breadpool Documentation

Release 0.0.5

Author

December 10, 2015

Contents

Contents:

CHAPTER 1

breadpool package

1.1 Subpackages

1.2 Submodules

1.3 breadpool.pool module

BreadPool intends to simply provide implementations for a thread pool and a scheduled executor, with easy to use interfaces and thread safety. Yes, it is a simple code to write your own implementations for these, however it can be a lot easier if they come in a *pip install*.

class `breadpool.pool.`**`AbstractRunnable`**

Bases: `object`

The tasks that should be executed using the ThreadPool or the ScheduledJobExecutor should be of a sub class of AbstractRunnable. Extend AbstractRunnable and write the task implementation inside the execute() method.

Take a look at the EasyTask implementation of AbstractRunnable for an example.

`execute``()`

class `breadpool.pool.`**`EasyTask`**`(`*function*, **args*, ***kwargs*`)`

Bases: *`breadpool.pool.AbstractRunnable`*

This is an implementation of the AbstractRunnable class which accepts a function to be executed.

EasyTask allows to easily submit functions as runnable tasks to the ThreadPool or the ScheduledJobExecutor.

`execute``()`

Executes the given function passing the given arguments and keyword arguments to the function :return:

class `breadpool.pool.`**`ScheduledJobExecutor`**`(`*task*, *thread_pool*, *delay*, *name*`)`

Bases: `threading.Thread`

A Scheduled executor which periodically executes a given task. This should be given a thread pool to work on. When a task is submitted to the scheduled task, it will repeatedly, periodically execute that task using the provided thread pool.

`run``()`

`terminate``()`

Sets the terminate event on the scheduled executor :return:

class `breadpool.pool.`**`ThreadPool`**(*size*, *name*, *daemon=False*, *polling_timeout=60*)

Bases: `object`

The ThreadPool class offers a simple Thread pool implementation for Python. It uses Python's Queues to coordinate tasks among a set of worker threads. The specified number of threads are created when the thread pool is created, and is maintained so that they do not increase more than that number.

`enqueue`(*task*)

Adds the specified task to the task queue for a worker thread to start working on it. If any free worker threads are waiting on the task queue, it will immediately pick up this task.

Parameters **`task`** – The task to be added to the task queue.

Returns

`get_pool_size`()

Returns the size of the thread pool.

Returns The size of the thread pool

Return type int

`terminate`()

Waits for the task queue to finish and sends the terminate event for all the worker threads. :return:

1.4 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index