consumers

Aug 24, 2018

Contents

Consumers is a simple, flexible way to parallelize processing in Python.

CHAPTER 1

Quick Links

- PyPi
- Source Code
- Documentation

1.1 Examples

1.1.1 Multiple Processes

Consumers run in separate processes.

```
import os

from consumers import Pool


def printer(numbers):
    pid = os.getpid()
    for number in numbers:
        print(pid, number)


pool = Pool(printer)

for number in range(5):
    pool.put(number)

pool.join()
```

```
3320 0
3320 2
```

(continues on next page)

(continued from previous page)

```
3321 1
3323 3
3324 4
```

1.1.2 Getting Results

Consumers can return data the end of execution.

```
from consumers import Pool


def concatenate(letters):
    return '-'.join(letters)


with Pool(concatenate, quantity=2) as pool:
    for letter in 'abcdef':
        pool.put(letter)

print(pool.results)
```

```
('b-c', 'a-d-e-f')
```

1.1.3 Multiple Data

Multiple pieces of data can be queued and consumed with ease.

```
from consumers import Pool


def print_letter_index(items):
    for index, letter in items:
        print('{} is at index {}'.format(letter, index))


with Pool(print_letter_index) as pool:
    for i, v in enumerate('abcdef'):
        pool.put(i, v)
```

```
a is at index 0
c is at index 2
b is at index 1
d is at index 3
e is at index 4
f is at index 5
```

1.1.4 Consumer Configuration

Consumers can be configured with positional and keyword arguments.

```
from consumers import Pool


def print_host(numbers, host, port=80):
    connection = '{}:{}'.format(host, port)

    for number in numbers:
        print(connection, number)


with Pool(print_host, 1, args=('remote',)) as pool:
    for i in range(3):
        pool.put(i)

with Pool(print_host, 1, args=('local',), kwargs={'port': 8123}) as pool:
    for i in range(3):
        pool.put(i)
```

```
remote:80 0
remote:80 1
remote:80 2
local:8123 0
local:8123 1
local:8123 2
```

1.1.5 Cross-Pool Communication

Consumers can put data into other pools.

```
from consumers import Pool


def square_sums(numbers, logger_pool):
    total = 0
    for number in numbers:
        total += number * number
    logger_pool.put(total)


def logger(totals):
    for total in totals:
        print('A consumer has finished with a total of', total)


logger_pool = Pool(logger, 1)
square_sums_pool = Pool(square_sums, args=(logger_pool,))

with logger_pool, square_sums_pool:
    for i in range(500):
        square_sums_pool.put(i)
```

```
A consumer has finished with a total of 10292214
A consumer has finished with a total of 10354035
A consumer has finished with a total of 10416304
A consumer has finished with a total of 10479197
```

1.2 API

1.2.1 Pool

class consumers.**Pool**(*consumer*, *quantity=None*, *args=None*, *kwargs=None*)

A *Pool* is responsible for the lifecycle of separate consumer processes and the queue upon which they consume from.

When used as a context manager, entering the context returns the pool object and exiting invokes its *join()* method.

Parameters

- **consumer** (*callable*) – A callable which will consume from the pool's queue.

  A generator will be passed as the first argument of the consumer. It will continue to yield the next item from the queue until the queue is both closed and empty.

  Additional parameters may be specified with *args* and *kwargs*.

- **args** (*tuple*) – Positional arguments to pass to the consumer function. Will take position after the generator argument provided by the Pool.
- **kwargs** (*dict*) – Keyword arguments to pass to the consumer function.
- **quantity** (*int*) – The number of consumer processes to create.

  Defaults to the number of CPUs in the system as determined by multiprocessing.cpu_count().

close()

Prevent any more items from being added into the pool's queue and inform consumers to shutdown after the remaining items have been processed. Non-blocking.

join()

Block until the pool's queue has drained and consumers have stopped.

Sets *results*.

Raises ***consumers.ConsumerError*** – One or more of the consumers did not cleanly exit.

put(**args*)

Enqueue all **args* as a single item in the queue.

results

Results from the consumers.

Only available after *join()* has completed.

Returns A tuple with a size of as many consumers in the pool.

Raises ***consumers.PoolError*** – Results are not available at this time.

terminate()

Terminate the consumer processes.

1.2.2 Exceptions

exception consumers.**ConsumerError**

An exception in a consumer occurred.

exception consumers.**PoolError**
An error occurred accessing a pool.

1.3 Changelog

1.3.1 0.6.1 (2018-04-02)

- Fix documentation.

1.3.2 0.6.0 (2018-03-03)

- Change consumer to be any callable type.
- Add consumer arg and kwarg specification.
- Change consumer process naming.

1.3.3 0.5.0 (2018-01-31)

- Remove `Pool.start()` in favor of starting the pool upon Pool instantiation.
- Rename `Pool.stop()` to `Pool.join()`
- Add `Pool.close()` to drain the queue and shutdown consumer processes in the background.
- Add `Pool.terminate()` to kill the consumer processes.
- Refactor a few things.

1.3.4 0.4.0 (2018-01-27)

- Essentially an API rewrite on what consumers are and how they consume.

1.3.5 0.3.0 (2018-01-25)

- Added queue results.
- Removed mechanism for queues to manage related queues.

1.3.6 0.2.1 (2018-01-23)

- Fixed consumers being created in the master process before being forked into separate processes.

1.3.7 0.2.0 (2018-01-22)

- Added mechanism for queues to manage related queues.
- Restructured package.

1.3.8 0.1.0 (2018-01-21)

- Initial release.

Index