OpenFOAM 中的运行时选择机制

2017-05-31 · OpenFOAM · 1960 words · 4 mins read · 12880 times read

运行时选择（run-time selection，RTS）机制是 OpenFOAM 的一大特点。RTS 做的工作其实很简单：通过运行时（从字典读入）的不同关键字构造不同对象，这些构造的对象类都派生自同一父类，具有一组相同的接口。RTS 的实现涉及到工厂模式（factory pattern）、哈希表（hash table）和宏（macro），本文将从这三个方面介绍 RTS 的实现原理。

工厂模式

工厂模式是一种设计模式（design pattern），是用来提高代码可维护性的一种技巧。常见的工厂模式有三类：简单工厂（simple factory）、工厂方法（factory method）和抽象工厂（abstract factory）。OpenFOAM 用的是简单工厂，即在工厂类中提供一个静态函数用来构造所需对象并返回，这个静态函数一般叫 New，OpenFOAM 称之为 Selector。所有涉及到 RTS 的工厂类（父类）中都可以找到这个函数。

传统的简单工厂模式有个比较严重的缺陷：在构造所需对象的时候，所有的可构造对象只能是事先考虑到的，如果需要添加对象类，则需要修改工厂。这样不利于代码的维护。OpenFOAM 使用哈希表构造了一个 constructor table，将可构造的对象类存储在这个哈希表中，从而避免了新增具体产品类时对工厂类的修改。

以时间离散格式类 `ddtScheme` 为例说明，相关代码：

```
src/finiteVolume/finiteVolume/ddtSchemes/ddtScheme/ddtScheme.H
src/finiteVolume/finiteVolume/ddtSchemes/ddtScheme/ddtScheme.C
src/finiteVolume/finiteVolume/ddtSchemes/ddtScheme/ddtSchemes.H
```

`ddtScheme::New` 的实现如下：

C++

```C++
template<class Type>
tmp<ddtScheme<Type> > ddtScheme<Type>::New
(
    const fvMesh& mesh,
    Istream& schemeData
)
{
    if (fv::debug)
    {
        Info<< "ddtScheme<Type>::New(const fvMesh&, Istream&) : "
               "constructing ddtScheme<Type>"
            << endl;
    }

    if (schemeData.eof())
    {
        FatalIOErrorIn
        (
            "ddtScheme<Type>::New(const fvMesh&, Istream&)",
            schemeData
        )   << "Ddt scheme not specified" << endl << endl
```

```
            << "Valid ddt schemes are :" << endl
            << IstreamConstructorTablePtr_->sortedToc()
            << exit(FatalIOError);
    }

    const word schemeName(schemeData);

    typename IstreamConstructorTable::iterator cstrIter =
        IstreamConstructorTablePtr_->find(schemeName);

    if (cstrIter == IstreamConstructorTablePtr_->end())
    {
        FatalIOErrorIn
        (
            "ddtScheme<Type>::New(const fvMesh&, Istream&)",
            schemeData
        )   << "Unknown ddt scheme " << schemeName << nl << nl
            << "Valid ddt schemes are :" << endl
            << IstreamConstructorTablePtr_->sortedToc()
            << exit(FatalIOError);
    }

    return cstrIter()(mesh, schemeData);
}
```

`New` 函数从哈希表中通过关键字 schemeName 查找与之对应的具体时间离散格式（如 Euler、backward 等）的构造函数，并返回其所构造的对象。这样就通过简单工厂实现了 RTS 的基本功能。

哈希表

哈希表的本质是一张 key -> value 键值对映射表，表中的 key 和 value 一一对应，可以通过 key 访问 value。OpenFOAM 使用哈希表用来维护工厂类可构造的具体产品类，这个哈希表使用类名作为 key，使用一系列静态函数的指针作为 value 。这样就可以通过类名调用其对应的静态函数构造并返回相应的对象。我们先看哈希表声明的相关代码：

C++

```
template<class Type>
class ddtScheme
:
    public refCount
{
    ...

public:

    ...

    /* Construct from argList function pointer type */
    typedef autoPtr<ddtScheme> (*IstreamConstructorPtr)(const fvMesh& mesh, Istream& schemeData);

    /* Construct from argList function table type */
    typedef HashTable<IstreamConstructorPtr, word, string::hash> IstreamConstructorTable;

    /* Construct from argList function pointer table pointer */
    static IstreamConstructorTable* IstreamConstructorTablePtr_;

    /* Table constructor called from the table add function */
    static void constructIstreamConstructorTables();

    /* Table destructor called from the table add function destructor */
    static void destroyIstreamConstructorTables();

    /* Class to add constructor from argList to table */
    template<class ddtSchemeType>
```

```
    class addIstreamConstructorToTable
    {
    public:

        static autoPtr<ddtScheme> New(const fvMesh& mesh, Istream& schemeData)
        {
            return autoPtr<ddtScheme>(new ddtSchemeType(mesh, schemeData));
        }

        addIstreamConstructorToTable
        (
            const word& lookup = ddtSchemeType::typeName;
        )
        {
            constructIstreamConstructorTables();
            if (!IstreamConstructorTablePtr_->insert(lookup, New))
            {
                std::cerr<< "Duplicate entry " << lookup
                    << " in runtime selection table " << "ddtScheme"
                    << std::endl;
                error::safePrintStack(std::cerr);
            }
        }

        ~addIstreamConstructorToTable()
        {
            destroyIstreamConstructorTables();
        }
    };

    ...
```

这里定义了一个哈希表 `IstreamConstructorTablePtr_` 用来存储类名-函数指针的键值对 。同时声明了一个 `addIstreamConstructorToTable` 类模板，类模板中声明了一个静态函数 `New` ，该函数将返回新构造的具体时间离散格式（如 Euler、backward等）实例。类的构造函数和析构函数分别调用了 `constructIstreamConstructorTables` 和 `destroyIstreamConstructorTables` 这两个函数，我们看其实现：

C++

```
    /* Define the constructor function table */
    template<>
    ddtScheme<scalar>::IstreamConstructorTable*
        ddtScheme<scalar>::IstreamConstructorTablePtr_ = NULL;

    /* Table constructor called from the table add function */
    template<>
    void ddtScheme<scalar>::constructIstreamConstructorTables()
    {
        static bool constructed = false;
        if (!constructed)
        {
            constructed = true;
            ddtScheme<scalar>::IstreamConstructorTablePtr_
                = new ddtScheme<scalar>::IstreamConstructorTable;
        }
    }

    /* Table destructor called from the table add function destructor */
    template<>
    ddtScheme<scalar>::destroyIstreamConstructorTables()
    {
        if (ddtScheme<scalar>::IstreamConstructorTablePtr_)
        {
            delete ddtScheme<scalar>::IstreamConstructorTablePtr_;
            ddtScheme<scalar>::IstreamConstructorTablePtr_ = NULL;
        }
```

```
    }

    ...
```

这里用到了 C++ 的模板特化（template specialization），将模板参数特化为 scalar、vector 等绝对类型。上面代码列举了模板参数特化为 scalar 类型的实现， `constructIstreamConstructorTables` 函数的作用是如果不存在则构造一个哈希表， `destroyIstreamConstructorTables` 函数的作用是如果存在则销毁哈希表。

有了哈希表，接下来需要向哈希表中添加记录，这些添加操作对应的代码在各具体产品类（子类）中可以找到。以 Euler 格式为例：

C++

```
ddtScheme<scalar>::addIstreamConstructorToTable<EulerDdtScheme<scalar> >
    addEulerScalarIstreamConstructorToTable_;
```

这里定义了一个 `addIstreamConstructorToTable<EulerDdtScheme<scalar>>` 类型的变量，其构造函数首先调用 `constructIstreamConstructorTables` 构造哈希表，再使用 `HashTable::insert` 方法往表中添加记录，这些记录用类名 `ddtSchemeType::typeName` 作为 key，用对应类的 New 函数的函数指针作为 value。在所有的子类中添加上述代码，可以把所有子类的记录都添加到哈希表中。

宏

上面提到的工厂模式和哈希表已经实现了 RTS 的所有功能，但是我们看到对于每个 RTS 类型（如 `gradSchemes` 、 `divSchemes` 等）都需要增加大量的代码，这些代码结构几乎没有区别。OpenFOAM 采用宏定义来压缩这些代码，相关代码：

```
src/OpenFOAM/db/runTimeSelection/construction/addToRunTimeSelectionTable.H
src/OpenFOAM/db/runTimeSelection/construction/runTimeSelectionTables.H
```

比如上面关于哈希表的代码都可以用 `declareRunTimeSelectionTable` 、 `defineTemplateRunTimeSelectionTable` 以及 `makeFvDdtTypeScheme` 等宏定义展开，具体展开过程这里不再赘述。

Author : wwzhao
LastMod : 2017-05-31

#OpenFOAM #runtime selection #ddt scheme

‹ OpenFOAM 中的有限体积离散　　OpenFOAM 中的 autoPtr ›

在 MARINECFD 上还有

OpenFOAM 中的对象注册机制

9 个月前 • 4条评论

对象注册（object registry）机制是 OpenFOAM …

OpenFOAM 中源项的实现

1 年前 • 6条评论

考虑如下所示的关于 ψ 的输运方程： \[\begin{equation} …

OpenFOAM …

2 年前 • 2条评论

背景 在 OpenFOAM 2.3.0 时，Henry G. Weller …

OpenFOAM 中的类

2 年前 • 1条评论

tmp 类是 OpenFOAM 来封装对象的一个类 将介绍 tmp …

realRabbita • 8 个月前

如果向哈希表添加记录addEulerScalarIstreamConstructorToTable_每次都会调用constructIstreamConstructorTables(), 那么岂不是每次都会重新定义一个新的、空的IstreamConstructorTablePtr_指针？忘指正

回复 • 分享 ›

Weiwen Zhao 管理员 realRabbita • 8 个月前

你好，`constructIstreamConstructorTables()` 是一个静态函数，其中定义了一个静态变量：`static bool constructed = false;`。

当第一次添加记录时，将 `constructed` 设为 `true`。

以后再添加记录，`constructed` 一直为 `true`，因此不会执行 if 条件中的语句。

1 回复 • 分享 ›