Пишем для asyncio

Андрей Светлов

О себе

- Python Core Developer
- Принимал участие в создании asyncio
- Соавтор нескольких библиотек на основе asyncio — aiohttp, aiozmq, aiopg
- Использую asyncio на работе

yield from

```
import asyncio

@asyncio.coroutine
def task():
    resp = yield from aiohttp.request('GET', 'http://python.org')
    if resp.status == 200:
        body = yield from resp.read()
        print(body)
    resp.close()

asyncio.get_event_loop().run_until_complete(task())
```

Упрощенный yield from

```
for i in EXPR:
    yield i
```

~~i.send(val)~~

~~i.throw(exc)~~

~~i.close()~~

Полный вариант yield from

```
_i = iter(EXPR)
try:
    _y = next(_i)
except StopIteration as _e:
    _r = _e.value
else:
    while 1:
        try:
            _s = yield _y
        except GeneratorExit as _e:
            try:
                _m = _i.close
            except AttributeError:
                pass
            else:
                _m()
            raise _e
        except BaseException as _e:
            _x = sys.exc_info()
            try:
                _m = _i.throw
            except AttributeError:
                raise _e
            else:
                try:
                    _y = _m(*_x)
                except StopIteration as _e:
                    _r = _e.value
                    break
    else:
        try:
            if _s is None:
                _y = next(_i)
            else:
                _y = _i.send(_s)
        except StopIteration as _e:
            _r = _e.value
            break
RESULT = _r
```

yield против yield from

- программный стек
- неудобные stack trace
- неприятная отладка
- Медленней в 20 раз

Всё ровно наоборот :)

Stack trace

```
@asyncio.coroutine
def inner():
    raise RuntimeError("Ooops!")

@asyncio.coroutine
def outer():
    yield from inner()

loop = asyncio.get_event_loop()
loop.run_until_complete(outer())
```

```
Traceback (most recent call last):
  File "s.py", line 13, in <module>
    loop.run_until_complete(outer())
  File ".../asyncio/base_events.py", line 208, in
run_until_complete
    return future.result()
  File ".../asyncio/futures.py", line 243, in result
    raise self._exception
  File ".../asyncio/tasks.py", line 302, in _step
    result = next(coro)
  File "s.py", line 9, in outer
    yield from inner()
  File ".../asyncio/tasks.py", line 84, in coro
    res = func(*args, **kw)
  File "s.py", line 5, in inner
    raise RuntimeError("Ooops!")
RuntimeError: Ooops!
```

Debug режим

```
@asyncio.coroutine
def f():
    asyncio.sleep(10)  # missing yield from

loop = asyncio.get_event_loop()
loop.run_until_complete(f())
```

Переменная окружения PYTHONASYNCIODEBUG

```
$ PYTHONASYNCIODEBUG=1 python3 d.py
Coroutine 'sleep' defined at /usr/lib/python3.4/asyncio/tasks.py:542 was never
yielded from
```

Отладка

```
def g():
    ret = yield from f()
```

- next
- step

Переключение контекста

```
def transfer(amount, payer, payee, server):
    if not payer.sufficient_funds_for_withdrawl(amount):
        raise InsufficientFunds()
    log("{payer} has sufficient funds.", payer=payer)
    payee.deposit(amount)
    log("{payee} received payment", payee=payee)
    payer.withdraw(amount)
    log("{payer} made payment", payer=payer)
    server.update_balances([payer, payee])
```

Переключение контекста 2

```
@coroutine
def transfer(amount, payer, payee, server):
    if not payer.sufficient_funds_for_withdrawl(amount):
        raise InsufficientFunds()
    log("{payer} has sufficient funds.", payer=payer)
    payee.deposit(amount)
    log("{payee} received payment", payee=payee)
    payer.withdraw(amount)
    log("{payer} made payment", payer=payer)
    yield from server.update_balances([payer, payee])
```

Переключение контекста 3

```
@coroutine
def transfer(amount, payer, payee, server):
    if not payer.sufficient_funds_for_withdrawl(amount):
        raise InsufficientFunds()
    yield from log("{payer} has sufficient funds.", payer=payer)
    payee.deposit(amount)
    yield from log("{payee} received payment", payee=payee)
    payer.withdraw(amount)
    yield from log("{payer} made payment", payer=payer)
    yield from server.update_balances([payer, payee])
```

Переключение контекста 4

```
@coroutine
def transfer(amount, payer, payee, server):
    with (yield from payer.hold(amount):
        if not payer.sufficient_funds_for_withdrawl(amount):
            raise InsufficientFunds()
        yield from log("{payer} has sufficient funds.", payer=payer)
        payee.deposit(amount)
        yield from log("{payee} received payment", payee=payee)
        payer.withdraw(amount)
        yield from log("{payer} made payment", payer=payer)
        yield from server.update_balances([payer, payee])
```

Явный event loop

```
yield from asyncio.sleep(1.5, loop=loop)


yield from asyncio.wait_for(f(), 15, loop=loop)
fut = asyncio.Future(loop=loop)


reader, writer = yield from asyncio.open_connection("www.python.org", 80,
loop=loop)
```

Тесты

```
class MyTest(unittest.TestCase):
    def setUp(self):
        self.loop = asyncio.new_event_loop()
        asyncio.set_event_loop(None)
    def tearDown(self):
        self.loop.close()
    def test_feature(self):
        @asyncio.coroutine
        def go():
            yielf from asyncio.sleep(1.5, loop=self.loop)
        self.loop.run_until_complete(go())
```

Ожидание одноразового события

```
fut = Future()

@asyncio.coroutine
def wait():
    yield from fut

def sinal():
    fut.set_result(None)
```

Transport/Protocol — не для вас

```
class Proto:
    def connection_made(self, transport):
        self.transport = transport
    def data_received(self, data):
        self.transport.write(b'echo' + data)
    def connection_lost(self, exc):
        self.transport = None
```

Stream API

```
reader, writer = open_connection(host, port)

data yield from reader.read()
writer.write(data)
yield from writer.drain()
writer.close()
```

Синхронный код — в executor

```
def syncronous_dns(host, port):
    return socket.getaddrinfo(host, port)

ret = yield from loop.run_in_executor(None, syncronous_dns, "www.python.org", 80)
```

Блокировки

```
with (yield from lock):
    code()
```

Очереди

```
yield from queue.put(value)
yield from queue.get()
```

Процессы

```
proc = yield from asyncio.create_subprocess_shell(cmd, **kwds)
result = yield from proc.communicate(request)
yield from proc.wait()
proc.returncode
```

call_soon/call_later

```
loop.call_soon(func, 1, 2, 3)
loop.call_later(1.5, func, 1, 2, 3)

@asyncio.coroutine
def func():
    f()
    yield from sleep(1.5)
    g()
```

Task/Future cancellation

task.cancel()

- grandfather
  - father
    - son

Таймауты

```
@asyncio.coroutine
def task():
    resp = yield from aiohttp.request('GET', 'http://python.org')
    body = yield from resp.read()
    resp.close()
    return body

try:
    body = yield asyncio.wait_for(task(), 10.5)
except asyncio.TimeoutError:
    handle_timeout()
```

Ждем нескольких событий

yield from asyncio.**gather**(f(), g(), h())

Магические методы

```
def __init__(self):
    yield from self.meth()

def __getattr__(self, name):
    return (yield from self.dispatch(name))
```

trollius

- yield From(f())
- Python 2
- Библиотеки

Резюме

Удобное API

TCP/UDP/Unix сокеты, unix pipes, процессы

Мало библиотек

Нет встроенного HTTP

aiohttp — доделаем

aiopg — сделали

aioredis — создается

Вопросы?

andrew.svetlov@gmail.com