**Who is responsible for what?**

先介绍下各个对象各自负责提供什么信息。

**FEValues objects**

例如求积分： 

转化为数值积分： 

FEValues的作用是给用户提供基函数(包括其导数等)在积分点处的值，以及积分点处的某些几何信息如法向量等。但FEValues本身不计算这些信息，它只是提供一个存储这些信息的地方。具体计算是通过调用mapping和finite\_element相关的函数来进行的。

**评注**：这里实际空间上的基函数的具体形态是不清楚的，是由参考单元上的基函数映射来的，虽然我们不知道其具体样子，但不妨碍我们的计算。因为我们知道，这样的基函数经过逆映射，回到参考单元后，它将退成为关于参考坐标ξ,η(而非x,y)的函数，而其函数形式就是由FiniteElement类定义的基函数形式。在这里虽然写的是，但实际上我们也不需要知道实际空间中积分点的坐标，需要的只是由Quadrature提供的参考单元上积分点位置，因为我们的积分是在参考单元上做的。故上面的应该换成。

**Mappings**

Mappings负责任何与坐标变换有关的东西。从参考单元到实际单元的映射为：



因为FEValues只需要在积分点上的值，因而mapping也只需要提供积分点上的映射。

mapping类有两种作用：1）计算几何信息啊（如法向量、雅可比阵的行列式）并存放到FEValues提供的地方；2）提供支持的有限元，用于把基函数及其导数从参考单元映射到实际单元。

**FiniteElement**

Finite Element负责定义参考单元上的基函数、导数等及计算实际单元上的映射值（当然，需要mapping对象的辅助）。

它能提供积分点处的信息并把计算结果存放到FEValues提供的地方。

**What to compute？**

为计算上面的积分，通常的代码为：

fe\_values(mapping, fe, quadrature, update\_gradients | update\_JxW);

fe\_values.reinit(cell);

fe\_values.shape\_grad(i, quadrature\_points); //i是对应自由度i的基函数

用户通过标识update\_gradients | update\_JxW指定了需要网格积分点上的梯度和权重数据。

首先，**构造fe\_values（fe\_values(mapping, fe, quadrature, update\_gradients | update\_JxW)）**：

调用fe\_values的构造函数时，它先通过调用**FiniteElement::requires\_update\_flags()**告知fe需要提供什么数据。为了计算实际网格积分点上的，需要通过参考网格对应点的梯度值乘以雅可比矩阵的逆，也就是需要提供参考网格上的基函数梯度及映射的雅可比矩阵的逆，所以在这里函数会返回两个flags: update\_gradients和update\_covariant\_transformation；

再调用函数**Mapping::requires\_update\_flags()**，告知mapping，为了提供update\_gradients和update\_covariant\_transformation，需要mapping提供什么。前者不关mapping的事，忽略；后者要求mapping提供雅可比矩阵，所以返回flag: update\_contravariant\_transformation。

也就是说，至此fe\_values找出了每个对象各自需要提供的信息（对应的flags）。

然后它**进行一些预运算**：

fe\_values先使用quadrature对象和上面得到的update flags来初始化它指向的mapping对象和fe对象，这个初始化过程就是计算并保存尽可能多的后续会用到的信息。

fe\_values调用**Mapping::get\_data(update\_flags, quadrature)**。把结果存到Mapping::InternalDataBase中，需要的时候返回给fe\_values；

fe\_values调用**FiniteElement::get\_data(update\_flags, mapping, quadrature, output\_data)**。把结果存到FiniteElement::InternalDataBase中，需要的时候返回给fe\_values。

然后，**在每个网格上进行具体计算（fe\_values.reinit(cell)）：**

调用**Mapping::fill\_fe\_values(cell, cell\_similarity, quadrature, internal\_data, output\_data)**，结果存放到internal::FEValues::MappingRelatedData output\_data中去。它会计算所有由update flags指定的与映射有关的数据，如JxW，雅可比矩阵及其逆，法向量等。

调用**FiniteElement::fill\_fe\_values(cell, cell\_similarity, quadrature, mapping, mapping\_internal, mapping\_data, fe\_internal, output\_data)**, 结果存放到internal::FEValues::MappingRelatedData output\_data中去。

下面具体分析一下代码：

**预运算：**

virtual InternalDataBase\* **Mapping< dim, spacedim >::get\_data**

**( const UpdateFlags update\_flags,**

**const Quadrature< dim > & quadrature )**

此函数的目的是保存那种只需要mapping计算一次并被用于实际网格上的计算的数据。

update\_flags：标识需要mapping在未来调用fill\_fe\_values()的时候提供的数据

quadrature：如积分点坐标和权重

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

virtual InternalDataBase\* **FiniteElement< dim, spacedim >::get\_data**

**( const UpdateFlags update\_flags,**

**const Mapping< dim, spacedim> & mapping,**

**const Quadrature< dim > & quadrature,**

**::internal::FEValues::FiniteElementRelatedData<dim,spacedim>&output\_data)**

此函数目的是节省实际网格上的计算消耗。比如计算实际网格积分点处的基函数梯度，需要由参考网格积分点处的基函数梯度乘以雅可比矩阵的逆，而参考网格上的梯度值对所有实际网格都是一样的，所以只需计算一次。计算后的返回值会传给FiniteElement::fill\_fe\_values()用于在实际网格上计算。

update\_flags：标识需要fe计算哪些信息

mapping：引用mapping，用于计算形函数的值(包括导数值等)

quadrature：引用积分对象，用于说明需要在哪些点计算形函数的值

output\_data：用于存放那种只需计算一次且已经是最终形式的数据(在参考网格和实际网格上都一样的数据)，例如基函数在积分点处的值。

**实际运算：**

virtual CellSimilarity::Similarity **Mapping< dim, spacedim >::fill\_fe\_values**

**( const typename Triangulation< dim, spacedim >::cell\_iterator & cell,**

**const CellSimilarity::Similarity cell\_similarity,**

**const Quadrature< dim > & quadrature,**

**const typename Mapping<dim, spacedim>::InternalDataBase& internal\_data,**

**::internal::FEValues::MappingRelatedData<dim, spacedim>& output\_data )**

这个函数计算如下数据：

1）应用坐标变换本身得到的数据，如实际网格上的积分点坐标 （注：定义坐标变换用到的点和积分点不是同一套点）

2）计算实际网格上的形函数需要的数据，如及其逆（在计算实际网格上的形函数梯度时要用到）

cell：此函数计算坐标变换所在的网格

cell\_similarity：第一个参数所给的网格是否是上次调用此函数时指向的网格的简单移动或旋转等

quadrature：引用积分公式。用于映射积分点，以及计算JxW(提供权重W)

internal\_data：引用由get\_data()函数创建的数据对象，这些数据是只需mapping计算一次的数据

output\_data：用于存放计算结果的对象。需要存放哪些结果，由internal\_data中的update flags来指定。

virtual void **FiniteElement< dim, spacedim >::fill\_fe\_values**

**( const typename Triangulation< dim, spacedim >::cell\_iterator & cell,**

**const CellSimilarity::Similarity cell\_similarity,**

**const Quadrature< dim > & quadrature,**

**const Mapping< dim, spacedim > & mapping,**

**const typename Mapping<dim, spacedim>::InternalDataBase& mapping\_internal,**

**const ::internal::FEValues::MappingRelatedData<dim,spacedim>&mapping\_data,**

**const InternalDataBase & fe\_internal,**

**::internal::FEValues::FiniteElementRelatedData<dim, spacedim>& output\_data )**

==========================================================================

在利用Manifold对象定义高阶边界并求高阶边界的法向量时的具体函数调用过程：

(可利用我自己写的test\_visualize\_shape\_function这个函数进行调试测试查看调用过程)

1. fe.normal\_vector(i)

fe内部有个mapping\_output对象保存了normal\_vectors[i]

这个mapping\_output对象其实是::internal::FEValuesImplementation::MappingRelatedData<dim>类，它是在

2. FEValues::reinit()🡪(call)Mapping::fill\_fe\_values()时返回的对象

3. virtual CellSimilarity::Similarity Mapping<dim>::fill\_fe\_values(cell\_iterator,

cell\_similarity,

quadrature,

internal\_data,

output\_data)

此函数在MappingManifold下的具体实现为：

virtual CellSimilarity::Similarity MappingManifold<dim>::fill\_fe\_values(cell\_iterator,

cell\_similarity,

quadrature,

internal\_data,

output\_data)

它在内部计算output\_data.normal\_vectors[point]时调用的函数为：

4. cross\_product\_2d(-DX\_t[0])，其参数是个Tensor，这个函数本身也是Tensor类的成员函数，它的作用是把切向量旋转90°得到法向量。

==========================================================================