asyncio-cancel-token Documentation

Release 0.2.0

Jason Carver

Sep 06, 2019

Contents

Task cancellation pattern for `asyncio` applications.

Inspired by https://vorpus.org/blog/timeouts-and-cancellation-for-humans/

CHAPTER 1

Contents

1.1 Cancel Token

1.1.1 Introduction

A *~cancel_token.CancelToken* is used to trigger cancellation of async operations. This is useful for `asyncio` based python applications which need a sane pattern for cancelling or timing out.

1.1.2 Quick Start

```
>>> import asyncio
>>> from cancel_token import CancelToken, OperationCancelled
>>> async def run_and_cancel_task():
...     async def some_task(token):
...         print("started task")
...         await token.wait()
...         print('task cancelled')
...     token = CancelToken('demo')
...     asyncio.ensure_future(some_task(token))
...     # give the task a moment to start
...     await asyncio.sleep(0.01)
...     # trigger the cancel token
...     token.trigger()
...     # give the task a moment to complete
...     await asyncio.sleep(0.01)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(run_and_cancel_task())
started task
task cancelled
```

1.1.3 Basic Usage

Creation of a *~cancel_token.CancelToken* simply requires providing a name.

```
>>> CancelToken('demo')
<CancelToken: demo>
```

Cancel tokens are triggered by calling the `trigger()` method. Triggering a cancel token causes the following behaviors.

- The property `triggered` will return `True`
- Any calls to the coroutine `wait()` will return.
- Any calls to the method `raise_if_triggered()` will raise an *~cancel_token.OperationCancelled* exception.

From within your application, you might use the cancel token any number of ways.

Loop exit condition

The property `triggered` can be useful as the conditional for a `while` loop.

```
async def my_task(token):
    while not token.triggered:
        ... # do something
```

Or you may want to break out of the loop in a less gracefull manner by raising the *~cancel_token.OperationCancelled* exception.

```
async def my_task(token):
    while True:
        token.raise_if_triggered()
        ... # do something
```

Waiting for an external signal

```
async def main():
    token = CancelToken('worker')

    asyncio.ensure_future(do_work(token))
    # wait for work to be completed before proceeding
    await token.wait()
```

1.1.4 Chaining Tokens

One of the more useful patterns is token chaining. Chaining can be used to create a single token which will trigger if any of the tokens it is chained to are triggered,

```
>>> token_a = CancelToken('token-a')
>>> token_b = CancelToken('token-b').chain(token_a)
>>> token_a.triggered
False
>>> token_b.triggered
```

(continues on next page)

(continued from previous page)

```
False
>>> token_a.trigger()
>>> token_a.triggered
True
>>> token_b.triggered
True
```

In this example we create `token_b` which has been chained with `token_a`. `token_b` can be triggered independently, not effecting `token_a`. However, if `token_a` is triggered, it also causes `token_b` to be triggered.

1.1.5 Integration with other async APIs

Within the boundaries of your own application it is easy to pass cancel tokens around as needed. However, you will often need cancellations to apply to async calls to apis which do not support the cancel token API.

The `cancel_token.CancelToken.cancellable_wait()` function can be used to enforce cancellations and timeouts on other async APIs. It expects any number of awaitables as positional arguments as well as an optional `timeout` as a keyword argument.

```
>>> import asyncio
>>> from cancel_token import CancelToken
>>> loop = asyncio.get_event_loop()
>>> token = CancelToken('demo')
>>> async def some_3rd_party_api():
...     await asyncio.sleep(10)
...
>>> loop.run_until_complete(token.cancellable_wait(some_3rd_party_api(), timeout=0.1))
TimeoutError
```

1.2 API

1.2.1 Cancel Token

1.2.2 Exceptions

1.3 Release Notes

1.3.1 v0.2.0

- Change to raise `asyncio.TimeoutError` instead of the `TimeoutError` from built-ins

1.3.2 v0.1.0-alpha.1

- Launched repository, claimed names for pip, RTD, github, etc

CHAPTER 2

Indices and tables

- genindex
- modindex