deal.II learning note

1.deal.II介绍

deal.II是一个开放源代码的有限元程序库，是一个C++软件库，支持创建有限元代码以及开放的用户和开发人员社区。

deal.II 是一个开源的有限元法求解器，支持大规模并行计算，自适应网格。采用C++编写，实现优雅。其文档完整丰富，文档共有三个级别，由浅入深：

- tutorial：一系列的教学程序，共有64步教学步骤，通过tutorial的学习可以对dealii有整体的认识。
- manual：对每个类以及相应的函数的介绍。适合用于查询类与函数的具体用法。
- Modules：介绍了实现某一个功能需要用到的一系列类与函数，比如 Sparsity patterns 介绍了存储稀疏矩阵相关的内容。

本deal.II学习笔记基于deal.II官方文档，学长的学习笔记和Wolfgang Bangerth's lectures。

2.deal.II安装

3.deal.II示例学习

3.1.Step-1

相关链接：deal.II step-1 tutorial program

step-1的源码位于**examples/step-1**目录，进入该目录并输入如下命令进行编译（对于其它的step项目同样需要进行如下的编译）：

```
cmake .         # 生成makefile，会查找deal.II库
make            # 生成可执行文件
make run        # 执行可执行文件，也会编译代码，所以第二句代码可忽略
```

如`cmake .`报错，如果此命令无法找到deal.II库，则需要使用以下命令提供安装路径：

```
cmake -DDEAL_II_DIR=/path/to/installed/deal.II .
```

step-1的主要内容是网格生成。网格有两种，一是方形网格，二是环形网格，并展示了网格迭代细化的过程。

有限元程序的三个过程：网格的Triangulation类型的对象；调用GridGenerator函数以生成网格；并在相关迭代器的所有单元上循环；

加载头文件

首先需要加载头文件，最为重要的是 Triangulation 类，其用于生成单元，Triangulation<1>表示一维单元，以此类推，可以表示二维单元和三维单元，对于边界单元，Triangulation<1,2>表示面边界上的曲线单元，Triangulation<2,3>表示体边界上的曲面单元。

```
#include <deal.II/grid/tria.h>
```

加载以下两个与在单元格和（或）面上的循环的头文件，实现单元存取和迭代：

```
#include <deal.II/grid/tria_accessor.h>
#include <deal.II/grid/tria_iterator.h>
```

生成具有一点过标准形状的网格：

```
#include <deal.II/grid/grid_generator.h>
```

网格的格式化输出：

```
#include <deal.II/grid/grid_out.h>
```

C++输入输出：

```
#include <iostream>
#include <fstream>
```

载入库cmath（数学运算库）：

```
#include <cmath>    // for std::sqrt and std::fabs
```

最后加上命名空间dealii，确保调用的函数来自dealii库。

```
using namespace dealii;
```

创建第一个网格（正方形网格）

首先申明二维单元triangulation，然后设置单元为立方体形状，并对单元进行四次全局细化，每次细化一分为四，总共将产生$4^4 = 256$个单位网格。最终将网格绘制成eps图像输出（或其它图像格式，如*.svg）。

```
void first_grid()
{
  Triangulation<2> triangulation;
  GridGenerator::hyper_cube(triangulation);
  triangulation.refine_global(4);
  std::ofstream out("grid-1.eps");
  GridOut       grid_out;
  grid_out.write_eps(triangulation, out);
  std::cout << "Grid written to grid-1.eps" << std::endl;
}
```

创建第二个网格（环形网格）

同样首先申明二维单元triangulation，然后设置中心点，内外圆的半径，并用hyper_shell函数生成网格，可以通过此函数自动调整圆周单元的数量，最后一个参数显式设置为10，即内外圈分为10个环带网格。

```
void second_grid()
{
  Triangulation<2> triangulation;
  const Point<2> center(1, 0);
  const double   inner_radius = 0.5, outer_radius = 1.0;
  GridGenerator::hyper_shell(
    triangulation, center, inner_radius, outer_radius, 10);
```

> 默认情况下，triangulation假定所有边界均为直线，为保证区域边界是弯曲的，引入一个manifold indicator，如果没有manifold描述与特定的manifold indicator关联，则表示产生笔直边缘的manifold。

然后通过五次（自定义）循环细化网格：

```
for (unsigned int step = 0; step < 5; ++step)
  {
```

> triangulation的单元格并不是数组方式储存，而是以迭代器（iterator）的方式进行储存，包含指针，迭代器的基本操作见C++学习笔记。
>
> active cell是未进一步细化的单元，并且是可以标记为进一步细化的单元，即已激活的网格，所以遍历的是最细的网格。

遍历所有active cell：

```
for (auto it = triangulation.active_cell_iterators().begin();
     it != triangulation.active_cell_iterators().end();
     ++it)
  {
    auto cell = *it;
    // Then a miracle occurs...
  }
```

上面的写法需要声明循环的初始与结束条件，在此还可以采用更为简便的方式：

```
for (auto &cell : triangulation.active_cell_iterators())
  {
```

获取单元的维度：

```
   for (const auto v : cell->vertex_indices())
    {
```

在此细化的策略为：如果网格是靠近圆环内边沿的网格，即有节点在圆环内径上，那么就加密这个网格。由于计算机机内小数表示有误差，因此不能直接判断等于，而是差值绝对值小于一个极小数，该极小数设置为等于环的内半径的10^{-6}倍。

```
      const double distance_from_center =
        center.distance(cell->vertex(v));  // 网格节点
      if (std::fabs(distance_from_center - inner_radius) <=
          1e-6 * inner_radius)
        {
          cell->set_refine_flag();  // 标记为active cell
          break;
        }
    }
}
```

已经标记了所有要细化的单元，接下来执行网格的稀疏和细化函数即可：

```
  triangulation.execute_coarsening_and_refinement();
}
```

最后输出网格的图像：

```
  std::ofstream out("grid-2.eps");
  GridOut       grid_out;
  grid_out.write_eps(triangulation, out);
  std::cout << "Grid written to grid-2.eps" << std::endl;
}
```

main函数

在main()调用之前生成两个网格的函数即可。

```
int main()
{
  first_grid();
  second_grid();
}
```