

ВВЕДЕНИЕ В THREADPOOLEXECUTOR

ВВЕДЕНИЕ

Задачи урока

- 01 Изучить основные особенности ThreadPoolExecutor в Python
- 02 Научиться эффективно использовать пул потоков для параллельного выполнения задач

ThreadPoolExecutor

это класс, предоставляемый модулем concurrent.futures в Python, который представляет пул потоков для эффективного выполнения асинхронных операций

ThreadPoolExecutor предоставляет простой и удобный интерфейс для параллельного выполнения задач в фоновых потоках

Основные особенности ThreadPoolExecutor:

◆ 01 Создание пула потоков

Вы можете создать экземпляр ThreadPoolExecutor указав количество потоков, которые вы хотите использовать, или использовать значение по умолчанию, которое равно количеству ядер процессора

Например:

```
...
from concurrent.futures import ThreadPoolExecutor

# Создание ThreadPoolExecutor с 5 потоками
executor = ThreadPoolExecutor(max_workers=5)

# Создание ThreadPoolExecutor с количеством потоков по умолчанию
executor = ThreadPoolExecutor()
```

◆ 02 Планирование задач

`ThreadPoolExecutor` предоставляет метод `submit()`, который используется для планирования выполнения задачи в пуле потоков. Метод `submit()` возвращает объект `Future`, который представляет результат выполнения задачи или ее текущий статус

Пример использования

```
...
def task(x):
    return x*x

# Планирование выполнения задачи
future = executor.submit(task, 5)

# Получение результата задачи
result = future.result()
print(result) # Вывод: 25
```

◆ 03 Параллельное выполнение задач

`ThreadPoolExecutor` автоматически распределяет задачи между потоками пула и параллельно выполняет их. Когда задачи завершены, результаты могут быть получены с использованием метода `result()` объекта `Future` или с помощью функций обратного вызова

◆ 04 Управление завершением потоков

`ThreadPoolExecutor` предоставляет метод `shutdown()`, который используется для корректного завершения работы пула потоков. После вызова метода `shutdown()`, `ThreadPoolExecutor` больше не принимает новые задачи и ожидает завершения выполняющихся задач

Кроме того, `ThreadPoolExecutor` также предоставляет методы, такие как `shutdown_now()`, `wait()`, `as_completed()` и другие, для управления потоками и получения результатов выполнения задач

`ThreadPoolExecutor` предоставляет простой интерфейс для параллельного выполнения асинхронных задач в фоновых потоках, позволяя эффективно использовать вычислительные ресурсы и ускорить выполнение программы, особенно при наличии задач, которые могут быть выполнены независимо друг от друга

Также доступны и другие методы

```
...
import time
import concurrent.futures
import random
import functools
from concurrent.futures import wait

def parser(site):
    time.sleep(random.randint(1,5))
    return f"data from site {site}"

def save_db(future):
    print(future.result())

def save_db2(site, future):
    print(f'{site}, {future.result()}')

def ex1():
    with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
        future_to_url = {executor.submit(parser, url) for url in ["site 1", "site 2", "site 3"]}
        for future in concurrent.futures.as_completed(future_to_url):
            print(future.result())

def ex2():
    with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
        for result in executor.map(parser, ["site 1", "site 2", "site 3"]):
            print(result)

def ex3():
    with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
        futures = [executor.submit(parser, site) for site in ["site 1", "site 2", "site 3"]]
        for future in futures:
            future.add_done_callback(save_db)

def ex4():
    with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
```

```
for site in ["site 1", "site 2", "site 3"]:  
    future = executor.submit(parser, site)  
    future.add_done_callback(functools.partial(save_db2, site))  
  
def ex5():  
    with concurrent.futures.ThreadPoolExecutor(max_workers=4) as  
executor:  
        futures = [executor.submit(parser, url) for url in ["site  
1", "site 2", "site 3"]]  
        done, not_done = wait(futures)  
        print(done)  
        for item in done:  
            print(item.result())  
  
if __name__ == '__main__':  
    start_time = time.time()  
  
    #ex1()  
    #ex2()  
    #ex3()  
    #ex4()  
    ex5()  
  
    print("--- %s seconds ---" % (time.time() - start_time))
```

ThreadPoolExecutor.add_done_callback()

это метод класса ThreadPoolExecutor, который используется для добавления функции обратного вызова (**callback**) к задаче, запланированной на выполнение в пуле потоков. Функция обратного вызова будет вызвана, когда задача будет завершена, независимо от того, была ли выполнена успешно или с ошибкой

Когда задача будет завершена, функция обратного вызова save_db() будет вызвана с объектом future в качестве аргумента. Метод result() объекта future используется для получения результата выполнения задачи

ThreadPoolExecutor.wait()

это метод класса ThreadPoolExecutor, который используется для ожидания завершения всех запланированных задач в пуле потоков. Этот метод блокирует текущий поток, пока все задачи не будут выполнены или пока не истечет указанный таймаут

```
...  
wait(fs, timeout=None, return_when=ALL_COMPLETED)
```

fs – это итерируемый объект, содержащий объекты **Future**, представляющие задачи, ожидание которых требуется

timeout – это опциональный аргумент, указывающий таймаут в секундах, после которого ожидание завершается, даже если не все задачи выполнены. Если **timeout** равен **None**, то ожидание продолжается до завершения всех задач

return_when – это опциональный аргумент, указывающий, когда ожидание должно завершиться

Возможные значения

ALL_COMPLETED – ожидание завершается только тогда, когда все задачи выполнены

FIRST_COMPLETED – ожидание завершается, когда хотя бы одна задача выполнена

Метод **wait()** возвращает кортеж, состоящий из трех множеств

01 Множество **done** содержит объекты **Future**, представляющие задачи, которые были завершены

02 Множество **not_done** содержит объекты **Future**, представляющие задачи, которые выполняются или на которых ожидается выполнение

03 Множество **failed** содержит объекты **Future**, представляющие задачи, которые завершились с ошибкой

ИТОГИ

- ✓ Мы изучили, как создавать экземпляры `ThreadPoolExecutor` с указанием количества потоков или использованием значения по умолчанию
 - ✓ Познакомились с методом `submit()` для планирования выполнения задачи в пуле потоков и получения объекта `Future` для отслеживания статуса и результата выполнения задачи
 - ✓ Узнали, что `ThreadPoolExecutor` автоматически распределяет задачи между потоками для параллельного выполнения
 - ✓ Изучили метод `shutdown()` для корректного завершения работы пула потоков и другие методы для управления потоками и получения результатов выполнения задач
- 