Асинхронное сетевое программирование
Андрей Светлов

andrew.svetlov@gmail.com
asvetlov.blogspot.com

PyCon RU 2015

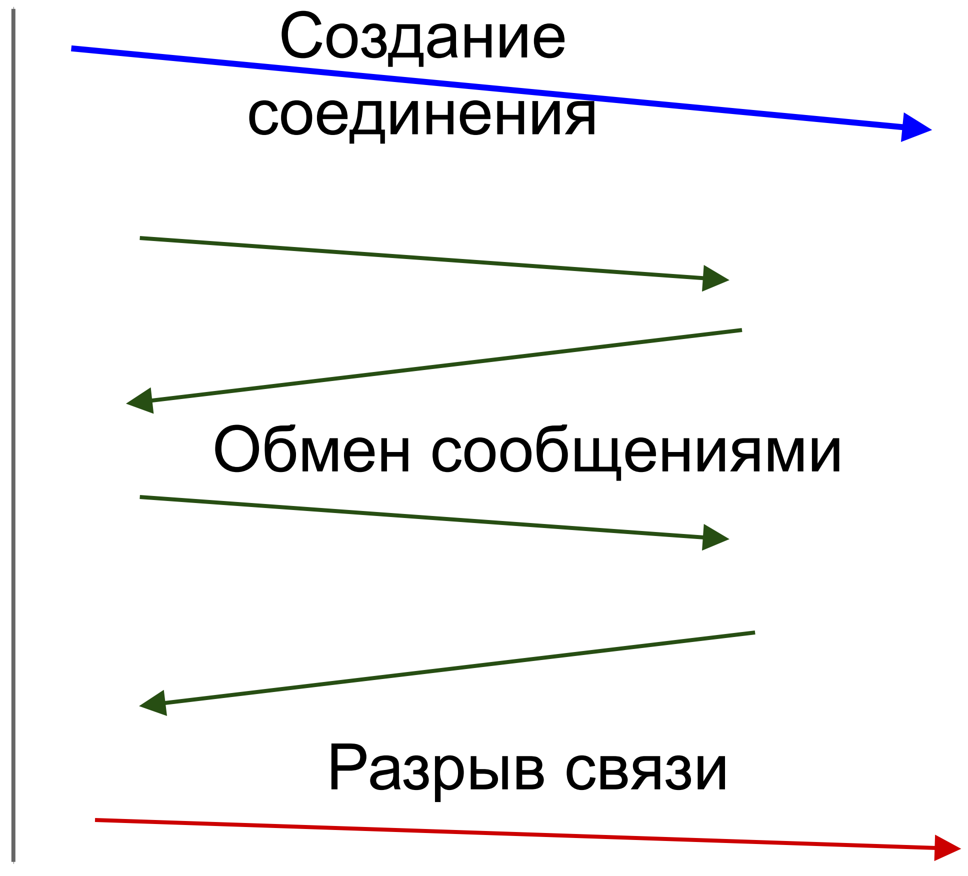
Екатеринбург

Сети = сокеты

- IPv4, IPv6, Unix, RAW и т.д.
- Для IPv4:
  - потоковые (TCP)
  - пакетные (UDP)

TCP, клиент-сервер

Transmission Control Protocol



Синхронные TCP сокеты

- Простые для программирования
- Легкие для изучения
- Практически непригодные для серверной стороны

TCP Сокет

Двунаправленный **байтовый** канал взаимодействия

+

средства управления этим каналом

Пассивные и активные сокеты

Создание соединения

```
import socket

s = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)

s.connect(('127.0.0.1', 7777))
```

Передача данных

```python
sent_size = s.send(b'binary data')
s.sendall(b'data again')

received_data = s.recv(4096)
if received_data:
    process(received_data)
else:
    do_shutdown(s)
```

Прием соединения

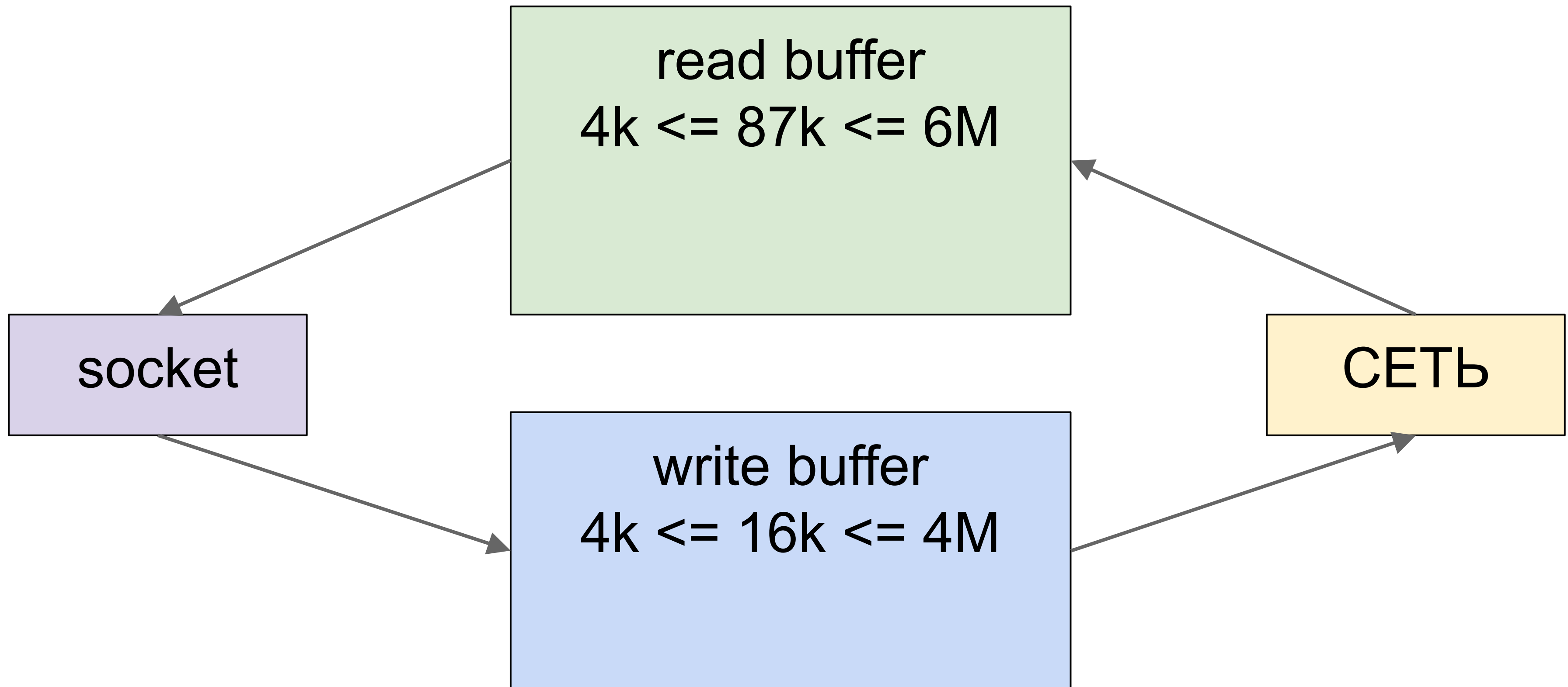
```
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)

s.bind(('127.0.0.1', 7777))
s.listen(10)
while True:
    client, addr = s.accept()
    data = client.recv(4096)
```

Завершение соединения

```python
s.close()
```

```python
s.sendall(b'closing socket')
s.shutdown(socket.SHUT_WR)
data = s.recv(4096)
if data:
    process_final_answer(data)
s.close()
```

Буфера



Настройки

```
opt = s.getsockopt(level, name)
s.setsockopt(level, name, opt)
```

Десятки платформозависимых настроек соединения

Используются относительно редко

`TCP_CORK,TCP_NODELAY`

TIME_WAIT

После закрытия сокета его адрес освобождается **закрывающей стороной** через 1-4 минуты

```
s.setsockopt(SOL_SOCKET,
    SO_REUSEADDR, 1)
```

Обрыв соединения

TCP ***шлёт пакеты*** только если ***есть новые данные*** или ***нужно подвердить доставку***

~~SO_KEEPALIVE~~

Пользовательский keep alive

Ошибки

- Исключение `OSError` (начиная с Python 3.3)
- Причины
  - Адрес не найден
  - Порт не обслуживается
  - Таймаут
  - Разрыв соединения
  - Прочее

Ошибки

```
+-- OSError
|    +-- BlockingIOError
|    +-- ChildProcessError
|    +-- ConnectionError
|    |    +-- BrokenPipeError
|    |    +-- ConnectionAbortedError
|    |    +-- ConnectionRefusedError
|    |    +-- ConnectionResetError
|    +-- FileExistsError
|    +-- FileNotFoundError
|    +-- InterruptedError
|    +-- IsADirectoryError
|    +-- NotADirectoryError
|    +-- PermissionError
|    +-- ProcessLookupError
|    +-- TimeoutError
```

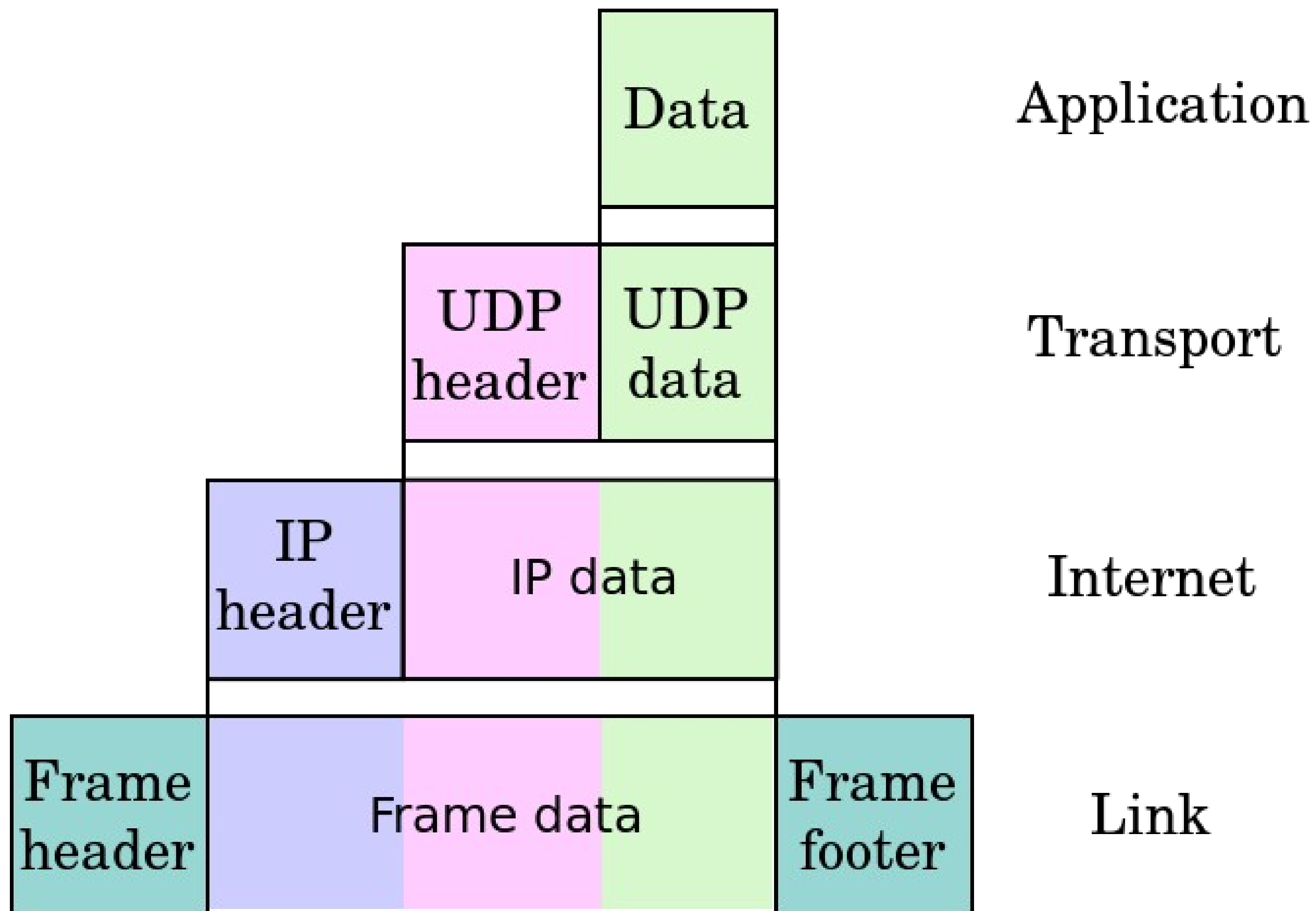
Пакеты

Пользовательские кадры сообщений

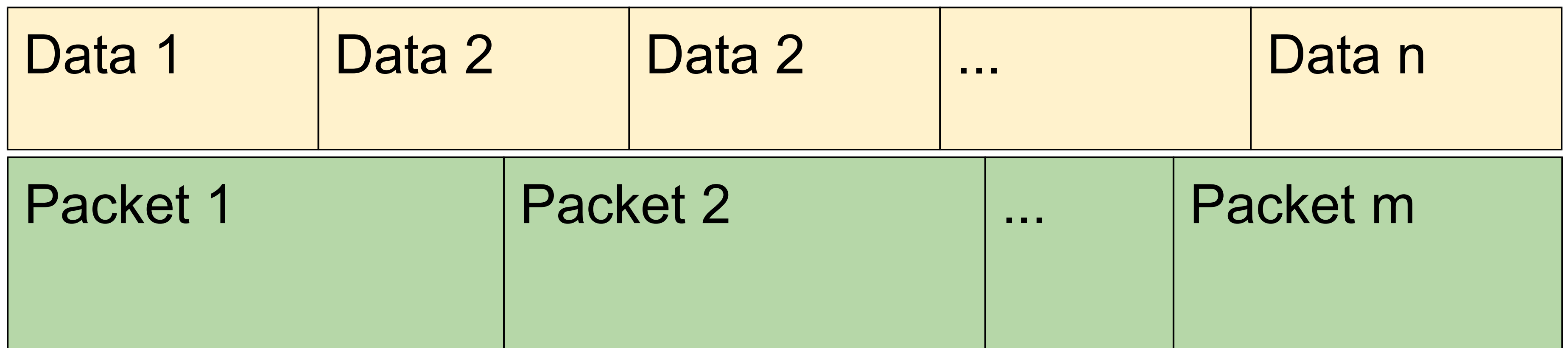
- длина
- тип
- данные
- контрольная сумма

Пара сокетов обменивается **бесконечными** потоками **байтов**, которые порциями проталкиваются через сеть

Под капотом



TCP поток



Минимальный пакет по стандарту 512 байт
На практике минимум 4 килобайта

Ждем событий от нескольких сокетов сразу

Неблокирующие сокеты

```
s = socket.socket(...)

s.setblocking(False)
```

Таймауты делаем сами

~~`s.settimeout(0.5)`~~

selectors

select

poll

epoll

kqueue

Описание интерфейса и принципа работы

Почему неудобно писать «в лоб»

select

```
r, w, x = select.select(rlist,
                        wlist,
                        xlist,
                        [timeout])
```

- 1024 сокета
- 64 на Windows

Пример

```
import select

s1.setblocking(False)
s2.setblocking(False)

reads, writes, errs = select.select(
    [s1, s2], [s2], [], 1.0)
for s in reads:
    data = s.recv(4096)
```

epoll

```
.register(fd, eventmask)
.unregister(fd)
.modify(fd, eventmask)
.poll(timeout)
.fileno()
.close()
.fromfd(fd)

EPOLLIN, EPOLLOUT, EPOLLERR, EPOLLHUP
```

Селекторы

```
poller = select.epoll()
poller.register(s1,EPOLLIN)
poller.register(s2, EPOLLIN|EPOLLOUT)
poller.poll(1.5)
...
poller.unregister(s1)
poller.register(s3, EPOLLIN)
poller.modify(s2, EPOLLIN)
poller.poll(5.7)
```

```
for fd, event in poller.poll(1.5):
    process_timers()
    if event & POLLIN:
        process_read(fd)
    if event & POLLOUT:
        process_write(fd)
```

BaseSelector

- `register(fileobj, events, data=None)`
- `unregister(fileobj)`
- `modify(fileobj, events, data=None)`
- `select(timeout=None)`
- `close()`
- `get_key(fileobj)`

Event loop

Главный и единственный «вечный» цикл программы

- ждём событий
- обрабатываем таймеры
- обрабатываем данные из сокетов

socket <= file descriptor (inotify и т.д.)

Таймеры

- call_soon(cb, *args)
- call_later(delay, cb, *args)
- *time()*
- *call_at(when, cb, *args)*

Handle

- __init__(callback, args)
- __repr__()
- cancel()
- _run()

TimerHandle(Handle)

- __init__(when, callback, args)
- hash & ordering
- __eq__, __ne__, __lt__, __gt__, __le__, __ge__
- __hash__

Event loop

Агрегирует экземпляр Selector`a

Регистраторы

- `add_reader(fd, callback, *args)`
- `remove_reader(fd)`
- `add_writer(fd, callback, *args)`
- `remove_writer(fd)`

Использование

Регистрируем сокет

```
s = socket.socket(...)
s.setblocking(False)

event_loop.add_reader(s.fileno(),
                      reader)

s.connect(('127.0.0.1', 7777))

event_loop.run_forever()
```

reader()

```
try:
    data = s.recv(16*1024)
    process_data(data)
except (BlockingIOError,
        InterruptedError):
    return
except ConnectionError as exc:
    force_close(exc)
except Exception as exc:
    fatal_error(exc)
```

Пассивные сокеты

```
s = socket.socket(...)
s.bind(('127.0.0.1', 7777))
s.listen(100)
s.setblocking(False)

event_loop.add_reader(s.fileno(),
                      acceptor)

event_loop.run_forever()
```

acceptor()

```
try:
    conn, addr = sock.accept()
    conn.setblocking(False)

   event_loop.add_reader(conn.fileno(),
                         reader)
except (BlockingIOError,
        InterruptedError):
    return
except Exception:
    self.remove_reader(sock.fileno())
    sock.close()
```

~~Tornado~~

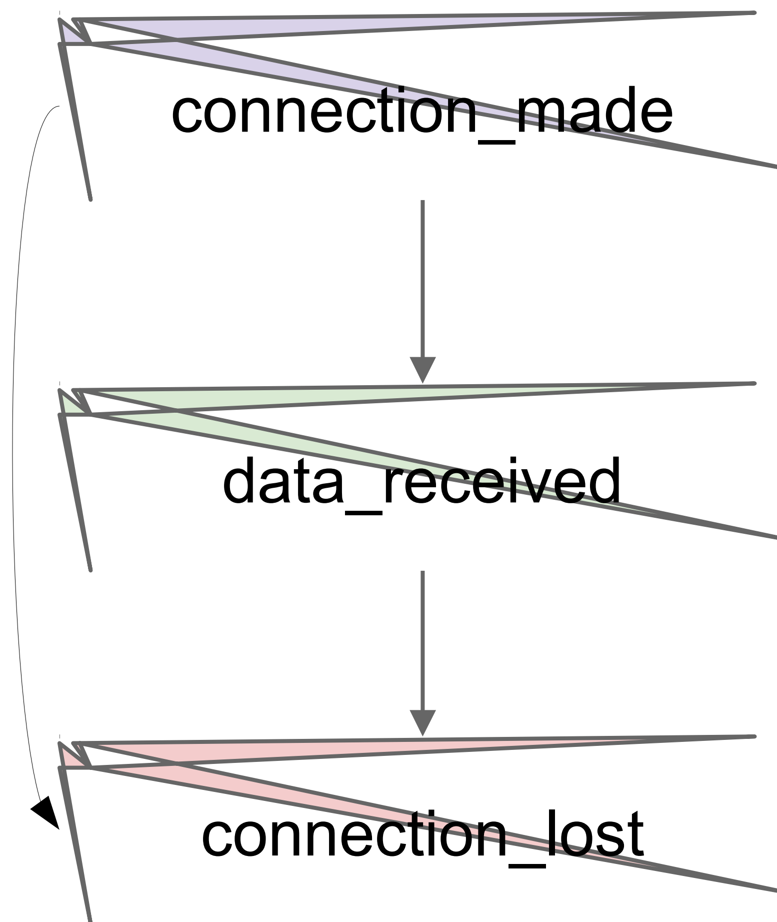
Transport

- **write(buffer)**
- ***close()***
- ***abort()***
- *pause()*
- *resume()*
- *pause_writing()*
- *resume_writing()*
- *discard_output()*
- *write_eof()*
- *can_write_eof()*

Protocol

- connection_made(transport)
- data_received(data)
- connection_lost(exc)
- *eof_received()*

Жизненный цикл протокола



~~**twisted**~~

Сложности в работе с transport-protocol

- Обратные вызовы остались
- Код нелинейный
- Любое действие программиста может быть только ответом на обратный вызов (пример с таймерами)

Отложенный результат: Deferred, Future, Promise

Future(*, loop=None)

- **cancel()**
- cancelled()
- done()
- **result()**
- **exception()**
- **add_done_callback(cb)**
- remove_done_callback(cb)
- ***set_result(result)***
- ***set_exception(exception)***

Создание отложенного вызова

```
def cb(f):
    pass

f = asyncio.Future()
f.add_done_callback(cb)
```

```
fut = asyncio.Future()

yield from fut


fut.set_result(123)
```

Высокоуровневое программирование

asyncio streams

```
reader, writer = yield from open_connection(
    'localhost', 8080)

while True:
    data = yield from reader.read(1024)
    writer.write(data)
    yield from writer.drain()
```

Генераторы как легковесные потоки

yield

```
def f():
    x = yield 1
    try:
        y = yield 2
    except Exception as ex:
        z = yield 10000
    raise RuntimeError("EXC")
    return 'done'
```

```
g = f()
a = g.send(None)
b = g.send('x')
c = g.throw(ValueError("err"))
try:
    d = g.send('z')
except RuntimeError ex:
    pass
```

Цепочки генераторов

```
for i in f():
    yield i

yield from f()
```

RESULT = yield from EXPR

```
_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
```

```python
transp, proto = yield from loop\
    .create_connection(
        proto_factory, host, port)

family, type, proto, name, addr = \
    yield from loop.getaddrinfo(
                              host, port)

data = yield from loop.sock_recv(
                             sock, 1024)
```

```python
@coroutine
def sleep(delay, result=None, *,
          loop=None):
    f = futures.Future(loop=loop)
    h = f._loop.call_later(
          delay, f.set_result, result)
    try:
        return (yield from f)
    finally:
        h.cancel()
```

task

- Наследник Future
- Выполняет себя пока не закончится
- coroutine должна быть прокручена yield from

Запуск легковесного процесса

```
task = loop.create_task(coro(1, 2, 3))



yield from task
```

Пример

```
@coroutine
def curl(url):
    client = aiohttp.ClientSession()
    resp = yield from client.get(url)
    txt = yield from resp.text()
    return txt

f = async(curl('www.python.org'))
txt = event_loop.run_until_complete(f)
```

Отмена задач

- По таймауту
- По явному запросу

```
t.cancel()
```

Синхронизация

- gather
- Lock
- Event
- Condition
- Semaphore

yield from asyncio.gather(f1(), f2())

yield from asyncio.wait_for(f3(), 1.5)

В лоб

```
lock = Lock()
yield from
  lock.acquire()
try:
    f()
finally:
    lock.release()
```

```
lock = Lock()
lock.acquire()
try:
    f()
finally:
    lock.release()
```

Контекстный менеджер

```
lock = Lock()
with (yield from
  lock):
    f()
```

```
lock = Lock()
with lock:
    f()
```

Очередь

```
q = asyncio.Queue(maxsize=5,
                  loop=loop)

val = yield from q.get()

yield from q.put(5)
```

Jason

```
queue = jason.Queue(maxsize=5,
  loop=loop)

val = yield from queue.async_q.get()

queue.sync_q.put(5)
```

Вопросы?

Андрей Светлов

andrew.svetlov@gmail.com