**並列処理・並行処理・非同期処理の違いとAPI待機処理の実例**

**1. 並列処理（Parallel Processing）**

**概要**

複数の処理を“同時に”実行する処理方式。主にマルチスレッドで実装され、CPUの複数コアを使用することで真に同時に複数のタスクを実行する。

**実装方法**

* Pythonでは threading モジュールを使用。

**実例コード（スレッドを使った並列実行）**

import threading

import time

def task(name):

print(f"{name} 開始")

time.sleep(2)

print(f"{name} 終了")

# スレッド作成

thread1 = threading.Thread(target=task, args=("タスク1",))

thread2 = threading.Thread(target=task, args=("タスク2",))

# スタート

thread1.start()

thread2.start()

# メインスレッドは待つ

thread1.join()

thread2.join()

**2. 並行処理（Concurrent Processing）**

**概要**

複数の処理を“同時のように”実行。マルチプロセスを用い、プロセスごとにメモリ空間を分けて動かす。

**実装方法**

* Pythonでは multiprocessing モジュールを使用。

**実例コード（プロセスを使った並行実行）**

from multiprocessing import Process

import time

def task(name):

print(f"{name} 開始")

time.sleep(2)

print(f"{name} 終了")

if \_\_name\_\_ == "\_\_main\_\_":

process1 = Process(target=task, args=("プロセス1",))

process2 = Process(target=task, args=("プロセス2",))

process1.start()

process2.start()

process1.join()

process2.join()

**3. 非同期処理（Asynchronous Processing）**

**概要**

処理が完了するのを待たずに次の処理へ進み、完了後に結果を受け取る形で動作する。主に I/O 処理（API通信、ファイル読み書き）と相性が良い。

**実装方法**

* Pythonでは asyncio モジュールを使用。

**実例コード（API呼び出しの非同期待機）**

import asyncio

import aiohttp

async def fetch(session, url):

async with session.get(url) as response:

print(f"{url} を取得中...")

return await response.text()

async def main():

urls = [

"https://httpbin.org/delay/2",

"https://httpbin.org/delay/3"

]

async with aiohttp.ClientSession() as session:

tasks = [fetch(session, url) for url in urls]

results = await asyncio.gather(\*tasks)

print("API応答完了")

asyncio.run(main())

**実行結果の流れ（APIの待ち合わせ）**

1. 2つのURLに対して非同期でGETリクエストを送る
2. それぞれ2秒, 3秒の遅延があるが、待ち時間を重ねずに非同期で進む
3. 約3秒後に両APIの応答が完了

**4. 処理の比較まとめ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **種類** | **同時実行の実態** | **メモリ空間** | **使用モジュール** | **向いている処理** |
| 並列処理 | 真に同時 | スレッド共有 | threading | CPU重い処理 |
| 並行処理 | 同時風に切替 | プロセス分離 | multiprocessing | 独立した重処理 |
| 非同期処理 | 完了を待たず実行 | シングルスレッド | asyncio / aiohttp | I/O処理（API, DB等） |

**5. まとめ**

* 並列処理：CPUパワーを生かして同時処理を行う
* 並行処理：プロセスを分けて安定した同時処理を実現
* 非同期処理：待ち時間の効率化に最適