Модуль concurrent.futures в Python

Параллельное выполнения задач в разных процессах или потоках

Модуль concurrent.futures предоставляет высокоуровневый интерфейс для асинхронного выполнения вызываемых объектов, с использованием пулов потоков threads или ядер процессора process.

Плюсы модуля concurrent.futures:

- прост для понимания как пять копеек, доступен даже новичку в Python;
- при использовании класса ThreadPoolExecutor, задачи выполняются в потоках;
- при использовании класса ProcessPoolExecutor, задачи выполняются на ядрах процессора;
- оба класса реализуют одинаковый API-интерфейс, который определяется абстрактным классом concurrent.futures.Executor (смотрите ниже по тексту), поэтому приложения могут переключаться между потоками и процессами с минимальными изменениями;
- легко интегрируется в модуль asyncio, для запуска блокирующих операции в отдельных потоках/процессах.

Минусы модуля concurrent.futures:

- менее гибок, по сравнению полноценными модулями threading и multiprocessing.

Чтобы использовать пул потоков/процессов с задачами, приложение создает экземпляр соответствующего класса исполнителя, а затем отправляет задачи для их запуска. При запуске каждой задачи создается экземпляр класса Future этой задачи, который ставится в очередь на выполнение в пуле потоков/процессов.

Для получения результатов задач, приложение может использовать объект Future для блокировки дальнейшего выполнения программы, пока результат задачи в пуле потоков/процессов не станет доступен. При этом не нужно напрямую управлять объектами Future, для удобства ожидания обработки данных, модулем предоставляются различные API-интерфейсы.

Модуль concurrent.futures определяет абстрактный класс *concurrent.futures.Executor()*, который предоставляет основные методы для асинхронного выполнения вызовов. Его следует использовать не напрямую, а через его конкретные подклассы, о которых говорилось выше. В данной документации к модулю про этот класс больше говорится не будет, а его методы (т.к. они наследуются) будут рассмотрены в конкретных классах.

Вверх

Как получать результаты работы пулов потоков/процессов?

1. При помощи метода ThreadPoolExecutor.map(), который ждет выполнения всех задач запущенных в пуле и по окончании возвращает результаты в виде итератора. Результаты обработки данных всегда будут возвращаться в том порядке, в каком они поступали при запуске пула потоков/процессов. Смотрите пример.
2. Подождать окончания работы пула потоков/процессов при помощи функции модуля as_completed(Future), а потом для каждого объекта Future вызвать метод Future.result() с результатами его работы. Используйте данный подход, если не имеет значения, в каком порядке будут поступать результаты обработки данных. Смотрите пример.

Использование ThreadPoolExecutor.map() для получения результатов работы.

Первый пример разберем на объект ThreadPoolExecutor, который управляет набором рабочих потоков, передавая им задачи, когда они становятся доступными для дополнительной работы.

В примере используется метод ThreadPoolExecutor.map() для одновременного создания множества результатов из итерируемого ввода. Для демонстрации того, что этот метод всегда возвращает значения в порядке, основанном на вводе, будем приостанавливать выполнение задач в потоках на разное количество времени при помощи функции time.sleep().

Возвращаемое значение из метода ThreadPoolExecutor.map() на самом деле является особым типом итератора, который ждет результаты выполнения от всех запущенных задач в пуле потоков/процессов.

```python
import concurrent.futures as pool
import threading, time

def worker(n):
    # имя конкретного рабочего потока
    th_name = threading.current_thread().name
    th_name = th_name.replace('PoolExecutor-0_', '-')
    print(f'{th_name}: обработка данных => {n}')
    result = n / 10
    time.sleep(result)
    print(f'{th_name}: обработка закончена => {n}')
    return result

# данные для обработки в пуле потоков
data = list(range(1, 10, 2))

# имя основного потока программы
th_name = threading.current_thread().name
print(f'{th_name}: запущен...')
# создаем и запускаем пул из 3 потоков
with pool.ThreadPoolExecutor(max_workers=3) as executer:
    # в метод map передаем функцию
    # worker()  и данные для обработки
    res = executer.map(worker, data)

# получаем результаты в виде списка
results = list(res)
print(f'{th_name}: результаты => {results}')


# MainThread: запущен...
# Thread-0: обработка данных => 1
# Thread-1: обработка данных => 3
# Thread-2: обработка данных => 5
# Thread-0: обработка закончена => 1
# Thread-0: обработка данных => 7
# Thread-1: обработка закончена => 3
# Thread-1: обработка данных => 9
# Thread-2: обработка закончена => 5
# Thread-0: обработка закончена => 7
# Thread-1: обработка закончена => 9
# MainThread: результаты => [0.1, 0.3, 0.5, 0.7, 0.9]
```

Использование метода Future.result() для получения результатов работы.

Этот пример аналогичен первому, только будет основываться на пуле ядер процессора (не потоков), а получение результатов обработки данных будет основываться на методе Future.result().

Вызов метода Future.result() объекта Future блокируется до тех пор, пока задача не будет завершена: либо путем возврата значения или возникновения исключения, либо до отмены самого задания. Функция модуля as_completed() возвращает объекты Future по мере их завершения, из которых уже можно получить результат работы методом Future.result().

Обратите внимание, что результаты будут возвращаться не по порядку поступления данных из data или очередности запуска процессов в пуле, а по окончанию обработки поступившей порции данных в worker() в каждом процессе. Для демонстрации этого поведения, функция worker() останавливает выполнение задач в потоках на разное время (строка с time.sleep(result)).


```
import concurrent.futures as pool
import multiprocessing, time

def worker(n):
    # имя конкретного рабочего процесса
    core_name = multiprocessing.current_process().name
    print(f'{core_name}: обработка данных => {n}')
    result = n / 10
    time.sleep(result)
    print(f'{core_name}: обработка закончена => {n}')
    return result

# данные для обработки в пуле процессов
data = list(range(1, 10, 2))

# имя основного процесса программы
core_name = multiprocessing.current_process().name
print(f'{core_name}: запущен...')
# создаем и запускаем пул с использованием
# 3-х ядер процессора
with pool.ProcessPoolExecutor(max_workers=3) as executer:
    wait_complete = []
    # проходимся по списку с данными для обработки
    for task in data:
        # запускаем worker() для каждого элемента данных
        future = executer.submit(worker, task)
        # в список складываем объекты 'future'
        # с будущими результатами
        wait_complete.append(future)

print(f'{core_name}: результаты работы пула:')
# ждем окончания выполнения пула процессов
for res in pool.as_completed(wait_complete):
    # возвращаем обработанные данные
    print(f':=> {res.result()}')


# MainProcess: запущен...
# Process-1: обработка данных => 1
# Process-2: обработка данных => 3
# Process-3: обработка данных => 5
# Process-1: обработка закончена => 1
# Process-1: обработка данных => 7
# Process-2: обработка закончена => 3
# Process-2: обработка данных => 9
# Process-3: обработка закончена => 5
# Process-1: обработка закончена => 7
# Process-2: обработка закончена => 9
# MainProcess: результаты работы пула:
# :=> 0.3
# :=> 0.1
# :=> 0.9
# :=> 0.5
# :=> 0.7
```

Содержание раздела:

Вверх

- КРАТКИЙ ОБЗОР МАТЕРИАЛА.

Вверх