**Integrators**

一个命名空间和函数的集合，用于简化有限元程序中泛函和双线性泛函的编程。汇集了两种目的的函数：MeshWorker命名空间中的抽象积分函数，以及LocalIntegrators命名空间中的针对实际问题的（如对流问题、散度问题等）用于计算cell term和face term积分的实际函数实现。

**有限元网格上的积分**

当我们在有限元空间上进行函数或泛函的积分时，积分循环的结构总是相同的。一般在3-5个循环层次，从外至内为：

1、在所有网格上循环

2、可选地，在所有界面上循环，从而计算通量

3、在网格/界面的所有积分点上循环

4、可选地，在所有检验函数上循环，从而计算泛函（forms）

5、可选地，在所有试探函数上循环，从而计算双线性泛函（bilinear forms）

这些循环自然地分归为两个类，即网格或界面上的贡献的类（循环3-5，称为局部贡献），以及在网格对象上的外层循环（此过程常称为装配）。

deal.II中用于支持外层循环的代码可以在命名空间MeshWorker中找到。为了网格及界面贡献，deal.II提供了FEValuesBase类及其派生类。虽然外层循环具有一般性，但局部贡献的计算依赖于具体的问题。因此，此处不可能存在一般性的算法。然而，我们可以提供一个一般性的用于计算积分的函数接口（命名空间MeshWorker下），并提供一个积分器库（存放在命名空间LocalIntegrators下）。

**LocalIntegrators命名空间**

其下又有命名空间：

|  |
| --- |
| Advection |
| Divergence |
| Elasticity |
| GradDiv |
| L2 |
| Laplace |
| Maxwell |
| Patches |

这个命名空间提供了用于具体应用中计算双线性泛函、泛函及误差估计的局部积分器。比如，命名空间Laplace下包含了拉普拉斯算子的计算网格矩阵及残值的函数，以及处理弱边界条件的函数。命名空间L2下包含计算质量矩阵及L2内积的函数。

**函数签名**

这个命名空间下的函数遵循一种一般性的函数签名（及声明的形式）。最简单的情况下，有两个相关的函数：

template<int dim>

void

cell\_matrix (

FullMatrix<double>& M，

const FEValuesBase<dim>& fe,

const double factor = 1.);

template<int dim>

void

cell\_residual (

BlockVector<double>\* v,

const FEValuesBase<dim>& fe,

const std::vector<Tensor<1,dim>>& input,

const double factor = 1.);

这实际上是对应同一算子的一对函数，函数cell\_residual实现了算子从有限元空间到它的对偶空间的映射，函数cell\_matrix生成了对应于cell\_residual的Frechet导数的双线性泛函。

这些函数的第一个参数是返回类型，即：

* FullMatrix<double> 对应矩阵
* BlockVector<double> 对应向量

下一个参数是FEValues对象，代表了用于积分的有限元。

**用途**

局部积分器在任何需要用到局部积分循环的地方都能使用。下面的例子源于Stokes求解器的代码实现，使用了MeshWorker::Assembler::LocalBlocksToGlobalBlocks。矩阵为：

* 0：速度的向量拉普拉斯算子
* 1：散度矩阵
* 2：用在预处理器中的压力质量矩阵

有了这些矩阵，则MeshWorker::loop()调用的函数就能写成这样：

using namespace ::LocalIntegrators;

template<int dim>

void MatrixIntegrator<dim>::cell(MeshWorker::DoFInfo<dim> &dinfo //计算cell\_term积分

MeshWorker::IntegrationInfo<dim> &info)

{

Laplace::cell\_matrix (dinfo.matrix(0, false).matrix, //这里直接调用现成的局部积分函数计算矩阵

info.fe\_values(0));

Divergence::cell\_matrix (dinfo.matrix(1,false).matrix,

info.fe\_valuse(0),

info.fe\_values(1));

L2::cell\_matrix (dinfo.matrix(2,false).matrix,

info.fe\_values(1));

}

**MeshWorker命名空间**

这个命名空间提供了有限元程序中普遍存在的用于网格循环的函数和类：

类：DoFInfo、DoFInfoBox、IntegrationInfo、IntegrationInfoBox、LocalIntegrator、LocalResults…

函数：cell\_action()、loop()、integration\_loop()、mesh\_loop…

这个命名空间下的主力函数是loop()函数，它提供了一般性的网格循环函数接口。由于其一般性，往往直接使用它容易出错，在很多应用中更建议使用MeshWorker::LocalIntegrator类的一个派生类以及不那么general的integration\_loop()函数。

loop()函数需要若干对象作为它的参数。这些对象分为两类，一类是info对象，如DoFInfo和IntegrationInfo，另一类是worker对象如LocalWorker和IntegrationWorker。

Worker对象通常做两个不同的工作：首先，它们计算网格或界面的局部贡献。然后，它们把这些局部贡献组装到全局结果，可以是泛函、form或双线性form。虽然第一个工作是依赖于具体问题的，但第二个工作具有一般性，仅仅依赖于数据结构。因此，我们在Assembler命名空间下提供了用于worker的基类。

**模板参数类型**

函数loop()和cell\_action()使用了一些模板参数。在这里列出这些类需要的最少的要求：

**1. ITERATOR**

任何含有operator++()的指向TriaObjectAccessor的对象。

**2. DOFINFO**

这个类的示例为类模板DoFInfo。为了与cell\_action()和loop()函数一起使用，DOFINFO需要依照下述接口：

class DOFINFO

{

private:

DOFINFO();

DOFINFO(const DOFINFO&);

DOFINFO& operator=(const DOFINFO&);

public:

template<class CellIt>

void reinit(const CellIt& c);

template<class CellIt, class FaceIt>

void reinit(const CellIt& c, const FaceIt& f, unsigned int n);

template<class CellIt, class FaceIt>

void reinit(const CellIt& c, const FaceIt& f, unsigned int n, unsigned int s);

friend template class DoFInfoBox<int dim, DOFINFO>;

};

三个私有成员函数被DoFInfoBox调用，不会在别处用到。

此外，你还需要至少一个共有构造函数。

DOFINFO对象汇集到DoFInfoBox中。在这些对象中，我们存放局部操作后的结果。一旦这些信息被汇集好后，使用一个ASSEMBLER把它装配到全局数据中去。

**3. INFOBOX**

这个类的示例为IntegrationInfoBox。它把用于局部操作的输入数据收集在INFO对象中。它提供了下述连接到loop()和cell\_action()的接口：

class INFOBOX

{

public:

template<int dim, class DOFINFO>

void post\_cell(const DoFInfoBox<dim, DOFINFO>&);

template<int dim, class DOFINFO>

void post\_faces(const DoFInfoBox<dim, DOFINFO>&);

INFO cell;

INFO boundary;

INFO face;

INFO subface;

INFO neighbor;

};

这个类的主要目的是把五个INFO对象收集起来，它们包含了用于每个网格和界面的临时数据。对于这些对象的要求列在下面。

这里的两个函数模板是在cell\_action()中被调用的回调函数。第一个的调用位于在faces上工作前，第二个位于在faces上工作后。

**4. INFO**

这个类的示例为IntegrationInfo类。它们包含了在网格和界面上计算结果所需要的临时数据。MeshWorker只使用下述接口：

class INFO

{

public:

void reinit(const DOFINFO& i);

}

**接口函数**

void

MeshWorker::cell\_action(ITERATOR cell,

DoFInfoBox<dim,DOFINFO>& dof\_info,

INFOBOX& info,

const std::function<void(DOFINFO&, typename INFOBOX::CellInfo&)>& cell\_worker,

const std::function<void(DOFINFO&, typename INFOBOX::CellInfo&)>& boundary\_worker,

const std::function<void(DOFINFO&, DOFINFO&, typename INFOBOX::CellInfo&, typename INFOBOX::CellInfo&)>&)& face\_worker,

const LoopControl& loop\_control

)

这个函数被loop()调用，在网格和界面上执行需要的操作。当函数返回后，DoFInfoBox中的DoFInfo对象会被计算得到的数据所填充。因而，在这个函数执行后，就可以调用DoFInfoBox::assemble()函数进行装配了。

参数：

cell 是当前操作所处的网格

dof\_info 是存放局部结果的对象

info 是包含内部过程额外数据的对象

cell\_worker 定义了在每个网格上进行的局部操作

boundary\_worker 定义了在边界面上进行的局部操作

face\_worker 定义了在内部界面上进行的局部操作

loop\_control 指定该进行哪些操作的控制结构

void

MeshWorker::loop(ITERATOR begin,

typename identity<>::type end,

DOFINFO& dinfo,

INFOBOX& info,

const std::function<void(DOFINFO&, typename INFOBOX::CellInfo&)>& cell\_worker,

const std::function<void(DOFINFO&, typename INFOBOX::CellInfo&)>& boundary\_worker,

const std::function<void(DOFINFO&, DOFINFO&, typename INFOBOX::CellInfo&, typename INFOBOX::CellInfo&)>&)& face\_worker,

ASSEMBLER& assembler,

const LoopControl& lctrl=LoopControl()

)

下面详述此命名空间下一些重要的类及其基类：

**MeshWorker::DoFInfo → MeshWorker::LocalResults**

LocalResults这个类用于存放局部积分的结果。它是MeshWorker::DoFInfo类的基类，其私有

* 成员变量为：

std::vector<number> J

std::vector<BlockVector<number>> R

std::vector<MatrixBlock<FullMatrix<number>>> M1

std::vector<MatrixBlock<FullMatrix<number>>> M2

Table<2, number> quadrature\_data

这个类提供了用于保存局部积分结果的地方。这些局部结果实际的类型取决于使用它们的Assembler。也是由Assembler把局部结果数组设置为正确的大小。下面是提供的数据类型以及使用它们的assembler：

1. n\_values()个数，由value()函数访问，存储在数据成员J中。

2. n\_vectors()个向量，向量长度等于一个网格上的自由度数，由vector()函数访问，保存在R中

3. n\_matrices()个矩阵，矩阵维度为每个方向上等于一个网格上的自由度数，由matrix()函数访问（第二个参数为false），保存在M1中，这些矩阵耦合了同一网格中的自由度。对于跨过界面的通量，有多余的M2用来存储同样多个矩阵，不过这些矩阵的维度则同时取决于两侧网格的自由度。这些由matrix()函数访问（第二个参数为true）。

* 共有函数接口：

number& value(int i)

BlockVector<number>& vector(int i)

MatrixBlock<FullMatrix<number>>& matrix(int i, bool external=false)

在应用中，会把局部结果写入如dinfo.matrix(0).matrix及dinfo.vector(0).block(0)这样的数据结构中。这里，dinfo.matrix(i)返回的是对应当前网格i的局部矩阵，dinfo.vector(i)返回的是对应当前网格的局部向量，进而dinfo.vector(0).block(0)返回的则是。

BlockVector<number>是基于deal.II的Vector对象实现的block vectors。可看做一个大向量，由若干block子向量组成。其基类是BlockVectorBase<Vector<number>>

BlockVector中的不同子向量可以有不同尺寸。BlockVector的功能包括了所有Vector具有的功能，外加通过函数block(i)来访问其中的某个Vector。它也含有完整的随机访问迭代器，就像别的Vector类或者std::vector。因此，所有适用于迭代器的算法都可以在这个类的对象上使用。

**MeshWorker::IntegrationInfo<dim, spacedim>**

这个类的对象传递给局部积分函数用于积分。

这个类的对象或多或少含有FEValues、FEFaceValues或FESubfaceValues类的对象。它们保存在一个指向基类FEValuesBase类的指针数组中。

此外，还含有空间global\_data，用于存放有限元函数在积分点处的值。这些向量在每个网格或界面上自动初始化。为了避免初始化不需要的向量，可使用initialize\_selector()来选择需要的向量。

* 积分模式：

这个类支持两种局部积分模式，对应了Assembler命名空间文档中的数据模式。

一种是通过使用FESystem的标准模式。即，这个类存在一个FEValuesBase对象，包含了整个系统的全部形函数，且与系统分量数保持一致。使用这种模式需要在所有系统形函数上循环。它需要对每个形函数分辨系统分量，并选择正确的双线性形式（通常用if或switch语句）。

另一种积分模式基于每个系统基元对应一个FEValuesBase对象。每个网格上的自由度按块重排，从而它们代表了与全局系统相同的分块结构。则进行积分的对象可分别在每个块上进行操作，可显著改善代码的重用性。

注：正如在DoFInfo中提到的，分块模型的使用是通过在这个类使用initialize()之前调用BlockInfo::initialize\_local()来触发的。

* 共有函数接口

const FEValuesBase<dim, spacedim>& fe\_values() const

const FEValuesBase<dim, spacedim>& fe\_values(const int i) const

上面两个函数都是访问某个有限元的函数。区别在于，第一个函数是当使用单一有限元来initialize()的时候(没有BlockInfo参数)才使用，第二个函数是当使用一组有限元来initialize()的时候用的。

在应用中，通常先定义const FEValuesBase<dim>& fe\_v=info.fe\_values()，然后调用fe\_v.shape\_value(j，point)来获得某个形函数(对应自由度索引j的)在某个积分点处(对应积分点point)的值。注意，这里相当于是函数info.fe\_values().quadrature\_point(point)，返回的是某个积分点在真实空间的坐标。故用户需要知道FEValuesBase类的结构，其详细内容见其他笔记。

* 成员变量

std::vector<std::shared\_ptr<FEValuesBase<dim, spacedim>>> fevalv

这其实就是个保存FEValues对象的向量。

std::vector<std::vector<std::vector<double>>> values

std::vector<std::vector<std::vector<Tensor<1,spacedim>>>> gradients

std::vector<std::vector<std::vector<Tensor<2,spacedim>>>> hessians

std::shared\_ptr<VectorDataBase<dim, spacedim>> global\_data

其中values是一个向量，它包含了有限元函数在积分点处的值。对每个有限元函数就有一个对应的向量，此向量又包含了一个向量表示各分量，内部又包含一个向量表示各积分点的值。