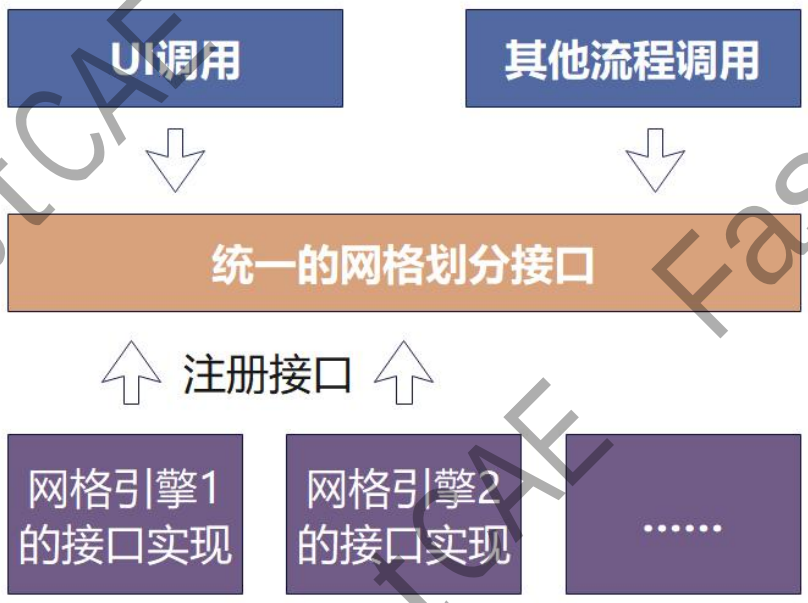
网格引擎集成与接口调用

说明书

青岛数智船海科技有限公司

时间	修改纲要	修改人
2024/09/02	首次编制	李宝君

为适应不同网格划分程序，FastCAE 定义了一套通用的网格划分接口，规定 UI 与其他逻辑操作只能访问该接口，组件中实现抽象接口，以达到隔离网格划分引擎的目的。基于这套接口与组件接口，可实现网格划分程序的集成。FastCAE 定义的抽象接口位于 FITKInterfaceMeshGen 中。接口主要包含尺寸场定义、网格划分引擎驱动、以及网格读入等部分。



一、网格划分接口定义

主要接口类及功能描述如下：

类名	功能描述
FITKMeshGenInterface	单例类，用于接口实现类的注册以及功能接口的调用，并且用于存储尺寸场等网格划分参数。
FITKAbstractMesherDriver	抽象网格划分引擎驱动类，定义网格划分引擎输入参数、启动网格划分引擎、终止网格划分引擎操作等接口。
FITKAbstractMeshSizeInfoGenerator	网格局部尺寸场生成器，指定某一个特性形状控件的尺寸场，相较于下者，该尺寸场是与几何模型无关的。

FITKAbstractGeometryMeshSizeGenerator	与几何相关的网格局部尺寸生成器，指定某一个几何集合（线面集合）的网格划分尺寸。可用于指定边界层网格参数。
FITKAbstractMeshProcessor	网格划分完成之后的网格处理器，定义网格处理（从文件读入或者内存转化）的过程
FITKGlobalMeshSizeInfo	定义网格划分全局尺寸，例如最大最小尺寸，尺寸因子等，该类继承 FITKVarientParams，可以自由添加以字符串为键标记的参数，参数可以为任意类型。
FITKGeometryMeshSize	定义与几何绑定的网格局部尺寸，通过几何组件的唯一 ID 对几何进行描述。默认有边界层厚度、增长率等参数。该类继承 FITKVarientParams，可以自由添加以字符串为键标记的参数。FITKAbstractGeometryMeshSizeGenerator 生成该类型。
FITKAbstractRegionMeshSize	抽象区域网格尺寸参数，与几何模型无关，继承自 FITKGlobalMeshSizeInfo，后面的具体形状派生自该类。该类中定义了简单的网格离散参数，例如各个维度的离散数量与增长率等。FITKAbstractMeshSizeInfoGenerator 生成该类型参数
FITKRegionMeshSizeBox	长方体区域尺寸场，继承自 FITKAbstractRegionMeshSize。
FITKRegionMeshSizeCylinder	圆柱体区域尺寸场，继承自 FITKAbstractRegionMeshSize。
FITKRegionMeshSizeSphere	球体区域尺寸场，继承自 FITKAbstractRegionMeshSize。
FITKRegionMeshSizeGeom	通过几何文件指定的区域尺寸场，继承自 FITKAbstractRegionMeshSize。需要首先导入几何

	文件，然后指定几何数据的唯一 ID。
FITKZonePoint	网格区域点定义，可用于指定材料点。

上述接口为网格划分的抽象接口，对某一个具体的网格划分引擎不一定会用到上述的全部接口，需要视具体情况而定。下面将举例说明接口实现与调用的过程。

二、网格引擎集成过程

在网格划分引擎集成之前需要明确三个问题：1.网格划分引擎的输入是什么，如何指定；2.网格剖分引擎如何驱动；3. 网格划分输出的网格如何获取。一般而言，网格划分引擎的输入主要分为两部分，分别是几何信息与尺寸场信息，可通过文件或者内存的形式进行数据传递；网格剖分引擎的驱动方式通常分为两种，一种通过 API 调用，另外一种是通过可执行程序方式驱动；网格划分的结果一般也是两种，通过文件传递或者通过内存传递。上述三个问题将会直接影响网格划分引擎的集成方案与实现方法，因此一定要提前确认。

确认上述问题之后，便可以基于 FastCAE 定义的接口实现网格引擎的集成，通常情况下可以将网格引擎封装为一个组件，在程序初始化对网格划分引擎进行初始化，在后续的仿真流程中便可以直接使用。具体流程如下：

1. 编写网格剖分引擎组件

遵循 FastCAE 基础底座中的组件定义接口，编写一个组件。具体操作为继承 AppFrame::FITKComponentInterface 类编写子类。并实现两个纯虚函数，代码如下：

```
class MyMeshGenInterface :
    public AppFrame::FITKComponentInterface
{
public:
    explicit MyMeshGenInterface()
    {
        // todo
    }
    virtual ~MyMeshGenInterface() = default;
    virtual QString getComponentName() override
```

```
    {
        return "MyMeshGenInterface" ;
    }
    virtual bool exec(const int indexPort) override
    {
       return true;
    }
};
```

然后在注册到应用程序框架中的组件工厂的 createComponents 函数中添加创建 MyMeshGenInterface。代码如下：

```
QList<AppFrame::FITKComponentInterface *> ComponentFactory::createComponents()
{
    // 自定义组件列表
    QList<AppFrame::FITKComponentInterface *> componentList;
    //****创建其他组件********

    //定义的网格划分
    componentList << new MyMeshGenInterface ;

     //****创建其他组件********
    return componentList;
}
```

2. 定义尺寸场生成器

由于每一个网格划分引擎对尺寸场的表示方法均有所差异，定义尺寸场生成器可以方便的进行尺寸场的自定义拓展。在第一章节中已经说明了接口中尺寸场的表达方式，全局尺寸场（FITKGlobalMeshSizeInfo）和两种局部尺寸场（FITKGeometryMeshSize、FITKAbstractRegionMeshSize），这些信息生成的方式被封装在两个尺寸场生成器中（FITKAbstractMeshSizeInfoGenerator、FITKAbstractGeometryMeshSizeGenerator），其中全局尺寸场的生成包含在前者生成器中。定义尺寸场生成器之前，需要首先对尺寸场的表示进行确认，若已有的接口不能满足尺寸场的表达，则需要对数据表达进行派生。例如若需要对全局尺寸场进行派生，则代码如下：

```
class MyGlobalMeshSize :
    public Interface::FITKGlobalMeshSizeInfo
{
public
     MyGlobalMeshSize () = default;
```

```
6.       virtual MyGlobalMeshSize () = default;
7.
8.       //****添加其他参数与访问接口****
9.   };
```

完成尺寸场的数据表达之后，则需要对两个生成器进行定义，对虚函数进行重写，代码可参考：

```
1.   class MyMeshSizeGenerator :
2.       public Interface::FITKAbstractMeshSizeInfoGenerator
3.   {
4.   public:
5.       explicit MyMeshSizeGenerator () = default;
6.       virtual ~MyMeshSizeGenerator () = default;
7.
8.       virtual Interface::FITKGlobalMeshSizeInfo* generateGlobalMesh
     SizeInfo() override
9.       {
10.          return MyGlobalMeshSize;
11.      }
12.      virtual Interface::FITKAbstractRegionMeshSize* createRegionMe
     shSize(Interface::FITKAbstractRegionMeshSize::RegionType t)
     override
13.      {
14.          //根据类型创建自定义尺寸场，也可以直接采用已经提供的类型
15.          //用户根据需要添加
16.      }
17.  };
```

```
1.   class MyGeometryMeshSizeGenerator :
2.       public Interface::FITKAbstractGeometryMeshSizeGenerator
3.   {
4.   public:
5.       explicit MyGeometryMeshSizeGenerator() = default;
6.       virtual ~MyGeometryMeshSizeGenerator() = default;
7.       virtual Interface::FITKGeometryMeshSize* createGeometryMeshSi
     ze() override
8.       {
9.           //创建尺寸场，可以自定义也可以采用已经提供的类型
10.          //用户根据需要添加
11.      }
12.  };
```

3. 定义网格划分引擎驱动器

网格划分引擎驱动器需要实现网格划分引擎的启动与终止，该部分是比较消耗时间的操作，因此需要考虑使用多进程与多线程技术。网格划分驱动器的定义方式是继承 Interface::FITKAbstractMesherDriver，并且对定义的虚函数进行重写。

```
class MyMesherDriver : public Interface::FITKAbstractMesherDriver
{
public:
    explicit MyMesherDriver() = default;
    virtual ~MyMesherDriver() = default;
    virtual void startMesher(QStringList info = QStringList()) override
    {
        //根据网格划分引擎的输入要求，生成网格划分引擎需要的信息
        //以文件或者内存的形式表达

        //执行网格和划分操作
        //API 方式调用网格划分引擎时使用多线程（底座提供线程池）
        //可执行程序方式调用使用多进程驱动（底座提供可执行程序驱动器）

       //程序运行结束需要触触发基类的信号 mesherFinished !!!!
    }
    virtual void stopMesher(QStringList info = QStringList()) override
    {
            //杀死正在运行的线程或者进程
    }
};
```

4. 网格处理器定义

网格处理器是用于网格划分结束后，对网格引擎输出的网格进行处理，一般来说可能是读取网格引擎输出的网格文件，或者通过网格引擎提供的 API 拿到网格数据。然后将网格数据进行内存转化，进行可视化或者进行进的数据优化等操作。网格处理器的定义方式为，继承 Interface::FITKAbstractMeshProcessor，并重写虚函数。

```
class MyMeshProcessor : public Interface::FITKAbstractMeshProcessor
{
```

```
3.  public:
4.      explicit MyMeshProcessor () = default;
5.      virtual ~MyMeshProcessor () = default;
6.      virtual void start(QStringList info = QStringList()) override
7.      {
8.          //读取网格文件或者通过API读取网格信息
9.          //根据业务逻辑与功能要求将数据存储到相应位置
10.         //若需要可视化则调用可视化刷新函数
11.     }
12.
13. };
```

5. 自定义类注册

上述几个步骤对尺寸场生成、网格划分引擎驱动、网格处理器进行了定义，完成定义之后需要将他们注册到对应的位置上，只有完成了注册，上述自定义的函数才能够通过抽象接口被调用。注册的方法也比较简单，就是在第 1 步的网格剖分组件的构造函数中完成。代码参考如下：

```
1.  MyMeshGenInterface::MyMeshGenInterface()
2.  {
3.      //获取注册单例
4.      Interface::FITKMeshGenInterface* mf = Interface::FITKMeshGenI
    nterface::getInstance();
5.      if (nullptr == mf) return;
6.      //注册相关类
7.      mf->regMeshSizeGenerator(new MyMeshSizeGenerator);
8.      mf->regGeometryMeshSizeGenerator(new MyGeometryMeshSizeGenera
    tor);
9.      mf->regMesherDriver(new MyMesherDriver);
10.     mf->regMeshProcessor(new MyMeshProcessor);
11. }
```

三、网格划分接口调用

所有的网格划分接口均通过抽象的单例类进行调用，通过单例的接口，可调用到上一章节描述的具体的几何引擎的功能实现，这样就达到了屏蔽具体几何引擎的目的。该单例既

是功能接口，同时也是网格划分的数据管理对象，单例的获取方式如下：

```
    //获取注册单例
    Interface::FITKMeshGenInterface* mInterface = Interface::FITKMeshGenInterface::getInstance();
```

1. 尺寸场创建

全局尺寸场数据由单例类直接创建，全局尺寸场是全局唯一的，直接从单例中获取即可，若为用户自定义类型，可通过强制类型转换获取。

```
Interface::FITKGlobalMeshSizeInfo* gloInfo = mInterface ->getGlobalMeshSizeInfo();
MyGlobalMeshSize *  myInfo = dynamic_cast<MyGlobalMeshSize*>(gloInfo );
if(myInfo)
{
    //do something
}
```

对于局部尺寸场，则通过单例获取创建器，需要根据类型获取不同的创建器，然后通过创建器创建后将创建出来的局部尺寸场对象放回到单例的管理器中进行管理，示例如下：

```
//局部区域尺寸场管理器（几何模型无关）
auto  meshSizeManager = Interface::FITKMeshGenInterface::getInstance()->getRegionMeshSizeMgr();
//尺寸场生成器
auto meshGenerator = Interface::FITKMeshGenInterface::getInstance()->getMeshSizeGenerator();
//根据分支条件等给定 t 的类型
Interface::FITKAbstractRegionMeshSize::RegionType  t /* = a type*/;
//创建局部尺寸场
currentObj = meshGenerator->createRegionMeshSize(t);
//加入管理器
meshSizeManager->appendDataObj(currentObj)
//****
//do something with currentObj
//****
```

2. 执行网格划分

执行网格划操作也是直接通过单例获取网格引擎驱动器，执行网格划分，需要注意的是，如果要想要实现网格划分执行完成之后自动进行网格处理，必须通过 mesherFinished 信号实现。示例代码如下：

```
// 获取单例
auto meshGen = Interface::FITKMeshGenInterface::getInstance();
// 网格划分驱动器
auto meshDriver = meshGen->getMesherDriver();
//设置参数
meshDriver->setValue("WorkDir", "XXXXXX");
//启动网格划分（多线程与多进程）
meshDriver->startMesher();
//网格划分完成之后，自动执行readMesh
connect(meshDriver, &Interface::FITKAbstractMesherDriver::mesherFinished, [this] { readMesh();});
```

3. 调用网格处理器

与上述两部分相同，网格处理器也是通过单例调用，2 示例代码中最后的 readMesh 函数就将调用网格处理器，简化后的代码如下：

```
// 获取单例
auto meshGen = Interface::FITKMeshGenInterface::getInstance();
// 读取网格
auto meshProcessor = meshGen->getMeshProcessor();
// 设置参数
meshProcessor->setValue("WorkDir", “XXXXXX”);
//启动数据处理
meshProcessor->start();
//****完成之后执行刷新渲染窗口等其他操作****
//*********************************
```