Java Thread

1. Phân biệt synchronous vs asynchronous

**Đồng bộ (Synchronous):** Là cách thực thi tác vụ **tuần tự**. Tác vụ thứ hai chỉ có thể bắt đầu khi tác vụ đầu tiên đã hoàn thành. Các luồng (thread) sẽ bị **chặn** (block) và chờ đợi cho đến khi tác vụ hiện tại kết thúc.

**Bất đồng bộ (Asynchronous):** Là cách thực thi tác vụ **không tuần tự**. Tác vụ thứ hai có thể bắt đầu ngay lập tức mà không cần đợi tác vụ đầu tiên hoàn thành. Các luồng không bị chặn và có thể tiếp tục thực hiện các tác vụ khác.

**Cơ chế hoạt động**

* **Đồng bộ:**
* Tác vụ A bắt đầu.
* Luồng thực thi chờ đợi cho đến khi A hoàn thành.
* Tác vụ B bắt đầu ngay sau đó.
* Luồng tiếp tục chờ B hoàn thành.
* **Bất đồng bộ:**
* Tác vụ A bắt đầu.
* Luồng chính (main thread) gửi tác vụ A đến một luồng nền (background thread) và **không chờ đợi**.
* Luồng chính tiếp tục thực hiện tác vụ B ngay lập tức.
* Khi tác vụ A trên luồng nền hoàn thành, nó sẽ thông báo kết quả cho luồng chính.

**Ví dụ thực tế**

* **Đồng bộ:** Khi bạn gọi điện thoại cho bạn bè. Bạn phải đợi bạn ấy nhấc máy (tác vụ 1) thì mới bắt đầu nói chuyện (tác vụ 2).
* **Bất đồng bộ:** Khi bạn gửi một tin nhắn văn bản. Bạn không cần đợi người nhận trả lời (tác vụ 1) mà có thể ngay lập tức làm việc khác (tác vụ 2), ví dụ như gửi tin nhắn cho người thứ hai.

2. Phân biệt trường hợp sử dụng, ưu nhược điểm của async và sync

|  |  |  |
| --- | --- | --- |
|  | Sync (đồng bộ) | Async (bất đồng bộ) |
| Ưu điểm | - Dễ hiểu, dễ viết vì luồng thực thi đi theo một trình tự rõ ràng.  - Việc debug đơn giản hơn vì có thể theo dõi luồng thực thi một cách tuyến tính | - Luồng chính không bị chặn khi chờ đợi các tác vụ tốn thời gian, giúp phản hồi nhanh hơn.  - Tối ưu hóa việc sử dụng tài nguyên cho chép xử lý nhiều tác vụ cùng một lúc  - Hỗ trợ môi trường làm việc song song và hiệu quả |
| Nhược điểm | - Ứng dụng có thể bị "treo" hoặc phản hồi chậm nếu gặp một tác vụ tốn nhiều thời gian.  - Luồng bị "ngủ" trong khi chờ đợi, lãng phí thời gian và tài nguyên của CPU. | - Khó viết và gỡ lỗi hơn. Cần các cơ chế như callbacks, promises, hoặc async/await để xử lý kết quả khi các tác vụ nền hoàn thành.  - Có thể gây khó khăn trong việc đồng bộ hóa dữ liệu nếu không được quản lý tốt.  - Có thể gặp khó khăn trong việc đảm bảo tính toàn vẹn của dữ liệu khi có nhiều tác vụ chạy song song. |

**Trường hợp sử dụng**

* **Sử dụng Đồng bộ (Synchronous)**

Nên dùng lập trình đồng bộ cho những tác vụ đơn giản, ngắn gọn hoặc khi thứ tự thực thi là yếu tố quan trọng nhất.

* **Các tác vụ tuần tự:** Khi kết quả của tác vụ này là đầu vào cho tác vụ tiếp theo.
* **Ứng dụng đơn giản:** Những ứng dụng không đòi hỏi hiệu suất cao hoặc không có các thao tác I/O phức tạp.
* **Các thuật toán tính toán:** Các thuật toán cần xử lý từng bước một cách có trình tự.

Một ví dụ thực tế là việc truy cập cơ sở dữ liệu để lấy dữ liệu. Nếu cần dữ liệu từ bảng A để truy vấn bảng B, thì phải đợi truy vấn A hoàn thành.

* **Sử dụng Bất đồng bộ (Asynchronous)**

Nên dùng lập trình bất đồng bộ cho những tác vụ tốn nhiều thời gian, đặc biệt là các thao tác I/O.

* **Gọi API và dịch vụ mạng:** Khi gửi yêu cầu đến một máy chủ từ xa, ứng dụng có thể tiếp tục xử lý các yêu cầu của người dùng mà không cần chờ đợi phản hồi.
* **Thao tác với tệp tin:** Đọc hoặc ghi các tệp lớn có thể thực hiện ngầm trong khi giao diện người dùng vẫn phản hồi.
* **Xử lý sự kiện:** Trong các ứng dụng có giao diện đồ họa (GUI), lập trình bất đồng bộ giúp giao diện người dùng không bị "đóng băng" khi có một tác vụ nền đang chạy.

Ví dụ: Các ứng dụng web hoặc di động, nơi ta muốn tải hình ảnh hoặc dữ liệu từ mạng mà không làm chậm giao diện. Các tác vụ này sẽ chạy ở chế độ nền, và khi hoàn thành, chúng sẽ cập nhật giao diện mà không làm gián đoạn trải nghiệm người dùng

3. Tìm hiểu từ khóa : **synchronized**trong java

**synchronized** trong Java là một từ khóa (keyword) được sử dụng để đồng bộ hóa các phương thức hoặc khối lệnh trong môi trường đa luồng (multithreading). Nó được sử dụng để kiểm soát việc truy cập đồng thời vào các tài nguyên được chia sẻ, ngăn chặn các vấn đề như tình trạng tranh chấp dữ liệu (race conditions).

**Một số đặc điểm nổi bật của Synchronized trong Java:**

* Đảm bảo tính đúng đắn của dữ liệu
* Đảm bảo an toàn của dữ liệu
* Tăng hiệu suất
* Không đảm bảo tuân thủ thứ tự

Tại sao lại cần sử dụng **synchronized** ?

Trong môi trường đa luồng, khi nhiều luồng cùng truy cập vào các tài nguyên chung như các biến, đối tượng, file,… thì có thể xảy ra các lỗi như race condition, deadlock, hoặc các giá trị không đồng bộ (inconsistent) của dữ liệu. Dẫn đến các lỗi không mong muốn, và làm ảnh hưởng đến tính đúng đắn và hiệu suất của chương trình.

**=> synchronized** để đảm bảo rằng chỉ một luồng có thể truy cập vào phương thức hoặc khối mã được đánh dấu **synchronized** tại một thời điểm. Điều này giúp giảm thiểu xung đột và đảm bảo tính ổn định của ứng dụng.

**Cách hoạt động của synchronized ?**

**synchronized** hoạt động bằng cách sử dụng cơ chế **khóa** (locking) của đối tượng hoặc lớp chứa phương thức hoặc khối mã đó. Khi một luồng (thread) truy cập vào một phương thức hoặc khối lệnh được đánh dấu synchronized, nó sẽ tự động lấy một khóa. Chỉ khi luồng này hoàn thành công việc và giải phóng khóa, các luồng khác mới có thể truy cập vào vùng mã được bảo vệ đó. Điều này đảm bảo rằng tại một thời điểm, chỉ có **một luồng duy nhất** được thực thi bên trong vùng synchronized, giữ cho dữ liệu luôn nhất quán.

**synchronized block** cho phép đồng bộ hóa chỉ một phần của một phương thức hoặc khối lệnh. Khi sử dụng synchronized block, chỉ phần được khai báo trong khối lệnh đó mới được đồng bộ hóa, giúp tăng hiệu suất của chương trình.

Process synchronization và thread synchronization:

<https://rikkei.edu.vn/synchronized-trong-java/>

**Phân loại synchronized trong Java:**

Có ba loại synchronized: synchronized method, synchronized block và static synchronization.

**Synchronized method**

Synchronized method là một phương thức được đánh dấu với từ khóa synchronized. Khi một luồng truy cập vào synchronized method, nó sẽ lấy khóa trên đối tượng chứa phương thức đó.

*Khi sử dụng synchronized method, chỉ nên áp dụng cho các phương thức liên quan đến truy cập vào tài nguyên chia sẻ giữa các luồng, việc sử dụng synchronized method không cần thiết sẽ làm giảm hiệu suất của ứng dụng.*

Ví dụ:

class Example {

private int count = 0;

public synchronized void increment() {

count++;

}

public synchronized int getCount() {

return count;}}

**Synchronized block**

Synchronized block là một khối mã được đánh dấu với từ khóa synchronized. Điều này cho phép đồng bộ hóa một phần nhỏ của mã thay vì toàn bộ phương thức, giúp cải thiện hiệu suất.

synchronized (object) {

// Khối lệnh được bảo vệ bởi synchronized

}

Synchronized block được sử dụng để đồng bộ hóa chỉ một phần của một phương thức hoặc một khối lệnh, giúp tăng hiệu suất của chương trình. Nó cũng cho phép các đối tượng khác nhau được sử dụng để đồng bộ hóa, giúp tránh tình trạng blocking ở mức đối tượng toàn cục (global lock) và giảm thiểu sự cạnh tranh truy cập vào khối lệnh được bảo vệ.

Ví dụ:

class Example {

private Object lock = new Object();

private int count = 0;

public void increment() {

synchronized (lock) {

count++;

}

}

public int getCount() {

synchronized (lock) {

return count;

}

}

}

**Static Synchronization**

Static synchronization là cơ chế đồng bộ hóa được áp dụng cho các phương thức hoặc khối lệnh static, sử dụng từ khóa synchronized với đối tượng khóa (lock object) là lớp hoặc đối tượng Class.

public static synchronized void methodName() {

// code here

}

Ví dụ:

public class Example {

private static int count = 0;

public static synchronized void increment() {

count++;

}

public static synchronized int getCount() {

return count;

}

}

**Một số lưu ý khi sử dụng synchronized trong Java**

* Khi sử dụng synchronized, hãy cẩn thận với việc lựa chọn đối tượng khóa (lock object). Sử dụng cùng một đối tượng khóa cho các tài nguyên không liên quan có thể dẫn đến hiệu suất kém và deadlock.
* Tránh sử dụng đồng bộ hóa chuỗi (string synchronization) vì điều này có thể gây ra các vấn đề về hiệu suất và deadlock.
* Hạn chế sử dụng synchronized trong phương thức toàn cục, điều này có thể giảm hiệu suất và khả năng mở rộng của ứng dụng.

**Ưu điểm**

* **An toàn cho luồng**: Nó đảm bảo rằng dữ liệu được chia sẻ không bị hỏng bởi các thao tác đồng thời.
* **Dễ sử dụng**: Cú pháp đơn giản, không yêu cầu thư viện phức tạp.

**Nhược điểm**

* **Giảm hiệu suất**: Việc khóa luồng có thể làm giảm hiệu suất của ứng dụng, đặc biệt khi nhiều luồng cùng chờ đợi để truy cập vào một tài nguyên.
* **Nguy cơ bế tắc (deadlock)**: Nếu các luồng lấy khóa trên các đối tượng khác nhau theo một thứ tự không nhất quán, chúng có thể rơi vào tình trạng deadlock, nơi tất cả các luồng đều chờ đợi lẫn nhau.