WorkStream是个命名空间，主要会用到它里面的模板函数，比如run函数，可用于共享式内存的多线程并行运算。

这个类用到intel的线程构建模块Threading Building Blocks(TBB)来负载平衡独立子范围到可用线程上去。

比如装配矩阵中，计算不同网格对应的局部贡献可以并行。但把局部贡献复制到全局矩阵的时候，有几点要注意：第一，几个线程不能同时进行写的操作，而是需要mutex进行同步写；其次，我们总是希望局部贡献被写入全局矩阵的顺序是一定的。因此，我们希望能保证每次只有一个线程进行写操作，且以一种稳定且可复现的顺序复制结果。

这个类就是为了完成这一目标的。它与由某个迭代器范围给出的对象流一起工作，运行一个函数，并行地作用在所有这些对象上；再把每个对象传给一个后处理函数，这些后处理函数有序地运行，按照它们在输入的迭代器范围中出现的顺序依次访问这些对象。

在内部，传给这个类的run()函数的范围会分成一系列“项目（items）”，之后会分布到允许的若干线程上去。一个“项目”就是我们将会在其上进行操作的迭代器范围的一个元素；例如，为了组装矩阵或者估计误差，一个项目就可以是一个网格。TBB模块决定创建多少线程（通常与处理器核心数量相同），但在任意给定时刻激活的项目数是由构造函数的参数决定的。它应该大于或等于处理器核心的数量——默认为当前系统核心数的4倍。

当工作线程是闲置状态或将成为闲置状态时，TBB模块发出请求，进行项目的创建。然后它被交给一个工作函数（worker function），通常是一个主类的成员函数。这些工作函数在若干线程上并行地运行，且它们作用在项目上并没有一定的顺序。

通常，工作函数需要一些额外的数据，比如FEValues对象，输入数据向量等，其中有些是不能在线程间被共享的。为此，run()函数会需要另外一个模板参数，ScratchData，它指定了一个保存在每个项目上的type对象，这个对象被线程用作私有数据，不会与其他线程进行共享。

此外，工作函数把它们计算的结果存储在模板类CopyData的对象中。然后这些被传给另外一个名为copier的函数，它们将会把这些结果转换为永久型的存储。比如，它可能会把工作函数计算得到的局部贡献的结果复制到全局矩阵中去。与工作函数相反，在任意给定时刻，只会有一个copier函数会运行，因而它可以安全地把局部贡献复制到全局矩阵，而不用使用mutex或类似的方式来锁定矩阵。另外，copier函数作用在CopyData对象上的顺序是与这些对象对应的项目被创建的顺序一致的。

一旦某个项目被copier函数处理，它就会被删除，而被用到的ScratchData和CopyData对象则被认为已经无用，或者可能在下次工作函数被激活时再被重复使用。

template<typename Worker , typename Copier , typename Iterator , typename ScratchData , typename CopyData >

void WorkStream::run (const std::vector< std::vector< Iterator > > & colored\_iterators,

Worker worker,

Copier copier,

const ScratchData & sample\_scratch\_data,

const CopyData & sample\_copy\_data,

const unsigned int queue\_length = 2\*MultithreadInfo::n\_threads(),

const unsigned int chunk\_size = 8

)

这是WorkStream下两个主要的函数之一。它的思想来源于Turcksin的paper。因此，它不是使用了begin和end迭代器的概念给出一个范围，而是使用了一个“上色”迭代器图的概念，也就是每个颜色代表了一些网格，写这些网格的贡献到全局矩阵不会发生冲突（及，这些网格不构成邻居关系）。每个“颜色”由网格的std::vectors来代表。这个函数的第一个参数，一个由一些网格集合构成的集合（由向量的向量来表示），通常通过调用GraphColoring::make\_graph\_coloring()函数来创建。

用作worker和copier对象的函数，要么是指向非成员函数的指针，或者是可通过操作符()进行调用的对象，比如，由std::bind创建的对象。

两个数据类型ScratchData和CopyData需要有可用的拷贝构造函数。ScratchData仅仅在worker函数中被使用，而CopyData则是从worker传给copier的对象。

queue\_length参数表示在给定时刻可激活的项目数。每个项目由输入流的chunk\_size个元素组成，这些元素之后会被worker和copier函数在同一线程上依次作用。

注：如果你的数据对象很大，或者他们的构造函数很费时，请记住生成了queue\_length个ScratchData对象的副本，及queue\_length\*chunk\_size个CopyData对象的副本。

template<typename Worker , typename Copier , typename Iterator , typename ScratchData , typename CopyData >

void WorkStream::run (const Iterator & begin,

const typename identity< Iterator >::type & end,

Worker worker,

Copier copier,

const ScratchData & sample\_scratch\_data,

const CopyData & sample\_copy\_data,

const unsigned int queue\_length = 2\*MultithreadInfo::n\_threads(),

const unsigned int chunk\_size = 8

)

template<typename MainClass , typename Iterator , typename ScratchData , typename CopyData >

void WorkStream::run (const Iterator & begin,

const typename identity< Iterator >::type & end,

MainClass & main\_object,

void(MainClass::\*)(const Iterator &, ScratchData &, CopyData &) worker,

void(MainClass::\*)(const CopyData &) copier,

const ScratchData & sample\_scratch\_data,

const CopyData & sample\_copy\_data,

const unsigned int queue\_length = 2\*MultithreadInfo::n\_threads(),

const unsigned int chunk\_size = 8

)