



Awesome async / await avec Python 3.5

Bordeaux Engineers Share & Talk #3
27 juin 2018

Thibaut Séguy

CTO @ Actiplay

Data marketing / Adtech

<https://github.com/thibauts>

<https://twitter.com/thibautseguy>

Python et concurrence, un historique tourmenté

- Threading / polling (vanilla)
- Callbacks, deferred, futures, promises (twisted, tornado)
- Green threads (gevent, greenlet, coroutines)
- yield from (Python 3.3)
- asyncio (Python 3.4)
- async / await (Python 3.5)

Convergence des mécanismes de concurrence

- Javascript : callbacks, promises ... async / await !
- C#, Javascript, Scala, Dart, Python l'ont adopté
- Une syntaxe uniforme, qui masque les complexités d'implémentation

Commençons par la documentation d'asyncio :

Bounded semaphores

Stream writers

Futures

Unix signals

Coroutines

Pipes

Locks

Protocols

Delayed calls

Tasks

Executors

Stream readers

Coroutine chaining

Queues

Events

Event loop

Transports

Cancellation

- 18.5.1.1. **BaseEvent** loop
 - 18.5.1.1.1. Run an event loop
 - 18.5.1.1.2. **Calls**
 - 18.5.1.1.3. Delayed calls
 - 18.5.1.1.4. **Runners**
 - 18.5.1.1.5. **Tasks**
 - 18.5.1.1.6. Creating connections
 - 18.5.1.1.7. Creating listening connections
 - 18.5.1.1.8. **Waiters** (connections)
 - 18.5.1.1.9. Low-level socket operations
 - 18.5.1.1.10. **Reactive** base name
 - 18.5.1.1.11. **Connections**
 - 18.5.1.1.12. **Flow** signals
 - 18.5.1.1.13. **IOStreams**
 - 18.5.1.1.14. **Error Handling API**
 - 18.5.1.1.15. **Debug mode**
 - 18.5.1.1.16. **Server**
 - 18.5.1.1.17. **Handle**
 - 18.5.1.1.18. **Event loop examples**
 - 18.5.1.1.18.1. **hello world with call back**
 - 18.5.1.1.18.2. **Measure the current date with call back**
 - 18.5.1.1.18.3. **Run a file stream for read events**
 - 18.5.1.1.18.4. **Use signal handlers for input and output**
- 18.5.2. **Event loop**
 - 18.5.2.1. **Event loop features**
 - 18.5.2.1.1. **Asynchronous event loop**
 - 18.5.2.1.2. **Platform support**
 - 18.5.2.1.3. **Non-blocking**
 - 18.5.2.1.4. **Use C++**
 - 18.5.2.2. **Event loop policies and the default policy**
 - 18.5.2.3. **Event loop policy interface**
 - 18.5.2.4. **Access to the global read policy**
 - 18.5.2.5. **Constructing an event loop policy**
- 18.5.3. **Tasks and runners API**
 - 18.5.3.1. **Callbacks**
 - 18.5.3.1.1. **Callback: Hello world coroutine**
 - 18.5.3.1.2. **Example: Coroutine displaying the current time**
 - 18.5.3.1.3. **Callback: CPU operations**
 - 18.5.3.2. **TaskRunner**
 - 18.5.3.3. **Task runner**
 - 18.5.3.3.1. **TaskRunner: Future with run, cancel, and sleeping**
 - 18.5.3.3.2. **Callback: Future with run, forever**
 - 18.5.3.4. **Task**
 - 18.5.3.4.1. **Example: Parallel execution of tasks**
- 18.5.4. **Transports and protocols (callback-based API)**
 - 18.5.4.1. **Transports**
 - 18.5.4.1.1. **BaseTransport**
 - 18.5.4.1.2. **BaseTransport**
 - 18.5.4.1.3. **NetTransport**
 - 18.5.4.1.4. **CoroutineTransport**
 - 18.5.4.1.5. **BaseTransportTransport**
 - 18.5.4.2. **Protocols**
 - 18.5.4.2.1. **Protocol class**
 - 18.5.4.2.2. **Converter callback**
 - 18.5.4.2.3. **Streaming protocol**
 - 18.5.4.2.4. **DataStreamProtocol**
 - 18.5.4.2.5. **Flow control callback**
 - 18.5.4.2.6. **Converter and protocols**
 - 18.5.4.3. **Protocol examples**
 - 18.5.4.3.1. **TCP echo client protocol**
 - 18.5.4.3.2. **TCP echo server protocol**
 - 18.5.4.3.3. **UDP echo client protocol**
 - 18.5.4.3.4. **UDP echo server protocol**
 - 18.5.4.3.5. **Tag-based API: sleep before wait for data using a protocol**
- 18.5.5. **Streams (coroutine-based API)**
 - 18.5.5.1. **Stream functions**
 - 18.5.5.1.1. **StreamAdapter**
 - 18.5.5.1.2. **StreamFilter**
 - 18.5.5.1.3. **StreamReadProtocol**
 - 18.5.5.1.4. **IncompleteAdapter**
 - 18.5.5.1.5. **StreamAdapterFilter**
 - 18.5.5.1.6. **Stream examples**
 - 18.5.5.1.6.1. **TCP echo client using streams**
 - 18.5.5.1.6.2. **TCP echo server using streams**
 - 18.5.5.1.6.3. **GET HTTP request**
 - 18.5.5.1.6.4. **Tag-based API: sleep before wait for data using streams**
- 18.5.6. **Sessions**
 - 18.5.6.1. **WebSocket event loop**
 - 18.5.6.2. **Create a high-level API using `WebSocket`**
 - 18.5.6.3. **Create a subprotocol: low-level API using `WebSocket`**
 - 18.5.6.4. **Channels**
 - 18.5.6.5. **Protos**
 - 18.5.6.6. **Subprotocols and threads**
 - 18.5.6.7. **Sessions examples**
 - 18.5.6.7.1. **Subprotocol using `WebSocket` and `WebSocket`**
 - 18.5.6.7.2. **Subprotocol using `Channels`**
- 18.5.7. **Serialization primitive**
 - 18.5.7.1. **Ioops**
 - 18.5.7.1.1. **IOV**
 - 18.5.7.1.2. **IOV**
 - 18.5.7.1.3. **Condition**
 - 18.5.7.2. **Sessions**
 - 18.5.7.2.1. **Sessions**
 - 18.5.7.2.2. **Serialized `WebSocket`**
- 18.5.8. **Queue**
 - 18.5.8.1. **Queue**
 - 18.5.8.2. **PriorityQueue**
 - 18.5.8.3. **Unbounded**
 - 18.5.8.3.1. **Exceptions**
- 18.5.9. **Queueing framework**
 - 18.5.9.1. **Queueing mode of `async`**
 - 18.5.9.2. **Queueing API**
 - 18.5.9.3. **Concurrency and multithreading**
 - 18.5.9.4. **Handle blocking functions in `async`**
 - 18.5.9.5. **Logging**
 - 18.5.9.6. **HTTP handling: request must be created**
 - 18.5.9.7. **Object-oriented: must be created**
 - 18.5.9.8. **Chain-oriented: connector**
 - 18.5.9.9. **Flowing: no connector**
 - 18.5.9.10. **HTTP handling: `async` and `HTTP`**

💀 ASYNCRIO 💀

Mais si, c'est beaucoup plus simple ! Reviens !

Live coding time !

Woohoo !

Le nécessaire pour commencer

- `async def`, `async with`, `async for`
- `await`
- `asyncio.get_event_loop()`
- `loop.run_forever()`

Utilisation en production

- Warning: modules spécifiques
- Ecosystème en évolution constante (<https://github.com/aio-libs>)
- OK si le périmètre est bien défini (notamment, choix des datastores)

Merci !

<https://github.com/thibauts/talk-asyncio-best>