中的下列设置构建对象的：

c++

```
thermoType
{
    type            hePsiThermo;
    mixture         reactingMixture;
    transport       sutherland;
    thermo          janaf;
    energy          sensibleEnthalpy;
    equationOfState perfectGas;
    specie          specie;
}
```

solver 中创建模型：

c++

```
autoPtr<psiReactionThermo> pThermo(psiReactionThermo::New(mesh));
```

这个函数定义于 `psiReactionThermo`：

c++

```
Foam::autoPtr<Foam::psiReactionThermo> Foam::psiReactionThermo::New
(
    const fvMesh& mesh,
    const word& phaseName
)
{
    return basicThermo::New<psiReactionThermo>(mesh, phaseName);
}
```

它调用了 `basicThermo` 中的 `New` 函数：

c++

```
3   {
4       // IOdictionary thermoDict---thermophysicalProperties
5       typename Thermo::fvMeshConstructorTable::iterator cstrIter =
6           lookupThermo<Thermo, typename Thermo::fvMeshConstructorTable>
7           (
8               thermoDict,
9               Thermo::fvMeshConstructorTablePtr_
10          );
11  //fvMeshConstructorTablePtr_
12  //这个是在 runTimeSelectionTables.H 的 declareRunTimeSelectionTable 函数中定义的。
13      return autoPtr<Thermo>(cstrIter()(mesh, phaseName));
14  }
```

调用了 2 参数的 lookupThermo：

basicThermoTemplates.C 中的 `lookupThermo` 函数：

c++

```
1   template<class Thermo, class Table>
2   typename Table::iterator Foam::basicThermo::lookupThermo
3   (
4       const dictionary& thermoDict,
5       Table* tablePtr    //即 fvMeshConstructorTablePtr_，它是哈希表add类型的一个指针，
typeName和模型都存放在这里
6   )
7   {
8       const word thermoTypeName
9       (
10      word(thermoTypeDict.lookup("type")) + '<'
11      + word(thermoTypeDict.lookup("mixture")) + '<'
12      + word(thermoTypeDict.lookup("transport")) + '<'
13      + word(thermoTypeDict.lookup("thermo")) + '<'
14      + word(thermoTypeDict.lookup("equationOfState")) + '<'
15      + word(thermoTypeDict.lookup("specie")) + ">>,"
16      + word(thermoTypeDict.lookup("energy")) + ">>>"
17      );
18  //通过字典文件读入的数据，构造 thermoTypeName:
//hePsiThermo<reactingMixture<sutherland<janaf<perfectGas<specie>>,sensibleEnthalpy>>>
19  //注意这里并不是类名，而是省略版
20
21      return lookupThermo<Thermo, Table>
22      (
23      thermoTypeDict,
24      tablePtr,
25      nCmpt,
26      cmptNames,
27      thermoTypeName
28      );
29  }
```

最后返回值是调用 5 个参数的 `lookupThermo` 函数：

c++

```
1   template<class Thermo, class Table>
2   typename Table::iterator Foam::basicThermo::lookupThermo
3   (
4       const dictionary& thermoTypeDict,
5       Table* tablePtr,
6       const int nCmpt,
7       const char* cmptNames[],
```

```
11      // Lookup the thermo package
12      typename Table::iterator cstrIter = tablePtr->find(thermoTypeName);
13      return cstrIter;
14  }
15  //上述步骤是根据字典文件在备选库中选择我们需要的组合，那么备选库是怎么形成的呢?
```

第一个 lookupThermo 函数的作用是，将 thermophysicalProperties 字典文件中的设置翻译成：

c++

```
1
hePsiThermo<reactingMixture<sutherland<janaf<perfectGas<specie>>,sensibleEnthalpy>>>
```

但该字符串并不是最终的 class，比如 janaf，实际上类名是 janafThermo。

第二个 lookupThermo 函数的作用就是，根据该字符串，在 RTS 库中找到我们选择的类。

那么上面提到的省略版字符串，怎么和 RTS 库中的组合匹配的呢？下面来看 RTS 库的生成过程。

RTS 库的生成

按照Giskard 博客提到的 RTS 流程：

1. 基类类体里调用 TypeName 和 declareRunTimeSelectionTable 两个函数，类体外面调用 defineTypeNameAndDebug 和 defineRunTimeSelectionTable 两个函数。
2. 基类中需要一个静态 New 函数作为 selector。
3. 派生类类体中需要调用 TypeName 函数，类体外调用 defineRunTimeSelectionTable 和 addToRunTimeSelectionTable 两个宏函数。

我们发现： makeReactionThermo.H 中有两个重要函数： defineThermoPhysicsReactionThermo 和 makeThermoPhysicsReactionThermos。

下面一一介绍：

defineThermoPhysicsReactionThermo

其中的 defineThermoPhysicsReactionThermo，这个函数通过一组形参列表得到 CThermo##Mixture##ThermoPhys，然后再转换成一串简化的字符串。这里的形参列表就是实例化所需要提供的各种类，对应 (psiReactionThermo,hePsiThermo,reactingMixture,sutherlandTransport)。

c++

```
1   #define
defineThermoPhysicsReactionThermo(BaseReactionThermo,CThermo,Mixture,ThermoPhys) \
2       typedef CThermo//hePsiThermo
\
3       <
\
4           BaseReactionThermo,//psiReactionThermo
\
5           SpecieMixture
\
6           <
\
7               Mixture//reactingMixture
```

目录


```
9                   ThermoPhys//sutherlandTransport
\
10              >
\
11          >
\
12      > CThermo##Mixture##ThermoPhys;
\
13
\
14      defineTemplateTypeNameAndDebugWithName
\
15      (
\
16          CThermo##Mixture##ThermoPhys,
\
17          (#CThermo"<" + Mixture<ThermoPhys>::typeName() + ">").c_str(),
\
18          0
\
19      )
20  //作用：把尖括号表示的类变成字符串 CThermo##Mixture##ThermoPhys，然后再变成简化的尖括号。
```

如：

c++

```
1   hePsiThermo
2   <
3   psiReactionThermo,
4       SpecieMixture
5       <
6           reactingMixture
7           <
8               sutherlandTransport
9               <
10                  species::thermo
11                  <
12                      janafThermo<perfectGas<specie>>,sensibleEnthalpy
13                  >
14              >
15          >
16      >
17  >
```

变成 RTS 库中保存的字符串：

`hePsiThermo<reactingMixture<sutherland<janaf<perfectGas<specie>>,sensibleEnthalpy>>>`

隐藏了一些信息，如 psiReactionThermo 和 SpecieMixture 。

具体过程：

typeName() 依次调用 reactingMixture， sutherlandTransport，species::thermo，janafThermo，sensibleEnthalpy，perfectGas，specie 的 typeName()。返回值依次是：

plain

```
1   "reactingMixture<" + ThermoType::typeName() + '>'
2   sutherland<" + Thermo::typeName() + '>'
3   Thermo::typeName() + ',' + Type<thermo<Thermo, Type>>::typeName()
4   "janaf<" + EquationOfState::typeName() + '>'
```

最底层的 specie 类只有一个 ClassName("specie")；

我们找到一个文件 className.H 中：

```c++
#define ClassName(TypeNameString)                                     \
    ClassNameNoDebug(TypeNameString);                                 \
    static int debug

#define ClassNameNoDebug(TypeNameString)                              \
    static const char* typeName_() { return TypeNameString; } \
    static const ::Foam::word typeName
```

所以这里的意义就在于用简化版的字符串表示完整的类型。而简化版的字符串则是读取 thermophysicalProperties 之后稍作处理得到的。（在前面的2 参数的 lookupThermo 中）

makeThermoPhysicsReactionThermos

定义如下：

```c++
#define
makeThermoPhysicsReactionThermos(BaseThermo,BaseReactionThermo,CThermo,Mixture,ThermoPhys)
    // 定义于 makeReactionThermo.H
        defineThermoPhysicsReactionThermo(BaseReactionThermo,CThermo,Mixture,ThermoPhys);

        addThermoPhysicsThermo(basicThermo, CThermo##Mixture##ThermoPhys);
        addThermoPhysicsThermo(fluidThermo, CThermo##Mixture##ThermoPhys);
        addThermoPhysicsThermo(BaseThermo, CThermo##Mixture##ThermoPhys);
        addThermoPhysicsThermo(BaseReactionThermo, CThermo##Mixture##ThermoPhys)
```

调用了上边的 defineThermoPhysicsReactionThermo 用于创建 RTS 中的 typeName，调用了 addThermoPhysicsThermo （位于 makeThermo.H ），用于写入到 RTS 哈希表。

makeThermoPhysicsReactionThermos 函数的形参：

(BaseThermo,BaseReactionThermo,CThermo,Mixture,ThermoPhys)，这是实例化需要提供的所有信息。

比如其中的一种组合：

makeThermoPhysicsReactionThermos(psiThermo,psiReactionThermo,hePsiThermo,reactingMixture,gasHThermoPhysics) （所有的组合在 psiReactionThermos.C 中列出）。

这里调用 4 个 addThermoPhysicsThermo 函数是为何？假定先分析第一个 addThermoPhysicsThermo 函数：

```c++
#define addThermoPhysicsThermo(BaseThermo_,CThermoMixtureThermoPhys_) //定义于
makeThermo.H
        addToRunTimeSelectionTable
        (
            BaseThermo_,//基类名
            CThermoMixtureThermoPhys_,//派生类名
            fvMesh
        );
```

BaseThermo_ 是 basicThermo， CThermoMixtureThermoPhys_ 是 hePsiThermo##reactingMixture##gasHThermoPhysics 。

```c++
#define addToRunTimeSelectionTable(baseType,thisType,argNames)
    baseType::add##argNames##ConstructorToTable<thisType>
        add##thisType##argNames##ConstructorTo##baseType##Table_
```

addfvMeshConstructorToTable 这个类是在 runTimeSelectionTables.H 文件中的 declareRunTimeSelectionTable 宏函数中定义的。

所以 addThermoPhysicsThermo 中创建了一个 baseType::addfvMeshConstructorToTable<thisType> 类的对象 add##thisType##fvMesh##ConstructorTo##baseType##Table_ ，而该对象构造时会往哈希表中插入键和值，即派生类的名字和 New 函数，此函数返回指向派生类对象的基类指针。
这里的基类 baseType 是 basicThermo ，派生类 thisType 是 hePsiThermo##reactingMixture##gasHThermoPhysics ，这里是省略版。

OF-8

OF-8 去除了 reactingMixture 这个类!
它的成员 speciesComposition_ 移至 multiComponentMixture 中。
另外 multiComponentMixture 还新增了 readSpeciesData 和 readSpeciesComposition 这两个函数，用于初始化 specieThermos_ 和 speciesComposition_ 。

之前（OF-7）化学反应的信息是保存在 reactingMixture 中，然后复制到 StandardChemistryModel 中的 reactions_
现在（OF-8）是直接保存在 StandardChemistryModel 中的 reactions_ 。

化学反应的信息在 StandardChemistryModel 中，以前（OF-7）是

```c++
const PtrList<Reaction<ThermoType>>& reactions_;
```

以前（OF-7）是下面这样，调用的是复制构造函数，因为所有的化学反应的信息都包含在 reactingMixture 中!

```c++
reactions_
(
    dynamic_cast<const reactingMixture<ThermoType>&>(this->thermo())
)
```

现在（OF-8）是

```c++
const ReactionList<ThermoType> reactions_;
```

现在在 StandardChemistryModel 中是这样构造:

```c++
reactions_
(
```

```
6       ).species(),
7       specieThermo_,
8       this->mesh(),
9       *this
10  )
```

调用的是这个构造函数：

c++

```
1   //- Construct from thermo list, objectRegistry and dictionary
2   ReactionList
3   (
4       const speciesTable& species,
5       const PtrList<ThermoType>& speciesThermo,
6       const objectRegistry& ob,
7       const dictionary& dict
8   );basicThermo
```

下图是热物理库部分类的 UML 类图。欲获取更佳阅读体验，可以右键在新标签页打开该图。

▶ OF-8 UML

thermophysicalModels RTS

您的肯定会给我更大动力~

使用 qtcreator 对 OpenFOAM 代码 debug

OpenFOAM 如何从焓得到温度

切换站点概览

目录

你已经读了 95 %

1. OF-7
2. 求解器中创建对象
3. RTS 选择热物理模型
4. RTS 库的生成
5. OF-8

OpenFOAM 成长之路

太棒了