

Android Tutorial using Kotlin 第二堂

（3）應用程式與使用者的互動

[Android Tutorial using Kotlin 第二堂（2）設計應用程式使用者介面 << 前情](#)

Android API提供應用程式使用者互動的設計架構，你可以根據使用者在應用程式的操作，設計與提供應用程式與使用者的互動能力。例如使用者點擊畫面元件、按下實體按鍵，還有在觸控螢幕上點擊或滑動，這些操作行為通常會稱為「事件」。應用程式依照需求加入事件的控制，當某一種事件發生的時候，也就是使用者執行某種操作，可以執行你為這些事件設計好的程式碼。

Android系統的使用者操作事件控制，都是一些已經設計好的作法，根據使用者操作事件和畫面元件的種類，通常是撰寫實作面（interface）的類別，根據這個介面的規定實作需要的函式，在函式裡面設計需要執行的工作。

7-1 畫面元件的onClick設定

想要讓應用程式提供的畫面元件，可以讓使用者點擊以後執行一個指定的工作，最簡單的作法就是在畫面元件加入「android:onClick」設定，例如最常用的按鈕元件（Button）。如果需要的話，也可以為文字元件（TextView）執行onClick這開啟「res/layout/activity_main.xml」檔案，參考下面的內容加入需要的設定：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    ...
    <!-- 加入「android:clickable="true"」的設定，TextView元件才可以點擊 -->
    <!-- 加入「android:onClick="函式名稱"」的設定 -->
    <TextView
        ...
        android:clickable="true"
        android:onClick="aboutApp" />

</LinearLayout>
```

TextView是用來顯示文字的元件，所以要特別加入讓它可以點擊的設定，如果是按鈕元件的話就不用特別設定。如果希望使用點擊TextView元件以後，在畫面顯示應用程式名稱的訊息框，就要加入需要的程式碼。開啟專案的「MainActivity.kt」，參考下面內容加入需要的程式碼：

```
package net.macdidi.atk
...
```

```
// 加入訊息框的API
import android.widget.Toast

class MainActivity : AppCompatActivity() {

    ...

    // 函式名稱與onClick的設定一樣，參數的型態是android.view.View
    fun aboutApp(view: View) {
        // 顯示訊息框，指定三個參數
        // Context：通常指定為「this」
        // String或int：設定顯示在訊息框裡面的訊息或文字資源
        // int：設定訊息框停留在畫面的時間
        Toast.makeText(this, R.string.app_name, Toast.LENGTH_LONG).show()
    }
}
```

執行這個應用程式，在應用程式畫面點擊最下面的**TextView**元件，檢查有沒有顯示訊息框。

7-2 選單事件控制

如果應用程式提供的功能比較多一些，為了讓畫面可以比較簡潔，通常會把功能設計為選單，選單資源的部份已經在「第二章 規劃與建立應用程式需要的資源」建立好了。需要讓使用者選擇選單項目後執行一些特定的工作，最簡單的作法是為選單項目「onClick」的設定。開啟「res/menu/main_menu.xml」檔案，參考下面的內容加入需要的設定：

```
<?xml version="1.0" encoding="utf-8"?>
<menu 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"
    tools:context=".MainActivity">

    <!-- 為選單項目加入「android:onClick」設定 -->
    <item
        android:id="@+id/search_item"
        android:icon="@android:drawable/ic_menu_search"
        android:title="SEARCH"
        app:showAsAction="always"
        android:onClick="clickMenuItem" />

    <item
        android:id="@+id/add_item"
        android:icon="@android:drawable/ic_menu_add"
        android:title="ADD"
        app:showAsAction="always"
        android:onClick="clickMenuItem" />

    <item
        android:id="@+id/revert_item"
```

```

        android:icon="@android:drawable/ic_menu_revert"
        android:title="REVERT"
        app:showAsAction="always"
        android:onClick="clickMenuItem" />

<item
    android:id="@+id/delete_item"
    android:icon="@android:drawable/ic_menu_delete"
    android:title="DELETE"
    app:showAsAction="always"
    android:onClick="clickMenuItem" />
</menu>

```

這裡執行的設定跟之前的說明不太一樣，所有選單項目設定的函式名稱都是「**clickMenuItem**」。你也可以為每一個選單項目相同的函式名稱，可是這樣做的話，**Activity**元件裡面就要宣告很多函式，所以使用這樣的作法。開啟「**MainActivity.kt**」檔案，下面的內容加入需要的程式碼：

```

package net.macdidi.atk

...
import android.support.v7.app.AlertDialog
import android.view.Menu
import android.view.MenuItem

class MainActivity : AppCompatActivity() {
    ...

    // 使用者選擇所有的選單項目都會呼叫這個函式
    fun clickMenuItem(item: MenuItem) {
        // 使用參數取得使用者選擇的選單項目元件編號
        val itemId = item.getItemId()

        // 判斷該執行什麼工作，目前還沒有加入需要執行的工作
        when (itemId) {
            R.id.search_item -> {
            }
            R.id.add_item -> {
            }
            R.id.revert_item -> {
            }
            R.id.delete_item -> {
            }
        }

        // 測試用的程式碼，完成測試後記得移除
        val dialog = AlertDialog.Builder(this@MainActivity)
        dialog.setTitle("MenuItem Test")
            .setMessage(item.getTitle())
    }
}

```

```

        .setIcon(item.getIcon())
        .show()
    }

}

```

執行這個應用程式，選擇畫面上方的選單項目，檢查有沒有顯示對話框。

7-3 監聽與事件介紹

「`android.view`」和「`android.widget`」套件宣告了許多「`Listener`」介面，這些介面通常會叫作「監聽介面」。每一個監聽介以控制使用者在應用程式中執行的一種操作，這些介面的名稱都很規則，都是使用「`On`種類`Listener`」的格式命名。例如下列在「`android.view.View`」類別中的基本監聽介面：

- `View.OnClickListener`：執行點擊事件。
- `View.OnLongClickListener`：執行長按事件。
- `View.OnKeyListener`：執行實體按鍵操作事件。
- `View.OnTouchListener`：執行觸控螢幕操作事件。

採用這種方式為某個畫面元件加入事件控制，因為需要在程式碼使用畫面元件，所以一定要為元件取一個名稱，設定元件名稱「`android:id="@+id/名稱"`」的格式：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    ...
    <!-- 加入「android:id="@+id/名稱"」的設定 -->
    <TextView
        android:id="@+id/show_app_name"
        ... />
</LinearLayout>

```

為需要執行事件控制的元件設定好名稱（`id`）以後，讓使用者在點擊這個元件以後，使用對話框顯示比較詳細的應用程式資訊。以先在文字資源檔（`res/values/strings.xml`）加入需要的資源：

```

<resources>
    ...
    <string name="about">這是Android Tutorial應用程式</string>
</resources>

```

開啟專案的「MainActivity.kt」，參考下面的內容加入需要的程式碼：

```
package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        // 讀取在畫面配置檔已經設定好名稱的元件
        val show_app_name: TextView = findViewById(R.id.show_app_name)

        // 建立點擊監聽物件
        val listener = View.OnClickListener {
            val d = AlertDialog.Builder(this@MainActivity)
            d.setTitle(R.string.app_name)
            d.setMessage(R.string.about)
            d.show()
        }

        // 註冊點擊監聽物件
        show_app_name.setOnClickListener(listener)
    }
    ...
}
```

執行這個應用程式，在應用程式畫面點擊最下面的TextView元件，檢查有沒有顯示對話框。因為你在這個TextView元件執行OnClickListener事件的註冊，它的「android:onClick」設定就被覆蓋了，所以點擊以後只會顯示對話框。

一般的應用程式也很常使用長按事件，開啟專案的「MainActivity.kt」，參考下面的內容，把原來的點擊事件改為長按事件：

```
package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        // 讀取在畫面配置檔已經設定好名稱的元件
        val show_app_name: TextView = findViewById(R.id.show_app_name)

        ...
        // 建立長按監聽物件
        val longListener = View.OnLongClickListener {
            val dialog = AlertDialog.Builder(this@MainActivity)
            dialog.setTitle(R.string.app_name)
            dialog.setMessage("Long Click: ${getString(R.string.about)}")
            dialog.show()

            false
        }
    }
}
```

```

    }

    // 註冊長按監聽物件
    show_app_name.setOnLongClickListener(longListener)

}

...

}

```

執行這個應用程式，在應用程式畫面點擊最下面的TextView元件，會顯示原來設定的訊息框，長按TextView元件會顯示對話框。由這兩個練習，就可以瞭解Android事件的設計方式，在大部份的情況下，監聽介面都會以「On」開頭，宣告與建立好監聽物件後，呼叫元件的「set監聽介面」函式執行註冊的工作。

7-4 ListView元件的事件控制

ListView元件在應用程式中的應用非常多，從應用程式的功能表、瀏覽大量的資料或是讓使用者執行資料的選擇，應用程式需列表資料的需求，都可以使用它來完成。它可以簡單的列出一些文字的項目在畫面上，讓使用者瀏覽與選擇。也可以自己設計的項目畫面，加入圖示、CheckBox或其它需要的畫面元件，它呈現的畫面與可以提供的操作功能都非常靈活。

這個記事本的主畫面使用ListView元件顯示所有的記事資料，選擇一個項目以後可以顯示詳細的內容與執行後續的工作，所以ListView設定選擇項目的事件控制。開啟專案的「MainActivity.kt」，參考下面的內容加入需要的程式碼：

```

package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        // 建立選單項目點擊監聽物件
        val itemListener = AdapterView.OnItemClickListener {
            // parent: 使用者操作的ListView物件
            // view: 使用者選擇的項目
            // position: 使用者選擇的項目編號，第一個是0
            // id: 在這裡沒有用途
            parent, view, position, id ->
            Toast.makeText(this@MainActivity,
                data[position], Toast.LENGTH_LONG).show()
        }

        // 註冊選單項目點擊監聽物件
        item_list.setOnItemClickListener(itemListener)
    }

    ...

}

```

執行這個應用程式，在應用程式畫面點擊`ListView`的選單項目，看看有沒有顯示選單項目的內容訊息框。`ListView`元件也提供點擊事件，你可以依照應用程式的需求，使用點擊與長按事件提供使用者的操作。開啟專案的「`MainActivity.kt`」，參考下面的人需要的程式碼：

```
package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        // 建立選單項目長按監聽物件
        val itemLongListener = AdapterView.OnItemLongClickListener {
            // parent: 使用者操作的ListView物件
            // view: 使用者選擇的項目
            // position: 使用者選擇的項目編號，第一個是0
            // id: 在這裡沒有用途
            parent, view, position, id ->
            Toast.makeText(this@MainActivity,
                "Long: ${data[position]}", Toast.LENGTH_LONG).show()
            false
        }

        // 註冊選單項目長按監聽物件
        item_list.setOnItemLongClickListener(itemLongListener)

        ...
    }
}
```

執行這個應用程式，在應用程式畫面長按`ListView`的選單項目，看看有沒有顯示選單項目的內容訊息框。

7-5 重新規劃Activity元件的程式碼

如果Activity元件需要的畫面元件比較多一些，使用者操作的功能也比較複雜，你應該可以想像得到，這個Activity元件類別的`onCreate`函式，會有一大堆呼叫`findViewById`函式取得畫面元件物件的敘述，還有宣告與建立監聽物件與執行註冊的敘述。這要的敘述通通寫在`onCreate`函式中，以程式設計的概念來說，一個函式的宣告有上百行的程式敘述，應該不是一種很好的寫法。開發人員來說，以後的維護與修改都會是一件不容易的工作。

為了讓所有Activity元件的程式碼，都可以使用一種比較固定而且容易的設計方式來完成需要的工作，建議你可以在開發每一個Activity元件類別的時候，使用像這個樣版的模式來開發Activity元件：

```
public class SampleActivity extends Activity {
    // 宣告所有需要的畫面元件物件欄位變數
    private ...;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

```

        setContentView(...);

        // 呼叫自己額外宣告的函式，執行所有註冊的工作
        processControllers();
    }

    private void processControllers() {
        // 在這個函式中，宣告或建立需要的監聽物件
        //      並執行所有需要的註冊工作
        ...
    }
    ...
}

```

熟悉這樣的寫法以後，原來需要執行的工作會在不同的函式中執行。你會先加入畫面元件欄位變數的宣告，然後在`processView`函式中取得與設定畫面元件物件，如果需要執行註冊監聽物件的工作，在`processControllers`函式中加入需要的程式碼。這樣把和依照工作簡單的分開在不同函式中執行，對以後的維護與修改都會有一個比較固定的作法，而且比較不容易出錯。所以就算是很簡單的`Activity`元件，都建議你使用這樣的寫法。

為了可以比較方便讀取畫面元件，先建立讀取畫面元件的函式，在應用程式主要套件目錄按滑鼠右鍵，選擇「New -> Kotlin File/Class」，在Name：欄位輸入「Bind」後按「OK」按鈕，建立一個新的Kotlin程式，參考下面的內容建立需要的函式：

```

package net.macdidi.atk

import android.app.Activity
import android.support.annotation.IdRes
import android.view.View

// 一般畫面元件
//  參數為畫面元件編號
//  回傳畫面元件物件
fun <T : View> Activity.bind(@IdRes res : Int) : Lazy<T> {
    return lazy { findViewById<T>(res) }
}

// 自定畫面元件
//  參數為畫面元件編號
//  回傳畫面元件物件
fun <T : View> View.bind(@IdRes res : Int) : Lazy<T> {
    return lazy { findViewById<T>(res) }
}

```

開啟專案的「MainActivity.kt」，不改變原來撰寫好的功能，把它改為下面的內容：


```

package net.macdidi.atk

import android.os.Bundle
import android.support.v7.app.AlertDialog
import android.support.v7.app.AppCompatActivity
import android.view.Menu
import android.view.MenuItem
import android.view.View
import android.widget.*

class MainActivity : AppCompatActivity() {

    private val item_list : ListView by bind(R.id.item_list)
    private val show_app_name: TextView by bind(R.id.show_app_name)

    private val data = arrayOf("關於Android Tutorial的事情",
                                "一隻非常可愛的小狗狗!", "一首非常好聽的音樂!")
    private val adapter : ArrayAdapter<String>
        by lazy { ArrayAdapter(this, android.R.layout.simple_list_item_1, data) }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        processControllers()

        item_list.adapter = adapter
    }

    private fun processControllers() {
        // 建立選單項目點擊監聽物件
        val itemListener = AdapterView.OnItemClickListener {
            // parent: 使用者操作的ListView物件
            // view: 使用者選擇的項目
            // position: 使用者選擇的項目編號，第一個是0
            // id: 在這裡沒有用途
            parent, view, position, id ->
            Toast.makeText(this@MainActivity,
                           data[position], Toast.LENGTH_LONG).show()
        }

        // 註冊選單項目點擊監聽物件
        item_list.setOnItemClickListener = itemListener

        // 建立選單項目長按監聽物件
        val itemLongListener = AdapterView.OnItemLongClickListener {
            // parent: 使用者操作的ListView物件
            // view: 使用者選擇的項目
            // position: 使用者選擇的項目編號，第一個是0
            // id: 在這裡沒有用途
            parent, view, position, id ->

```

```

        Toast.makeText(this@MainActivity,
            "Long: ${data[position]}", Toast.LENGTH_LONG).show()
        false
    }

    // 註冊選單項目長按監聽物件
    item_list.onItemLongClickListener = itemLongListener

    // 建立長按監聽物件
    val listener = View.OnLongClickListener {
        val dialog = AlertDialog.Builder(this@MainActivity)
        dialog.setTitle(R.string.app_name)
        dialog.setMessage(R.string.about)
        dialog.show()
        false
    }

    // 註冊長按監聽物件
    show_app_name.setOnLongClickListener(listener)
}

// 載入選單資源
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.menu_main, menu)
    return true
}

// 使用者選擇所有的選單項目都會呼叫這個函式
fun clickMenuItem(item: MenuItem) {
    // 使用參數取得使用者選擇的選單項目元件編號
    val itemId = item.itemId

    // 判斷該執行什麼工作，目前還沒有加入需要執行的工作
    when (itemId) {
        R.id.search_item -> {
        }
        R.id.add_item -> {
        }
        R.id.revert_item -> {
        }
        R.id.delete_item -> {
        }
    }
}

// 測試用的程式碼，完成測試後記得移除
val dialog = AlertDialog.Builder(this@MainActivity)
dialog.setTitle("MenuItem Test")
    .setMessage(item.title)
    .setIcon(item.icon)
    .show()
}

```

```
// 函式名稱與onClick的設定一樣，參數的型態是android.view.View
fun aboutApp(view: View) {
    // 顯示訊息框
    // Context：通常指定為「this」；如果在巢狀類別中使用，要加上這個Activity元件類別的名稱，例如
    // String或int：設定顯示在訊息框裡面的訊息或文字資源
    // int：設定訊息框停留在畫面的時間，使用宣告在Toast類別中的變數，可以設定為「LENGTH_LONG」和
    Toast.makeText(this, R.string.app_name, Toast.LENGTH_LONG).show()
}

}
```

完成這個階段的工作了，執行這個應用程式，確認所有功能都可以正確的運作。

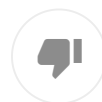
相關的檔案都可以在[GitHub](#)瀏覽與下載：

GitHub


<https://github.com/macdidi5/Android-Tutorial-Kotlin>

後續 >> [Android Tutorial using Kotlin 第二堂（4）建立與使用Activity元件](#)

Does Clearly work fine?



Shortcuts: **SHIFT+CTRL+C** to Toggle, **ESC** to Close.

 Give us feedback

Build upon  with Clearly