Генераторы. Корутины. Асинхронность

@pvavilin

2 апреля 2023 г.

Outline

Что такое асинхронное программирование?

концепция программирования, которая заключается в том, что результат выполнения функции доступен не сразу же, а через некоторое время в виде некоторого асинхронного (нарушающего обычный порядок выполнения) вызова.

Wiki

Пример синхронного программирования

```
def func(x, y):
    import time
    time.sleep(1)
    return x + y
def run():
    fut = pool.submit(func, 2, 3)
    r = fut.result()
    print('Got:', r)
run()
print("DONE")
```

- Got: 5
- DONE

Пример асинхронного программирования

```
def result_handler(fut):
    result = fut.result()
    print('Got:', result)
def run():
    fut = pool.submit(func, 2, 3)
    fut.add_done_callback(
        result_handler
    )
run()
print("DONE")
```

```
- DONE
- Got: 5
```

Что такое генераторы в Python

Генератор это функция, которая производит *последовательность результатов* а не единичный ответ.

```
print(str(x) for x in range(10))
- <generator object <genexpr> at 0x7fd86d1ffac0>
def countdown(n):
    while n > 0:
        yield n
        n -= 1
for i in countdown(3):
    print(i, sep='', end='...')
- 3...2...1...
```

Генераторы

Python понимает, что функция это генератор по наличию в функции метода yield.

Генераторы не запускаются автоматически при вызове, а только инициализируются

```
def countdown(n):
    print(f"Обратный отсёт для {n}")
    while n > 0:
        yield n
        # точка остановки
        n -= 1
g = countdown(3)
print(g)
- <generator object countdown at 0x7f3337b34e40>
```

Генераторы

Чтобы запустить генератор, надо вызывать метод next

```
g = countdown(3)
next(g)
- Обратный отсёт для 3
```

Генераторы

Генератор будет работать до тех пор пока не случится return

```
g = countdown(2)
print(next(g))
print(next(g))
try:
    print(next(g))
except StopIteration:
    print("КОНЕЦ")
```

```
- Обратный отсёт для 2
- 2
- 1
- КОНЕЦ
```

Генераторы как контекстные менеджеры

```
from contextlib import contextmanager
import time

@contextmanager
def timeit():
    import time
    try:
        start = time.time()
        yield start
    finally:
        end = time.time()
        print(f"{end-start:.2f}")
with timeit():
    time.sleep(2)
```

2.00

Пример использования генераторов

В Bash можно направлять результат работы одной программы в другую, причём данные в первую программу могут поступать даже после запуска *пайпа*

```
# на случай если такого файла
# не существовало
# или в нём что-то уже было,
# запишем в него пустоту
:> /tmp/t.txt
# tail -f => "follow" новые строки
#                      в файле
# grep -i python => искать вхождение
#                   подстроки python
tail -f /tmp/t.txt | grep -i python
```

Пример использования генераторов

Как реализовать такое на Python?

```
def follow(filepath, grepper):
    with open(filepath, "r") as fd:
        # "сикнемся" в конец файла
        fd.seek(0, 2)
        while True:
            line = fd.readline()
            if not line:
                # небольшая пауза
                time.sleep(0.1)
                continue
            grepper(line)
```

Пример использования генераторов

```
def grep(pattern, lines):
    pattern = pattern.lower()
    for line in lines:
        if pattern in line.lower():
            yield line
```

Пример использования генераторов

```
def follow(filepath):
    with open(filepath, "r") as fd:
        fd.seek(0, 2)
        while True:
            line = fd.readline()
            if not line:
                time.sleep(0.1)
                continue
            yield line
for line in grep("org", grep(
    "python", follow("/tmp/t.txt")
)):
    print(line)
```

Пример использования генераторов

```
with open(
    "/usr/share/doc/python3.10/copyright"
) as fd:
    print(
        '\n'.join(grep(
            "http",
                grep("python", fd.readlines())
            )
        )
    )
```

Корутины это генераторы

На самом деле yield принимает значение и возвращает его внутрь генератора.

```
# docs.python.org/3/library/typing.html
G = Generator[int, int, None]
def countdown(n) -> G:
    while n > 0:
        shift = (yield n)
        n -= 1
        if shift is not None:
            n += shift
g = countdown(1)
print(next(g))
print(g.send(10))
```

```
- 1
- 10
```

Корутины

В корутины можно передать эксепшен

```
def cor(n):
    while n > 0:
        try:
            yield n
            n -= 1
        except ValueError:
            print("Поймал!")
g = cor(3)
next(g)
g.throw(ValueError, "foobar")
- Поймал!
```

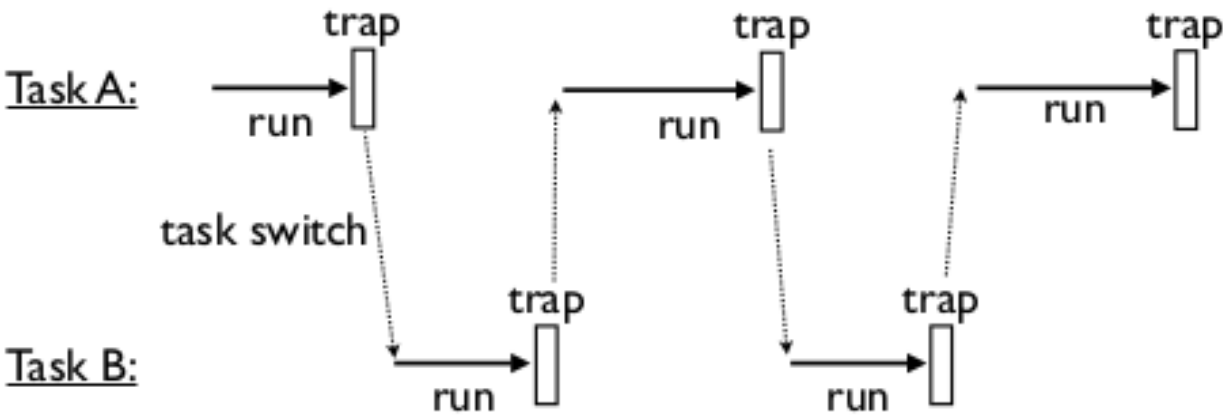
Пример использования корутин

```
def follow(filepath, target):
    with open(filepath, "r") as fd:
        fd.seek(0,2)
        while True:
            line = fd.readline()
            if not line:
                time.sleep(0.1)
                continue
            target.send(line)
```

Пример использования корутин

```
@coroutine
def printer():
    while True:
        line = (yield)
        print(line)
@coroutine
def broadcast(targets):
    while True:
        item = (yield)
        for target in targets:
            target.send(item)
follow("/tmp/t.txt", broadcast(
    [printer(), printer()]))
```

Task scheduling



Task

```
class Task:
    task_id = 0
    def __init__(self, target):
        Task.task_id += 1
        self.tid = Task.task_id
        # target coroutine
        self.target = target
        # value to send
        self.sendval = None
    def run(self):
        return self.target.send(
            self.sendval
        )
```

Task example

```
def foo():
    for i in range(2):
        yield i
t1 = Task(foo())
print(t1.run())
print(t1.run())
```

```
- 0
- 1
```

Scheduler

```
from queue import Queue

class Scheduler:
    def __init__(self):
        self.ready = Queue()
        self.taskmap = {}

    def new(self, target):
        newtask = Task(target)
        self.taskmap[newtask.tid] \
            = newtask
        self.schedule(newtask)
        return newtask.tid
```

Scheduler

```
def schedule(self,task):
    self.ready.put(task)

def mainloop(self):
    while self.taskmap:
        task = self.ready.get()
        result = task.run()
        self.schedule(task)
```

Scheduler example

```
def foo():
    while True:
        print("I'm foo")
        yield

def bar():
    while True:
        print("I'm bar")
        yield

sched = Scheduler()
sched.new(foo())
sched.new(bar())
sched.mainloop()
```

Scheduler Exit

```
def exit(self,task):
    print(f"Task {task.tid} terminated")
    del self.taskmap[task.tid]

def mainloop(self):
     while self.taskmap:
         task = self.ready.get()
         try:
             _ = task.run()
         except StopIteration:
             self.exit(task)
             continue
         self.schedule(task)
```

Scheduler example

```
def foo(n):
    for i in range(n):
        print("I'm foo")
        yield

def bar(n):
    for i in range(n):
        print("I'm bar")
        yield

sched = Scheduler()
sched.new(foo(2))
sched.new(bar(2))
sched.mainloop()
```

Дополнительные материалы

dabeaz.com

презентация старая, там используется Python2, будьте внимательны, синтаксис немного отличается!

AsyncIO

```
import time
import asyncio
async def count():
    print("One")
    await asyncio.sleep(1)
    print("Two")
async def main():
    await asyncio.gather(
        count(), count()
    )
asyncio.run(main())
```

```
- One
- One
- Two
- Two
```

AsyncIO event loop

```
loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(main())
finally:
    loop.close()
```

Запуск sync в async

```
import asyncio
from requests import get
from contextlib import asynccontextmanage

@asynccontextmanager
async def web_page(url):
    loop = asyncio.get_event_loop()
    yield await loop.run_in_executor(
        None, get, url)
async def main():
    async with web_page(
            "https://ya.ru") as data:
        print(data.content.decode("utf-8
loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

Запуск async в sync

```
# async downloader
async def rng(n):
    for i in range(n):
        yield i

async def foo(n):
    async for i in rng(n):
        print(i)
```

Запуск async в sync

```
# sync scheduler
def task(n):
    import asyncio
    loop = asyncio.get_event_loop()
    loop.run_until_complete(foo(n))

# register sync task in the sceduler
task(5)
```

```
0
1
2
3
4
```

Запуск async в Jupyter

Проблема
Обсуждение

```
# скорее всего даже не надо
%autowait asyncio

await foo(5)
```

Дополнительная литература

- AsyncIO in Python
- Using AsyncIO in Python

Вопросы-ответы