

【Kotlin】coroutines

相關資源

官方文件:[coroutines-guide](#)

一. 什麼是coroutines? (協同程序)

(一) 意義：

- coroutines的中文翻譯為「**協程**」，「深入淺出kotlin」一書中形容協程就像「輕量的執行緒」(light-weight threads)。使用協同程序的意義就在於，啟動一個背景工作時，同時讓其他程式不需要等待工作完成就可以做別的事情，用戶體驗會比較好
- Coroutines，分別是 cooperation + routine，cooperation 意指合作，routine 意指例行作業、慣例，照這裡直接翻譯就會是合作式例行作業。
- 協程是一種輕量級的線程，可以被掛起和恢復。它允許你以同步的方式編寫非同步程式碼，從而避免回調地獄。

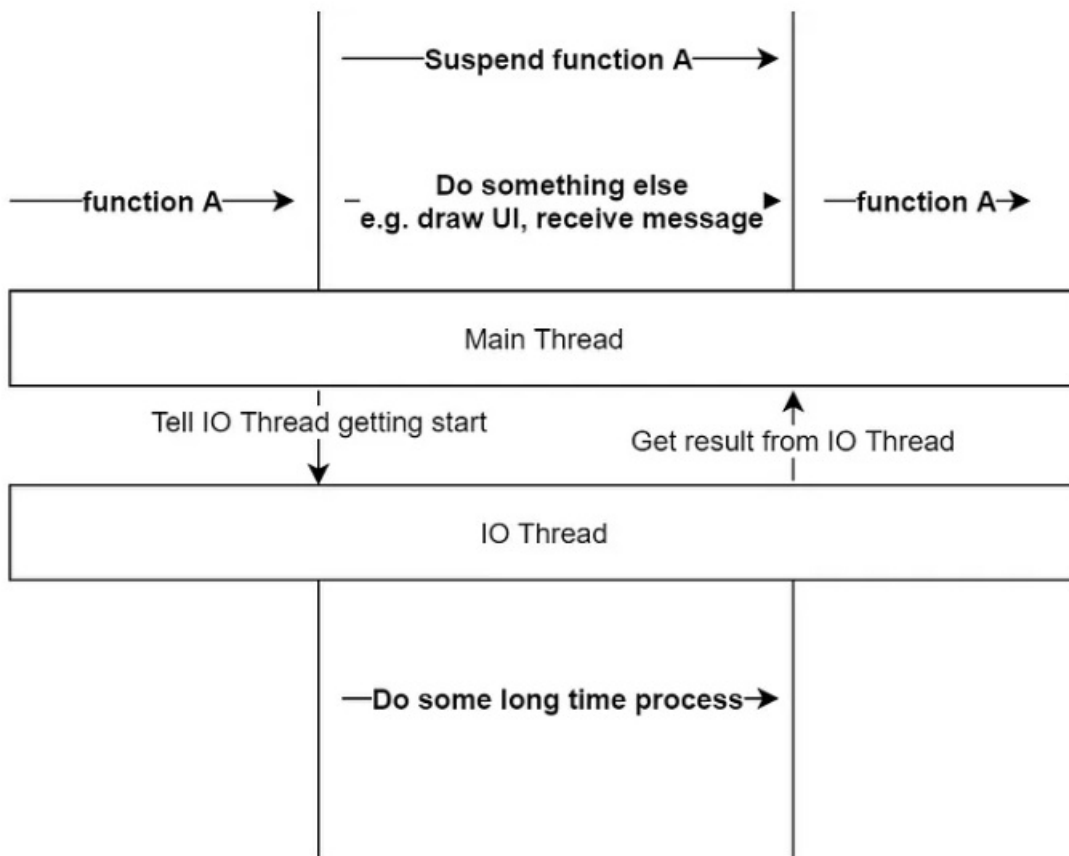
(二) 優點：

1. 解決傳統執行緒的困擾：

影片及書本中都以「在main thread連接internet」為例，由於android明令禁止應用程式在主執行緒存取網路，因此，此時主執行緒會被阻塞。而傳統的執行序在主執行緒阻塞未被移除前，是不會再執行其他作業的，coroutines也是執行緒，只不過在這種情形會被暫停，但是不會阻塞父執行緒。

2. 很適合執行背景工作：

如 (一) 所述，使用協同程序的好處就在於，啟動一個背景工作時，同時讓其他程式不需要等待工作完成就可以做別的事情，用戶體驗會比較好。



就是我在 main thread 執行到 function A 需要等 IO thread 耗時處理的結果，那我先暫停 function A，協調讓出 main thread 讓 main thread 去執行其他的事情，等到 IO thread 的耗時處理結束後得到結果後再回復 function A 繼續執行

二. 基本概念

掛起函數 (Suspending Function)

掛起函數是可以在不阻塞執行緒的情況下掛起的函數。它使用suspend關鍵字定義。例如：

```
suspend fun doSomething() {  
    // Your code here  
}
```

協程作用域（Coroutine Scope）

協程作用域管理協程的生命週期。協程只能在某個作用域內啟動。

```
runBlocking {
    launch {
        // Coroutine code
    }
}
```

啟動協程

Kotlin 提供了幾種方式來啟動協程，常用的方法有 `launch` 和 `async`。

- `launch`

```
runBlocking {
    launch {
        delay(1000L)
        println("World!")
    }
    println("Hello,")
}

// Hello
// World!
```

- `async`
- 與 `launch` 類似，但它會傳回一個 `Deferred` 對象，可以在將來取得結果。

```
runBlocking {
    val deferred = async {
        delay(1000L)
        "World!"
    }
    println("Hello,")
    println(deferred.await())
}

// Hello
// World!
```

協程上下文與調度器

協程上下文包含協程的各種元素，例如調度器（Dispatcher）、作業（Job）等。

- **Dispatchers.Main**：在主執行緒上運行，用於更新UI。
- **Dispatchers.IO**：用於執行IO 操作，如網路請求、檔案讀寫。
- **Dispatchers.Default**：用於執行CPU 密集型任務。
- **Dispatchers.Unconfined**：啟動協程時繼承目前線程，只有在協程掛起後才會切換到其他線程。

```
runBlocking {
    launch(Dispatchers.Main) {
        // 在主线程运行
    }
    launch(Dispatchers.IO) {
        // 在IO线程运行
    }
    launch(Dispatchers.Default) {
        // 在默认线程池运行
    }
}
```

```
fun main() = runBlocking { // this: CoroutineScope
    launch { // launch a new coroutine and continue
        delay(1000L) // non-blocking delay for 1 second (default time unit is ms)
        println("World!") // print after delay
    }
}
```

```
println("Hello") // main coroutine continues while a previous one is delayed
}

// Hello
// World!
```

🕒 修訂版本 #3

★ 由 treeman 建立於 20 🕒 2024 14:41:12

✍ 由 treeman 更新於 20 🕒 2024 15:18:09