concurrently Documentation

Release 0.8

Konstantin Enchant

Feb 22, 2018

Contents

Library helps to easily write concurrent executed code blocks.

Quick example:

```
import asyncio
from concurrently import concurrently


async def amain(loop):
    """
    How to fetch some web pages with concurrently.
    """
    urls = [  # define pages urls
        'http://test/page_1',
        'http://test/page_2',
        'http://test/page_3',
        'http://test/page_4',
    ]
    results = {}

    # immediately run wrapped function concurrent
    # in 2 thread (asyncio coroutines)
    @concurrently(2)
    async def fetch_urls():
        for url in urls:
            # some function for download page
            page = await fetch_page(url)
            results[url] = page

    # wait until all concurrent threads finished
    await fetch_urls()
    print(results)


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(amain(loop))
```

Decorator `@concurrently()` makes to main thinks:

- starts concurrent execution specified count of decorated function
- returns special *Waiter* object to control the running functions

By default, the code runs as asyncio coroutines, but there are other supported ways to execute, by specifying the argument *engine*.

CHAPTER 1

Details

1.1 Waiter

The `@concurrently()` returns special object `Waiter` to control the running functions, like a wait until complete, stop and other.

class concurrently.engines.**AbstractWaiter**

__call__(*suppress_exceptions: bool = False*, *fail_hard: bool = False*)
The call blocks until the completion of all concurrent functions.

All exceptions in concurrent functions are captured and re-raise as `UnhandledExceptions`.

You can customize this behavior with following options:

Parameters

- **suppress_exceptions** – don't raise `UnhandledExceptions`
- **fail_hard** – stop all functions and raise error if one of function abort with error

exceptions() → List[Exception]
Returns list of all exception.

Useful with option `suppress_exceptions`.

stop()
Interrupts execution functions.

1.1.1 UnhandledExceptions

exception concurrently.**UnhandledExceptions**(*exceptions*)

Parameters **exceptions** – list of exception

1.2 Supported engines

1.2.1 AsyncIOEngine

Runs code as asyncio coroutines:

```
from concurrently import concurrently, AsyncIOEngine

...
@concurrently(2, engine=AsyncIOEngine, loop=loop)  # loop is option
async def fetch_urls():
    ...

await fetch_urls()
```

class concurrently.**AsyncIOEngine**(*loop: asyncio.base_events.BaseEventLoop = None*)

Parameters **loop** – specific asyncio loop or use default if *None*

1.2.2 AsyncIOThreadEngine

Runs code in threads with asyncio:

```
from concurrently import concurrently, AsyncIOThreadEngine

...
@concurrently(2, engine=AsyncIOThreadEngine, loop=loop)
def fetch_urls():  # not async def
    ...

await fetch_urls()
```

class concurrently.**AsyncIOThreadEngine**(*loop: asyncio.base_events.BaseEventLoop = None*)

Parameters **loop** – specific asyncio loop or use default if *None*

1.2.3 ThreadEngine

Runs code in system threads:

```
from concurrently import concurrently, ThreadEngine

...
@concurrently(2, engine=ThreadEngine)
def fetch_urls():  # not async def
    ...

fetch_urls()  # not await
```

class concurrently.**ThreadEngine**

1.2.4 ProcessEngine

Runs code in system process:

```
from concurrently import concurrently, ProcessEngine

...
@concurrently(2, engine=ProcessEngine)
def fetch_urls():
    ...

fetch_urls()
```

class concurrently.**ProcessEngine**

1.2.5 GeventEngine

Runs code as `gevent` greenlets:

```
from concurrently import concurrently, GeventEngine

...
@concurrently(2, engine=GeventEngine)
def fetch_urls():
    ...

fetch_urls()
```

Note: You must install `gevent` module for use this engine:

```
$ pip install concurrently[gevent]
```

class concurrently.**GeventEngine**

Python Module Index

C

Index

Symbols

A

C

E

G

P

S

T

U