Parallel Streams en Java 8+

- **¿Qué son los parallel streams?**

Para entender los parallel streams, primero tenemos que saber qué es un stream y cómo funciona.

Un stream en Java es una secuencia de elementos que puede procesarse de manera paralela o secuencial, ofrecen un enfoque conciso y funcional para expresar manipulaciones complejas de datos.

Por ejemplo, podemos ver cómo en pocas líneas podemos imprimir por pantalla el doble de todos los números pares que estén en un array:

```java
import java.util.Arrays;

public class DobleDeNumerosPares {
    public static void main(String[] args) {
        // Definir un array de números
        int[] numeros = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // Utilizar streams para filtrar los números pares y luego imprimir el doble de cada uno
        Arrays.stream(numeros)
                .filter(numero -> numero % 2 == 0)
                .map(numero -> numero * 2)
                .forEach(System.out::println);
    }
}
```

La salida será la siguiente:

```
4
8
12
16
20
```

Una de las grandes ventajas de los streams es la capacidad de trabajar en paralelo.

Puedes crear streams que se ejecuten en paralelo y aprovechen múltiples núcleos del procesador para procesar los datos más rápido.

Esto se logra utilizando el método parallelStream() en lugar de stream() al trabajar con colecciones.

(No es necesario incorporar ninguna librería externa, pues viene incorporado en Java)

- **¿Cuándo se podría usar, y qué ventajas o inconvenientes tendría?**

Los parallel Streams son recomendables para tareas intensivas en procesamiento y grandes volúmenes de datos, por ejemplo:

1. Grandes conjuntos de datos:
Cuando trabajas con colecciones de datos muy extensas, como grandes listas o mapas y necesitas realizar operaciones como filtrado, mapeo o reducción, los parallel Streams *pueden acelerar el procesamiento al distribuir la carga en múltiples núcleos.*

2. Operaciones intensivas en CPU:
Tareas que implican un uso intensivo del procesador, como cálculos matemáticos complejos, manipulación de imágenes o procesamiento de grandes cantidades de texto, pueden beneficiarse de los parallel Streams al *aprovechar los múltiples núcleos de la CPU.*

3. Transformaciones independientes:
Permite *realizar operaciones independientes entre elementos de la colección*, como aplicar un mismo cambio a cada elemento.
Esto significa que cada elemento no depende del resultado de otro para ser procesado.

- **¿Cuándo NO usar Parallel streams?**

El resultado puede ser menos eficiente o erróneo si los usamos con …

1. Tareas cortas y ligeras para el procesador.
2. Tareas que requieren un orden específico de los resultados.
3. Tareas que no son compatibles con el paralelismo pues producen efectos secundarios inesperados.
    a) En colecciones: add(), remove(), clear()...
    b) Usando recursos compartidos: read(), write()...

En parallelStreams.java se puede ver claramente esto. Si el tamaño del Array es 1000, tardará más que parallelStreams. Si es 1000000, tardará menos.

- **¿Cuáles pueden ser algunos de sus <u>inconvenientes</u>?**

• Gasto de tiempo de trabajo útil:
Costos adicionales asociados con la creación y gestión de hilos.
Estos costos pueden incluir el consumo de recursos del sistema, la gestión de la concurrencia y otros factores que afectan el rendimiento de la ejecución concurrente.

• Posibles problemas de concurrencia:
Trabajar en paralelo puede provocar condiciones de carrera que afecten a la consistencia de los datos.

Bibliografía:

https://www.baeldung.com/java-when-to-use-parallel-stream
https://www.geeksforgeeks.org/what-is-java-parallel-streams/
https://www.geeksforgeeks.org/parallel-vs-sequential-stream-in-java/
https://www.arquitecturajava.com/java-parallel-stream-y-rendimiento/

Sergio Simón
Sofía Beltrán
Lucía Simón