Trio

a report from the wilds of async

Nathaniel J. Smith
May 17, 2017
Python Language Summit

Slides:
tinyurl.com/trio-language-summit
Outline

• You fool, why are you writing a new async library?
• Trio’s approach
• A litany of complaints
Outline

- You fool, why are you writing a new async library?
- Trio’s approach
- A litany of complaints
asyncio is the standard async library

“Ideally all third party frameworks would give up their own event loop implementation in favor of the standard implementation”

– Guido, PEP 3156
async: the story so far
The joy of order

When you write Python code, one prevailing, deep, unassailed assumption is that a line of code within a block is only ever executed after the preceding line is finished.

```
pod_bay_doors.open()
pod.launch()
```

The pod bay doors open, and only then does the pod launch. That's wonderful. One-line-after-another is a built-in mechanism in the language for encoding the order of execution. It's clear, terse, and unambiguous.

But what if we had to do it differently?

One solution: Deferred

Twisted tackles this problem with **Deferreds**, a type of object designed to do one thing, and one thing only: encode an order of execution separately from the order of lines in Python source code.
async: the story so far

• Twisted etc. couldn’t use Python to express control flow, so they invented complicated data structures that do the same thing in a clunkier way.

• asyncio standardized those data structures...

• ...then Yury added async/await to make them easier to use...

• ...then Dave Beazley pointed out: now that we have async/await, can’t we just... use Python instead?
“[...] curio is a beacon of clarity compared to asyncio. However, asyncio [...] must continue to support its existing APIs, [...] whether we like them or not.

I would love to see a design for a new API that focuses more on coroutines. But it should be a new PEP aimed at Python 3.7 or 3.8.”

Outline

- You fool, why are you writing a new async library?
- Trio’s approach
- A litany of complaints
# Simple echo client example

```python
import trio

async def main():
    with trio.socket.socket() as sock:
        with trio.open_cancel_scope(deadline=trio.current_time() + 5):
            await sock.connect((HOST, PORT))
            async with trio.open_nursery() as nursery:
                nursery.spawn(sender, sock)
                nursery.spawn(receiver, sock)

            print("goodbye language summit")

async def sender(sock):
    while True:
        await sock.sendall(b"hello language summit")
        await trio.sleep(1)

async def receiver(sock):
    while True:
        print("got data:", await sock.recv(MAX_CHUNK_SIZE))

trio.run(main)
```
# Simple echo client example

```python
import trio

async def main():
    with trio.socket.socket() as sock:
        with trio.open_cancel_scope(deadline=trio.current_time() + 5):
            await sock.connect((HOST, PORT))
            async with trio.open_nursery() as nursery:
                nursery.spawn(sender, sock)
                nursery.spawn(receiver, sock)

                print("goodbye language summit")

async def sender(sock):
    while True:
        await sock.sendall(b"hello language summit")
        await trio.sleep(1)

async def receiver(sock):
    while True:
        print("got data:", await sock.recv(MAX_CHUNK_SIZE))

trio.run(main)
```

https://gist.github.com/njsmith/b7bdc9cf153dc73cebd7b64fd5d39e0b
# Simple echo client example

```python
import trio

async def main():
    with trio.socket.socket() as sock:
        with trio.open_cancel_scope(deadline=trio.current_time() + 5):
            await sock.connect((HOST, PORT))
            async with trio.open_nursery() as nursery:
                nursery.spawn(sender, sock)
                nursery.spawn(receiver, sock)
            print("goodbye language summit")

async def sender(sock):
    while True:
        await sock.sendall(b"hello language summit")
        await trio.sleep(1)

async def receiver(sock):
    while True:
        print("got data:", await sock.recv(MAX_CHUNK_SIZE))

trio.run(main)
```

[async functions]

[https://gist.github.com/njsmith/b7bdc9cf153dc73cebd7b64fd5d39e0b]
# Simple echo client example

```python
import trio

async def main():
    with trio.socket.socket() as sock:
        with trio.open_cancel_scope(deadline=trio.current_time() + 5):
            await sock.connect((HOST, PORT))
            with trio.open_nursery() as nursery:
                nursery.spawn(sender, sock)
                nursery.spawn(receiver, sock)

            print("goodbye language summit")

async def sender(sock):
    while True:
        await sock.sendall(b"hello language summit")
        await trio.sleep(1)

async def receiver(sock):
    while True:
        print("got data:", await sock.recv(MAX_CHUNK_SIZE))

trio.run(main)
```

https://gist.github.com/njsmith/b7bdc9cf153dc73cebd7b64fd5d39e0b
# Simple echo client example

```python
import trio

async def main():
    with trio.socket.socket() as sock:
        with trio.open_cancel_scope(deadline=trio.current_time() + 5):
            await sock.connect((HOST, PORT))
            async with trio.open_nursery() as nursery:
                nursery.spawn(sender, sock)
                nursery.spawn(receiver, sock)

print("goodbye language summit")
```

```python
async def sender(sock):
    while True:
        await sock.sendall(b"hello language summit")
        await trio.sleep(1)

async def receiver(sock):
    while True:
        print("got data:", await sock.recv(MAX_CHUNK_SIZE))
```

trio.run(main)

https://gist.github.com/njsmith/b7bdc9cf153dc73cebd7b64fd5d39e0b
# Simple echo client example

```python
import trio

async def main():
    with trio.socket.socket() as sock:
        with trio.open_cancel_scope(deadline=trio.current_time() + 5):
            await sock.connect((HOST, PORT))

            async with trio.open_nursery() as nursery:
                nursery.spawn(sender, sock)
                nursery.spawn(receiver, sock)

            print("goodbye language summit")

async def sender(sock):
    while True:
        await sock.sendall(b"hello language summit")
        await trio.sleep(1)

async def receiver(sock):
    while True:
        print("got data:", await sock.recv(MAX_CHUNK_SIZE))
```

Not included:
- futures, callbacks
- implicit concurrency
- lost exceptions
Outline

• You fool, why are you writing a new async library?

• Trio’s approach

• A litany of complaints
(Just kidding)
Actual language issues

- `.throw()` is broken / adding Result builtin?
- Warning: coroutine ‘...’ was never awaited
- MultiError exception handling
  - Make traceback objects instantiable / mutable / subclassable
  - Mechanism to disable implicit chaining
  - More standard way to override exception printing?
  - More ergonomic MultiError handlers?
- Async generator cleanup / PEP 533

Slides: tinyurl.com/trio-language-summit
Actual language issues

- `.throw()` is broken / adding `Result` built-in?
- Warning: coroutine ‘...’ was never awaited
- MultiError exception handling
  - Make traceback objects instantiable / mutable / subclassable
  - Mechanism to disable implicit chaining
  - More standard way to override exception printing?
  - More ergonomic MultiError handlers?
- Async generator cleanup / PEP 533

Slides:
tinyurl.com/trio-language-summit
.throw() is broken :-(

- bpo-29590: breaks stack introspection (bad for profilers, debuggers, control-C handling)
- bpo-29587: no implicit exception chaining → trio is careful to never, ever use .throw()
```python
class Result(ABC):
    @abstractmethod
    def unwrap(self):
        pass

    @staticmethod
    def capture(fn, *args, **kwargs):
        try:
            return Value(fn(*args, **kwargs))
        except BaseException as exc:
            return Error(exc)

class Value(Result):
    def __init__(self, value):
        self.value = value

    def unwrap(self):
        return self.value

class Error(Result):
    def __init__(self, error):
        self.error = error

    def unwrap(self):
        raise self.error
```
If Result were built-in:

class Generator:
    def send(self, value):
        return self.resume(Value(value))

    def throw(self, error):
        return self.resume(Errror(error))

    def resume(self, result):
        if not isinstance(result, Result):
            raise TypeError

# ...
Actual language issues

- `.throw()` is broken / adding Result builtin?
- **Warning:** coroutine ‘...’ was never awaited
- MultiError exception handling
  - Make traceback objects instantiable / mutable / subclassable
  - Mechanism to disable implicit chaining
  - More standard way to override exception printing?
  - More ergonomic MultiError handlers?
- Async generator cleanup / PEP 533

Slides: tinyurl.com/trio-language-summit
f()
g()
await g()
f() 

```javascript
async g() {
    // Code
}
```

```javascript
await g()
```
async def test_f():
    f()

async def test_g():
    g()
async def test_f():
    f()

async def test_g():
    g()
>>> print("hi"); f(1); print("bye")
hi
Warning: f() takes 0 positional arguments but 1 was given; function call skipped
bye
>>>
The trio tutorial from 10,000 feet
The trio tutorial from 10,000 feet

Intro to async/await
The trio tutorial from 10,000 feet

Intro to async/await

Task spawning and nurseries
The trio tutorial from 10,000 feet

Intro to async/await

Task spawning and nurseries

Explaining

Warning: coroutine ‘...’ was never awaited
Are we just doomed?

• In asyncio, g() calls are common :-(

• Minimal proposal: add a global (or thread-local) counter of times this warning fired, so test harness / trio main loop can detect the problem and blow up? (without having to monkeypatch warnings)

• More ambitious proposal →
Straw proposal for 3.7

- `await x` stays the same
- `await g(*args, **kwargs) →`
  
  ```python
  if hasattr(g, "__coroutine__"):
      coro = g.__coroutine__(*args, **kwargs)
  else:
      coro = g.__call__(*args, **kwargs)
  await coro
  ```

- Trio: use decorator or `__future__` to make `__call__` an error

- asyncio: use decorator to make `__call__` return Task object; no more `ensure_future`
Actual language issues

- `.throw()` is broken / adding `Result` builtin?
- Warning: coroutine ‘...' was never awaited

- `MultiError` exception handling
  - Make traceback objects instantiable / mutable / subclassable
  - Mechanism to disable implicit chaining
  - More standard way to override exception printing?
  - More ergonomic `MultiError` handlers?

- Async generator cleanup / PEP 533

Slides: tinyurl.com/trio-language-summit
main()
├─ f1()
└─ f2()
    ├─ f2a()
    └─ f2b()
main()
  └── f1() → KeyError
    └── f2()
        └── f2a() → ValueError
        └── f2b() → RuntimeError
main()
├─ f1() → KeyError
└─ f2()
    ├─ f2a() → ValueError
    └─ f2b() → RuntimeError

MultiError([
    KeyError,
    MultiError([
        ValueError,
        RuntimeError,
    ]),
])
main()
  └─ f1() → KeyError
  └─ f2()
      └─ f2a() → ValueError
      └─ f2b() → RuntimeError

MultiError([KeyError, MultiError([ValueError, RuntimeError,]),])

Traceback (most recent call last):
  File "multierror.py", line 22, in <module>
    trio.run(main)
  File "multierror.py", line 6, in main
    nursery.spawn(f2)
 trio.MultiError: KeyError(), <MultiError: RuntimeError(), ValueError()>

Details of embedded exception 1:
Traceback (most recent call last):
  File "multierror.py", line 9, in f1
    raise KeyError
 KeyError

Details of embedded exception 2:
Traceback (most recent call last):
  File "multierror.py", line 14, in f2
    nursery.spawn(f2b)
 trio.MultiError: RuntimeError(), ValueError()

Details of embedded exception 1:
Traceback (most recent call last):
  File "multierror.py", line 17, in f2a
    raise ValueError
 ValueError

Details of embedded exception 2:
Traceback (most recent call last):
  File "multierror.py", line 20, in f2b
    raise RuntimeError
 RuntimeError
def catch_block(exc):
    if isinstance(exc, (RuntimeError, KeyError)):
        return None
    else:
        return exc

with trio.MultiError.catch(catch_block):
    ...
Problems

• PyPy has a special extension to specifically allow mutating tracebacks w/o ctypes, b/c both jinja2 and trio need it
Problems

• PyPy has a special extension to specifically allow mutating tracebacks w/o ctypes, b/c both jinja2 and trio need it

• Want to hide/annotate parts of traceback
Traceback (most recent call last):
  File "multiterror.py", line 22, in <module>
    trio.run(main)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 1130, in run
    return result.unwrap()
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_result", line 118, in unwrap
    task.coro.send(next_send)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_result", line 747, in init
    return system_nursery.reap_and_unwrap(task)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_result", line 286, in reap_and_unwrap
    raise self.error
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 1230, in run_impl
    msg = task.coro.send(next_send)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 253, in asend
    return await self._invoke(self._it.send, value)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/async_generator_impl.py", line 269, in __do_it
    return await self._invoke(self._it.send, *args)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/async_generator_impl.py", line 192, in _invoke
    result = fn(*args)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 213, in open_nursery
    await nursery._clean_up(pending_exc)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 353, in _clean_up
    raise mexc
trio.MultiError: KeyError(), <MultiError: RuntimeError(), ValueError()>

Details of embedded exception 1:
Traceback (most recent call last):
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 1230, in run_impl
    msg = task.coro.send(next_send)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 9, in f1
    raise KeyError
KeyError

Details of embedded exception 2:
Traceback (most recent call last):
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 1230, in run_impl
    msg = task.coro.send(next_send)
  File "multiterror.py", line 14, in f2
    nursery.spawn(f2)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/async_generator_impl.py", line 253, in asend
    return await self._invoke(self._it.send, value)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/async_generator_impl.py", line 192, in _invoke
    result = fn(*args)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 213, in open_nursery
    await nursery._clean_up(pending_exc)
  File "home/njs/.user-python3.5-64bit/local/lib/python3.5/site-packages/trio_core_run", line 353, in _clean_up
    raise mexc
trio.MultiError: RuntimeError(), ValueError()
Problems

- PyPy has a special extension to specifically allow mutating tracebacks w/o ctypes, b/c both jinja2 and trio need it.
- Want to hide/annotate parts of traceback.
Problems

- PyPy has a special extension to specifically allow mutating tracebacks w/o ctypes, b/c both jinja2 and trio need it.
- Want to hide/annotate parts of traceback.
- Implicit chaining corrupts context information when re-raising :-(

Problems

- PyPy has a special extension to specifically allow mutating tracebacks w/o ctypes, b/c both jinja2 and trio need it
- Want to hide/annotate parts of traceback
- Implicit chaining corrupts context information when re-raising :-(
- pytest, IPython, ... don’t know that MultiErrors are special
Problems

• PyPy has a special extension to specifically allow mutating tracebacks w/o ctypes, b/c both jinja2 and trio need it

• Want to hide/annotate parts of traceback

• Implicit chaining corrupts context information when re-raising :-(

• pytest, IPython, ... don’t know that MultiErrors are special

• MultiError.catch is kinda awkward
Actual language issues

• `.throw()` is broken / adding Result builtin?
• Warning: coroutine ‘...’ was never awaited
• MultiError exception handling
  – Make traceback objects instantiable / mutable / subclassable
  – Mechanism to disable implicit chaining
  – More standard way to override exception printing?
  – More ergonomic MultiError handlers?

• Async generator cleanup / PEP 533

Slides:
tinyurl.com/trio-language-summit
# Simple async generator that uses a websocket internally

```python
async def stock_ticker(symbol):
    async with open_websocket("wss://stock-service/" + symbol) as ws:
        while True:
            yield json.loads(ws.next_message())
```

```python
async def record_ticks(symbol):
    async for tick in stock_ticker(symbol):
        await db.record_price(symbol, tick["time"], tick["price"])
```

```python
async def main():
    with trio.open_cancel_scope(deadline=trio.current_time() + 60):
        await record_ticks("PYCON")
```
# Simple async generator that uses a websocket internally

```python
async def stock_ticker(symbol):
    async with open_websocket("wss://stock-service/" + symbol) as ws:
        while True:
            yield json.loads(ws.next_message())
```

```python
async def record_ticks(symbol):
    async with trio.aclosing(stock_ticker(symbol)) as agen:
        async for tick in agen:
            await db.record_price(symbol, tick["time"], tick["price"])
```

```python
async def main():
    with trio.open_cancel_scope(deadline=trio.current_time() + 60):
        await record_ticks("PYCON")
```
PEP 533

• Proposes to add __(a)close__ methods to iterator protocol
• Basic idea: (async) for will call __(a)close__ no matter how it exits (i.e.: adding little with flavor to for loops)
• There are a bunch of details + transition plan in the PEP
• Is it a good idea? Dunno. But so far the main alternative proposal is “never use with/try/finally in (async) generators”…
Thanks for listening!

• `.throw()` is broken / adding Result builtin?
• Warning: coroutine ‘...’ was never awaited
• MultiError exception handling
  – Make traceback objects instantiable / mutable / subclassable
  – Mechanism to disable implicit chaining
  – More standard way to override exception printing?
  – More ergonomic MultiError handlers?
• Async generator cleanup

Slides: tinyurl.com/trio-language-summit