**FEValuesBase**

template<int dim, int spacedim>

class FEValuesBase< dim, spacedim >

FEValues, FEFaceValues和FESubfaceValues对象一方面是通往有限元和映射类的**接口**，另一方面是通往网格和积分规则的**接口**。它本身几乎没有什么成员变量，而几乎全是成员函数。它们允许计算某个积分点处形函数的值或其导数值。这样进行抽象的原因是为了之后的优化：取决于不同类型的有限元和映射方式，有些值可以只在单位网格上计算一次。其他的则必须在每个网格上进行计算。但由于可以在同一时刻计算若干不同的值，这就提供了可能的优化方式。由于这种相互作用很复杂且依赖于实际的有限元，故而不能交给应用用户进行操作。

FEValues, FEFaceValues和FESubfaceValues只提供数据控制：具体计算是交给Mapping和FiniteElement类对象进行的。这就提供了get\_\*\_data和fill\_\*\_values函数，它们分别被FEValues\*类的构造函数和reinit函数调用。

**一般用法**

通常，一个FEValues\*类对象是在关于全部网格(或界面)的积分循环中被用到。为了利用其优化特性，应该在循环开始前创建它，则不依赖于具体网格地址或形状的信息会一次性地被计算出来（比如，可计算出在单位网格上形函数在积分点上的值，它们在映射到具体网格前对于所有网格都是一样的）。

然后，在网格循环中，对于每个网格它必须被重新初始化。比如，形函数的梯度值等于在单位网格上的梯度值（它在循环之前就已经一次性地为所有网格计算出了）乘以从单位网格到实际网格映射的jacobian矩阵（需要在每个网格上重新计算））

一个典型的，加和Laplace矩阵的局部贡献的代码如下：

FEValues values (mapping, finite\_element, quadrature, flags);

for (cell = dof\_handler.begin\_active();

cell != dof\_handler.end();

++cell)

{

values.reinit(cell);

for (unsigned int q=0; q<quadrature.size(); ++q)

for (unsigned int i=0; i<finite\_element.dofs\_per\_cell; ++i)

for (unsigned int j=0; j<finite\_element.dofs\_per\_cell; ++j)

A(i,j) += fe\_values.shape\_value(i,q) \*fe\_values.shape\_value(j,q) \*fe\_values.JxW(q);

...

}

注意，根据设计，在FEValues对象内部使用的积分点顺序和上面传给FEValues对象的构造函数的积分公式所定义的积分点顺序是一样的。

**成员函数**

这个类的成员函数分为不同的类型：

* shape\_value(), shape\_grad()等：每次返回这个对象的一个值
* shape\_value\_component(), shape\_grad\_component()等：用于向量有限元
* get\_function\_values(), get\_function\_gradients()等：计算积分点处的有限元函数或其导数值
* reinit：对某个网格初始化上面的FEValues。这个函数不在当前类中，而是在其派生类中。

例如：

void FEValuesBase<dim, spacedim>::get\_function\_laplacians(const InputVector& fe\_function

std::vector<typename InputVector::value\_type>& laplacians)

计算某个有限元在积分点处的(标量)laplacian。参数为：

[in] fe\_function (全局地)描述当前有限元函数的值的向量。

[out] laplacians fe\_function指定的函数在当前网格积分点上的laplacian值。laplacian值是在实际空间中计算的。