Aggregator Library

Documentation

Version 1.0

(Documentation created using Doxygen)

Class Project For COMS 4995

Language Library Design C++

RAKESH YARLAGADDA - ry2294

SHARAN SURYANARAYANAN - ss4951

SRIHARI SRIDHAR - ss4964

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

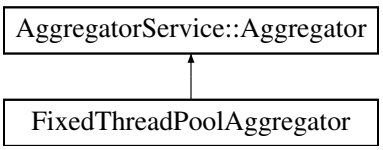
Chapter 3

Class Documentation

3.1 AggregatorService::Aggregator Class Reference

`#include <aggregator.h>`

Inheritance diagram for AggregatorService::Aggregator:



Public Member Functions

- virtual void addNode (std::shared_ptr< Runnable >)=0
- virtual void execute ()=0 throw (CyclicDependencyFoundException)

3.1.1 Detailed Description

Represents the Aggregator Interface which will used by clients to interact with the Aggregator Library.

3.1.2 Member Function Documentation

3.1.2.1 virtual void AggregatorService::Aggregator::addNode (std::shared_ptr< Runnable >) [pure virtual]

The addNode function is used to add a node/task into the graph. The user passes the shared_ptr of the task which he wants to add to the graph. The function takes the task and adds it into the dependency graph.

3.1.2.2 virtual void AggregatorService::Aggregator::execute ( ) throw CyclicDependencyFoundException) [pure virtual]

The execute function is used to process the graph and run the tasks added to the graph. The user can the execute function whenever he wants to run the Aggregator.

Implemented in FixedThreadPoolAggregator.

The documentation for this class was generated from the following file:

- AggregatorLibrary/AggregatorService/aggregator.h

3.2 AggregatorService::AggregatorFactory Class Reference

`#include <aggregator.h>`

Static Public Member Functions

- static std::shared_ptr< Aggregator > **newFixedThreadPoolAggregator** ()

3.2.1 Detailed Description

Represents the Factory which will be generate aggregators depending on the client's use-case.

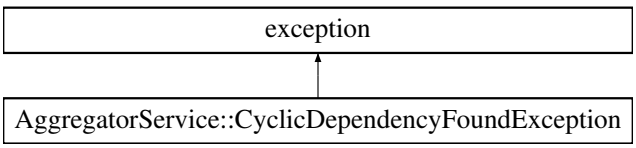
The documentation for this class was generated from the following files:

- AggregatorLibrary/AggregatorService/aggregator.h
- AggregatorLibrary/AggregatorService/aggregatorfactory.cpp

3.3 AggregatorService::CyclicDependencyFoundException Class Reference

`#include <aggregator.h>`

Inheritance diagram for AggregatorService::CyclicDependencyFoundException:



3.3.1 Detailed Description

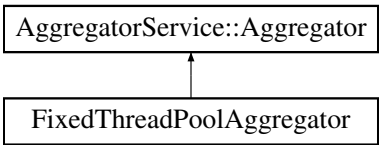
Exception thrown by Aggregator when the dependency graph is Cyclic.

The documentation for this class was generated from the following file:

- AggregatorLibrary/AggregatorService/aggregator.h

3.4 FixedThreadPoolAggregator Class Reference

Inheritance diagram for FixedThreadPoolAggregator:



Public Member Functions

- void **addNode** (shared_ptr< Runnable > node)
- void execute () throw (CyclicDependencyFoundException)

3.4.1 Member Function Documentation

3.4.1.1 void FixedThreadPoolAggregator::execute ( ) throw CyclicDependencyFoundException) [inline], [virtual]

The execute function is used to process the graph and run the tasks added to the graph. The user can the execute function whenever he wants to run the Aggregator.

Implements AggregatorService::Aggregator.

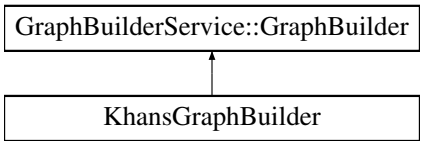
The documentation for this class was generated from the following file:

- AggregatorLibrary/AggregatorService/fixedthreadpoolaggregator.cpp

3.5 GraphBuilderService::GraphBuilder Class Reference

`#include <graphbuilder.h>`

Inheritance diagram for GraphBuilderService::GraphBuilder:



Public Member Functions

- virtual void **addVertex** (std::shared_ptr< Runnable >)=0
- virtual void **constructGraph** ()=0
- virtual bool **isGraphCyclic** ()=0
- virtual std::shared_ptr< Task > **getTask** (std::string)=0
- virtual std::queue< std::shared_ptr< Task > > **getIndependentTasks** ()=0
- virtual int **size** ()=0

3.5.1 Detailed Description

Represents the GraphBuilder interface which will be used by Aggregtors to create a Graph and construct edges.

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphBuilderService/graphbuilder.h

3.6 GraphBuilderService::GraphBuilderFactory Class Reference

`#include <graphbuilder.h>`

Static Public Member Functions

- static std::shared_ptr< GraphBuilder > **newKhansGraphBuilder** ()

3.6.1 Detailed Description

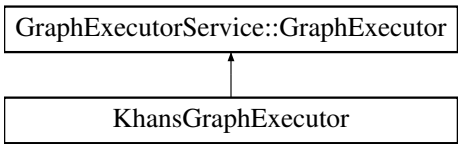
Factory for generating Graph builders

The documentation for this class was generated from the following files:

- AggregatorLibrary/GraphBuilderService/graphbuilder.h
- AggregatorLibrary/GraphBuilderService/graphbuilderfactory.cpp

3.7 GraphExecutorService::GraphExecutor Class Reference

Inheritance diagram for GraphExecutorService::GraphExecutor:



Public Member Functions

- virtual void execute (std::shared_ptr< GraphBuilder >)=0

3.7.1 Member Function Documentation

3.7.1.1 virtual void GraphExecutorService::GraphExecutor::execute (std::shared_ptr< GraphBuilder >) [pure virtual]

The execute function communicates with the TaskPoolExecutor to execute the graph. It takes a shared_ptr of a GraphBuilder and executes the graph.

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/graphexecutor.h

3.8 GraphExecutorService::GraphExecutorFactory Class Reference

Static Public Member Functions

- static std::shared_ptr< ThreadSafeQueue< std::shared_future< std::shared_ptr< Task > > > > **newSingleLockQueue** ()
- static std::shared_ptr< TaskPoolExecutor > **newQueueTaskExecutor** ()
- static std::shared_ptr< GraphExecutor > **newKhansGraphExecutor** ()
- static std::shared_ptr< ThreadPool > **newSingletonThreadPool** ()

The documentation for this class was generated from the following files:

- AggregatorLibrary/GraphExecutorService/graphexecutor.h
- AggregatorLibrary/GraphExecutorService/graphexecutorfactory.cpp

3.9 join_threads Class Reference

Public Member Functions

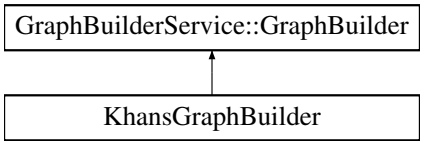
- **join_threads** (std::vector< std::thread > &threads_)

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/singletonthreadpool.cpp

3.10 KhansGraphBuilder Class Reference

Inheritance diagram for KhansGraphBuilder:



Public Member Functions

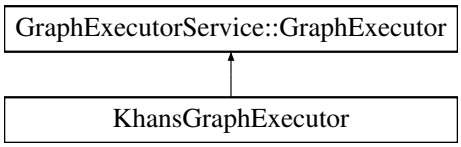
- void **addVertex** (shared_ptr< Runnable > vertex)
- int **size** ()
- void **constructGraph** ()
- bool **isGraphCyclic** ()
- queue< shared_ptr< Task > > **getIndependentTasks** ()
- shared_ptr< Task > **getTask** (string label)

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphBuilderService/khansgraphbuilder.cpp

3.11 KhansGraphExecutor Class Reference

Inheritance diagram for KhansGraphExecutor:



Public Member Functions

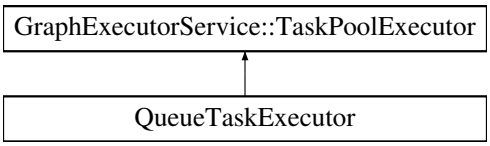
- void **execute** (shared_ptr< GraphBuilder > graph)

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/khansgraphexecutor.cpp

3.12 QueueTaskExecutor Class Reference

Inheritance diagram for QueueTaskExecutor:



Public Member Functions

- void **submit** (shared_ptr< Task > task)
- shared_ptr< Task > **take** ()

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/queuetaskexecutor.cpp

3.13 AggregatorService::Runnable Class Reference

`#include <aggregator.h>`

Public Member Functions

- virtual std::string getLabel () const =0
- virtual std::unordered_set< std::string > getDependencies () const =0
- virtual void run (std::unordered_map< std::string, std::shared_ptr< Runnable >>)=0

3.13.1 Detailed Description

Represents the interface that clients have to implement for each task. They should override the getLabel, get↩ Dependencies and run functions. The Aggregator will take the implemented class, draw the dependency graph and will execute the task.

3.13.2 Member Function Documentation

3.13.2.1 virtual std::unordered_set<std::string> AggregatorService::Runnable::getDependencies ( ) const [pure virtual]

The getDependencies function is used to get the set of dependencies of the node. The user has to return an unordered_set of std::string specifying the labels of all the classes on which this class depends upon.

3.13.2.2 virtual std::string AggregatorService::Runnable::getLabel ( ) const [pure virtual]

The getLabel is a const function returns the name of the task. The user can set anyname for the class, but would recommend setting it the actual name of the class. This function would be internally used while constructing and executing the graph.

3.13.2.3 virtual void AggregatorService::Runnable::run (std::unordered_map< std::string, std::shared_ptr< Runnable >>) [pure virtual]

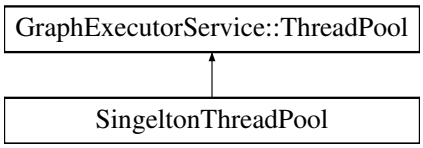
The run method gets an unordered_map of the task labels and the objects as input. This input is internally provided by the Aggregator, the map contains the labels and objects of the tasks on which this task is dependent on. The user writes his user code which has to be implemented for this task inside the run, function.

The documentation for this class was generated from the following file:

- AggregatorLibrary/AggregatorService/aggregator.h

3.14 SingeltonThreadPool Class Reference

Inheritance diagram for SingeltonThreadPool:



Public Member Functions

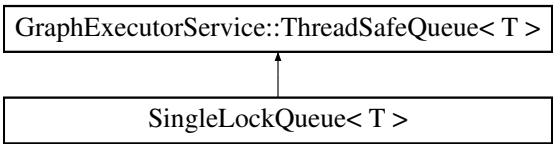
- void **submit** (pair< shared_ptr< Task >, return_queue > task)

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/singletonthreadpool.cpp

3.15 SingleLockQueue< T > Class Template Reference

Inheritance diagram for SingleLockQueue< T >:



Public Member Functions

- void push (T task)
- T wait_and_pop ()
- bool empty () const
- int size () const

3.15.1 Member Function Documentation

3.15.1.1 template<typename T> bool SingleLockQueue< T >::empty ( ) const [inline], [virtual]

The empty function checks if the queue is empty or not.

Implements GraphExecutorService::ThreadSafeQueue< T >.

3.15.1.2 template<typename T> void SingleLockQueue< T >::push (T) [inline], [virtual]

The push function allows the program to add an element on to the threadsafequeue.

Implements GraphExecutorService::ThreadSafeQueue< T >.

3.15.1.3 template<typename T> int SingleLockQueue< T >::size ( ) const [inline], [virtual]

The size function returns the number of elements in the queue.

Implements GraphExecutorService::ThreadSafeQueue< T >.

3.15.1.4 template<typename T> T SingleLockQueue< T >::wait_and_pop ( ) [inline], [virtual]

The wait_and_pop function pops the top element and returns it. In case the queue is empty and a thread is requesting for wait and pop, then the thread must be asked to wait and notified as soon as an element is added into the queue.

Implements GraphExecutorService::ThreadSafeQueue< T >.

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/singlelockqueue.cpp

3.16 GraphBuilderService::Task Class Reference

`#include <graphbuilder.h>`

Public Member Functions

- **Task** (std::shared_ptr< Runnable >)
- std::unordered_set< std::string > **getInDegree** ()
- void **addInDegree** (std::string, std::shared_ptr< Runnable >)
- void **removeInDegree** (std::string)
- bool **isInDegreeEmpty** ()
- std::unordered_set< std::string > **getOutDegree** ()
- void **addOutDegree** (std::string)
- std::shared_ptr< Runnable > **getNode** ()
- std::string **getLabel** ()
- void **run** ()

3.16.1 Detailed Description

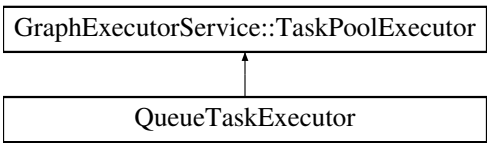
Represents the Task wrapper class that the aggregator library uses to wrap a Client's task.

The documentation for this class was generated from the following files:

- AggregatorLibrary/GraphBuilderService/graphbuilder.h
- AggregatorLibrary/GraphBuilderService/task.cpp

3.17 GraphExecutorService::TaskPoolExecutor Class Reference

Inheritance diagram for GraphExecutorService::TaskPoolExecutor:



Public Member Functions

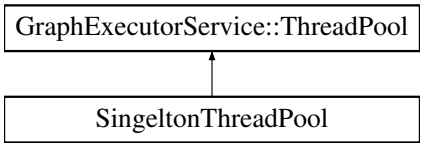
- virtual void **submit** (std::shared_ptr< Task >)=0
- virtual std::shared_ptr< Task > **take** ()=0

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/graphexecutor.h

3.18 GraphExecutorService::ThreadPool Class Reference

Inheritance diagram for GraphExecutorService::ThreadPool:



Public Member Functions

- virtual void submit (std::pair< std::shared_ptr< Task >, return_queue >)=0

3.18.1 Member Function Documentation

3.18.1.1 virtual void GraphExecutorService::ThreadPool::submit (std::pair< std::shared_ptr< Task >, return_queue >) [pure virtual]

The submit functions takes a pair of shared_ptr of Task and shared_ptr to a threadsafe queue and adds this pair into the task_queue.

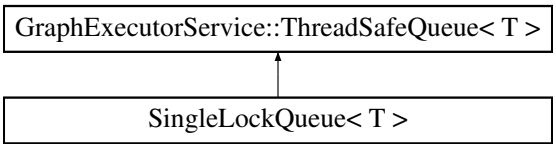
The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/graphexecutor.h

3.19 GraphExecutorService::ThreadSafeQueue< T > Class Template Reference

`#include <graphexecutor.h>`

Inheritance diagram for GraphExecutorService::ThreadSafeQueue< T >:



Public Member Functions

- virtual void push (T)=0
- virtual T wait_and_pop ()=0
- virtual bool empty () const =0
- virtual int size () const =0

3.19.1 Detailed Description

template<typename T>
class GraphExecutorService::ThreadSafeQueue< T >

Interface for a ThreadSafeQueue which is used by the ThreadPool and used for implementing task queue.

3.19.2 Member Function Documentation

3.19.2.1 template<typename T > virtual bool GraphExecutorService::ThreadSafeQueue< T >::empty ( ) const [pure virtual]

The empty function checks if the queue is empty or not.

Implemented in SingleLockQueue< T >.

3.19.2.2 template<typename T > virtual void GraphExecutorService::ThreadSafeQueue< T >::push (T) [pure virtual]

The push function allows the program to add an element on to the threadsafequeue.

Implemented in SingleLockQueue< T >.

3.19.2.3 template<typename T > virtual int GraphExecutorService::ThreadSafeQueue< T >::size ( ) const [pure virtual]

The size function returns the number of elements in the queue.

Implemented in SingleLockQueue< T >.

3.19.2.4 template<typename T > virtual T GraphExecutorService::ThreadSafeQueue< T >::wait_and_pop ( ) [pure virtual]

The wait_and_pop function pops the top element and returns it. In case the queue is empty and a thread is requesting for wait and pop, then the thread must be asked to wait and notified as soon as an element is added into the queue.

Implemented in SingleLockQueue< T >.

The documentation for this class was generated from the following file:

- AggregatorLibrary/GraphExecutorService/graphexecutor.h

Index