# JAVA THREAD

# I.Synchronous vs Asynchronous

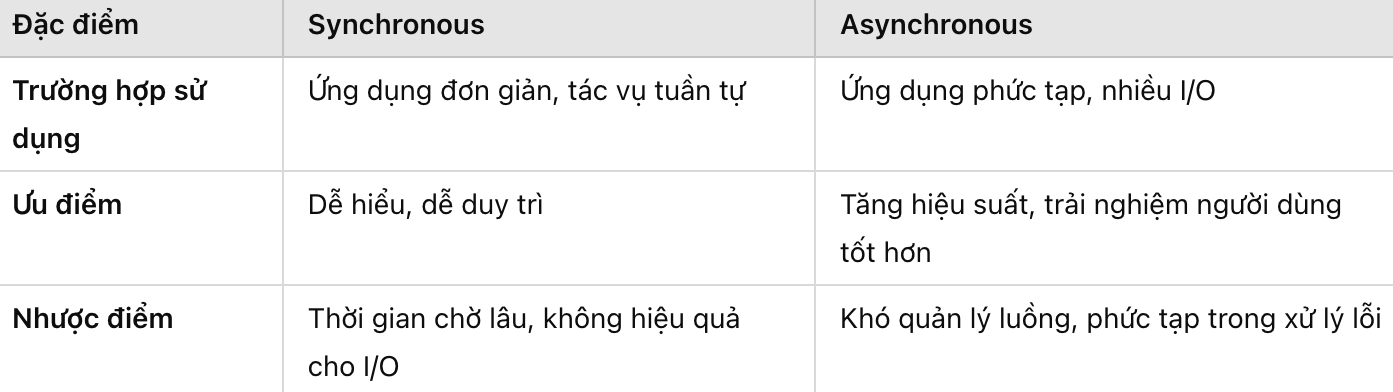
### 1. **Synchronous (Đồng bộ)**

* **Khái niệm**: Trong lập trình đồng bộ, các tác vụ được thực hiện theo thứ tự tuần tự. Điều này có nghĩa là một tác vụ phải hoàn thành trước khi tác vụ tiếp theo bắt đầu.
* **Hành vi**: Khi một hàm đồng bộ được gọi, chương trình sẽ chờ cho hàm đó hoàn thành và trả về kết quả trước khi tiếp tục thực hiện các dòng lệnh sau.

### 2. **Asynchronous (Không đồng bộ)**

* **Khái niệm**: Trong lập trình không đồng bộ, các tác vụ có thể được thực hiện mà không cần chờ cho tác vụ trước đó hoàn thành. Điều này cho phép chương trình tiếp tục thực hiện các tác vụ khác trong khi đang chờ kết quả từ một tác vụ không đồng bộ.
* **Hành vi**: Khi một hàm không đồng bộ được gọi, chương trình sẽ tiếp tục thực hiện các dòng lệnh khác mà không chờ cho hàm đó hoàn thành.

## Phân biệt trường hợp sử dụng, ưu nhược điểm của async và sync?



## Tìm hiểu từ khóa : synchronized trong java

-Trong Java, từ khóa **synchronized** được sử dụng để kiểm soát truy cập đến một phương thức hoặc một khối mã (block of code) bởi nhiều luồng (threads) cùng một lúc. Điều này giúp đảm bảo rằng chỉ một luồng có thể truy cập vào tài nguyên chung tại một thời điểm nhất định, tránh tình trạng **race condition** (điều kiện tranh chấp) và đảm bảo tính nhất quán của dữ liệu.

# II.Thread

## Process là gì ?

+process là một chương trình phần mềm **đang được thực** thi trên máy tính, có rất nhiều process được khởi tạo và chạy đồng thời.

+ các process không liên quan đến nhau, không sử dụng chung vùng nhớ, không truy cập trực tiếp dữ liệu của nhau.

Bên trong process bao gồm:

* **Instruction** (công thức chế biến, các chỉ dẫn, hay nói cách khác là code).
* **Data** (nguyên liệu của món salad).
* Còn một phần quan trọng nữa, đó là thông tin về trạng thái (**state**) của process.

## 2.Thread là gì ?

-Thead nằm trong process.

-Thread là **đơn vị cơ bản** để hệ điều hành quản lý và thực thi. Phần sau sẽ tìm hiểu kĩ hơn về cách hệ điều hành làm việc với thread.

## 3. Có bao nhiêu cách để tạo 1 thread trong java ? Khác biệt giữa việc sử dụng cách cách đó gì ?

-Có 2 cách để tạo thread trong java

**1. Kế thừa lớp Thread**

Cách đầu tiên để tạo một thread là bằng cách kế thừa lớp Thread và ghi đè phương thức run().

class MyThread extends Thread {

@Override

public void run() {

System.out.println("Thread is running.");

}

}

public class Main {

public static void main(String[] args) {

MyThread thread = new MyThread();

thread.start(); // Bắt đầu thread

}

}

### 2. **Triển khai giao diện** Runnable

Cách thứ hai là triển khai giao diện Runnable, sau đó tạo một đối tượng của lớp này và truyền nó cho một đối tượng của lớp Thread.

class MyRunnable implements Runnable {

@Override

public void run() {

System.out.println("Thread is running.");

}

}

public class Main {

public static void main(String[] args) {

MyRunnable myRunnable = new MyRunnable();

Thread thread = new Thread(myRunnable);

thread.start(); // Bắt đầu thread

}

}



## 4. Thế nào là multi thread ? Sử dụng multi thread mang lại ưu nhược điểm gì ?

**Multi-threading** (đa luồng) là một kỹ thuật trong lập trình cho phép nhiều luồng (threads) thực thi đồng thời trong cùng một chương trình. Mỗi luồng là một đơn vị nhỏ hơn trong một tiến trình (process) và có thể thực hiện các tác vụ riêng biệt mà không ảnh hưởng đến nhau.

### **1. Định nghĩa Multi-threading**

* **Multi-threading** cho phép một chương trình chia nhỏ công việc thành nhiều luồng, mỗi luồng có thể thực hiện một tác vụ cụ thể.
* Tất cả các luồng trong một chương trình chia sẻ không gian bộ nhớ chung, cho phép chúng giao tiếp và trao đổi dữ liệu dễ dàng hơn.

### **2. Lợi ích của Multi-threading**

* **Tăng hiệu suất**:
  + Chạy nhiều luồng đồng thời có thể cải thiện hiệu suất tổng thể của ứng dụng, đặc biệt trên các hệ thống đa nhân (multi-core). Các tác vụ có thể được thực hiện song song, giảm thiểu thời gian hoàn thành.
* **Cải thiện trải nghiệm người dùng**:
  + Trong ứng dụng GUI, multi-threading cho phép giao diện người dùng vẫn phản hồi nhanh chóng trong khi thực hiện các tác vụ nặng (như tải dữ liệu từ mạng hoặc thực hiện tính toán nặng).
* **Quản lý tài nguyên hiệu quả**:
  + Multi-threading cho phép sử dụng tài nguyên máy tính một cách hiệu quả hơn, vì các luồng có thể chờ đợi một tác vụ khác hoàn thành mà không chiếm dụng CPU.

### **3. Nhược điểm của Multi-threading**

* **Khó khăn trong việc lập trình**:
  + Việc viết và duy trì mã đa luồng phức tạp hơn so với lập trình đơn luồng. Lập trình viên phải quản lý trạng thái, đồng bộ hóa, và các vấn đề liên quan đến race condition.
* **Deadlock**:
  + Nếu không quản lý đúng cách, multi-threading có thể dẫn đến tình trạng deadlock, nơi hai hoặc nhiều luồng chờ đợi lẫn nhau để giải phóng tài nguyên, gây treo ứng dụng.
* **Quản lý đồng bộ hóa**:
  + Cần phải đảm bảo rằng các luồng không truy cập vào tài nguyên chung một cách đồng thời mà không có đồng bộ hóa, điều này có thể gây ra lỗi không mong muốn.

## 5. Làm thế nào để biết được 1 thread, multi thread đã hoàn thành hay chưa?

Để kiểm tra xem một thread hoặc multi-thread đã hoàn thành hay chưa trong Java, bạn có thể sử dụng một số phương pháp khác nhau. Dưới đây là các cách phổ biến:

### 1. **Sử dụng phương thức** isAlive()

Trong lớp Thread, có phương thức isAlive() cho phép bạn kiểm tra xem thread đang chạy hay đã hoàn thành. Phương thức này trả về true nếu thread vẫn đang hoạt động và false nếu nó đã kết thúc.

**Ví dụ:**

java

Copy code

class MyThread extends Thread {

@Override

public void run() {

try {

Thread.sleep(2000); // Giả lập công việc trong 2 giây

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class Main {

public static void main(String[] args) {

MyThread thread = new MyThread();

thread.start();

// Kiểm tra trạng thái của thread

while (thread.isAlive()) {

System.out.println("Thread vẫn đang chạy...");

try {

Thread.sleep(500); // Kiểm tra mỗi 500ms

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("Thread đã hoàn thành.");

}

}

### 2. **Sử dụng** join()

Phương thức join() của lớp Thread cho phép một thread chờ đợi cho một thread khác hoàn thành trước khi tiếp tục thực hiện. Khi thread chính gọi join() trên một thread, nó sẽ chờ cho đến khi thread đó hoàn thành.

class MyThread extends Thread {

@Override

public void run() {

try {

Thread.sleep(2000); // Giả lập công việc trong 2 giây

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class Main {

public static void main(String[] args) {

MyThread thread = new MyThread();

thread.start();

try {

thread.join(); // Chờ cho thread hoàn thành

System.out.println("Thread đã hoàn thành.");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

### 3. **Sử dụng** Future **và** ExecutorService

Khi sử dụng ExecutorService để quản lý các thread, bạn có thể nhận một đối tượng Future khi gửi một tác vụ. Đối tượng Future cho phép bạn kiểm tra xem tác vụ đã hoàn thành hay chưa và lấy kết quả nếu có.

**Ví dụ:**

java

Copy code

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutionException;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

public class Main {

public static void main(String[] args) {

ExecutorService executor = Executors.newSingleThreadExecutor();

// Tạo một tác vụ

Callable<String> task = () -> {

Thread.sleep(2000); // Giả lập công việc trong 2 giây

return "Hoàn thành";

};

Future<String> future = executor.submit(task);

// Kiểm tra trạng thái của tác vụ

while (!future.isDone()) {

System.out.println("Tác vụ vẫn đang chạy...");

try {

Thread.sleep(500); // Kiểm tra mỗi 500ms

} catch (InterruptedException e) {

e.printStackTrace();

}

}

// Lấy kết quả

try {

String result = future.get(); // Lấy kết quả (nếu có)

System.out.println(result);

} catch (InterruptedException | ExecutionException e) {

e.printStackTrace();

} finally {

executor.shutdown();

}

}

}

## 6. Có giới hạn việc tạo ra bao nhiêu thread trong 1 ứng dụng java hay không?

Có giới hạn , phụ thuộc vào 1 số yếu tố dưới đây :

### 1. **Giới hạn về bộ nhớ**

### 2. **Giới hạn về tài nguyên hệ thống**

### 3. **Giới hạn trong JVM**

### 4. **Khả năng lập trình**