про асинхронное сетевое программирование

Стас Рудаков

Да что мы всё обо мне, да обо мне...

О чем будем говорить

- Что нам дает асинхронность
- Как это работает
- Twisted
- Gevent
- Tornado
- PEP 3156

https://raw.github.com/nott/talks/async.pdf

Работа с сетью: зачем что-то изобретать?

```
1 #!/usr/bin/env python2
2 import urllib2
3 f = urllib2.urlopen('http://python.org/')
4 print f.read(10)
```

Listing 1: urllib_example.py

Работа с сетью: зачем что-то изобретать?

```
#!/usr/bin/env python2
import urllib2
f = urllib2.urlopen('http://python.org/')
print f.read(10)
```

Listing 2: urllib_example.py

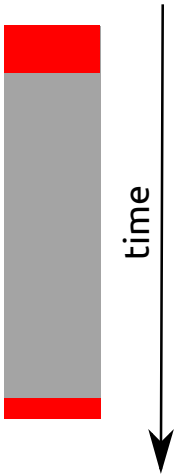
```
$ time python urllib_example.py
<!DOCTYPE

real 0m0.384s
user 0m0.067s
sys 0m0.017s
```

Работа с сетью: зачем что-то изобретать?

```
1 #!/usr/bin/env python2
2 import urllib2
3 f = urllib2.urlopen('http://python.org/')
4 print f.read(10)
```

Listing 3: urllib_example.py

```
1 $ time python urllib_example.py
2 <!DOCTYPE
3
4 real 0m0.384s
5 user 0m0.067s
6 sys 0m0.017s
```



Работа с сетью: дальше будет только хуже

```
1 #!/usr/bin/env python2
2 import sys
3 import urllib2
4 for i in xrange(5):
5     f = urllib2.urlopen('http://python.org/')
6     print f.read(10),
```

Listing 4: urllib_example2.py

Работа с сетью: дальше будет только хуже

```
#!/usr/bin/env python2
import sys
import urllib2
for i in xrange(5):
    f = urllib2.urlopen('http://python.org/')
    print f.read(10),
```

Listing 5: urllib_example2.py

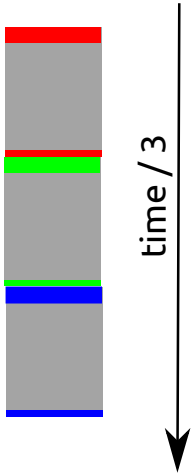
```
$ time python urllib_example2.py
<!DOCTYPE  <!DOCTYPE  <!DOCTYPE

real 0m1.007s
user 0m0.050s
sys 0m0.003s
```

Работа с сетью: дальше будет только хуже

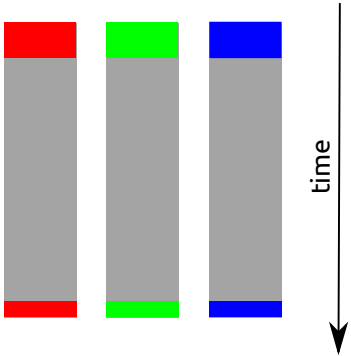
```
1 #!/usr/bin/env python2
2 import sys
3 import urllib2
4 for i in xrange(5):
5     f = urllib2.urlopen('http://python.org/')
6     print f.read(10),
```

Listing 6: urllib_example2.py

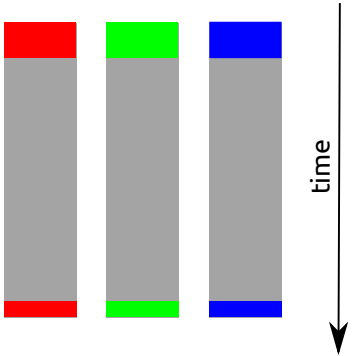
```
1 $ time python urllib_example2.py
2 <!DOCTYPE  <!DOCTYPE  <!DOCTYPE
3
4 real 0m1.007s
5 user 0m0.050s
6 sys 0m0.003s
```



I know, I'll use...

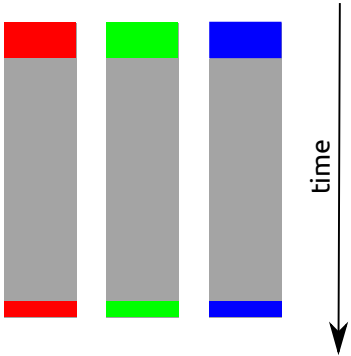


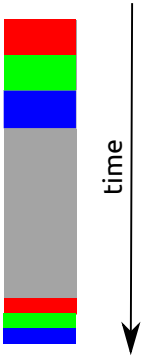
I know, I'll use...



Some people, when confronted with a problem, think,
"I know, I'll use threads,"
and then two they hav erpoblesms. *(Ned Batchelder)*

I know, I'll use...

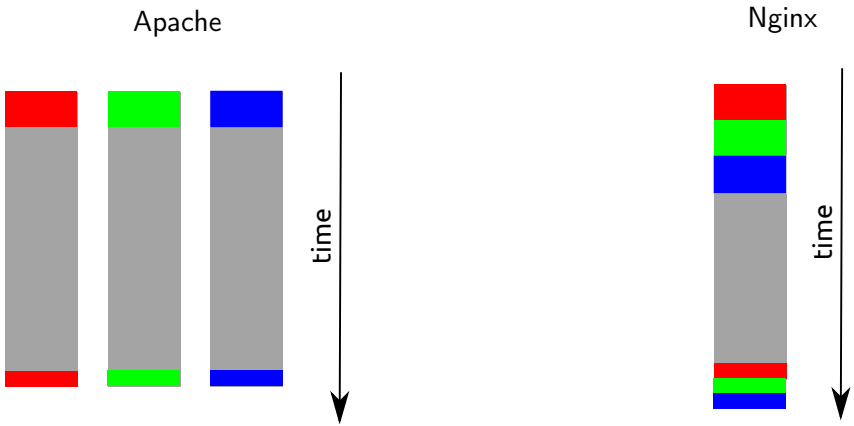




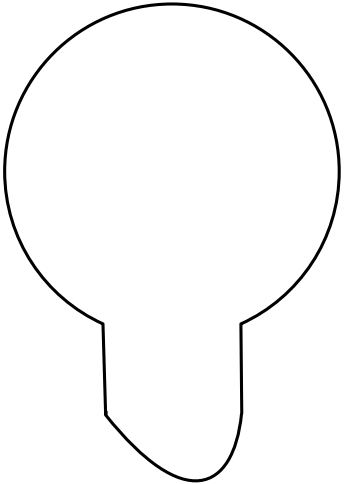
Some people, when confronted with a problem, think,
"I know, I'll use threads,"
and then two they hav erpoblesms. *(Ned Batchelder)*

Apache vs Nginx

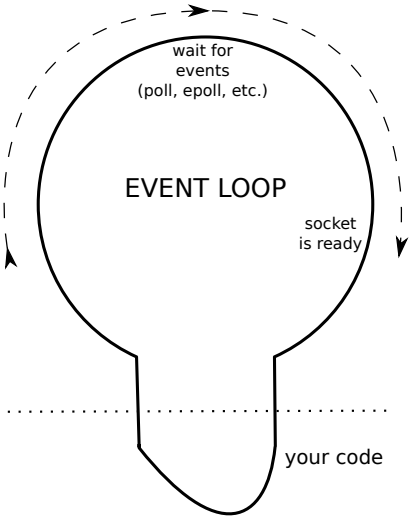
Apache vs Nginx



Лампочка

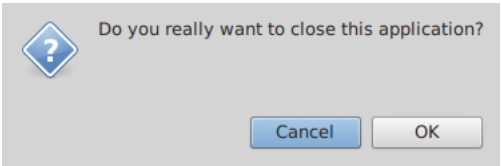


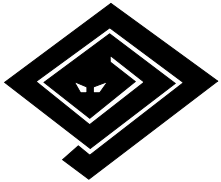
Event loop


wait for
events
(poll, epoll, etc.)
EVENT LOOP
socket
is ready
your code

Event loop in GUI

```
1      def OnClose(self, event):
2          dlg = wx.MessageDialog(self,
3              "Do you really want to close this application?",
4              "Confirm Exit", wx.OK|wx.CANCEL|wx.ICON_QUESTION)
5          result = dlg.ShowModal()
6          dlg.Destroy()
7          if result == wx.ID_OK:
8              self.Destroy()
```





Twisted

Twisted: transport vs protocol

Транспорт

- знает, как пересылать байты
- примеры: socket, UNIX pipe, SSL socket, etc.

Протокол

- знает, какие байты пересылать
- примеры: HTTP, SMTP, XMPP, Memcached, etc.

Twisted: transport vs protocol на примере

```
#!/usr/bin/env python2
from twisted.internet import reactor, protocol
class Counter(protocol.Protocol):
    def __init__(self, factory):
        self.factory = factory
    def connectionMade(self):
        num = self.factory.num = self.factory.num + 1
        self.transport.write("open connections: %d\n" % num)
    def connectionLost(self, reason):
        self.factory.num -= 1
    dataReceived = lambda self, data: None
class CounterFactory(protocol.Factory):
    def __init__(self):
        self.num = 0
    buildProtocol = lambda self, addr: Counter(self)
reactor.listenTCP(8123, CounterFactory())
reactor.run()
```

Twisted: transport vs protocol

```
$ telnet 127.0.0.1 8123
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
open connections: 2
```

Twisted: Deferred

- Deferred — обещание вернуть результат.
- К нему прикрепляются callback'и и errback'и.
- Само использование Deferred еще не делает программу асинхронной.
- Важно, чтобы callback'и вызывались по событиям (в любых смыслах).

Twisted: Deferred на примере

```
1  #!/usr/bin/env python2
2  import sys
3  from twisted.internet import defer, reactor
4  from twisted.python.util import println
5  from twisted.web.client import getPage
6
7  results = []
8  for i in xrange(int(sys.argv[1])):
9      d = getPage('http://www.python.org/')
10     d.addCallback(lambda value: value[:10])
11     d.addCallback(lambda value: println(value[::-1]))
12     d.addErrback(lambda error: println("error:", error))
13     results.append(d)
14
15 d_list = defer.DeferredList(results)
16 d_list.addBoth(lambda ignored: reactor.stop())
17 reactor.run()
```

Twisted: Deferred

```
$ time python twisted_example.py 1
 EPYTCOD!<

real 0m0.833s
user 0m0.250s
sys 0m0.030s
$ time python twisted_example.py 5
 EPYTCOD!<
 EPYTCOD!<
 EPYTCOD!<
 EPYTCOD!<
 EPYTCOD!<

real 0m1.116s
user 0m0.257s
sys 0m0.043s
```

Twisted: параллельная вселенная

- последовательность синхронных операторов $\approx$ цепочка `Deferred`;
- вызов функции внутри функции $\approx$ возврат `Deferred` из `Deferred`;
- `Deferred` $\approx$ `stack frame`;
- блок `try..except` $\approx$ цепочка `errback`'ов;
- `threading.join` $\approx$ `DeferredList`;

Twisted: параллельная вселенная

- последовательность синхронных операторов ≈ цепочка `Deferred`;
- вызов функции внутри функции ≈ возврат `Deferred` из `Deferred`;
- `Deferred` ≈ `stack frame`;
- блок `try..except` ≈ цепочка `errback`'ов;
- `threading.join` ≈ `DeferredList`;

- `logging`;
- `plugins`;
- несовместимость с `stdlib`;
- отлично проработанная инфраструктура;
- достаточно (но далеко не самый) быстрый;

Gevent

и прекрасный код

Gevent

```
1  #!/usr/bin/python2
2  import sys
3  from gevent import joinall, monkey, spawn
4
5  monkey.patch_all()
6
7  import urllib2
8
9  def fetch():
10     f = urllib2.urlopen('http://www.python.org/')
11     print f.read(10)
12
13 num = int(sys.argv[1])
14 # equivalent can be also implemented with threading
15 joinall([spawn(fetch) for i in xrange(num)])
```

Gevent: oh wow!

```
$ time python gevent_example.py 1
<!DOCTYPE

real 0m0.434s
user 0m0.077s
sys 0m0.020s

$ time python gevent_example.py 5
<!DOCTYPE
<!DOCTYPE
<!DOCTYPE
<!DOCTYPE
<!DOCTYPE

real 0m0.428s
user 0m0.093s
sys 0m0.023s
```

Greenlets inside

```
1  #!/usr/bin/env python
2  from greenlet import greenlet
3
4  def test1():
5      print(1)
6      gr2.switch()
7      print(3)
8  def test2():
9      print(2)
10     gr1.switch()
11     print(4)
12
13 gr1 = greenlet(test1)
14 gr2 = greenlet(test2)
15 gr1.switch()
```

Listing 8: greenlet_example.py

```
$ python greenlet_example.py
1
2
3
```

Gevent: есть нюанс

- Текущая версия 0.13.
- В версии 1.0 запланирован переход с `libevent` на `libev`.

Gevent: есть нюанс

- Текущая версия 0.13.
- В версии 1.0 запланирован переход с `libevent` на `libev`.
- Первая β-версия для 1.0 была выпущена около 2 лет назад.

Gevent: есть нюанс

- Текущая версия 0.13.
- В версии 1.0 запланирован переход с `libevent` на `libev`.
- Первая β-версия для 1.0 была выпущена около 2 лет назад.
- Пугают страшилки про гринлеты.

Gevent: есть нюанс

- Текущая версия 0.13.
- В версии 1.0 запланирован переход с `libevent` на `libev`.
- Первая β-версия для 1.0 была выпущена около 2 лет назад.
- Пугают страшилки про гринлеты.
- Проблемы с совместимостью с существующими библиотеками.

Tornado

и быстрые HTTP серверы

Tornado

```
#!/usr/bin/env python
import tornado.ioloop
import tornado.httpclient
import tornado.web

class Handler(tornado.web.RequestHandler):
    @tornado.web.asynchronous
    def get(self, path):
        http = tornado.httpclient.AsyncHTTPClient()
        http.fetch("http://www.python.org/" + path,
                   self._on_download)
    def _on_download(self, response):
        self.write(response.body)
        self.finish()
application = tornado.web.Application([(r"/(.*)", Handler),])
application.listen(8888)
tornado.ioloop.IOLoop.instance().start()
```

Tornado: we need an HTTP client!

```
1  #!/usr/bin/env python
2  import sys, tornado.ioloop, tornado.httpclient
3
4  def fetch():
5      http = tornado.httpclient.AsyncHTTPClient()
6      http.fetch("http://www.python.org/", on_download)
7  def on_download(response):
8      print(response.body[:10])
9      global num
10     num -= 1
11     if not num: ioloop.stop()
12
13 num = int(sys.argv[1])
14 ioloop = tornado.ioloop.IOLoop.instance()
15 for i in range(num):
16     ioloop.add_callback(fetch)
17 ioloop.start()
```

Tornado: the results

```
$ time python gevent_example.py 5
<!DOCTYPE
<!DOCTYPE
<!DOCTYPE
<!DOCTYPE
<!DOCTYPE

real 0m0.428s
user 0m0.093s
sys 0m0.023s

$ time python tornado_httpclient.py  1
<!DOCTYPE

real 0m0.469s
user 0m0.163s
sys 0m0.027s
```

PEP 3156

Asynchronous IO Support Rebooted

PEP 3156: ключевые моменты

- Разделение на транспорт и протокол — как в `Twisted`.
- `Future` — объект обещания — подсмотрен в `Java Futures`.
- API построено на `yield from` (до свидания, Python 2).
- `Event loop` — один на поток.
- `Tulip` — reference implementation.
- Вероятно, `Tulip` войдет в Python 3.4 под каким-нибудь скучным именем (осень 2013).

yield from на примере

```
1  #!/usr/bin/env python3
2
3  def a():
4      yield 1
5      yield 2
6      yield 3
7      return 'done_1'
8
9  def b():
10     done = yield from a()
11     print(done)
12     return 'done_b'
13
14 for i in b():
15     print(i)
```

Listing 9: yield_from.py

```
$ python yield_from.py
1
2
3
done_1
```

Tulip — reference implementation

```
#!/usr/bin/env python3
import sys
import tulip
import tulip.http

def curl(url):
    response = yield from tulip.http.request('GET', url)
    data = yield from response.read()
    print(data.decode('utf-8', 'replace')[:10])

if __name__ == '__main__':
    loop = tulip.get_event_loop()
    loop.run_until_complete(curl(sys.argv[1]))
```

“Эй ты, заканчивай там!”

- Asynchronous networking — это правильный инструмент. . . для правильной задачи.
- Python настолько крутой, что допускает как минимум 3 возможных подхода для написания асинхронных сетевых фрэймворков.
- PEP 3156 выглядит многообещающе.

“Эй ты, заканчивай там!”

- Asynchronous networking — это правильный инструмент. . . для правильной задачи.
- Python настолько крутой, что допускает как минимум 3 возможных подхода для написания асинхронных сетевых фрэймворков.
- PEP 3156 выглядит многообещающе.
- Стас, ты забыл рассказать про совместимость на уровне event loop’ов!

СПАСИБО ЗА ВНИМАНИЕ. ВОПРОСЫ?

Стас Рудаков
mailto:stas@garage22.net
mailto:s_rudakou@wargaming.net
https://raw.github.com/nott/talks/async.pdf

http://twistedmatrix.com/
http://www.tornadoweb.org
http://www.gevent.org/
http://www.python.org/dev/peps/pep-3156/
https://code.google.com/p/tulip/