coworker Documentation

Release 0.0.1

Max Zheng

Jun 06, 2018

Contents

Contents:

CHAPTER 1

coworker

Generic worker that performs concurrent tasks using coroutine.

CHAPTER 2

Quick Start Tutorial

Define how a task is performed and create the worker:

```
from coworker import Coworker

class SquareWorker(Coworker):
    async def do_task(self, task):
        return task * task

worker = SquareWorker(max_concurrency=5)     # Only 5 tasks will run concurrently
                                             # As do_task is fast, 35,000 tasks can be␣
↪done in 1 second.
```

To run in the background forever and add tasks:

```
import asyncio

async def background_worker_example():
    # Start worker / Run in background
    asyncio.ensure_future(worker.start())

    # Mulitiple tasks
    tasks = list(range(100))
    results = await asyncio.gather(*worker.add_tasks(tasks))
    print(results)  # results = [0, 1, 4, 9, ...]

    # Single task
    result = await worker.add_tasks(2)
    print(result)   # result = 4

    # Stop worker
    await worker.stop()

# Run async usage example
asyncio.get_event_loop().run_until_complete(background_worker_example())
```

To run for a list of tasks and stop worker when finished:

```
task_futures = asyncio.get_event_loop().run_until_complete(worker.start([1, 2, 3]))
print([t.result() for t in task_futures])    # [1, 4, 9]
```

CHAPTER 3

Links & Contact Info

Documentation: http://coworker.readthedocs.org

PyPI Package: https://pypi.python.org/pypi/coworker
GitHub Source: https://github.com/maxzheng/coworker
Report Issues/Bugs: https://github.com/maxzheng/coworker/issues

Connect: https://www.linkedin.com/in/maxzheng
Contact: maxzheng.os @t gmail.com

CHAPTER 4

API Documentation

4.1 Coworker

class coworker.**Coworker**(*max_concurrency=10*, *sliding_window=True*)

Generic worker to perform concurrent tasks using coroutine IO loop.

Initialize worker

Parameters

- **max_concurrency** (*int*) – How many tasks can be done at the same time. Defaults to 10.
- **sliding_window** (*bool*) – Start a task as soon as there is an available slot based on concurrency instead of waiting for all concurrent tasks to be completed first.

add_tasks(*tasks*)

Add task(s) to queue

Parameters **tasks** (*object | list*) – A single or list of task(s) to add to the queue.

Returns

If a single task is given, then returns a single task future that will contain result from `self.do_task()`. If a list of tasks is given, then a list of task futures, one for each task.

Note that if hash(task) is the same as another/existing task, the same future will be returned, and the task is only performed once. If it is desired to perform the same task multiple times / distinctly, then the task will need to be wrapped in another object that has a unique hash.

available_slots

Number of available slots to do tasks based on concurrency and window settings

cancel_task(*task*)

Cancel a task

do_task(*task*)

Perform the task. Sub-class should override this to do something more meaningful.

idle
Worker has nothing to do and is doing nothing

on_finish()
Invoked after worker completes all tasks before exiting worker. Subclass should override if needed.

on_finish_task(*task*, *result*)
" Invoked after the task is completed. Subclass should override if needed.

Parameters

- **task** – Task that was finished
- **result** – Return value from `self.do_task(task)()`

on_start()
Invoked before worker starts. Subclass should override if needed.

on_start_task(*task*)
Invoked before starting the task. Subclass should override if needed.

Parameters **task** – Task that will start

start(*tasks=None*)
Start the worker.

Parameters **tasks** (*list*) – List of tasks to do. If provided, worker will exit immediately after all tasks are done. If that's not desired, use `self.add_task()` instead.

Returns List of futures for each task in the same order.

stop()
Stop the worker by canceling all tasks and then wait for worker to finish.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index