測試环境：

1）Android Studio

2) java&Kotlin

電子三甲

裘翀皓

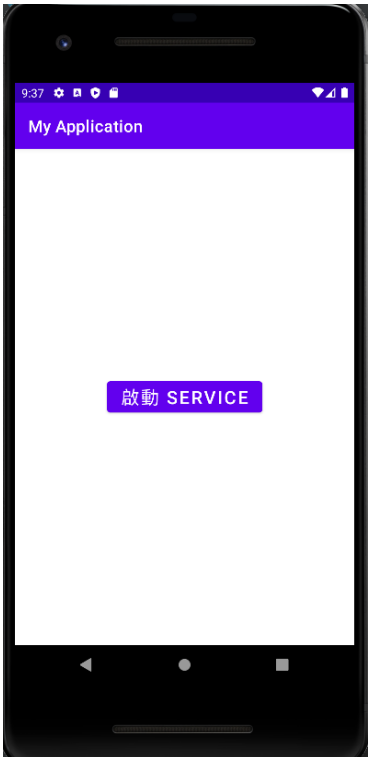
108360150

心得:

LAB12:

lab12讓我們了解了Service的用途以及使用時機，使用Service執行背景工作。

當Activity離開畫面後會進入停止狀態，開發者需要使用Service處理應用程式消失後要繼續執行的特定任務。lab12的kotlin專案如下圖所示，點擊啟動SERVICE按鈕後，會啟動Service並且結束MainActivity，延時3S後Service會啟動SecActivity。



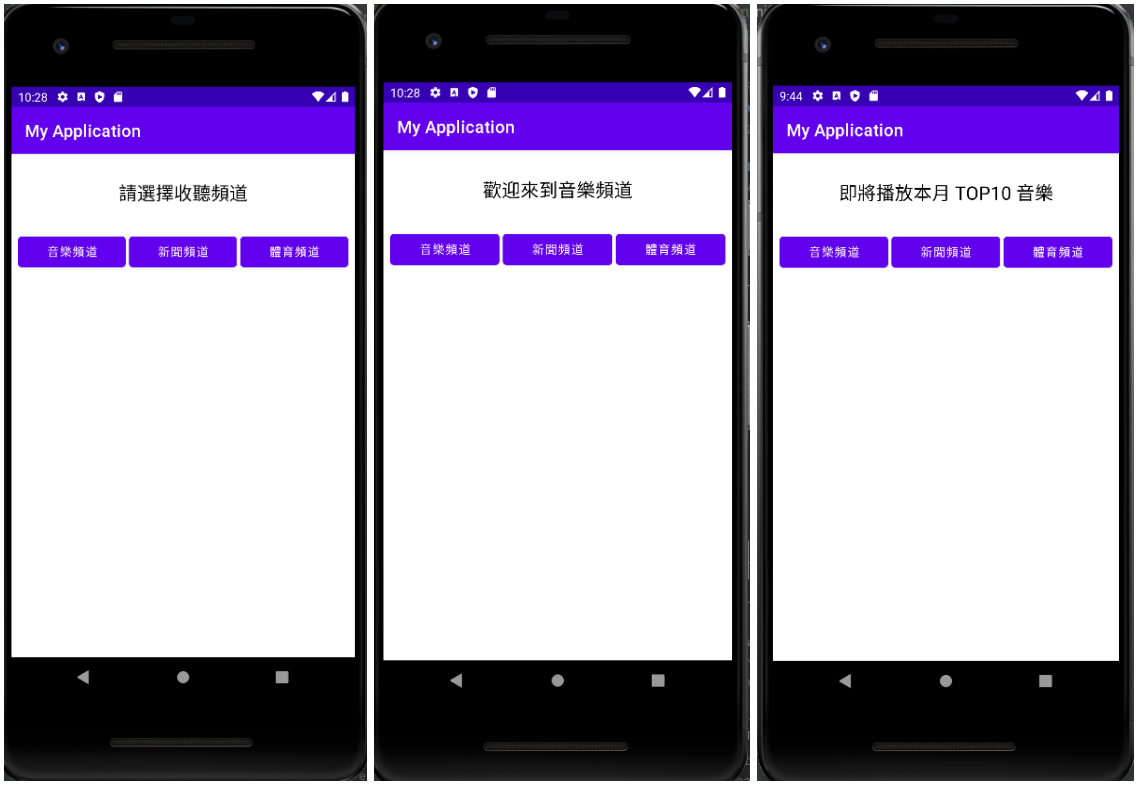
lab13:

lab13用到了BroadcastReceiver，例如手機亮點不足的提醒就是用到了BroadcastReceiver,

其創建路徑是Flie-New-Other-Broadcast Receiver。

lab13也會用到Service，幫助我們傳送廣播信息。

如下圖所示，如果我們點擊「音樂頻道」，系統在3S延遲過後會顯示即將播放本月TOP10音樂。其他兩個按鈕功能相近。



程式碼有使用到Coroutines

記得要做API的引入（大象）

```
dependencies {
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.9'
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9'
```

Coroutines：

```
val job: Job = GlobalScope.launch(Dispatchers.Main) {  this: CoroutineScope
    // launch coroutine in the main thread
    try {
        delay( timeMillis: 3000) //延遲三秒
        broadcast(
            when(channel) {
                "music" -> "即將播放本月 TOP10 音樂"
                "new" -> "即將為您提供獨家新聞"
                "sport" -> "即將播報本週 NBA 賽事"
                else -> "頻道錯誤"
            }
        )
    }catch (e:InterruptedException){
        e.printStackTrace()
    }
}
job.start() //啟動執行緒
return START_STICKY
```

Thread：

```
//lin AsyncTask
/*
//若 thread 被初始化過且正在運行，則中斷它
if (::thread.isInitialized && thread.isAlive)
    thread.interrupt()
thread = Thread {
    try {
        Thread.sleep(3000) //延遲三秒
        broadcast(
            when(channel) {
                "music" -> "即將播放本月 TOP10 音樂"
                "new" -> "即將為您提供獨家新聞"
                "sport" -> "即將播報本週 NBA 賽事"
                else -> "頻道錯誤"
            }
        )
    } catch (e: InterruptedException) {
        e.printStackTrace()
    }
}

thread.start() //啟動執行緒
return START_STICKY
 */
```

Kotlin 官網中提到 Coroutine 是輕量化的 Thread (Light-weight Thread), 從官方附帶的 Wikipedia 中看到 Coroutine 屬於協同式 (Cooperative) 多工, 而 Thread 通常屬於搶佔式 (Preemptive) 多工。

協同式多工:程式會定時放棄已佔有的執行資源讓其它程式可以執行。由程式自己讓出執行資源, 作業系統不會干涉。

搶佔式多工:程式有各自的優先權, 作業系統會根據程式的優先權安排當下哪個程式能擁有執行資源去執行, 另外作業系統有權中斷任何正在執行中的程式, 變更執行資源的擁有者。

Thread

Android 透過 Java 層建立的 Thread 實際上是對應到底層作業系統的 Thread, 因此在 Android 中直接使用多個 Thread 實現程式並行運作, 這些 Thread 都會由作業系統來排程 (搶佔式多工), 當 Thread 數量過多時就容易增加作業系統切換 Thread (上下文切換 Context Switch) 的負擔, 影響整體效能。

Coroutine

為什麼說 Coroutine 是輕量化的 Thread?首先建立一個 Coroutine 不會綁定到作業系統的 Thread, 此外 Coroutine 使用協同式多工來排程, Coroutine 之間的切換由當前正在執行的 Coroutine 主動讓出執行權給其它 Coroutine 執行, 藉此達到並行運作, 因為 Coroutine 的切換是在上層, 不需要由底層的作業系統來處理, 所以 Coroutine 交替時所產生的上下文切換負擔比 Thread 小。

參考資料:

https://medium.com/gogolook-tech/kotlin-coroutines-%E5%85%A5%E9%96%80%E6%A6%82%E5%BF%B5-coroutine-vs-thread-e7d112b0d8ba

Github:

lab12:

layout:
https://github.com/108360150-Qiuchonghao/Android_project_108360150/tree/master/kotlin_4/lab12/app/src/main/res/layout

kotlin程式碼:
https://github.com/108360150-Qiuchonghao/Android_project_108360150/tree/master/kotlin_4/lab12/app/src/main/java/com/example/myapplication

lab13:

layout:
https://github.com/108360150-Qiuchonghao/Android_project_108360150/tree/master/kotlin_4/lab13/app/src/main/res/layout

kotlin程式碼:
https://github.com/108360150-Qiuchonghao/Android_project_108360150/tree/master/kotlin_4/lab13/app/src/main/java/com/example/myapplication