在建立了网格之后，现在我们来说明如何在网格上定义自由度。在此例中，将使用最低阶的有限元(Q1)，即自由度是附着在网格的格点上的。

术语“自由度”在有限元中往往表示两种截然不同的，但相关的含义。

第一种是，我们希望把有限元解表示成形函数(shape function)的线性组合，即。这里，是展开系数。因为我们还不知道它们的值(我们将在一个线性或非线性系统中把它们求解出来)，所以称之为“未知数”或者“自由度”。

第二种含义是：有限元问题的数学描述往往表述为，我们希望找到一个有限维的函数，它满足一些方程(例如)。换句话说，就是说解必须处在某种空间中。然而，为了在计算机上进行实际计算，我们必须选取这个空间的一组基；这就是我们上面使用的带有系数的形函数集合。当然，空间的基有很多组，但我们特别地选择了由有限元函数进行描述的，局部地定义在每个网格上的那组。在这种语义下，所谓“自由度”要求我们指定(enumerate)出空间的基函数。对于Q1有限元，则我们必须在网格的格点上进行分派，但对于高阶的有限元，则必须指定出依附在边、面、或网格内部的形函数。提供这种对的基函数的enumeration的类称为DoFHandler。

在网格上定义自由度很简单，事实上，只需要先创建一个有限元对象，然后通过DoFHandler::distribute\_dofs函数将其交给DoFHandler对象即可(我们使用“分布自由度”这个术语来表示上面描述的指定基函数的过程)。DoFHandler类用于控制哪个自由度处于何处，即，它能回答诸如“全局共有多少自由度”或“在这个网格上，告诉我处于这里的所有基函数的全局索引indices是多少”这样的问题。当决定系统的矩阵应该为多大，以及把某个网格上的局部贡献复制到全局矩阵时，将会用到这些信息。

下一步就是求解对应于某个偏微分方程的离散后的矩阵和右端项。我们将把这个任务留到Step-3，在这里我们只讲另外一个关键的问题，即有限元矩阵的稀疏性，即矩阵中大多数元素都是零（更准确地讲，只有当每一行的非零元素个数存在某个固定的上界，而不会随着全局自由度的增加而增长时，才称其为稀疏矩阵）。稀疏性是有限元方法区别于别的方法，例如泰勒展开或使用傅里叶基函数的一个显著特定。

稀疏性来源于有限元形函数是在不同的网格上局部地定义的，而非全局，双线性form的局部的微分运算只会导致在有部分重叠的形函数间产生耦合。默认情况下，DoFHandler类在网格上以随意的方式进行自由度分派（自由度标号顺序是随机的）；因而，产生的sparsity pattern是非最优化的。然而，对于某些线性求解器或预处理器，把自由度按某种有序方式进行排列会更有利，我们将使用Cuthill和McKee的算法来实现。可把这看做选择一组不同的，变更过次序的有限元空间的基函数。

void make\_grid2(Triangulation<2> &triangulation){

GridGenerator::hyper\_cube(triangulation);

triangulation.refine\_global(1);

}

void distribute\_dofs(DoFHandler<2> &dof\_handler){

static const FE\_Q<2> finite\_element(1);

dof\_handler.distribute\_dofs(finite\_element); //how's the distribution of the dofs?

DynamicSparsityPattern dsp(dof\_handler.n\_dofs(),dof\_handler.n\_dofs());

//now the dsp knows it should have 9 rows, but doesn't know which entry shoud be nonzero, so it doesn't have entries to denote the location of nozero elements yet.

DoFTools::make\_flux\_sparsity\_pattern (dof\_handler, dsp);

//compute which entries of a matrix built on the given dof\_handler may possibly be nonzero, and create a sparsity pattern object that represents these nonzero locations. So now the dsp correctly describes the location of nonzero element in the matrix.

SparsityPattern sparsity\_pattern;

sparsity\_pattern.copy\_from(dsp);

}

int main(){

Triangulation<2> triangulation;

make\_grid2(triangulation); //生成网格

DoFHandler<2> dof\_handler(triangulation); //生成dof\_handler

distribute\_dofs(dof\_handler); //分布自由度到网格上并生成sparsity pattern

}