aqueue

Tim Martin

Oct 17, 2022

CONTENTS

`aqueue` is an async task queue with live progress display.

You put items in, and they get processed, possibly creating more items which get processed, and so on, until all items are completed. A typical use case would be to scrape a website.

Note: `aqueue`, or any asynchronous framework, is only going to be helpful if you're performing **I/O-bound** work.

CHAPTER

ONE

INSTALLATION

aqueue is a Python package hosted on PyPI. The recommended installation method is pip-installing into a virtual environment:

```
pip install aqueue
```

CHAPTER

TWO

GETTING STARTED

There's two things you need to do to use aqueue:

1. Implement your *Item* subclasses.
2. *Start your queue* with one of those items.

CHAPTER

THREE

EXAMPLE

```python
import aqueue


class RootItem(aqueue.Item):
    async def process(
        self, enqueue: aqueue.EnqueueFn, set_desc: aqueue.SetDescFn
    ) -> None:
        # display what we're doing in the worker status panel
        set_desc("Processing RootItem")

        # make an HTTP request, parse it, etc
        ...

        # when you discover more items you want to process, enqueue them:
        for _ in range(5):
            enqueue(ChildItem())

    async def after_children_processed(self) -> None:
        # run this method when this Item and all other Items it enqueued are done
        print("All done!")


class ChildItem(aqueue.Item):

    # track the enqueueing and completion of these items in the overall panel
    track_overall: bool = True

    async def process(
        self, enqueue: aqueue.EnqueueFn, set_desc: aqueue.SetDescFn
    ) -> None:
        set_desc("Processing ChildItem")


if __name__ == "__main__":
    aqueue.run_queue(
        initial_items=[RootItem()],
        num_workers=2,
    )
```

CHAPTER

FOUR

ITEMS

Items are your units of work. They can represent whatever you'd like, such as parts of a website that you're trying to scrape: an item for the index page, for subpages, for images, etc.

Each item must be an instance of a subclass of `aqueue.Item`. Imperatively, you must implement the `aqueue.Item.process` method, which defines the work of the item, such as making an HTTP request, parsing it, downloading something, etc.

Note: `aqueue` is built on top of Trio, and, therefore, you may only use Trio-compatible async primitives inside `Item` methods.

Fundamentally, items can make other items to be processed later. To enqueue them, use the `enqueue` method passed to the `process` method.

As a rule of thumb, you should make a new item class whenever you notice a one-to-many relationship. For example, "this *one* page has *many* images I want to download".

CHAPTER

FIVE

STARTING YOUR QUEUE

Once you’ve implemented some `aqueue.Item` classes, start your queue to kick things off.

CHAPTER

SIX

SHARING STATE

Often, its beneficial to share state between the items. Using the website scrape example again, you may want to keep track of the URLs you've visited so you don't scrape them twice.

If this is needed, simply keep a global set/dict/list and store a key for the item. For example, a URL string may be a good key.

If you don't want to or can't use a global variable, consider a `contextvars.ContextVar`.

CHAPTER

SEVEN

PERSISTING STATE

During development, its probably likely that your program will crash after doing some work. For example, maybe your HTTP request timed out or you had a bug in your HTML parsing.

It's a shame to lose that work that's been done. So, if you're looking for a really handy way to persist state across runs, check out the built-in `shelve` module. It's like a dict that automatically saves to a file each time you set a key in it.

CHAPTER

EIGHT

OTHER COOL THINGS

The API is fully docstringed and type-hinted