cooperative Documentation

Release 0.1.10

John W Lockwood IV

April 11, 2015

Contents:

CHAPTER 1

Getting Started with cooperative

Write non-blocking computationally expensive code to go along with non-blocking io, without having to think about everything in callbacks.

`batch_accumulate` will iterate over a generator in batches, yielding to other iterators passed into *twisted.internet.task.cooperate*

CHAPTER 2

Examples

2.1 Write computation code to cooperative.

```
#!/usr/bin/env python
# _*_ coding: utf-8 _*_
from operator import add

import sys
from twisted.internet import defer
from twisted.internet.task import react
from twisted.python import log

from cooperative import batch_accumulate


def expensive(number):
    log.msg("starting {}".format(number))
    for value in range(100000):
        if 25000 == value:
            log.msg("1/4 for {}".format(number))
        if 50000 == value:
            log.msg("1/2 for {}".format(number))
        if 75000 == value:
            log.msg("3/4 for {}".format(number))
        yield number * value / 3.0


@defer.inlineCallbacks
def do_some_expensive_things(number):
    """
    Perform one expensive computation cooperatively with any
     other iterator passed into twisted's cooperate, then
     use it's result to pass into the second computation.

    :param number:
    :return:
    """
    result = yield batch_accumulate(1000, expensive(number))
    total = reduce(add, result, 0)
    log.msg("first for {}: {}".format(number, total))

    result = yield batch_accumulate(1000, expensive(int(total/1e9)))
    total = reduce(add, result, 0)
    log.msg("second for {}: {}".format(number, total))
    defer.returnValue(total)


def main(reactor):
    d1 = do_some_expensive_things(54.0)
    d2 = do_some_expensive_things(42)
    d3 = do_some_expensive_things(10)
    d4 = do_some_expensive_things(34)

    # Enqueue events to simulate handling external events
    d5 = defer.Deferred().addCallback(log.msg)
    reactor.callLater(0.3, d5.callback, "########## simulated request 1 ############")

    d6 = defer.Deferred().addCallback(log.msg)
    reactor.callLater(0.5, d6.callback, "########## sim request 2 ############")

    d7 = defer.Deferred().addCallback(log.msg)
    reactor.callLater(1.0, d7.callback, "########## simulated request 3 ############")

    return defer.gatherResults([d1, d2, d3, d4, d5, d6, d7]).addCallback(log.msg)

if __name__ == "__main__":
    log.startLogging(sys.stdout)
    react(main, [])
```

CHAPTER 3

cooperative Package

3.1 `cooperative` Package

class `cooperative.`**`ValueBucket`**

Bases: `object`

Produces a callable that accumulates all non-None values it is called with in order.

The contents may be accessed or collected and drained, to make room for new content.

`contents``()`

Returns contents

`drain_contents``()`

Starts a new collection to accumulate future contents and returns all of existing contents.

`cooperative.`**`accumulate`**`(`*a_generator*, *cooperator=None*`)`

Start a Deferred whose callBack arg is a deque of the accumulation of the values yielded from a_generator.

Parameters **a_generator** – An iterator which yields some not None values.

Returns A Deferred to which the next callback will be called with the yielded contents of the generator function.

`cooperative.`**`accumulation_handler`**`(`*stopped_generator*, *spigot*`)`

Drain the contents of the bucket from the spigot.

Parameters

- **stopped_generator** – Generator which as stopped
- **spigot** – a Bucket.

Returns The contents of the bucket.

`cooperative.`**`batch_accumulate`**`(`*max_batch_size*, *a_generator*, *cooperator=None*`)`

Start a Deferred whose callBack arg is a deque of the accumulation of the values yielded from a_generator which is iterated over in batches the size of max_batch_size.

It should be more efficient to iterate over the generator in batches and still provide enough speed for non-blocking execution.

Parameters

- **max_batch_size** – The number of iterations of the generator to consume at a time.
- **a_generator** – An iterator which yields some not None values.

Returns A Deferred to which the next callback will be called with the yielded contents of the generator function.

3.2 `_meta` Module

3.3 Subpackages

3.3.1 tests Package

`test_cooperative` Module

class `cooperative.tests.test_cooperative.`**`Doer`**(*own_reactor*, *own_cooperator*)

Bases: `object`

`count` = **0**

`run`(**args*, ***kwargs*)

Cooperatively iterator over two iterators consecutively and the result of the final one is returned.

Returns

class `cooperative.tests.test_cooperative.`**`TestAccumulate`**(*methodName='runTest'*)

Bases: `twisted.trial._asynctest.TestCase`

`test_accumulate`(**args*, ***kwargs*)

Ensure that within an inline callback function, a accumulate wrapped generator yields the result of the output of the generator.

Returns

`test_failure`(**args*, ***kwargs*)

Ensure that within an inline callback function, a accumulate based function yields the result if it's cooperative generator.

Since and_the_winner_is is designed to always log and error, Ensure one IndexError is logged.

Returns

`test_multi_deux_batched`(**args*, ***kwargs*)

Ensure multiple inline callback functions will run cooperatively.

Ensure the result of gatherResults can be chained together in order.

Ensure cooperatively run generators will complete no matter the length.

Ensure the longest one will continue to iterate after the others run out of iterations.

Ensure those called with batch_accumulate will iterate over the generator in batches the size of max_size.

Returns

`test_multi_deux_chain`(**args*, ***kwargs*)

Ensure multiple inline callback functions will run cooperatively.

Ensure the result of gatherResults can be chained together in order.

Ensure cooperatively run generators will complete no matter the length.

Ensure the longest one will continue to iterate after the others run out of iterations.

Returns

`test_multi_winner``(`**args*, ***kwargs*`)`
Ensure multiple inline callback functions will run cooperatively.

Returns

`test_multi_winner_chain``(`**args*, ***kwargs*`)`
Ensure multiple inline callback functions will run cooperatively.

Ensure the result of gatherResults can be chained together in order.

Returns

`test_trice_winner``(`**args*, ***kwargs*`)`
Ensure multiple inline callback functions will run cooperatively.

Returns

class `cooperative.tests.test_cooperative.`**`TestHandler`**`(`*methodName='runTest'*`)`
Bases: `twisted.trial._asynctest.TestCase`

`test_accumulation_handler``()`
Ensure the return value of accumulation_handler is the contents of a Bucket instance with it's contents drained.

Returns

class `cooperative.tests.test_cooperative.`**`TestOwnCooperator`**`(`*methodName='runTest'*`)`
Bases: `twisted.trial._asynctest.TestCase`

`setUp``()`
Create a reactor and Cooperator that can be controlled.

Instantiate a Doer with the reactor and cooperator.

Create a Looping Call and set it's clock to the reactor.

Returns

`tearDown``()`

`test_control_coop``()`
Ensure control of own cooperator.

Returns

`cooperative.tests.test_cooperative.`**`i_get_tenth_11`**`(`*value*`)`
Yield the tenth and eleventh item of value.

Parameters **value** –

Returns

`cooperative.tests.test_cooperative.`**`run_some_with_error`**`(`**args*, ***kwargs*`)`

Cooperatively iterator over two iterators consecutively, but the second one will always raise an IndexError, which is caught, logged and a message is returned.

Returns

`cooperative.tests.test_cooperative.`**`run_some_without_error`**`(`**args*, ***kwargs*`)`
Cooperatively iterator over two iterators consecutively and the result of the final one is returned.

Parameters **value** – Any sequence.

Returns

CHAPTER 4

tests Package

4.1 `test_cooperative` Module

class `cooperative.tests.test_cooperative.`**`Doer`**(*own_reactor*, *own_cooperator*)

Bases: `object`

`count` = **0**

`run`(**args*, ***kwargs*)

Cooperatively iterator over two iterators consecutively and the result of the final one is returned.

Returns

class `cooperative.tests.test_cooperative.`**`TestAccumulate`**(*methodName='runTest'*)

Bases: `twisted.trial._asynctest.TestCase`

`test_accumulate`(**args*, ***kwargs*)

Ensure that within an inline callback function, a accumulate wrapped generator yields the result of the output of the generator.

Returns

`test_failure`(**args*, ***kwargs*)

Ensure that within an inline callback function, a accumulate based function yields the result if it's cooperative generator.

Since and_the_winner_is is designed to always log and error, Ensure one IndexError is logged.

Returns

`test_multi_deux_batched`(**args*, ***kwargs*)

Ensure multiple inline callback functions will run cooperatively.

Ensure the result of gatherResults can be chained together in order.

Ensure cooperatively run generators will complete no matter the length.

Ensure the longest one will continue to iterate after the others run out of iterations.

Ensure those called with batch_accumulate will iterate over the generator in batches the size of max_size.

Returns

`test_multi_deux_chain`(**args*, ***kwargs*)

Ensure multiple inline callback functions will run cooperatively.

Ensure the result of gatherResults can be chained together in order.

Ensure cooperatively run generators will complete no matter the length.

Ensure the longest one will continue to iterate after the others run out of iterations.

Returns

`test_multi_winner`(**args*, ***kwargs*)
Ensure multiple inline callback functions will run cooperatively.

Returns

`test_multi_winner_chain`(**args*, ***kwargs*)
Ensure multiple inline callback functions will run cooperatively.

Ensure the result of gatherResults can be chained together in order.

Returns

`test_trice_winner`(**args*, ***kwargs*)
Ensure multiple inline callback functions will run cooperatively.

Returns

class `cooperative.tests.test_cooperative.`**`TestHandler`**(*methodName='runTest'*)
Bases: `twisted.trial._asynctest.TestCase`

`test_accumulation_handler`()
Ensure the return value of accumulation_handler is the contents of a Bucket instance with it's contents drained.

Returns

class `cooperative.tests.test_cooperative.`**`TestOwnCooperator`**(*methodName='runTest'*)
Bases: `twisted.trial._asynctest.TestCase`

`setUp`()
Create a reactor and Cooperator that can be controlled.

Instantiate a Doer with the reactor and cooperator.

Create a Looping Call and set it's clock to the reactor.

Returns

`tearDown`()

`test_control_coop`()
Ensure control of own cooperator.

Returns

`cooperative.tests.test_cooperative.`**`i_get_tenth_11`**(*value*)
Yield the tenth and eleventh item of value.

Parameters value –

Returns

`cooperative.tests.test_cooperative.`**`run_some_with_error`**(**args*, ***kwargs*)

Cooperatively iterator over two iterators consecutively, but the second one will always raise an IndexError, which is caught, logged and a message is returned.

Returns

`cooperative.tests.test_cooperative.`**`run_some_without_error`**`(`**args*, ***kwargs*`)`

Cooperatively iterator over two iterators consecutively and the result of the final one is returned.

Parameters **value** – Any sequence.

Returns

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

Index