Java Avancé

Cours 5 : Programmation Parallèle et Asynchrone

Arsène Lapostolet

2 Février 2024

Introduction

Thread

Thread : fil d'exécution. Plusieurs en même temps possible.

- Améliorer la vitesse d'un process : attention à l'overhead
- Permettre un cas d'utilisation : ex modèle client-serveur

Thread Java

- Classe `Thread`, construit à partir d'un `Runnable`
- Lancer avec la méthode `start()`

```java
var oddThread = new Thread(() -> {
    IntStream
            .range(0, 100)
            .filter(number -> number % 2 != 0)
            .forEach(System.out::println);
});


```

```
9  var evenThread = new Thread(() -> {
10     IntStream
11             .range(0, 100)
12             .filter(number -> number % 2 == 0)
13             .forEach(System.out::println);
14 });
15
16 evenThread.start();
17 oddThread.start();
18
19 Thread.sleep(5000);
```

Synchronisation

Joindre

- Attendre la fin d'un thread

```java
evenThread.join();
oddThread.join();
```

Ressource partagée et section critique

- Section critique : à excéuter atomiquement sinon problème
- `synchronized` : un seul thread à la fois sur la section
- Collections syncrhonisées : `Collections.synchronizedList(new ArrayList<>())`
- Attention au coût
- Section critiques synchronisées : **code thread-safe**

Ressource partagée et section critique

```java
public class Hotel {

    private final int roomsCount;
    private int bookedRoomsCount = 0;

    public Hotel(int roomsCount) {
        this.roomsCount = roomsCount;
    }

    public int getAvailableRoomCount(){
        return roomsCount - bookedRoomsCount;
    }
```

```
13      public void bookRooms(int numberOfBookedRooms){
14          if(getAvailableRoomCount() - numberOfBookedRooms < 0){
15              throw new IllegalArgumentException("Not enough
    rooms available");
16          }
17          else {
18              bookedRoomsCount += numberOfBookedRooms;
19          }
20      }
21  }
22  var hotel = new Hotel(20);
23  var reservationThread1 = new Thread(() -> hotel.bookRooms(3));
24  var reservationThread2 = new Thread(() -> hotel.bookRooms(7));
25
```

Ressource partagée et section critique

```Java
public synchronized void bookRooms(int numberOfBookedRooms)
{
    if(getAvailableRoomCount() - numberOfBookedRooms < 0){
        throw new IllegalArgumentException("Not enough rooms
available");
    }
    else {
        bookedRoomsCount += numberOfBookedRooms;
    }
}
```

Coordonner

- Coordonner des threads entre eux
- `wait` : attendre un signal
- `notify` : envoyer un signal
- Dans un bloc `synchronized`

Coordonner

```java
var token = new Object();

var oddThread = new Thread(() -> {
    var oddNumbers = ...
    for (var oddNumber : oddNumbers) {
        synchronized (token) {
            token.wait();
            System.out.println(oddNumber);
            token.notify();
        }
    }
});
```

```
13 var evenThread = new Thread(() -> {
14     var evenNumbers = ...
15     for (var evenNumber : evenNumbers) {
16         System.out.println(evenNumber);
17         synchronized (token) {
18             token.notify();
19             token.wait();
20         }
21     }
22 });
```

Programmation Asynchrone

Définition

- Abstraction au dessus des threads : tâches
- Mutualiser les threads
- Optimiser le temps CPU (I/O non bloquantes)

Notions :

- Promesse : tâche qui va être réalisée de façon asynchrone, on aura la résolution dans le futur
- Continuation : s'exécute sur le résultat de la promesse

Notion d'I/O non bloquantes

- CPU-bound : calculs, algos
- I/O-bound : attendre (appel réseau, système de fichiers)

Attente I/O = Temps CPU gaché

Objectif : le CPU fait autre chose quand il attend l'I/O

Asynchrone en Java : ThreadPool

Groupe de threads qui vont traiter une série de tâches

```java
ExecutorService threadPool =
Executors.newFixedThreadPool(4);

threadPool.submit(() -> {

  ...

} )
```

Promesse en Java : CompletableFuture

```java
CompletableFuture<String> task = CompletableFuture
.supplyAsync(() -> {
    var result = ... opération longue async...
    return result;
});

task.thenAccept((String result) -> println(result));

task.exceptionally(exception -> {
   exception.printStackTrace();
   return "";
});
```