aiohttp

Андрей Светлов

О себе

- Python Core Developer
- Committer в asyncio
- Соавтор aiohttp и еще десятка asyncio библиотек

Асинхронное программирование

- Неблокирующие операции ввода-вывода
- Недостижимая для традиционного (основанного на потоках) производительность web серверов.
- Объединение в одном процессе веб-обработчиков и “долгоиграющих” задач

Асинхронное программирование 2

- Цена -- неудобство написания кода

Два года назад

twisted
tornado

\+ десяток никому не известных библиотек

Год назад

- вышел asyncio
- http -- нет
- баз данных -- нет
- есть tcp sockets, udp sockets, unix sockets, pipes, processes

Сегодня

aiohttp
aiopg
aiomysql
aioredis
aiozmq
aioes
aiomcache

aiohttp_jinja2
aiohttp_session
aiorwlock
aiokafka

aiocouchdb
aio-s3
aio-beanstalk

Сегодня 2

aiodns
asyncio-mongo
aioamqp
aioetcd
qualmash (Qt)
aiogevent

asyncssh
irc3
autobahn
pulsar
aiogreen (eventlet)
gbulb (GLib)
rose (libuv)
aiomultiprocessing

Клиентское API

Клиентское API

```
def f(loop):
    response = yield from aiohttp.request(
        'GET', 'http://python.org',
        loop=loop)
    body = yield from response.read()
    return body

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

Keep-alive

```
conn = aiohttp.TCPConnector()

r = yield from aiohttp.request(
      'get', 'http://python.org',
      connector=conn)
```

Сессии

```
conn = aiohttp.TCPConnector(
           shared_cookies=True)

r = yield from aiohttp.request(
        'get', 'http://python.org',
        connector=conn)
```

Таймауты

```
r = yield from asyncio.wait_for(
        aiohttp.request(
            'get', 'http://python.org'),
        0.5)
```

Серверное API

Простейший сервер

```
import asyncio
from aiohttp import web

@asyncio.coroutine
def handler(request):
    return web.Response(
        body=b"Hello, world")

app = web.Application()
app.router.add_route('GET', '/', handler)
```

Простейший сервер 2

```
loop = asyncio.get_event_loop()
f = loop.create_server(app.make_handler(),
                       '0.0.0.0', 8080)
srv = loop.run_until_complete(f)
print('serving on',
      srv.sockets[0].getsockname())
try:
    loop.run_forever()
except KeyboardInterrupt:
    pass
```

Параметры

```
@asyncio.coroutine
def handler(request):
    args = request.GET
    accepts = request.headers.getall(
                  'ACCEPT', [])
    name = request.match_info['name']
    return web.HTTPOk()

app.router.add_route('GET', '/{name}',
                     handler)
```

Загрузка файлов

```
<form action="/store_mp3" method="post"
    accept-charset="utf-8"
    enctype="multipart/form-data">

  <label for="mp3">Mp3</label>
  <input id="mp3" name="mp3"
      type="file" value="" />

  <input type="submit" value="submit" />
</form>
```

Загрузка файлов 2

```
@asyncio.coroutine
def store_mp3_view(request):
    data = yield from request.post()
    filename = data['mp3'].filename
    input_file = data['mp3'].file
    content = input_file.read()
    return web.Response(
        body=content,
        headers=MultiDict(
            {'CONTENT-DISPOSITION':
                filename})
```

Web-sockets

```
@asyncio.coroutine
def websocket_handler(request):
    ws = web.WebSocketResponse()
    ws.start(request)
    while True:
        try:
            data = yield from ws.receive_str()
            if data == 'close':
                ws.close()
            else:
                ws.send_str(data + '/answer')
        except web.WebSocketDisconnectedError as exc:
            print(exc.code, exc.message)
            return ws
```

Middleware

```
app = web.Application(
    middlewares=[
        middleware_factory_1,
        middleware_factory_2])
```

Middleware 2

```
@asyncio.coroutine
def middleware_factory(app, handler):
    # configure middleware
    @asyncio.coroutine
    def middleware(request):
        try:
            # preprocess
            ret = yield from handler(request)
        except Exception as exc:
            # process exception
            raise
        else:
            # postprocess
            return ret
    return middleware
```

Примеры middleware

- CORS
- sessions
- database transactions

aiohttp_session

```
from aiohttp_session import get_session

@asyncio.coroutine
def hander(request):
    session = yield from get_session(request)
    session['key'] = 'value'
    return web.HTTPOk(text='Response text')
```

aiohttp_session 2

```
def session_middleware(storage):
    @asyncio.coroutine
    def factory(app, handler):
        @asyncio.coroutine
        def middleware(request):
            request[STORAGE_KEY] = storage
            response = yield from handler(request)
            yield from storage.save_session(request, response)
            return response
        return middleware
    return factory
```

Базы данных

```
import sqlalchemy as sa
from aiopg.sa import create_engine
metadata = sa.MetaData()
tbl = sa.Table('tbl', metadata,
   sa.Column('id', sa.Integer,
             primary_key=True),
   sa.Column('val', sa.String(255)))

app = web.Application()
app['db'] = yield from create_engine(...)
```

Базы данных

```
@asyncio.coroutine
def handler(request):
    with (yield from request.app['db']) as conn:
        yield from conn.execute(
            tbl.insert().values(val='abc'))

        res = yield from conn.execute(tbl.select())
        row = yield from res.fetchone()
        while row is not None:
            print(row.id, row.val)
            row = yield from res.fetchone()
```

Производительность

- Примерно как у tornado
- twisted.web быстрее (грязный тест)
- WSGI – медленнее

Дальнейшие планы

Вопросы?

andrew.svetlov@gmail.com
http://asvetlov.blogspot.com