YieldPoints Documentation

Release 0.1.+

A. Jesse Jiryu Davis

Oct 05, 2017

Contents

Simple extensions to Tornado's gen module.

CHAPTER 1

Examples

Use *WaitAny* to begin two tasks and handle their results in the order completed:

```
>>> @gen.coroutine
... def f():
...     callback0 = yield gen.Callback(0)
...     callback1 = yield gen.Callback(1)
...
...     # Fire callback1 soon, callback0 later
...     IOLoop.current().add_timeout(
...         timedelta(seconds=0.1), partial(callback1, 'foo'))
...
...     IOLoop.current().add_timeout(
...         timedelta(seconds=0.2), partial(callback0, 'bar'))
...
...     keys = set([0, 1])
...     while keys:
...         key, result = yield yieldpoints.WaitAny(keys)
...         print 'key:', key, ', result:', result
...         keys.remove(key)
...
>>> IOLoop.current().run_sync(f)
key: 1 , result: foo
key: 0 , result: bar
```

If you begin a task but don't wait for it, use *Cancel* or to *CancelAll* avoid a `LeakedCallbackError`:

```
>>> @gen.coroutine
... def f():
...     yield gen.Callback('key') # never called
...     yield yieldpoints.Cancel('key')
...
>>> IOLoop.current().run_sync(f)
```

Wait with a timeout. *WithTimeout* can take a `YieldPoint` or a key name as its second argument:

```
>>> @gen.coroutine
... def f():
...     callback = yield gen.Callback('key') # never called
...     try:
...         key, result = yield yieldpoints.WithTimeout(
...             timedelta(seconds=0.1), 'key')
...     except yieldpoints.TimeoutException:
...         print 'Timeout!'
...
>>> IOLoop.current().run_sync(f)
Timeout!
```

CHAPTER 2

Contents

Extended YieldPoints example

An example for *WaitAny*, *WithTimeout*, and *CancelAll*: download several web pages at once and take action as each completes. After 0.5 seconds, stop waiting.

```
import time

from tornado import web, gen, ioloop, httpclient

import yieldpoints


class PageRaceHandler(web.RequestHandler):
    """A 'page race': start downloading many pages at once, and see how long
    each takes.
    """
    @gen.coroutine
    def get(self):
        urls = set([
            'http://google.com', 'http://apple.com', 'http://microsoft.com',
            'http://amazon.com'])

        self.write('<table border="1">')

        start = time.time()
        def duration():
            return time.time() - start

        # Set max_clients so all fetches can happen at once
        client = httpclient.AsyncHTTPClient(max_clients=len(urls))

        # Start all the fetches
        for url in urls:
            client.fetch(url, callback=(yield gen.Callback(url)))
```

```
        # Handle them as they complete
        pending_urls = urls.copy()
        while pending_urls:
            try:

                url, response = yield yieldpoints.WithTimeout(
                    start + 0.5, yieldpoints.WaitAny(pending_urls))

            except yieldpoints.TimeoutException:
                self.finish("""
                    </table>

                    <p>These URLs did not complete after %.1f seconds: %s</p>
                """ % (duration(), ', '.join(pending_urls)))

                # Avoid LeakedCallbackError
                yield yieldpoints.CancelAll()
                return

            pending_urls.remove(url)
            self.write("""
                <tr>
                    <td>%s</td>
                    <td>HTTP %s</td>
                    <td>%.1f seconds</td>
                </tr>
            """ % (url, response.code, duration()))

            yield gen.Task(self.flush)

        self.finish("""
            </table>

            <p>Completed all in %.1f seconds</p>
        """ % duration())


if __name__ == '__main__':
    print 'Listening on http://localhost:8888'
    web.Application([('.*', PageRaceHandler)], debug=True).listen(8888)
    ioloop.IOLoop.current().start()
```

`yieldpoints` Classes

class `yieldpoints.`**`WaitAny`**(*keys*)

Wait for several keys, and continue when the first of them is complete.

Inspired by Ben Darnell in a conversation on the Tornado mailing list.

class `yieldpoints.`**`WithTimeout`**(*deadline*, *yield_point*, *io_loop=None*)

Wait for a YieldPoint or a timeout, whichever comes first.

Parameters

- *deadline*: A timestamp or timedelta

- *yield_point*: A `gen.YieldPoint` or a key
- *io_loop*: Optional custom `IOLoop` on which to run timeout

class `yieldpoints.`**`Cancel`**(*key*)
Cancel a key so `gen.engine` doesn't raise a LeakedCallbackError

class `yieldpoints.`**`CancelAll`**
Cancel all keys for which the current coroutine has registered callbacks

class `yieldpoints.`**`TimeoutException`**

Changelog

Version 0.1

First release.

CHAPTER 3

Source

Is on GitHub: https://github.com/ajdavis/yieldpoints

CHAPTER 4

Bug Reports and Feature Requests

Also on GitHub: https://github.com/ajdavis/yieldpoints/issues

CHAPTER 5

Indices and tables

- genindex
- search

Python Module Index

e

y

Index