

Lab9

Android 的非同步執行

本節目的：

- 了解執行緒與非同步執行的觀念。
- 學習使用 Thread 類別與 AsyncTask 類別。

9.1 ANR（應用程式無回應）

在前面章節中，我們知道應用程式的執行是在 Activity 之上，而 Activity 的運行好壞會直接影響到使用者的操作。假如說程式上設計不良，出現類似無窮迴圈導致程式執行時間過久呢？這時就會發生圖 9-1 的 ANR（應用程式無回應）。



圖 9-1 ANR（應用程式沒有回應）

要解決此方法的關鍵就是使用非同步執行方式來執行程式，以下會做說明。

9.1.1 執行緒與非同步執行

在沒有特別設計下，所有的任務（Task）都會在 Main Thread（或稱為 UI Thread）上執行，如下圖 9-2 所示。



圖 9-2 Main Thread 的 Task 排程

Main Thread 負責處理畫面更新的作業，如果 Main Thread 其中一個任務（Task）非常耗時間，或是完成時間不可預期，如網路相關的動作、資料庫的動作、檔案操作或複雜的計算，使 Main Thread 無法執行更新畫面相關的操作，就會造成畫面卡住，甚至出現 ANR，如圖 9-3 所示。

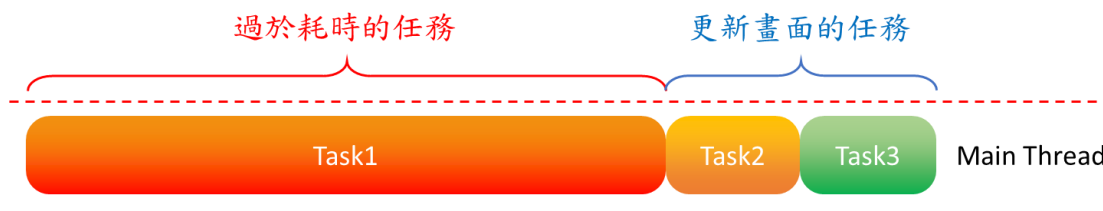


圖 9-3 Task1 無法在預期時間完成，導致無法執行 Task2 與 Task3

所以，我們需要把非常耗時間，或是完成時間不可預期的任務（Task），使用**非同步**的方式來處理，透過非同步的方式，放到 **Background Thread** 來執行，這樣就可以避免 Main Thread 出現任務（Task）卡住的問題，如圖 9-4 所示。



圖 9-4 將 Task1 移到背景工作，讓其他 task 可以執行

下面會介紹兩個非同步執行的方法：(1) Thread 類別 (2) AsyncTask 類別，Thread 是 Java 本來就有的語法，而 AsyncTask 是 Android SDK 提供的類別，使用這兩個類別方法來實現出非同步執行，可以把過於耗時的任務 (Task) 放到 Background Thread，讓 Main Thread 的任務 (Task) 執行不會受到影響。

9.1.2 非同步執行方法

在 Android 中，我們很常使用到非同步執行的方法，例如我們前面所教導的 Toast 就是很典型的非同步執行，他能在執行之後獨立運作，顯示期間 Activity 依然可以繼續的執行下去。

針對我們應用程式中的需求，我們可以產生出新的 Thread 去執行我們要實做的耗時作業，要在程式中實做一個新的 Thread 最簡單的方法可以用以下寫法：

```
Thread(Runnable {  
    ... //要在 Thread 中進行的工作  
}).start()
```

產生一個 Thread 之後，我們需要調用 Runnable() 介面，Runnable() 會提供 run() 方法執行我們要跑的程式，因此要將要實作的程式碼寫在 Runnable{} 內，然後使用 Thread.start() 方法將任務啟動。

由於 Thread 類別會產生出新的 Background Thread 去執行任務，但是 Background Thread 由於不是 UI Thread，無法操作畫面的更新，因此當 Background Thread 中的任務需要操作畫面時，就必須要嘗試與 UI Thread 溝通，透過 UI Thread 來對畫面更新，這時就會需要使用到 Handler 類別。

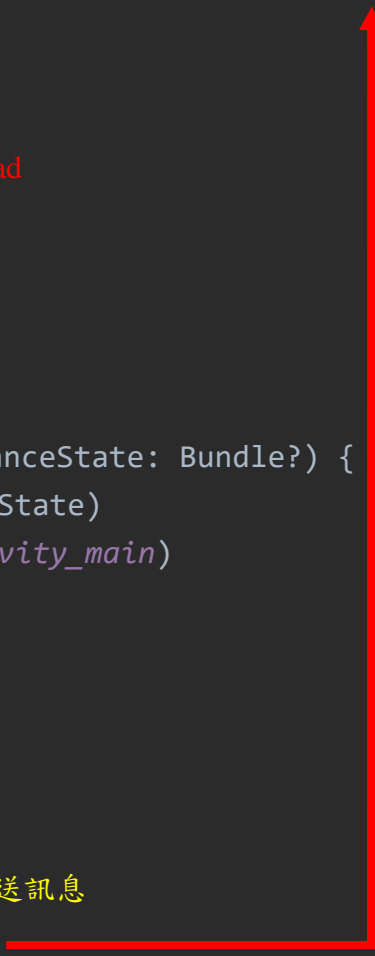
Handler 是一種跨 Thread 的溝通機制，可以在一個 Thread 中把訊息丟到 Message 類別中，再讓另外一個 Thread 從 Message 中取得訊息。在 Thread 類別中，我們需要額外加入以下程式碼：

```

class MainActivity : AppCompatActivity() {
    //Step1：建立 Handler 物件等待接收訊息
    private val handler = Handler(Handler.Callback { msg ->
        //判斷 msg 代號
        when (msg.what) {
            1 ->{
                ... //執行於 Main Thread
            }
        }
        true
    })

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //執行於 Background Thread
        Thread(Runnable {
            //Step2：建立 Message 物件
            val msg = Message()
            //加入代號
            msg.what = 1
            //Step3：透過 sendMessage 傳送訊息
            handler.sendMessage(msg)
        }).start()
    }
}

```



Step1 首先，我們需要在建立一個 Handler 類別，透過 Handler 提供的 Callback 介面實現 handleMessage()方法，利用 handleMessage 這個事件來接收 Thread 送出的 Message，並在 UI Thread 執行對應的操作。

Step2 Thread 類別中需要建立一個對應的 Message 物件，Message 會用於傳遞至 handleMessage()，Message 的 what 屬性可以加入一組代號，讓 Handler 可根據代號來決定要做什麼。

Step3 當 Thread 類別要發出 Message 時使用 Handler.sendMessage()方法來觸發 handleMessage()，如此就能把畫面操作透過 Handler 來執行處理。

9.1.3 AsyncTask 類別

AsyncTask 是 Android 1.5 加入的用於實現非同步操作的一個類別，是 Android 平台自己的非同步工具，融入了 Android 平台的特性，讓非同步操作更加的安全，方便和實用。

AsyncTask 可以方便的執行非同步操作（doInBackground），又能方便的與 Main Thread 進行聯繫。AsyncTask 出現的目的就是在提供簡單易用的方式達成 Handler、Thread 與 Message 的功能，AsyncTask 只要定義幾個方法就可以達到他們的效果。

一個完整的 AsyncTask 的架構如下：

```
object : AsyncTask<Int, Void, String?>() {  
    //Step2:初始化（執行於 Main Thread）  
    override fun onPreExecute() {  
    }  
    //Step3:執行（執行於 Back Thread）  
    override fun doInBackground(vararg params: Int?): String? {  
    }  
    //Step4:進度更新（執行於 Main Thread）  
    override fun onProgressUpdate(vararg params: Void) {  
    }  
    //Step5:結果處理（執行於 Main Thread）  
    override fun onPostExecute(result: String) {  
    }  
}.execute() //Step1:啟動 AsyncTask
```

首先 AsyncTask 必須明確定義整個流程中的輸出入資料型態。因此一開始 AsyncTask 就需要帶三個標籤—<Int, Void, String>，這三個標籤必須放入一個變數型態，如「Integer」、「Void」、「String」。這三個參數別用於定義輸入的資料型態、進度更新的資料型態、輸出的資料型態，程式中會對應到 doInBackground()、onProgressUpdate()與 onPostExecute()的輸入值。

```
object : AsyncTask<Int, Void, String?>() {  
    輸入 進度 結果  
    override fun onPreExecute() {  
    }  
  
    override fun doInBackground(vararg params: Int?): String? {  
    }  
  
    override fun onProgressUpdate(vararg params: Void) {  
    }  
  
    override fun onPostExecute(result: String) {  
    }  
}.execute()
```

下面我們就 AsyncTask 提供的四種方法分別做介紹：

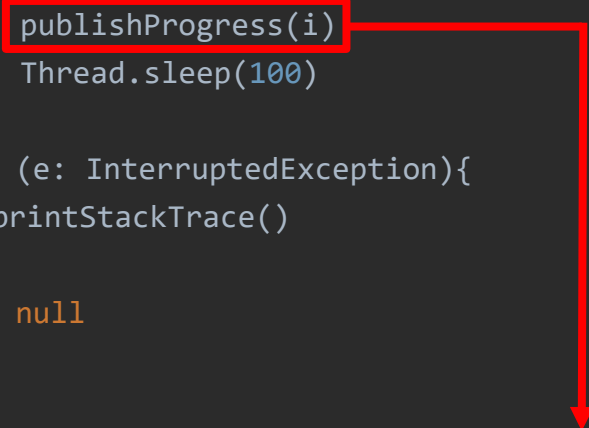
onPreExecute：這個方法會在 AsyncTask 開始非同步執行時被執行，這方法中可以讓我們初始化一些資訊或是保存一些變數在 AsyncTask 的區域變數之中，如果沒有必要資料的話此步驟可以被省略。

doInBackground：doInBackground()是 AsyncTask 中唯一執行於新的 Thread 的方法，也是 AsyncTask 中唯一必須被實作的方法。執行於 doInBackground()中的方法會獨立被執行，直到完成後回傳結果。

onProgressUpdate：onProgressUpdate() 是 AsyncTask 中用於監聽 doInBackground()進度的方法，我們可以使用他搭配 ProgressBar 來實現進度條效果。我們可以在 doInBackground() 中加入 publishProgress() 方法來觸發 onProgressUpdate()，如下方程式碼所示：

```
override fun doInBackground(vararg params: Int?): String? {
    try{
        for(i in 0 until 100){ //會執行 onProgressUpdate()
            publishProgress(i)
            Thread.sleep(100)
        }
    }catch (e: InterruptedException){
        e.printStackTrace()
    }
    return null
}

override fun onProgressUpdate(vararg params: Int?) {
    super.onProgressUpdate(*values)
}
```



範例中，我們使用 `Thread.sleep()` 這個方法讓這個 `Thread` 延遲一段時間來呈現出等待效果，在迴圈中我們加入了 `publishProgress(i)` 將進度時間值發送出去，而 `onProgressUpdate` 就能接收到發出來的訊息。

onPostExecute: `onPostExecute` 的主要工作在於處理任務結束後的回傳結果，當 `doInBackground()` 完成了工作並回傳之後 `onPostExecute()` 會接收到 `doInBackground()` 傳來的結果，可以在這方法中對結果實作後續處理。

總結以上方法後，`AsyncTask` 透過 `execute()` 方法啟用，`execute()` 也可以傳入參數，該參數會被傳入到 `doInBackground()` 之中，整理 `AsyncTask` 一次輸出的流程如下：


```
val input = 0
val out = arrayListOf("")

object : AsyncTask<Int, Void, String>() {
    //Step2:初始化
    override fun onPreExecute() {}
    //Step3:執行
    override fun doInBackground(vararg params: Int?) ← String? {
        publishProgress()
        return "執行完畢"
    }
    //Step4:進度更新
    override fun onProgressUpdate(vararg values: Void?) {
    }
    //Step5:結果處理
    override fun onPostExecute(result: String) {
        out[0] = result
    }
}

//Step1:啟動 AsyncTask，並傳入參數
}.execute(input)
```

9.2 龜兔賽跑

- 畫面中會使用兩個進度條 (SeekBar) 來模擬烏龜與的兔子跑步路線，如圖 9-5 所示。
- 由於烏龜與的兔子的移動要同時進行，且都為耗時的工作，因此烏龜與的兔子的移動分別由 AsyncTask 與 Thread 執行。
- 按下開始後，同時啟動 AsyncTask 與 Thread，並每 0.1 秒隨機讓各自的進度條增加 0~2% 的值，使兩個進度條同時的增加長度。
- 先抵達終點（進度條達到 100%），則會用 Toast 顯示出贏家。

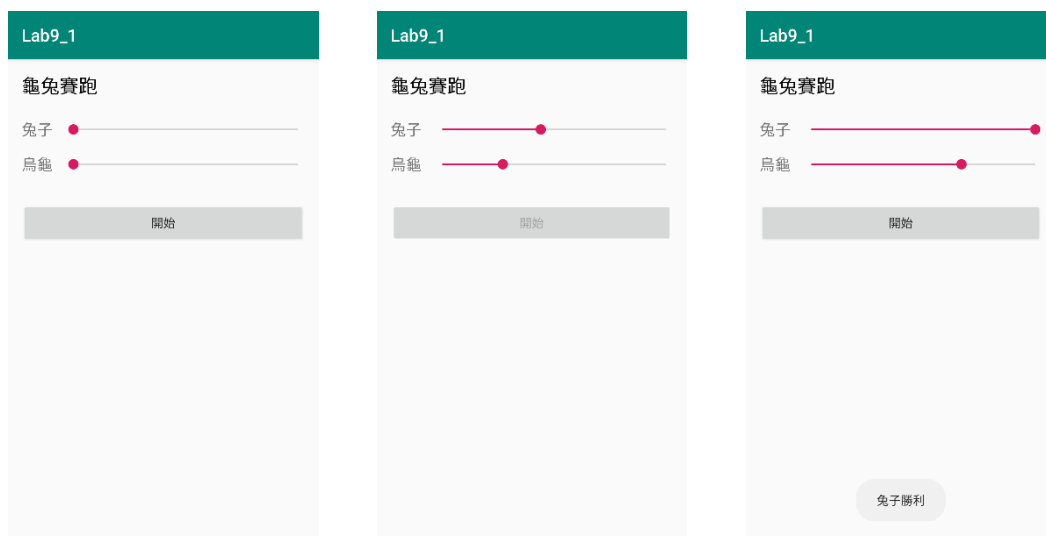


圖 9-5 初始畫面（左）、開始賽跑（中）、兔子勝利（右）

9.2.1 SeekBar 畫面設計

Step1 新增專案，以及圖 9-6 對應的 class 和 xml 檔。

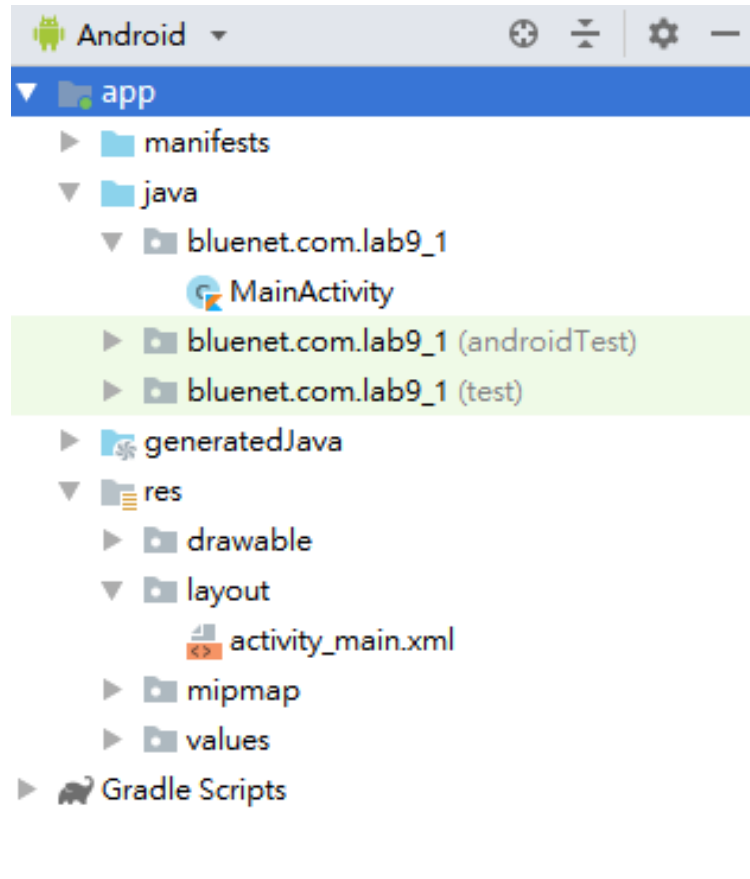


圖 9-6 龜兔賽跑專案架構

Step2 繪製 activity_main.xml 檔，如圖 9-7 所示。

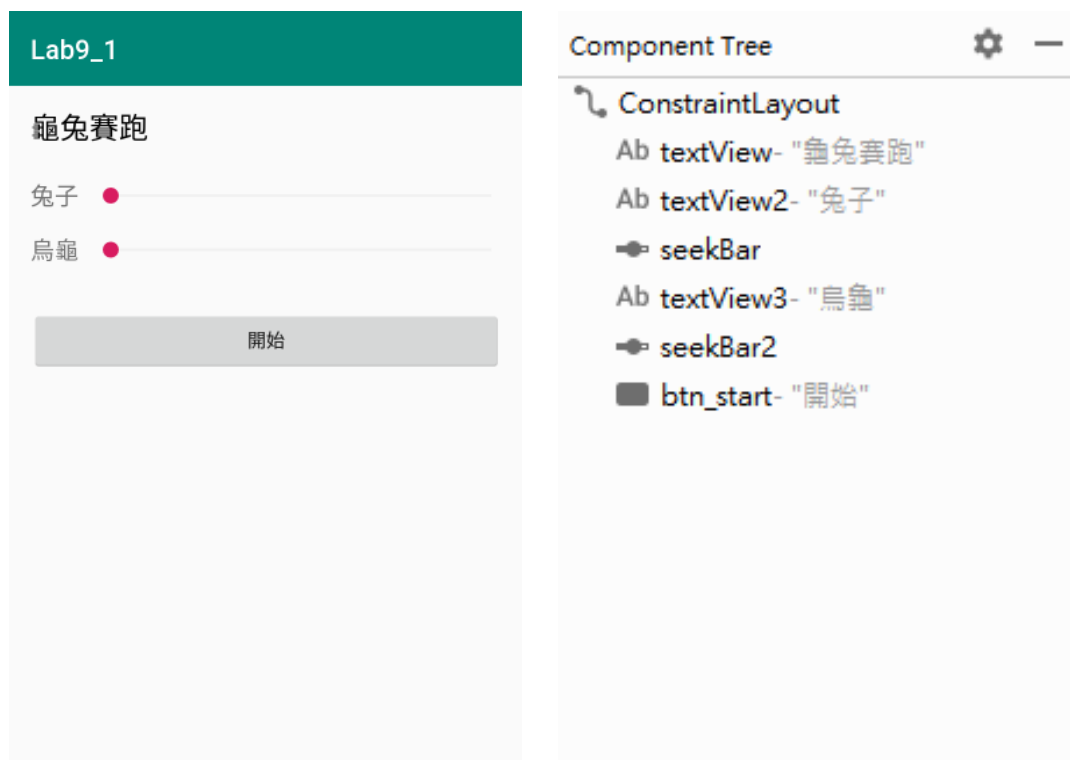


圖 9-7 龜兔賽跑預覽畫面（左）與布局元件樹（右）

對應的 xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="16dp"
        android:text="龜兔賽跑"
```

```
        android:textSize="22sp"
        android:textColor="@android:color/black"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

<TextView

```
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:text="兔子"
        android:textSize="18sp"
        app:layout_constraintStart_toStartOf="@+id/textView"
        app:layout_constraintTop_toBottomOf="@+id/textView" />
```

<SeekBar

```
        android:id="@+id/seekBar"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        app:layout_constraintBottom_toBottomOf="@+id/textView2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView2"
        app:layout_constraintTop_toTopOf="@+id/textView2" />
```

<TextView

```
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="烏龜"
        android:textSize="18sp"
        app:layout_constraintStart_toStartOf="@+id/textView"
        app:layout_constraintTop_toBottomOf="@+id/textView2" />
```

<SeekBar

```
        android:id="@+id/seekBar2"
```

```

        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        app:layout_constraintBottom_toBottomOf="@+id/textView3"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView3"
        app:layout_constraintTop_toTopOf="@+id/textView3" />

<Button
    android:id="@+id/btn_start"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="32dp"
    android:layout_marginEnd="16dp"
    android:text="開始"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView3" />
</android.support.constraint.ConstraintLayout>

```

9.2.2 Thread 與 AsyncTask 比較

Step1 編寫 MainActivity，按下按鈕後，分別執行 runThread()與 runAsyncTask() 兩個副程式。

```

class MainActivity : AppCompatActivity() {
    //建立兩個計數器，用於計算烏龜與兔子的進度
    private var rabprogress = 0
    private var torprogress = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //開始按鈕監聽事件
        btn_start.setOnClickListener {
            btn_start.isEnabled = false

```

```

        //初始化兔子的計數器
        rabprogress = 0
        //初始化烏龜的計數器
        torprogress = 0
        seekBar.progress = 0
        seekBar2.progress = 0
        //執行副程式來執行 Thread
        runThread()
        //執行副程式來執行 AsyncTask
        runAsyncTask()
    }
}
}

```

Step2 runThread()中編寫執行一個 Thread 來模擬兔子的移動，每 0.1 秒隨機增加計數器 0~2 的值，透過 Handler 來顯示到 SeekBar 之上。

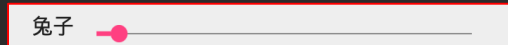
```

private fun runThread() {
    object : Thread() {
        override fun run() {
            while (rabprogress <= 100 && torprogress < 100) {
                try {
                    Thread.sleep(100) //延遲 0.1 秒
                } catch (e: InterruptedException) {
                    e.printStackTrace()
                }
                //隨機增加計數器 0~2 的值
                rabprogress += (Math.random() * 3).toInt()
                //建立 Message 物件
                val msg = Message()
                //加入代號
                msg.what = 1
                //透過 sendMessage 傳送訊息
                mHandler.sendMessage(msg)
            }
        }
    }.start() //啟動 Thread
}

//建立 Handler 物件等待接收訊息
private val mHandler = Handler(Handler.Callback { msg ->
    when (msg.what) { //判斷代號，寫入計數器的值到 SeekBar
        1 -> seekBar.progress = rabprogress
    }
})

//重複執行到計數器不小於 100 為止
if (rabprogress >= 100 && torprogress < 100) {

```



```

        //用 Toast 顯示兔子勝利
        Toast.makeText(this, "兔子勝利",
            Toast.LENGTH_SHORT).show()

        btn_start.isEnabled = true
    }
    true
})

```

Step3 runAsyncTask()中編寫執行一個 AsyncTask 來模擬烏龜的移動，每 0.1 秒隨機增加計數器 0~2 的值，並顯示到 SeekBar 之上。


```

private fun runAsyncTask() {
    object : AsyncTask<Void, Int, Boolean>() {
        override fun doInBackground(vararg voids: Void):
Boolean? {
            while (torprogress <= 100 && rabprogress < 100) {
                try {
                    Thread.sleep(100) //延遲 0.1 秒
                } catch (e: InterruptedException) {
                    e.printStackTrace()
                }
                //隨機增加計數器 0~2 的值
                torprogress += (Math.random() * 3).toInt()
                //更新進度條進度，傳入烏龜計數器
                publishProgress(torprogress)
            }
            return true
        }

        override fun onProgressUpdate(vararg values: Int?) {
            super.onProgressUpdate(*values)
            values[0]?.let { //寫入計數器的值到 SeekBar 的進度中
                seekBar2.progress = it
            }
        }

        override fun onPostExecute(status: Boolean?) {
            if (torprogress >= 100 && rabprogress < 100) {
                //用 Toast 顯示烏龜勝利
                Toast.makeText(this@MainActivity, "烏龜勝利",
                    Toast.LENGTH_SHORT).show()
                btn_start.isEnabled = true
            }
        }
    }.execute()
}

```



9.3 體脂肪計算機

- 圖 9-8 輸入身高（整數）和體重（整數）以及選擇性別後按下計算，會執行 5 秒左右，然後將體脂肪的結果做顯示。
- 為了呼應使用 AsyncTask 類別的情境，計算公式中我們加入 Thread.Sleep()方法暫停執行 5 秒，模擬這個計算需要長時間執行，計算後得到標準體重和體脂肪。

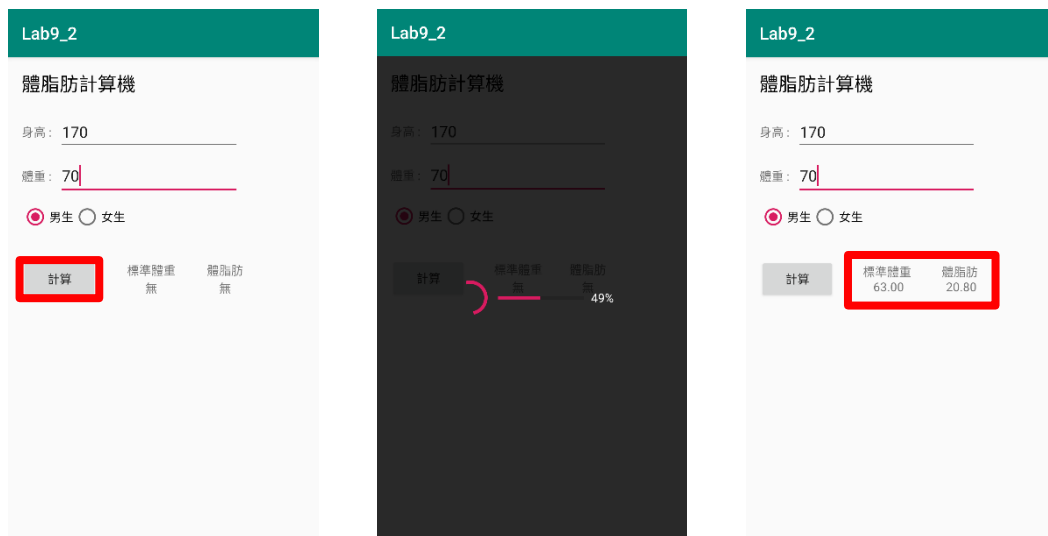


圖 9-8 輸入身高體重（左）、等待 5 秒（中）、顯示計算結果（右）

9.3.1 ProgressBar 畫面設計

Step1 新增專案，以及圖 9-9 對應的 class 和 xml 檔：

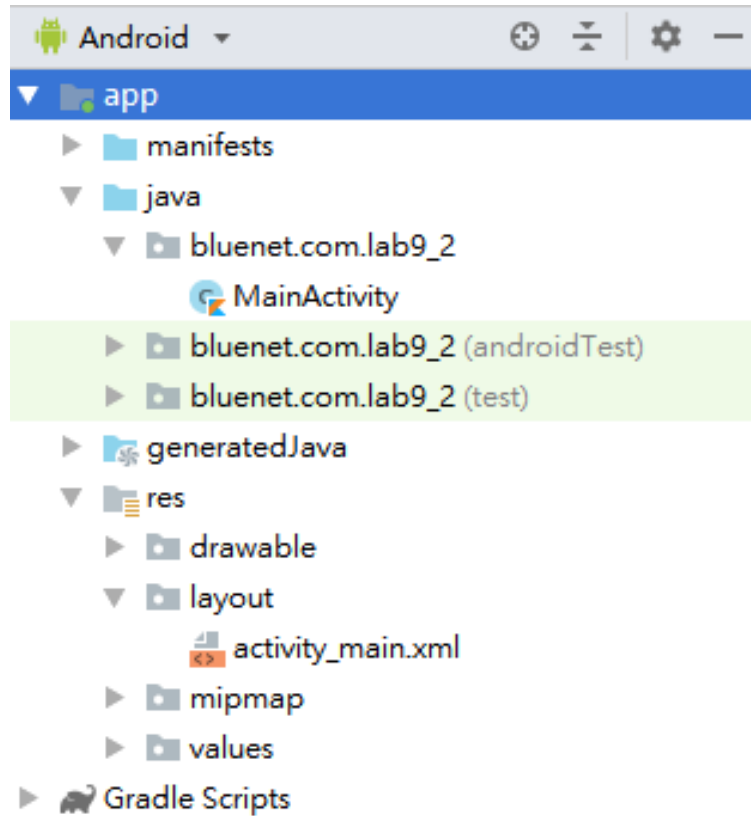


圖 9-9 計算機專案架構

Step2 繪製 activity_main.xml 檔，如圖 9-10 所示。



圖 9-10 計算機預覽畫面（左）與布局元件樹（右）

對應的 xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:text="體脂肪計算機"
```

```
        android:textSize="22sp"
        android:textColor="@android:color/black"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

<TextView

```
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="身高 :"
        app:layout_constraintStart_toStartOf="@+id/textView"
        app:layout_constraintTop_toBottomOf="@+id/textView" />
```

<EditText

```
        android:id="@+id/ed_height"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:ems="10"
        android:inputType="numberSigned"
        app:layout_constraintBottom_toBottomOf="@+id/textView2"
        app:layout_constraintStart_toEndOf="@+id/textView2"
        app:layout_constraintTop_toTopOf="@+id/textView2" />
```

<TextView

```
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="體重 :"
        app:layout_constraintStart_toStartOf="@+id/textView"
        app:layout_constraintTop_toBottomOf="@+id/textView2" />
```

<EditText

```
        android:id="@+id/ed_weight"
        android:layout_width="wrap_content"
        android:layout_height="47dp"
```

```
        android:layout_marginStart="8dp"
        android:ems="10"
        android:inputType="numberSigned"
        app:layout_constraintBottom_toBottomOf="@+id/textView3"
        app:layout_constraintStart_toEndOf="@+id/textView3"
        app:layout_constraintTop_toTopOf="@+id/textView3" />
```

<RadioGroup

```
        android:id="@+id/radioGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:orientation="horizontal"
        app:layout_constraintStart_toStartOf="@+id/textView2"
        app:layout_constraintTop_toBottomOf="@+id/ed_weight">
```

<RadioButton

```
        android:id="@+id/btn_boy"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="男生"
        android:checked="true"/>
```

<RadioButton

```
        android:id="@+id/btn_girl"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="女生"/>
```

</RadioGroup>

<Button

```
        android:id="@+id/btn_calculate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="計算"
        app:layout_constraintStart_toStartOf="@+id/textView2"
```

```
app:layout_constraintTop_toBottomOf="@+id/radioGroup" />
```

```
<TextView
```

```
    android:id="@+id/tv_weight"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="32dp"
    android:text="標準體重\n 無"
    android:gravity="center"
    app:layout_constraintBottom_toBottomOf="@+id/btn_calculate"
    app:layout_constraintStart_toEndOf="@+id/btn_calculate"
    app:layout_constraintTop_toTopOf="@+id/btn_calculate" />
```

```
<TextView
```

```
    android:id="@+id/tv_bmi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="32dp"
    android:text="體脂肪\n 無"
    android:gravity="center"
    app:layout_constraintBottom_toBottomOf="@+id/tv_weight"
    app:layout_constraintStart_toEndOf="@+id/tv_weight"
    app:layout_constraintTop_toTopOf="@+id/tv_weight" />
```

```
<LinearLayout
```

```
    android:id="@+id/ll_progress"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:elevation="3dp"
    android:background="#cc000000"
    android:clickable="true" android:visibility="gone">
```

```
<ProgressBar
```

```
    style="?android:attr/progressBarStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

```

        <ProgressBar
            android:id="@+id/progressBar2"
            style="?android:attr/progressBarStyleHorizontal"
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:progress="0"/>

        <TextView
            android:id="@+id/tv_progress"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:text="0%"
            android:textColor="@android:color/white"/>

    </LinearLayout>
</android.support.constraint.ConstraintLayout>

```

9.3.2 AsyncTask 進度更新

Step3 在方法 onCreate 中建立 Button 監聽按下事件執行 AsyncTask。

```

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //計算按鈕監聽事件
        btn_calculate.setOnClickListener {
            when{
                ed_height.length()<1 ->Toast.makeText(this,
                    "請輸入身高",Toast.LENGTH_SHORT).show()
                ed_weight.length()<1 ->Toast.makeText(this,
                    "請輸入體重",Toast.LENGTH_SHORT).show()
                else-> runAsyncTask() //執行副程式來執行 AsyncTask
            }
        }
    }
}

```

Step4 runAsyncTask()中編寫 AsyncTask。

```
private fun runAsyncTask(){
    object : AsyncTask<Void, Int, Boolean>() {
        override fun onPreExecute() {
            super.onPreExecute()
            tv_weight.text = "標準體重\n無"
            tv_bmi.text = "體脂肪\n無"
            //初始化進度條
            progressBar2.progress = 0
            tv_progress.text = "0%"
            //顯示進度條
            ll_progress.visibility = View.VISIBLE
        }

        override fun doInBackground(vararg voids: Void):
Boolean? {
            var progress = 0
            //建立迴圈執行一百次共延長 5 秒
            while (progress <= 100) {
                try {
                    //執行緒延遲 50ms 後執行
                    Thread.sleep(50)
                    //執行進度更新
                    publishProgress(progress)
                    //計數+1
                    progress++
                } catch (e: InterruptedException) {
                    e.printStackTrace()
                }
            }
            return true
        }

        override fun onProgressUpdate(vararg values: Int?) {
            super.onProgressUpdate(*values)
            values[0]?.let {
                //更新進度條進度
                progressBar2.progress = it
                tv_progress.text = "$it%"
            }
        }

        override fun onPostExecute(status: Boolean?) {
            ll_progress.visibility = View.GONE

            val cal_height = //身高
                ed_height.text.toString().toDouble()
            val cal_weight = //體重
                ed_weight.text.toString().toDouble()
            val cal_standweight: Double
            val cal_bodyfat: Double
            if (btn_boy.isChecked) {
                cal_standweight = (cal_height - 80) * 0.7
                cal_bodyfat = (cal_weight - 0.88 *
                    cal_standweight) / cal_weight * 100
            } else {
```




```
        cal_standweight = (cal_height - 70) * 0.6
        cal_bodyfat = (cal_weight - 0.82 *
                        cal_standweight) / cal_weight * 100
    }

    tv_weight.text = "標準體重 \n${String.format("%.2f",
                                                    cal_standweight)}"
    tv_bmi.text = "體脂肪 \n${String.format("%.2f",
                                              cal_bodyfat)}"
    }
}.execute()
}
```



標準體重	體脂肪
63.00	20.80