並列処理  
複数のスレッドを用いて**同一プロセス内で複数の処理を同時に行う手法**です。CPUのコアが複数ある場合に、各スレッドが異なるコア上で同時に実行されることで、実際に「並列」に動作します。  
public class ParallelExample {

public static void main(String[] args) {

Runnable task1 = () -> {

for(int i=0; i<5; i++) {

System.out.println("Task1: " + i);

}

};

Runnable task2 = () -> {

for(int i=0; i<5; i++) {

System.out.println("Task2: " + i);

}

};

Thread thread1 = new Thread(task1);

Thread thread2 = new Thread(task2);

thread1.start();

thread2.start();

}

}

並行処理  
**複数のプロセスが独立して並行に動く処理方式**です。プロセス間のメモリは共有されず、通信はIPC（パイプやソケットなど）を介して行います。  
from multiprocessing import Process

def task1():

for i in range(5):

print(f"Task1: {i}")

def task2():

for i in range(5):

print(f"Task2: {i}")

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

p1 = Process(target=task1)

p2 = Process(target=task2)

p1.start()

p2.start()

p1.join()

p2.join()

非同期処理

処理の**完了を待たずに次の処理に進み、後で結果を受け取る**方式です。非同期処理では**イベントループ**が主に利用されます。  
function fetchData() {

return new Promise((resolve) => {

setTimeout(() => {

resolve("API response data");

}, 2000); // 擬似的に2秒後にデータが返ってくる

});

}

async function main() {

console.log("API呼び出し開始");

const result = await fetchData(); // ここで一旦待機

console.log("APIの応答:", result);

}

main();

API応答結果の待ち合わせ処理

Javaで非同期API呼び出し後に結果を待ち受ける

import java.util.concurrent.\*;

public class AsyncApiExample {

public static CompletableFuture<String> callApi() {

return CompletableFuture.supplyAsync(() -> {

try {

Thread.sleep(2000); // APIの応答を待つ模擬

} catch (InterruptedException e) {

e.printStackTrace();

}

return "APIの応答データ";

});

}

public static void main(String[] args) throws Exception {

System.out.println("API呼び出し開始");

CompletableFuture<String> future = callApi();

// 非同期に処理を継続しながら結果を待つ

String result = future.get(); // 結果が来るまでブロック

System.out.println("APIの応答: " + result);

}

}