□□□□ □□□□□□□□□□□□□□□

التسلسل المتعدد يسمح فعال. بشكل Executor إطار استخدام وكيفية جافا في التسلسل المتعدد في نغوص دعونا
وحدة على المعتمدة أو الإدخال□الخروج على المعتمدة للعمليات الأداء يحسن مما متزامن، بشكل مهام عدة بتشغيل لبرنامجك
مستوى على وتسييرها الخيوط إنشاء إخفاء خلال من أبسط بشكل الخيوط تسيير Executor إطار ويوفر المركزية. المعالجة
الصلة. ذات والأدوات Executor استخدام كيفية سأريك ثم الأساسيات، عبر سأرشدك منخفض.

---

1. أساسيات التسلسل المتعدد في جافا

يوفر جافا طريقتين رئيسيتين لإنشاء الخيوط: □ **توسيع فئة Thread:**

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running: " + Thread.currentThread().getName());
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start(); //
    }
}
```

□ **تطبيق واجهة Runnable:**

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Runnable running: " + Thread.currentThread().getName());
    }
}

public class Main {
    public static void main(String[] args) {
        Thread thread = new Thread(new MyRunnable());
        thread.start();
    }
}
```

الحالي. الخيط في الكود ينفذ فقط run() الخيط لإطلاق run() لا start() استدعاء

---

2. لماذا استخدام Executor؟

إنشاء الخيوط، تكلفة مثل: المهام من العديد إدارة في فعال غير هو ولكن البسيطة، الحالات في يعمل يدويًا الخيوط إنشاء يجعل مما المهام، إدارة ومجموعة خيوط مجموعة توفر Executor إطار مقدمة في Java 5 تحت java.util.concurrent. والموارد. للتوسيع. قابلية وأكثر نظافة أكثر التسلسل المتعدد

---

3. استخدام ExecutorService

استخدامها: كيفية إلى الأساسية. الأداة هي Executor من فرعية واجهة ExecutorService الواجهة هي

الخطوة 1: إنشاء ExecutorService في Executors فئة استخدم خيوط: مجموعة لإنشاء المساعدة

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Main {
    public static void main(String[] args) {
        //              4
        ExecutorService executor = Executors.newFixedThreadPool(4);

        //
        for (int i = 0; i < 10; i++) {
            executor.submit(() -> {
                System.out.println("Task executed by: " + Thread.currentThread().getName());
                try {
                    Thread.sleep(1000); //
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            });
        }

        //         `executor`
        executor.shutdown(); //
```

```
    }
}
```

- `newFixedThreadPool(4)` ينشئ مجموعة من 4 خيوط. المهام الزائدة تنتظر في قائمة الانتظار.
- `submit()` يقبل المهام Runnable أو Callable. Callable يعيد نتيجة.

أنواع Executor الشائعة:

- `Executors.newSingleThreadExecutor()`: خيط واحد، يعالج المهام بشكل تسلسلي.
- `Executors.newCachedThreadPool()`: ينشئ خيوط حسب الحاجة، يعيد استخدام الخيوط الفارغة. جيد للمهام قصيرة العمر.
- `Executors.newScheduledThreadPool(n)`: لتأخير أو تكرار المهام دورية.

---

4. Future و Callable مع النتائج والاستثناءات

إذا كنت تحتاج إلى نتائج، استخدم Callable بدلاً من Runnable:

```
import java.util.concurrent.*;

public class Main {
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newFixedThreadPool(2);

        //        `Callable`
        Future<Integer> future = executor.submit(() -> {
            Thread.sleep(1000);
            return 42;
        });

        //
        System.out.println("Task submitted...");

        //              )       (
        Integer result = future.get();
        System.out.println("Result: " + result);

        executor.shutdown();
    }
}
```

- Future.get() يحصل على النتيجة، إذا يتوقف لم تنتهي المهمة.
- future.isDone() استخدم للتحقق من الانتهاء دون توقف.

---

5. مجموعة ضبط متقدم: الخيوط

مخصصة: ThreadPoolExecutor مجموعة في ترغب قد الحقيقية، للتطبيقات

```java
import java.util.concurrent.*;

public class Main {
    public static void main(String[] args) {
        //        :    2-4                   10
        ThreadPoolExecutor executor = new ThreadPoolExecutor(
            2, //
            4, //
            60L, //
            TimeUnit.SECONDS,
            new LinkedBlockingQueue<>(10) //
        );

        for (int i = 0; i < 15; i++) {
            executor.execute(() -> {
                System.out.println("Task by: " + Thread.currentThread().getName());
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            });
        }

        executor.shutdown();
    }
}
```

- حجم النواة 2: عدد الخيوط التي يتم الاحتفاظ بها على قيد الحياة.
- حجم القصوى 4: عدد الخيوط القصوى إذا كانت القائمة ممتلئة.
- القائمة: تحتفظ بالمهام عندما تكون جميع الخيوط مشغولة.

---

6. أفضل الممارسات

- **إيقاف التشغيل بشكل صحيح**: استدعِ دائمًا `shutdown()` أو `shutdownNow()` لمنع تسرب الموارد.
- **معالجة التقطيعات**: تحقق من `Thread.currentThread().isInterrupted()` في الدوال.
- **تجنب التحميل الزائد**: الكثير من الخيوط يمكن أن يضر بالأداء؛ اضبط حجم المجموعة بناءً على وحدات المعالجة المركزية (مثل: `Runtime.getRuntime().availableProcessors()`).
- **استخدام `try-finally` مع `ExecutorService`**:

```java
ExecutorService executor = Executors.newFixedThreadPool(4);
try {
    //
} finally {
    executor.shutdown();
}
```

ربطها معًا: مثال مع Log4j

مزج التسلسل المتعدد مع سؤالك السابق حول Log4j:

```java
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import java.util.concurrent.*;

public class Main {
    private static final Logger logger = LogManager.getLogger(Main.class);

    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(3);

        for (int i = 0; i < 5; i++) {
            final int taskId = i;
            executor.submit(() -> {
                logger.info("Task {} started by {}", taskId, Thread.currentThread().getName());
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    logger.error("Task {} interrupted", taskId, e);
                    Thread.currentThread().interrupt();
                }
```

```
                logger.info("Task {} completed", taskId);
            });
        }

        executor.shutdown();
    }
}
```

والملف. الشاشة إلى المهمة تقدم يسجل هذا السابق، من log4j2.xml مع