Asynchronous Programming In Dart

Dart’s Asynchronous Programming

640710555 นายภูริวัฒน์ วุฒิเจริญวงศ์

Asynchronous Programming คืออะไร?

- การทำงานในรูปแบบ Asynchronous ก็คือการทำงานที่ช่วยให้โปรแกรมของเราสามารถทำงานต่อเนื่องอย่างสมบูรณ์ได้ ในขณะที่ยังมีการดำเนินการอื่นอยู่ ยกตัวอย่างเช่น การเรียกดูข้อมูลผ่านเครือข่ายเน็ตเวิร์ค การบันทึกข้อมูลลงฐานข้อมูล การอ่านข้อมูลจากไฟล์ เป็นต้น
- Asynchronous จะทำงานแบบไม่ประสานเวลา (ตรงข้ามกับ Synchronous ที่แปลว่าการทำงานแบบประสานเวลา)
- สำหรับการใช้งาน Asynchronous ใน Dart เราสามารถใช้ Future class ในการจัดการ ร่วมกับคีย์เวิร์ด async และ await

Synchronous Programming คืออะไร?

ในการเขียนโปรแกรมแบบ Synchronous คือโปรแกรมที่รอคอยให้คำสั่งหนึ่งทำงานจนเสร็จสิ้นก่อนจึงจะดำเนินการต่อไป ข้อเสียในการใช้วิธีการนี้คือหากส่วนหนึ่งของโค้ดใช้เวลานานในการประมวลผล บล็อกถัดๆ มาที่ไม่เกี่ยวข้องกันจะถูกบล็อกจากการดำเนินการ

ตัวอย่างการเขียนโปรแกรมแบบ Synchronous

```
void main() {
  print("First Operation");
  print("Second Big Operation");
  print("Third Operation");
  print("Last Operation");
}
```

▼ Output

```
First Operation
Second Big Operation
Third Operation
Last Operation
```

ในตัวอย่างนี้สามารถเห็นได้ว่าโปรแกรมจะพิมพ์ออกมาทีละบรรทัด ถ้าสมมติว่า Second Big Operation ต้องใช้เวลา 3 วินาทีในการทำงาน Third Operation และ Last Operation ก็จะต้องรอเป็นเวลา 3 วินาทีถึงจะทำงานได้ โดยปัญหานี้เราสามารถนำการเขียนโปรแกรมแบบ Asynchronous มาแก้ไขได้

ตัวอย่างการเขียนโปรแกรมแบบ Asynchronous

```dart
void main() {
  print("First Operation");
  Future.delayed(Duration(seconds:3),()=>print('Second Big Operation'));
  print("Third Operation");
  print("Last Operation");
}
```

▼ Output

```
First Operation
Third Operation
Last Operation
Second Big Operation
```

ในตัวอย่างนี้สามารถเห็นได้ว่าจะพิมพ์ Second Big Operation เป็นอย่างสุดท้าย เนื่องจากต้องใช้เวลา 3 วินาทีในการทำงานแต่ Third Operation กับ Last Operation นั้นไม่จำเป็นต้องรอ 3 วินาที

การเขียนโปรแกรมแบบ Asynchronous ในภาษาอื่นๆ

```java
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;

public class Main {
    public static void main(String[] args) {
        System.out.println("First Operation");

        CompletableFuture<Void> future = CompletableFuture.runAsync(() -> {
            try {
                Thread.sleep(3000); // Simulate a 3-second delay
                System.out.println("Second Big Operation");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        System.out.println("Third Operation");
        System.out.println("Last Operation");

        try {
            future.get(); // Wait for the asynchronous task to complete
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }
    }
}
```

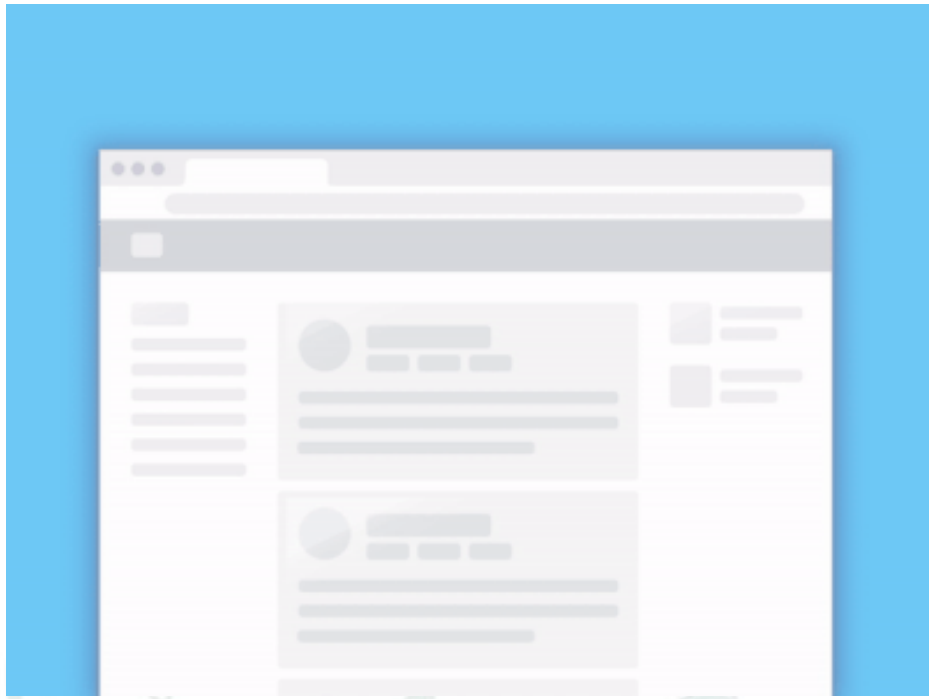
Java

```python
import asyncio

async def delayed_operation():
    await asyncio.sleep(3) # Simulate a 3-second delay
    print('Second Big Operation')

async def main():
    print("First Operation")

    asyncio.create_task(delayed_operation())

    print("Third Operation")
    print("Last Operation")

asyncio.run(main())
```

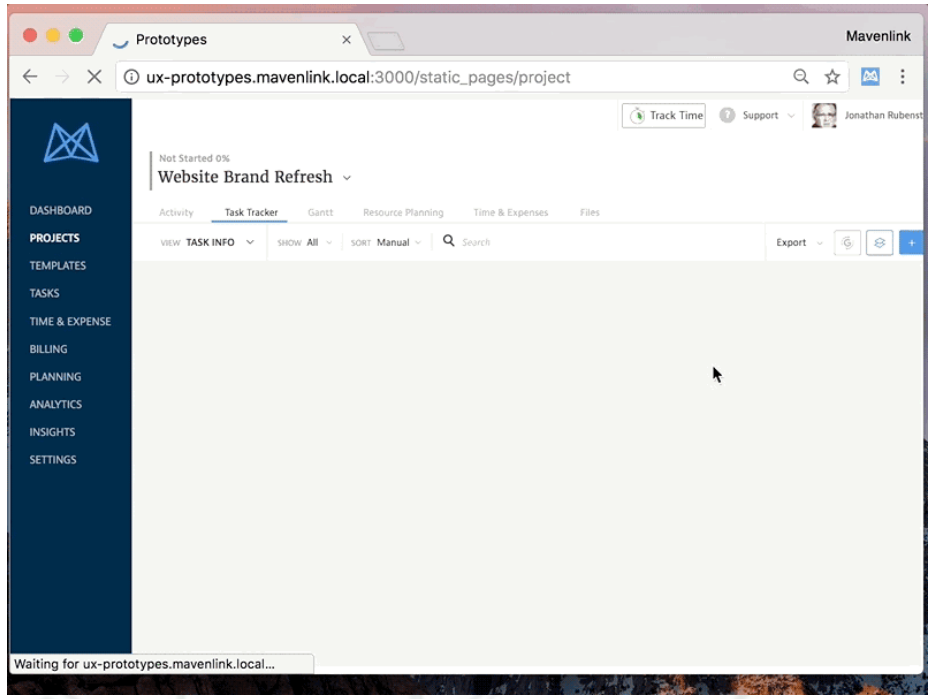
Python

ทำไมเราต้องใช้การเขียนโปรแกรมแบบ Asynchronous?

เพราะการเชื่อมต่อ I/O จะเกิดสิ่งที่เรียกว่า "Delay" ขึ้น



การติดต่อกับแหล่งข้อมูลจากภายนอกจะต้องเสียเวลาในการเชื่อมต่อ โดยโปรแกรมหรือแอพที่เราใช้ๆ กันอยู่ทุกวันนี้นอกจากโปรแกรมจะต้องทำการโหลดข้อมูลแล้ว มันยังต้องควบคุมส่วน UI ด้วย ดังนั้นถ้าเราเขียนโปรแกรมแบบเดิมๆ กับโปรแกรมที่มี UI เป็นกราฟิกโหมดจะเกิดอะไรขึ้น??



คำตอบคือ .. จังหวะที่โปรแกรมเราสั่งโหลดข้อมูลและมันกำลังรอให้ข้อมูลตอบกลับมาอยู่นั้น CPU จะไม่สามารถปลีกตัวไปทำงานอย่างอื่นได้เลยแม้แต่การจัดการกับ UI โปรแกรมจะกลับมาทำงานต่อได้อีกทีก็ต้องรอคำสั่งที่ทำการโหลดข้อมูลทำงานจนเสร็จแล้ว

การดำเนินการแบบ Asynchronous ที่พบได้บ่อย

- รับข้อมูลจากอินเทอร์เน็ต
- เขียนข้อมูลลงในฐานข้อมูล
- ดำเนินการที่ใช้เวลานาน
- อ่านข้อมูลจากไฟล์
- ดาวน์โหลดไฟล์ ฯลฯ

**Note:

การดำเนินการแบบ Asynchronous มักใช้เวลานาน ดังนั้นจึงมักจะให้ผลลัพธ์ในรูปแบบของ Future หากผลลัพธ์มีส่วนหลาย ๆ ส่วน จะให้ผลลัพธ์ในรูปแบบของ Stream ซึ่งเราจะได้เรียนรู้เพิ่มเติมเกี่ยวกับ Future และ Stream ในเนื้อหาส่วนถัดไป

คำศัพท์สำคัญ

- Synchronous Operation: รูปแบบการทำงานที่เป็นไปอย่างต่อเนื่องตามลำดับ โดยถ้าการทำงานในส่วนใด ยังไม่เสร็จเรียบร้อย ก็จะไม่มีการทำงานต่อในส่วนอื่นต่อได้
- Synchronous Function: เป็นฟังก์ชั่นที่ทำงานในรูปแบบการทำงานแบบ synchronous เท่านั้น
- Asynchronous Operation: รูปแบบการทำงานที่เมื่อถูกเรียกใช้งานแล้ว จะทำการรอผลลัพธ์ค่าสุดท้าย และในขณะเดียวกัน การทำงานของส่วนอื่น ก็ยังทำงานต่อเนื่องไปได้ แม้ส่วนแรกยังทำงานไม่เสร็จก็ตาม
- Asynchronous Function: เป็นฟังก์ชั่นที่ทำงานอยู่ในรูปแบบการทำงานแบบ asynchronous และสามารถใช้งานในรูปแบบการทำงาน synchronous อีกด้วย

Reference

- Asynchronous programming: futures, async, await | Dart
- Dart Programming - Async (tutorialspoint.com)
- Asynchronous Programming :: Dart Tutorial - Learn Dart Programming (dart-tutorial.com)
- การใช้งาน Asynchronous Programming ในภาษา Dart เบื้องต้น เนื คอร์สเรียน เรียนฟรี ออนไลน์ บทความ (ninenik.com)
- Async in Dart (1) รู้จัก Isolates และ Event Loop กับการทำงานไม่ประสานเวลากัน – TAmemo.com