Twisted

Silnik Twojego Internetu

Jan Urbański
wulczer@wulczer.org

Ducksboard

PyWaw #25, Warszawa, 10 czerwca 2013

Zarys

Programowanie asynchroniczne

- wiekszość programów jest ograniczona przez I/O
- Twisted stara się zmaksymalizować przepustowość przez wykonywanie operacji asynchronicznie
  - w czasie, gdy aplikacja czeka na zakończenie zapytania w bazie danych...
  - ... wysyła już zrenderowany szablon innemu klientowi
  - ... w czasie, gdy klient odczytuje dane z sieci...
  - ... parsuje nagłówki od następnego klienta
- metoda holywoodzka: nie dzwoń, my zadzwonimy

Krótki opis

- biblioteka do pisania programów sieciowych
  - zarówny klientów, jak i serwerów
- asynchroniczny model działania
- wsparcie dla wielu protokołów
  - HTTP, FTP, SMTP
  - DNS, NNTP, SSH
  - ... i bardziej egzotyczne
- dużo pomocniczych bibliotek: parsowanie linii poleceń, obsługa wątków itp.

Przykładowe zastosowania

- serwer pocztowy, który parsuje pocztę i przesyła ją przez HTTP do innego systemu
- serwer przyjmujący połączenia WebSockets i wsyłający klientom dane z kolejki AMQP + interfejs administracyjny przez SSH
- serwer FTP, który publikuje na Twitterze i na kanale jabberowym każdy nowy upload
- program wyświetlający na tablicy LED najnowsze pytania ze StackOverflow z tagiem Python

Zarys

Filozofia w kilku słowach

- niespotykany nacisk na jakość kodu
- jedna z najinteligentniejszych społeczności wolnego oprogramowania
- dużo interfejsów i dokumentacji, wyłącznie przemyślane decyzje
- niepowstrzymany, nawet jeśli powolny, marsz ku ideałowi
- celem jest zniszczenie amerykańskiej waluty promieniami TCP IP

Historia

- Twisted powstaje w połowie 2000 roku
  - Python 2.0 został wydany w październiku 2000 roku
  - PEP8 powstaje w sierpniu 2001 roku
  - moduł logging został zaimportowany do Pythona w listopadzie 2002 roku
- część projektu Ultima Online 2
- początkowo w Javie, kod został szybko przepisany w Pythonie

Proces

- UDQS (Ultimate Quality Development System)
  - każda zmiana w kodzie ma odpowiadający ticket
  - każda zmiana musi być przez kogoś sprawdzona
  - wszystkie zmiany dokonywane są w gałęziach
- nowy kod wymaga 100% pokrycia testami
- pełna dokumentacja wszytkich publiczych funkcji i zmiennych
- niekompatybilne zmiany są zabronione

Zarys

Reaktor

- reaguje na wydarzenia i wywołuje kod użytkownika
  - dane czekające na odczytanie z gniazda sieciowego
  - kliknięcia myszki, dane z klawiatury
  - dane na porcie szeregowym
  - upływ czasu
- łatwy do zastąpienia dzięki dobrze zdefiniowanym interfejsom
- różne reaktory używają różnych mechanizmów systemowych
  - `select`
  - `poll`, `epoll`
  - `kqueue`
  - `I/O Completion Ports`
  - pętle wydarzeń `GTK`, `Qt`

Zarys

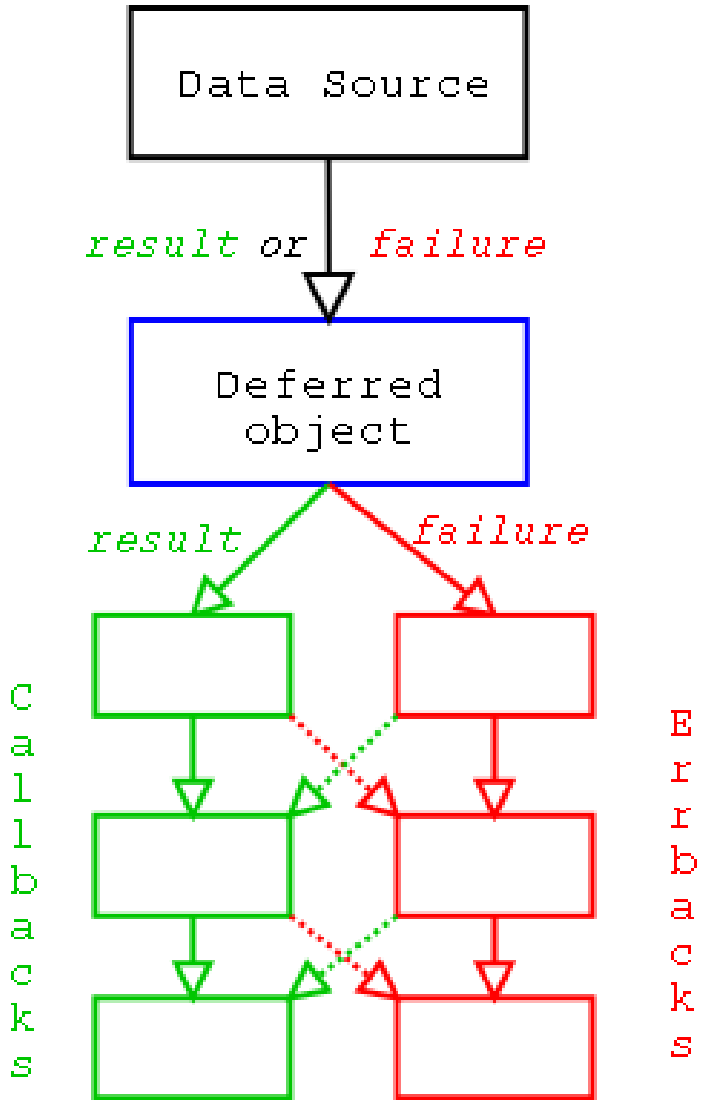
Abstracja Deferred

- Deferred to inaczej Promise albo Future
- wartość, która nie jest jeszcze gotowa do użycia
  - wynik zapytania do bazy danych
  - rezultat operacji wykonanej w innym wątku lub procesie
  - wykonanie metody na zdalnym obiekcie
- Deferred mogą wypalić albo zawieść

Callbacki i errbacki

- do Deferred można dołączyć funkcje `callback` i `errback`
- callback dostaje rezultat operacji, na którą Deferred czekał
- w razie błędów, sterowanie przechodzi do łańcucha errbacków
- `errback` może obsłużyć błąd, albo rzucić go dalej

Callbacki i errbacki, cd.



Sterowanie w Deferreds

- jeśli callback zwróci Deferred, wykonanie na niego czeka
- jeśli callback zawiedzie, sterowanie przekazywane jest do najbliższego errbacku
- jeśli errback nie zwróci błędu, sterowanie wraca do callbacków
- wynik Deferred jest wynikiem zwróconym przez ostatni callback

Łączenie Deferreds

- połączenie HTTP generuje apytanie do bazy danych
- ... które zwraca identyfikator...
- ... który jest użyty w zapytaniu do memcache...
- ... którego odpowiedź zwraca adres email...
- ... do którego wysyłana jest poczta i zapytanie HTTP się kończy

Łączenie Deferreds, cd

Łańcuch callbacków

```
def handle_request(req):
    sid = req.cookies.get('session_id')
    d = db.query('select id from ppl where sid = %s', sid)
    d.addCallback(email_from_cache)
    d.addCallback(send_mail)
    return d.addCallback(build_response, req)
```

Łączenie Deferreds 2, cd

Łańcuch callbacków

```
def email_from_cache(user_id):
    return memcache.get(user_id).addCallback(
        lambda res: res['email'])

def send_mail(email):
    return smtp.send_mail(email, 'so twisted!')

def build_response(res, req):
    req.finish(200, 'sent' if res else 'not sent')
```

Obsługa błędów

Łańcuch errbacków

```
def handle_request(req):
    sid = req.cookies.get('session_id')
    d = db.query('select id from ppl where sid = %s', sid)
    d.addCallback(email_from_cache)
    d.addErrback(not_in_cache)
    d.addCallback(send_mail)
    d.addCallback(build_response, req)
    return d.addErrback(request_failed, req)
```

Obsługa błędów

Łańcuch errbacków

```
def not_in_cache(f):
    f.trap(NoResultFound)
    return 'default@example.org'

def request_failed(f, req):
    req.finish(500, f.getErrorMessage())
```

Dzięki! Pytania?