

12

Android Fragment

- 12-1 Android Fragment
- 12-2 Fragment UI 佈局與管理
- 12-3 使用內建 Fragment UI
- 12-4 建構靈活的 UI 界面
- 12-5 分頁指示器 ViewPagerIndicator 與傳值



為了讓 Android UI 在平板上有更好地演示，Android 3.0 之後加入了 Fragment (碎片) 功能，增加互動介面上的彈性，Fragment 原本是給平板設備的專屬，Android 4.0 手機與平板 SDK 整合之後，手機也因此受惠自動提供此界面功能。

12-1 Android Fragment

在開發上您可以想成 Fragment 是一個模組是 Activity 組成的一部分，可以很輕易的建立、替換或移除 Fragment，甚至有時為了要保留之前的 Fragment 可以透過 `addToBackStack()` 將 Fragment 資源保留，讓使用者方便使用 BACK 鍵來回朔查找之前所建立過的 Fragment。Fragment 有自己的生命週期並且可以交互於 Activity 的生命週期之中，進而可以透過 `getActivity()` 來分享所有目前依附在 Activity 中的所有 Fragments 的 UI 資源。Fragment 也如同一個子 Activity(sub activity) 可以自由的部署與共享在任何 Activity 中。

開發上使用 Fragment 的好處有：

1. 模組化並有自己的生命週期。
2. 可以重複利用降低開發成本。
3. 一個 Fragment 在平板 (tablet) 與手機 (handset) 上可以有不同的使用者體驗。

開發 Fragment UI 方式有：

1. 客製化 Fragment。
2. 內建 Fragment。
 - A. DialogFragment
 - B. ListFragment
 - C. WebViewFragment ... 等

關於 Fragment API 相容性版本 API Level

◆ Android 3.0 (API Level 11) (含) 以上

直接可以使用 Fragment 相關 API。在取得 FragmentManager 與 LoaderManager 實體方面可調用 getFragmentManager() 取得 android.app.FragmentManager 實體，調用 getLoaderManager() 取得 android.app.LoaderManager 實體。

◆ Android 3.0 (API Level 11) 以下

- 匯入 v4 支援函式庫套件 (android-support-v4.jar)【註】使專案得以支援 Fragment API。

v4 支援函式庫.jar 檔路徑：

{Android ADT 安裝路徑 }/sdk/extras/android/support/v4/ android-support-v4.jar。

- 專案若有使用 action bar 機制則需同時匯入 v7 支援函式庫中的 v7 appcompat 函式套件 (android-support-v7-appcompat.jar) 與匯入 v4 支援函式庫套件 (android-support-v4.jar)，因為 v7 appcompat 函式庫必須依賴在 v4 支援函式庫上。

v7 與 v4 支援函式庫.jar 檔路徑：

{Android ADT 安裝路徑 }/sdk/extras/android/support/v7/appcompat/libs/ android-support-v7-appcompat.jar。

{Android ADT 安裝路徑 }/sdk/extras/android/support/v7/appcompat/libs/ android-support-v4.jar (因 v7 支援函式庫必須依賴在 v4 支援函式庫上，所以可以發現在 v7 函式庫路徑內有多付一個 android-support-v4.jar)。

Android 3.0 (API Level 11) 含以上	Android 3.0 (API Level 11) 以下
import android.app.Fragment with Activity 所使用的 Fragment 類別不可以 import 自 android.support.v4.app.Fragment，否則將會導致執行時期錯誤	import android.support.v4.app.Fragment with FragmentActivity
import android.app.FragmentManager 調用 getFragmentManager() 取得 FragmentManager 實例	import android.support.v4.app.FragmentManager 調用 getSupportFragmentManager() 取得 FragmentManager 實例

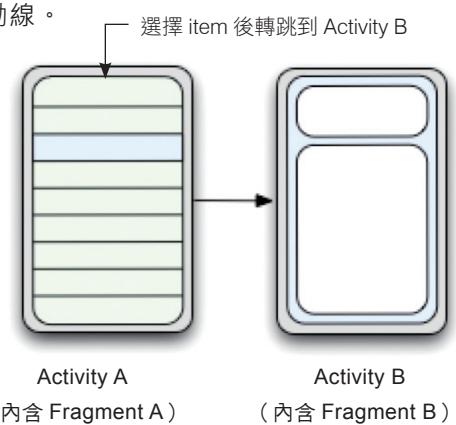
Android 3.0 (API Level 11) 含以上	Android 3.0 (API Level 11) 以下
<pre>import android.app.LoaderManager 調用 getLoaderManager() 取得 LoaderManager 實例</pre>	<pre>import android.support.v4.app. LoaderManager 調用 getSupportFragmentManager() 取得 LoaderManager 實例</pre>



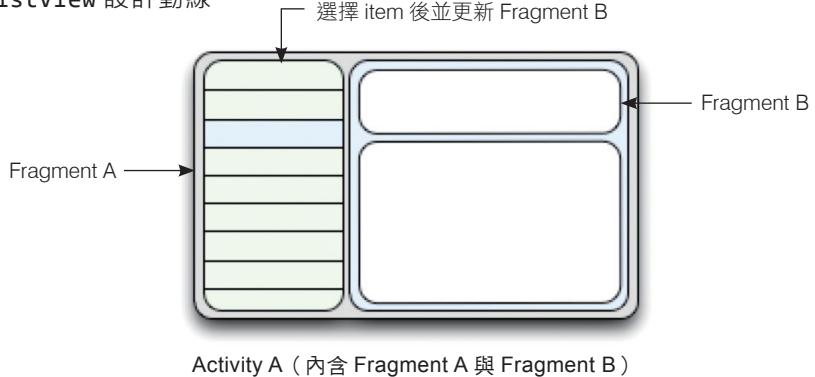
注意！Android 3.0 (API Level 11) 以下在使用支援函式庫不論是 v4 或 v7 時，Fragment 應使用在 FragmentActivity 而非 Activity 環境。

設計 Fragment 動線

手機 ListView 設計動線。



平板 ListView 設計動線。





Android 支援函式庫

Android 支援函式庫主要是讓早期的 Android 版本能夠享有新版 Android API 的功能與效率，降低在開發上因著 Android 版本不同而發生 API 不一致或沒有新 API 的窘境。

- 安裝 Android 支援函式庫：開啟 Android SDK Manager > Extras > Android Support Library (請參考下圖)。



安裝 Android 支援函式庫

- Android 支援函式庫安裝路徑：`{Android ADT 安裝路徑}/sdk/extras/android/support/{vXX 版本}`。
- Android 支援函式庫 jar 檔的取得：
 - 以 v4 Support Library (`android-support-v4.jar`) 為例：
`{Android ADT 安裝路徑}/sdk/extras/android/support/v4/android-support-v4.jar`
 - 以 v7 appcompat (`android-support-v7-appcompat.jar`) 為例：
`{Android ADT 安裝路徑}/sdk/extras/android/support/v7/appcompat/libs/ android-support-v7-appcompat.jar`

截至目前為止 Android 官方支援的函式庫有：

<http://developer.android.com/tools/support-library/index.html>

- v4 Support Library

支援 Android 1.6 (API Level 4) 含以上，內含四類支援套件：
App Components (內含支援 Fragment 類別)、User Interface、Accessibility 與 Content。

路徑：`{Android ADT 安裝路徑}/sdk/extras/android/support/v4`

- v7 Libraries

支援 Android 2.1 (API Level 4) 含以上，內含 v7 appcompat、v7 gridlayout 與 v7 mediarouter library 三種支援套件。

路徑：{Android ADT 安裝路徑 }/sdk/extras/android/support/v7

注意：v7 appcompat library 需依附在 v4 Support Library 上，使用時 android-support-v7-appcompat.jar 與 android-support-v4.jar 必須同時設定類別路徑（classpath）與置入 res\libs。

- v8 Support Library

支援 Android 2.2 (API Level 8) 含以上，內含 Renderscriptel 高性能 3D 渲染平行運算新框架。

路徑：{Android ADT 安裝路徑 }/sdk/extras/android/support/v8

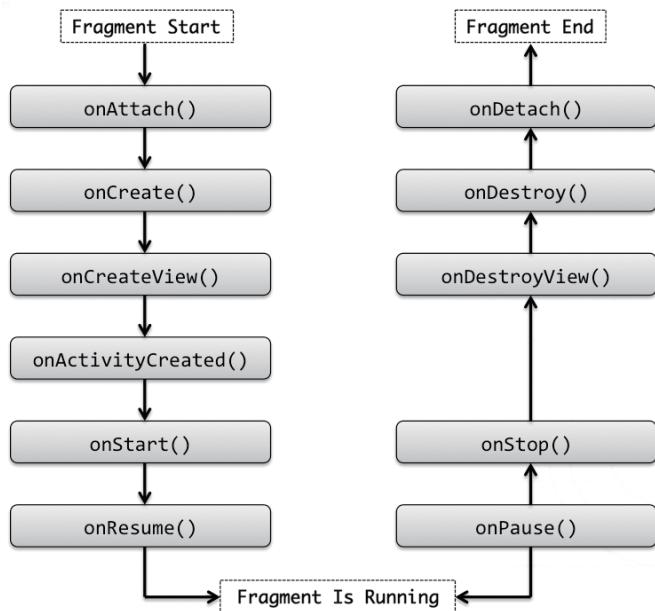
- v7 Libraries

支援 Android 3.2 (API Level 13) 含以上，內含 FragmentCompat 類別可以向後兼容 Fragment 的功能。

路徑：{Android ADT 安裝路徑 }/sdk/extras/android/support/v13

Fragment 生命週期

Fragment 如同 Activity 一樣有著自己的生命週期，值得注意的是 Fragment 不能獨立存活，必須依附 Activity，因此 Fragment 生命週期也會受到 Activity 生命週期的約束。Fragment 生命週期如右圖所示：

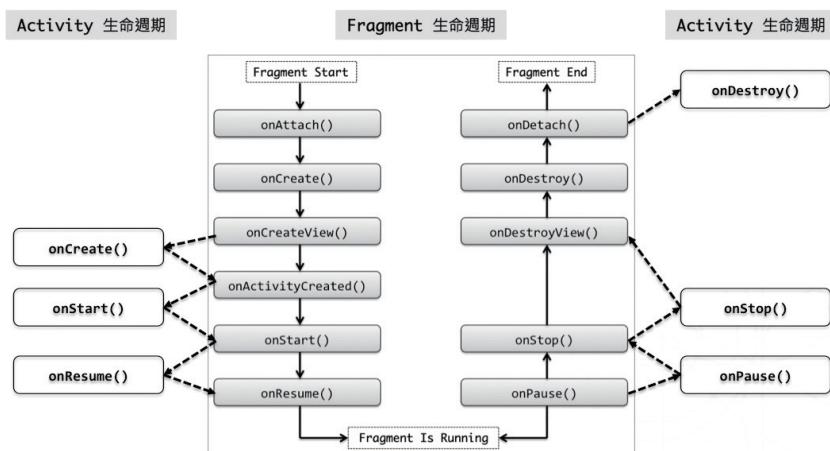


以下針對 Fragment 一些重要的生命週期回呼方法做說明，如下表：

表：Fragment 生命週期回呼函式

回呼方法名稱	說明
onAttach()	Activiry 與 Fragment 建立關係。
onCreate()	Fragment 剛被建立可以用來設定 Fragement 物件資源初始值。
onCreateView ()	Fragment 即將可以顯示，可用來設定 Fragment Layout 界面佈局
onActivityCreated()	所依附的 Activity onCreate() 已經執行完畢並準備進入 Activity onStart() 時所回呼的函式，用來設定 Fragment (使用 getView()) 或取得 Activity (使用 getActivity()) Layout 上的 UI View 物件。
onDestroy()	Fragment Layout 佈局資料被移除。
onDetach()	Fragment 已與 Activity 脫離。

Activity 與 Fragment 生命週期 (靜態 Fragment 佈局) 回呼函式交互運作圖：



Activity 與靜態 Fragment 生命週期回呼函式調用關係圖

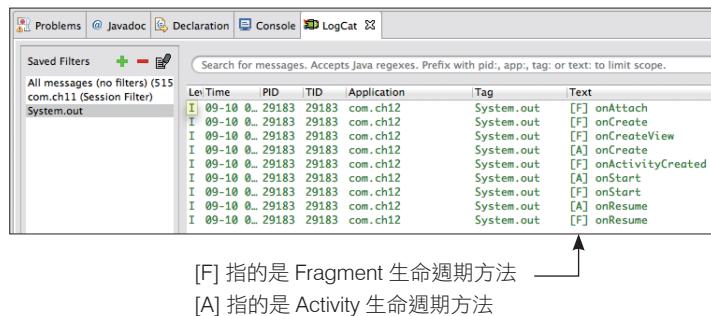
由上圖可知，Fragment 生命週期中 onCreateView() 方法執行完畢之後，緊接著才會調用執行 Activity 生命週期中的 onCreate() 方法。

範例程式 (CH12_01_Lifecycle)：檢視 Android Fragment 生命週期

執行結果：



於 LogCat 中可以檢視 System.out 跑出的執行結果。



The screenshot shows the Eclipse IDE's LogCat view. The title bar includes tabs for Problems, Javadoc, Declaration, Console, and LogCat. The LogCat tab is active. A search bar at the top says "Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope." Below the search bar, there is a table with the following columns: Level, Time, PID, TID, Application, Tag, and Text. The table contains ten entries, all of which are of level "I" (Information) and timestamped "09-10 0_". The Application column shows "com.ch12". The Tag column shows "System.out". The Text column contains various lifecycle method names: "onAttach", "onCreate", "onCreateView", "onCreate", "onActivityCreated", "onStart", "onStart", "onResume", and "onResume". An arrow points from the text "[F]" to the "onAttach" entry, and another arrow points from the text "[A]" to the "onCreate" entry.

Level	Time	PID	TID	Application	Tag	Text
All messages (no filters) (515 com.ch12 (Session Filter))						
I	09-10 0_	29183	29183	com.ch12	System.out	[F] onAttach
I	09-10 0_	29183	29183	com.ch12	System.out	[F] onCreate
I	09-10 0_	29183	29183	com.ch12	System.out	[F] onCreateView
I	09-10 0_	29183	29183	com.ch12	System.out	[A] onCreate
I	09-10 0_	29183	29183	com.ch12	System.out	[F] onActivityCreated
I	09-10 0_	29183	29183	com.ch12	System.out	[A] onStart
I	09-10 0_	29183	29183	com.ch12	System.out	[F] onStart
I	09-10 0_	29183	29183	com.ch12	System.out	[A] onResume
I	09-10 0_	29183	29183	com.ch12	System.out	[F] onResume

[F] 指的是 Fragment 生命週期方法
[A] 指的是 Activity 生命週期方法

CH12_01_Lifecycle 範例程式執行結果 LogCat 截圖

12-2 Fragment UI 佈局與管理

Fragment 在 UI 畫面佈局上可分為靜態部署與動態部署，這二者最大的差別在於靜態部署必須在 Activity xml 佈局畫面中事先宣告一定數量與固定位置的 xml 標籤 (`<fragment>`)，而動態部署不必事先在 Activity xml 佈局畫面中事先宣告，只要有設定 `android:id` 的 layout 都可以成為動態生成 Fragment 的容器 (`fragments` 部署地)，且數量與擺放位置皆可以自由決定增加 UI 部署上的彈性。

12-2-1 Fragment - 靜態部署

透過在 Activity xml 佈局畫面中的 `<fragment>` 標籤來部署 fragment UI，使得 fragment UI 可以重複使用，以下就以 CH12_02_Static 範例來說說明 Fragment 靜態部署。

範例程式 (CH12_02_Static) 相關 Java API

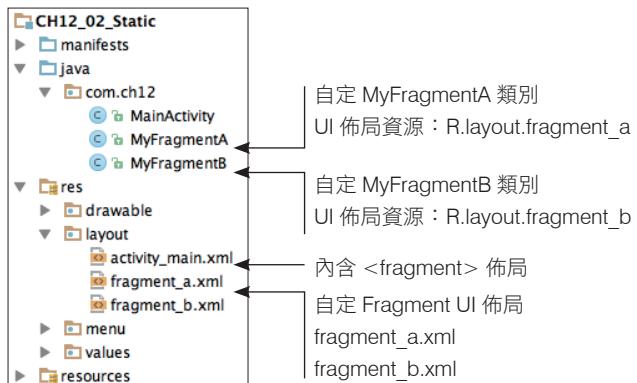
自定 Fragment 繼承宣告

```
import android.app.Fragment;
public class MyFragmentA extends Fragment {
    ...
}
```

↑ 繼承 Fragment

範例程式 (CH12_02_Static)：Fragment 靜態部署

專案結構：



執行結果與 activity_main.xml 對應：



<fragment> 標籤

```
<fragment
    android:id="@+id/fragment1"
    android:name="com.ch12.MyFragmentA" ← 所使用的 Fragment 類別路徑
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp" />
```

重點程式碼：

```
CH12_02_Static/com.ch12.MyFragmentA.java
```

..	...
20	@Override
21	public void onCreate(Bundle savedInstanceState) {
22	super.onCreate(savedInstanceState);
23	context = getActivity(); ← 取得 context 物件實體
24	}
25	
26	// 回呼狀態：Fragment 即將可以顯示在螢幕時

```
27 // 使用時機：用來設定 Fragment Layout 界面佈局
28 @Override
29 public View onCreateView(LayoutInflater inflater,
30     ViewGroup container, Bundle savedInstanceState) {
31     /*
32      * inflater 的 inflate 方法的三個參數分別是：
33      * int resource : fragment 的 layout 資源 ID 。
34      * ViewGroup root : 存放 fragment 的 layout 的 ViewGroup
35      * boolean attachToRoot : 指定展開的佈局是否依附到 ViewGroup 中
36      */
37     View view = inflater.inflate(R.layout.fragment_a, container, false);
38
39     // 記得要 return Fragment Layout 物件
40     return view;
41 }
42
43 // 回呼狀態：可以開始取得 Fragment Layout 界面物件
44 // 使用時機：用來設定 Layout 上的 UI View 物件
45 @Override
46 public void onActivityCreated(Bundle savedInstanceState) {
47     super.onActivityCreated(savedInstanceState);
48
49         ← 取得 Fragment 的 Layout 實例
50     textView1 = (TextView) getView().findViewById(R.id.textView1);
51     textView1.setText("X'mas Tree");
52
53     imageView1 = (ImageView) getView().findViewById(R.id.imageView1);
54     imageView1.setImageResource(R.drawable.tree);
55 }
...
...
```

👉 重點說明：

程式第 29~41 行用來設定與宣告 Fragment Layout 界面佈局實例 view，最重要的是第 40 行要將所建立的 view 實例回傳。

程式第 46~55 行設定 Fragment Layout 界面上的 View 元件，宣告這些 View 元件可以調用 `getView().findViewById()` 方法來取得，值得注意的是 `getView()` 所取得的實例其實就是第 40 行中所回傳的 view 實例（也就是 Fragment Layout）。

`com.ch12.MyFragmentB.java` 基本上與 `com.ch12.MyFragmentA.java` 程式碼相同，在此就不加以贅述，請自行查閱。

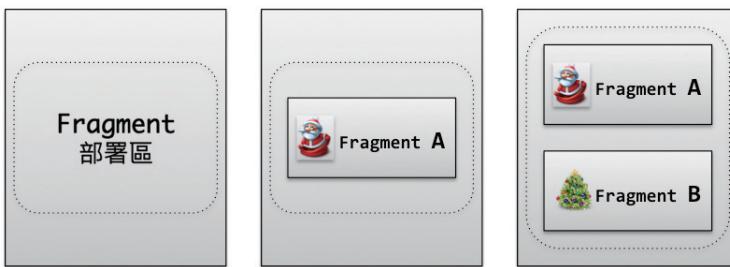
12-2-2 Fragment - 動態部署

Fragment 在界面部署上分為靜態與動態部署，不過若 Fragment 的配置為非固定數量或不定顯示時機與方式時，此時可以使用 Fragment 動態部署機制來滿足。

Fragment 動態部署主要分為動態新增 (add) 與動態置換 (replace) 二種：

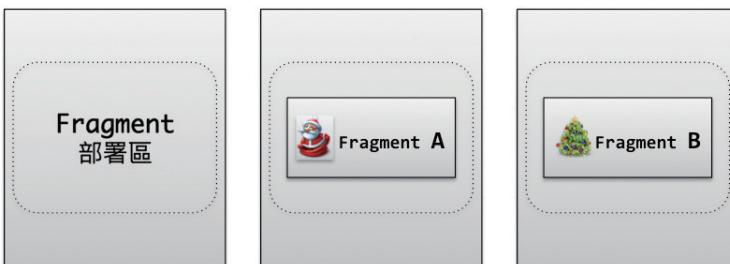
- Fragment 動態新增示意圖：

在 Fragment 部署區中先 add Fragment A 再 add Fragment B。



- Fragment 動態置換示意圖：

在 Fragment 部署區中先 replace Fragment A 再 replace Fragment B。注意：第一次 replace Fragment A 與調用 add Fragment A 部署狀況相同。



👉 範例程式 (CH12_02_Dyn) 相關 Java API

```
public abstract FragmentTransaction beginTransaction()
```

要求 FragmentManager 管理器提供一個 Fragment 交易機制 (FragmentTransaction)，並透過 FragmentTransaction 進行 Fragment 的新增、替換、移除等事宜。

```
public abstract FragmentTransaction add(int containerViewId, Fragment fragment, String tag)
```

把指定之 Fragment 加入到指定 UI 容器並定義 fragment 別名。

參數說明：

- containerViewId：指定 UI 容器。
- fragment：指定 Fragment 物件。
- tag：指定標籤別名以便日後利用 findFragmentByTag() 方法將該 Fragment 物件取出。

```
public abstract FragmentTransaction replace(int containerViewId, Fragment fragment, String tag)
```

把指定之 Fragment 加入到指定 UI 容器並定義 fragment 別名，若該容器已有 Fragment 則將予以替換。

```
public abstract FragmentTransaction remove(Fragment fragment)
```

從 FragmentManager 管理器與 UI 容器中移除指定 Fragment。注意，透過 remove() 移除的 Fragment 因為並不會保留在 FragmentManager 管理器中因此無法透過 attach() 换回。

```
public abstract FragmentTransaction attach(Fragment fragment)
```

於 FragmentManager 管理器中再次叫用之前被 detach 的 Fragment。

```
public abstract FragmentTransaction detach(Fragment fragment)
```

將指定 Fragment 從 UI 容器中移除。注意，透過 detach 移除 Fragment 時，該 Fragment 仍會繼續保留在 FragmentManager 管理器中，未來若有需要可透過 attach() 重新喚醒叫用。

```
public abstract FragmentTransaction addToBackStack(String name)
```

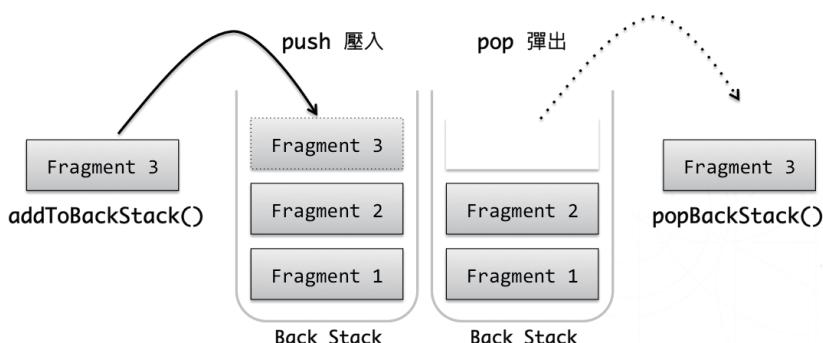
把此次的 Fragment 交易加入至 Back Stack 中，以便日後可以利用 popBackStack() 或手機返回鍵回復到之前的交易狀態。

```
public abstract int commit()
```

要求系統開始執行 `FragmentTransaction` 的交易行為（例如：`add()`、`remove()`、`replace()` 或 `addToBackStack()` 等行為）。注意，下達 `commit()` 時，該交易行為不會立即發生，因為該交易行為是由主執行緒負責執行，倘若主執行緒正處於忙碌狀態時，該交易行為會被安排在主執行緒的下一個工作排程中實施。

`Back Stack` 又稱退線堆疊，舉個例子：當我們由 `Activity A` 轉跳到另一個 `Activity B` 時可以按下手機上的返回鍵回復到前一個 `Activity`（也就是 `Activity A`）的狀態，這是因為當我們進行 `Activity A` 與 `Activity B` 之間的轉換時 `Android` 會自動將 `Activity` 個別之行為與狀態記錄在 `Back Stack` 退線堆疊中，以便我們日後可以利用返回鍵進行回溯，這在 `Activity` 間是可行的。不過針對 `Fragment` 部分，`Android` 並沒有預設自動記錄 `Fragment` 與 `Fragment` 之間的交易關係，所以我們必須手動調用 `addToBackStack()` 方法，將此次交易關係依序記錄在 `Back Stack` 中，以便日後可以透過返回鍵或 `popBackStack()` 方法回溯。

`Back Stack` 與 `addToBackStack()`、`popBackStack()` 運作關係圖：

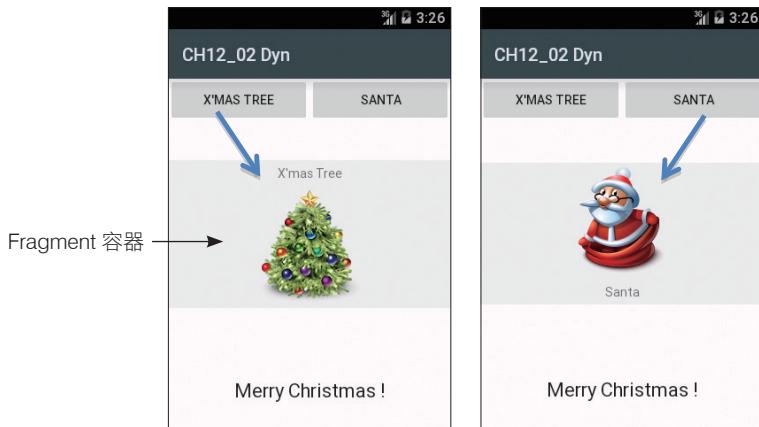


Back Stack 與 `addToBackStack()`、`popBackStack()` 運作關係圖

👉 範例程式（CH12_02_Dyn）：建立動態 Fragment

執行結果：

按下 X'mas Tree 與 Santa 鈕 fragment 容器將會自動切換內容。



👉 UI 界面部署與設計

```
CH12_02_Dyn/res/layout/activity_main.xml
01 <LinearLayout
02     xmlns:android="http://schemas.android.com/apk/res/android"
03     android:layout_width="fill_parent"
04     android:layout_height="fill_parent"
05     android:orientation="vertical" >
06 ...
07     <LinearLayout
08         android:id="@+id/fragment_addin_linearlayout"
09         android:layout_width="fill_parent"
10         android:layout_height="wrap_content"      ←—— Fragment 置入容器
11         android:layout_weight="1"
12         android:orientation="vertical" />
13 ...
14     </LinearLayout>
```

👉 重點說明：

第 32~48 行提供一個 layout 標籤當作執行時期動態 Fragment 置入用的容器。

第 33 行該容器 id 為 fragment_addin_linearlayout。

👉 重點程式碼：

CH12_02_Dyn /com.ch12.MainActivity.java

```
...
16     public void onClick(View view) {
17         // 建立 Fragment 交易服務機制
18         FragmentTransaction ft = getFragmentManager().beginTransaction();
19         switch(view.getId()) {
20             case R.id.buttonA:
21                 // 置換 fragment
22                 ft.replace(R.id.fragment_addin_linearlayout,
23                             new MyFragmentA(), "f_a");
24                 break;
25             case R.id.buttonB:
26                 // 置換 fragment
27                 ft.replace(R.id.fragment_addin_linearlayout,
28                             new MyFragmentB(), "f_b");
29                 break;
30         }
31         // 提交
32         ft.commit();
33     }
...

```

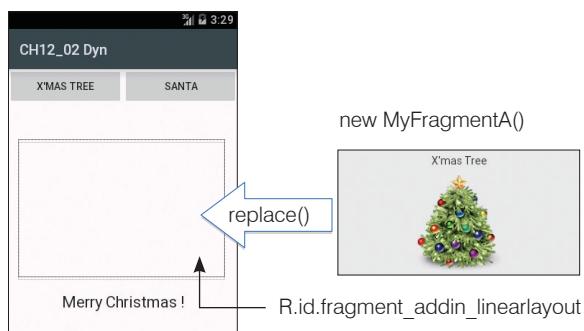
↑ 將自定 Fragment 部署至指定 Layout
並給予別名（若無別名可設定 null）

👉 重點說明：

程式第 18 行取得並建立 Fragment 交易機制，動態 Fragment 的所有設置皆必須在 Fragment 交易機制環境下使得設定與修改。

程式第 22 與 27 行於指定 layout 中替換指定的 Fragment 實例。

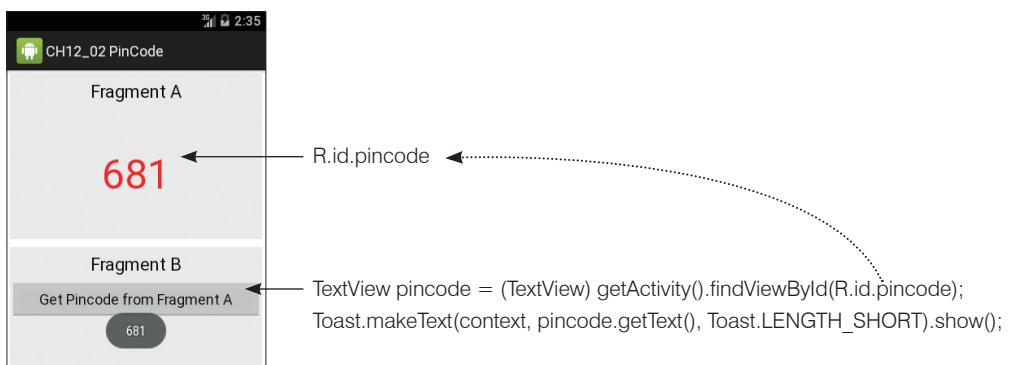
Fragment 動態置入圖如下：



程式第 32 行會將程式第 18 行之後與 32 行之前的所有針對 Fragment 的設定要求提交給主執行緒執行，若主執行緒忙碌則會將此提交作業放入到主執行緒的排程中待消化。

12-2-3 Fragments 間分享視圖資訊

本章將討論在同一個 Activity 下進行 Fragments 間的視圖 (view) 資訊分享。Fragments 可以說是 Activity 畫面佈局中的子項，因此若要達到 Fragments 間可以相互分享與存取視圖資訊就必須透過 Activity 來做為橋接，我們知道在 Fragment 中可以透過 `getView().findViewById()` 來宣告本身 Fragment 的視圖，若要在 Fragment 中取得其他 Fragment 的視圖則需提高層級到 Activity 的觀點並透過 `getActivity().findViewById()` 來實現。以下針對範例程式 CH12_2_PinCode 來做說明：Fragment A 內含 TextView 視圖 (`R.id.pincode`) 用來存放與顯示每 3 秒更換一次的 pincode 碼。Fragment B 內含 Button 視圖用來抓取 Fragment A 中的 TextView 視圖 (`R.id.pincode`) 內容值 (pincode 碼)。



於 Fragment B 中利用 `getActivity()` 來取得 Fragment A 的 TextView 視圖 (`R.id.pincode`) 實例，參考語法如下：

```
TextView pincode = (TextView) getActivity().findViewById(R.id.pincode);
```

所以於 Fragment 中可以利用 `getActivity().findViewById()` 實現在同一個 Activity 中跨域取得各 fragments 的視圖實例。

其他更多請參考範例程式：[CH12_02_PinCode](#)。

12-2-4 FragmentActivity

我們知道 **Fragment** 是 Android 3.0 (API Level 11) 才開始支援的機制，為了要讓 Android 3.0 以前的設備也能享有此資源因而提供了 v4 支援套件 (android-support-v4.jar) 來實現，v4 支援套件另外創建了 **FragmentActivity** 來取代 Android 3.0 (API Level 11) 以前的 Activity 以便得以支援 Fragment，不過也因為如此在實作上就要特別注意，以下就以 Android 3.0 (API Level 11) 作為分水嶺來說明使用 Fragment 時在程式撰寫與宣告上的差異：

Android 3.0 (API Level 11) (含) 以上：

```
import android.app.Fragment;

public class MainActivity extends Activity {
    // Block of code ...
}
```

Android 3.0 (API Level 11) 以下：

```
import android.support.v4.app.Fragment;

public class MainActivity extends FragmentActivity {
    // Block of code ...
}
```

結論請參考下表：

Android 3.0 (API Level 11) (含) 以上	Android 3.0 (API Level 11) 以下
import android.app.Fragment 並於 Activity 中實現 所使用的 Fragment 類別不可以 import 自 android.support.v4.app.Fragment 否則將會導致執行時期錯誤	import android.support.v4.app.Fragment 並於 FragmentActivity 中實現

當然若要讓 Android 1.6 (API Level 4) 含以上皆能支援 Fragment 可以統一使用以下程式撰寫方式與宣告：

```
import android.support.v4.app.Fragment;

public class MainActivity extends FragmentActivity {
    // Block of code ...
}
```

12-2-5 FragmentManager (Fragment 管理器)

要管理現存的 Fragment 可以使用 FragmentManager，包含：取得 Activity 上的 Fragments、BackStack 退線堆疊管理與監聽 ... 等。

👉 範例程式 (CH12_02_Lotto) 相關 Java API

```
public abstract Fragment findFragmentByTag(String tag);
public abstract Fragment findFragmentById(int id);
```

根據 fragment 當初給定的 tag 名稱與資源 id 來查找指定 fragment。
findFragmentByTag() 方法多用在動態 Fragment 佈局使用，findFragmentById() 則多用在靜態 Fragment 佈局使用（於靜態 fragment 佈局可以設定 <android:id> ）。

範例，動態佈局使用 findFragmentByTag：

```
ft.add(R.id.layout, new LottoFragment(), "f_lotto");
...
Fragment fragment = fm.findFragmentByTag("f_lotto");
```

```
public abstract void popBackStack();
public abstract void popBackStack(String name, int flags);
```

移除（彈出 =pop）退線堆疊內的 Fragment。

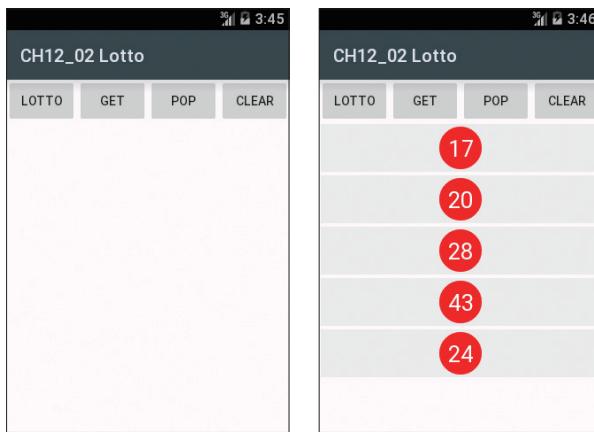
參數 name 會尋找 Fragment 當初透過 addBackStack(別名) 加入至退線堆疊中的別名並彈出，若 name=null 則會彈出目前退線堆疊中最上層的 Fragment。

參數 flags 可以置入 0 或 POP_BACK_STACK_INCLUSIVE，注意：若此參數是設為 POP_BACK_STACK_INCLUSIVE 不論 name 參數是否有給指定值或 null，一律清空退線堆疊內的所有資訊。

範例程式 (CH12_02_Lotto)：Lotto 數字

執行結果：

- 【LOTTO】鍵：產出 fragment 並顯示樂透數字。
- 【Get】鍵：取得指定 fragment 並 Toast 顯示樂透數字。
- 【POP】鍵：移除退線堆疊中最上層的 fragment。
- 【CLEAR】鍵：移除退線堆疊中所有 fragments。



重點程式碼：

```
CH12_02_Lotto/com.ch12.MainActivity.java
```

```
.. ...
23     public void onClick(View view) {
24         // 取得 Fragment 管理物件
25         FragmentManager fm = getFragmentManager();
26         FragmentTransaction ft = fm.beginTransaction();
27         switch (view.getId()) {
28             case R.id.lotto:
29                 ft.add(R.id.fragment_addin_linearlayout, new LottoFragment(),
30                         "f_lotto");
31                 ft.addToBackStack(null);
32                 break;
33             case R.id.get:
34                 Fragment fragment = fm.findFragmentByTag("f_lotto");
35                 if (fragment != null) {
```

↑ Fragment tag 名稱

```
36     TextView lotto_num = (TextView) fragment.getView()  
37         .findViewById(R.id.lotto_num);  
38     Toast.makeText(context, lotto_num.getText().toString(),  
39                     Toast.LENGTH_SHORT).show();  
40     }  
41     break;  
42 case R.id.pop:  
43     fm.popBackStack();  
44     break;  
45 case R.id.clear:  
46     fm.popBackStack(null,  
47                     FragmentManager.POP_BACK_STACK_INCLUSIVE);  
48     break;  
49 }  
50 // 提交  
51 ft.commit();  
52 }  
... ...
```

👉 重點說明：

程式第 25 行取得 `FragmentManager` 管理者物件。

程式第 31 行將 `Fragment` 置入退線堆疊。

程式第 34 行調用 `findFragmentByTag()` 於 `Activity` 中找到指定 `Fragment`。

程式第 43 行移除在退線堆疊中最上層的 `Fragment`。

程式第 46~47 行透過 `POP_BACK_STACK_INCLUSIVE` 參數清空退線堆疊內所有 `Fragments`。

12-3 使用內建 Fragment UI

Android 提供許多內建的 Fragment 方便程式開發者使用，常見的有：

- DialogFragment
- ListFragment
- WebViewFragment
- PreferenceFragment
- MapFragment 等…

以下就針對最簡單的 DialogFragment 來做介紹並說明傳值方法。DialogFragment 是一個浮動在 Activity 上的對話框，它有著與 Dialog 相同的功能，唯一最大的差別就是 DialogFragment 有著 Fragment 的生命週期可以獨立存在於任何一個 Activity 並且共用共享資源，另外 DialogFragment 可以使用 Handler 物件設計出複雜的 Dialog 功能（如果願意的話），雖然這些功能也可以用傳統的 Dialog 自行實作出，但額外撰寫的程式碼就必須多很多，這應該也算是使用 DialogFragment 的優勢吧。

DialogFragment 在傳值上可分為二種方式：

1. 建立 Bundle 物件：將要傳的內容透過 Bundle 物件封裝再調用 getArguments() 方法傳遞參數，接收方 DialogFragment 調用 getArguments() 取得 Bundle 物件。
2. 設定 tag：調用 show(FragmentManager manager, String tag) 方法時除了顯示 DialogFragment 對話框之外也可以透過第二個參數 tag 來將內容傳給 DialogFragment，接收方 DialogFragment 則可調用 getTag() 取得字串內容。值得注意一點的就是這個 tag 實質是指 DialogFragment 在被 add() 調用時的別名，所以若要當參數傳遞時請注意此參數真正的含義，請參考源碼：

```
android.app.DialogFragment.java
...
225     ...
226     mDismissed = false;
227     mShownByMe = true;
228     FragmentTransaction ft = manager.beginTransaction();
```

```
229     ft.add(this, tag);
230     ft.commit();
231 }
...
...
```

👉 範例程式 (CH12_03_DialogFragment) 相關 Java API

android.app.DialogFragment

自定 DialogFragment 時需繼承，語法如下：

```
public class MyDialogFragment extends DialogFragment {
    ...
}
```

public void show(FragmentManager manager, String tag)

顯示 DialogFragment。

參數說明：

- manager：FragmentManager 物件。
- tag：DialogFragment 別名設置（這個 tag 其實是指 DialogFragment 在被 add() 調用時的別名）。

public void setArguments(Bundle args)

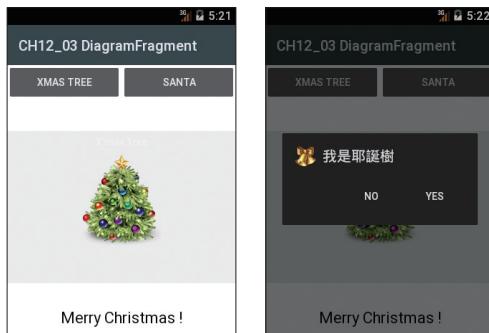
設置參數 Bundle。

public final Bundle getArguments()

取得參數 Bundle。

👉 範例程式 (CH12_03_DialogFragment)：Fragment 對話框

執行結果：



重點程式碼：

```
CH12_03_DialogFragment/com.ch12. MyDialogFragment.java
...
09     public class MyDialogFragment extends DialogFragment {
10
11     @Override
12     public Dialog onCreateDialog(Bundle savedInstanceState) {
13
14         String title = getArguments().getString("title"); ← 取得參數
15
16         AlertDialog.Builder dialog = new AlertDialog.Builder(getActivity());
17         dialog.setIcon(R.drawable.bells);
18         dialog.setTitle(title);
19         dialog.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
20             public void onClick(DialogInterface dialog, int whichButton) {
21                 // dosomething
22             }
23         });
24         dialog.setNegativeButton("No", new DialogInterface.OnClickListener() {
25             public void onClick(DialogInterface dialog, int whichButton) {
26                 // dosomething
27             }
28         });
29
30         return dialog.create(); ← 根據 AlertDialog.Builder 所設置的
31                                         參數來創建 AlertDialog
32
33     }
}
```

重點說明：

程式第 14 行取得參數 Bundle。

程式第 16~28 行利用 AlertDialog.Builder 設置 AlertDialog 相關參數。

程式第 30 行根據 AlertDialog.Builder 所設置的參數來創建 AlertDialog。

CH12_03_DialogFragment/com.ch12. MyFragmentA.java

```
.. ...
42 @Override
43 public void onActivityCreated(Bundle savedInstanceState) {
44     super.onActivityCreated(savedInstanceState);
45
46     // 透過 getView() 取得 Fragment 在 onCreateView() 方法中所定義的 Layout
47     textView1 = (TextView)getView().findViewById(R.id.textView1);
48     textView1.setText("X'max Tree");
49     imageView1 = (ImageView)getView().findViewById(R.id.imageView1);
50     imageView1.setImageResource(R.drawable.tree);
51
52     // 設定 DialogFragment
53     imageView1.setClickable(true);
54     imageView1.setOnClickListener(new OnClickListener() {
55         @Override
56         public void onClick(View view) {
57             // 建立 MyDialogFragment
58             MyDialogFragment dialogfragment = new MyDialogFragment();
59             // 建立 Bundle
60             Bundle args = new Bundle();
61             args.putString("title", "我是聖誕樹");
62             dialogfragment.setArguments(args);
63             // 顯示 MyDialogFragment
64             dialogfragment.show(getFragmentManager(), "dialog");
65         }
66     });
67 }
```

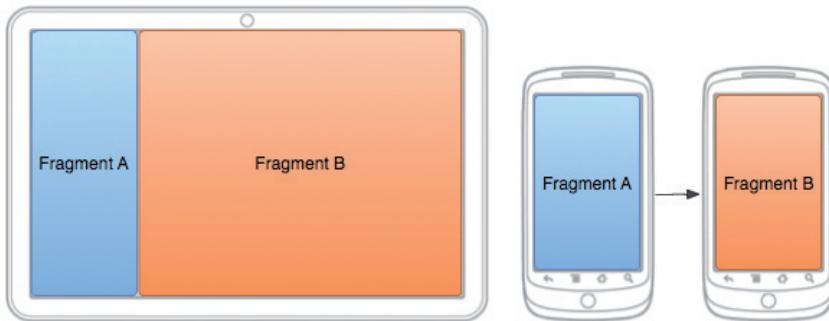
 重點說明：

程式第 62 行設置參數 Bundle。

程式第 64 行顯示 Dialog 對話框。

12-4 建構靈活的 UI 界面

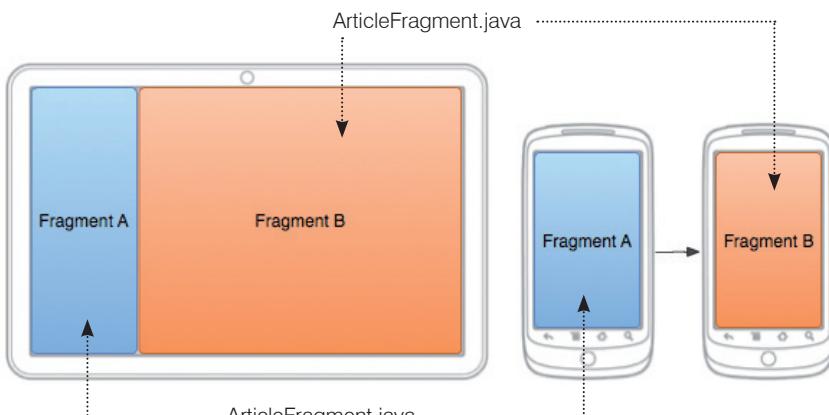
由 12-2、12-3 的章節當中學習到了 Fragment 靜態、動態的建制與內建 Fragment 的用法，接下來我們來做一些 Fragment 的特效變化讓我們於不同螢幕大小在 UI 的呈現上能夠有彈性，請參考下圖：



Tablet 與 Handset UI 上的佈局

由上圖可知在手持行動裝置（Handset）中畫面佈局是採用 FragmentA 與 FragmentB 交互替換達成的 UI 顯示效果，而在平板裝置（Tablet）中由於螢幕較大 FragmentA 與 FragmentB 將會一同呈現。CH12_04_Flexible 範例將採用 android-support-v4.jar 套件完成此項需求任務。

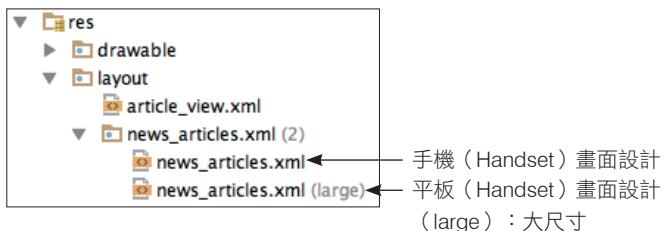
範例程式（CH12_04_Flexible）Fragment UI 畫面與 java 程式對應關係簡圖：



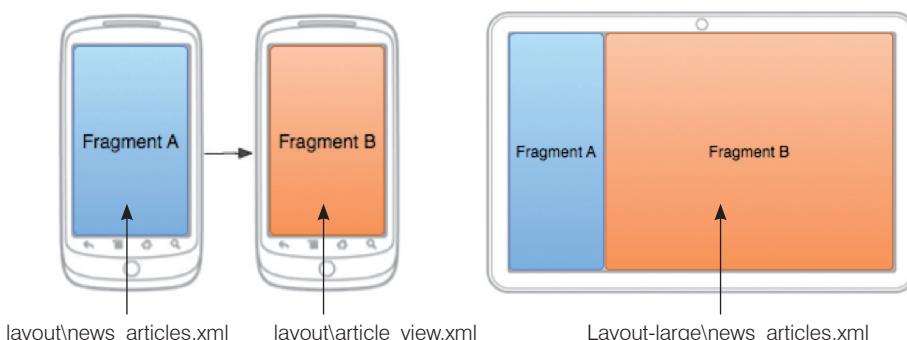
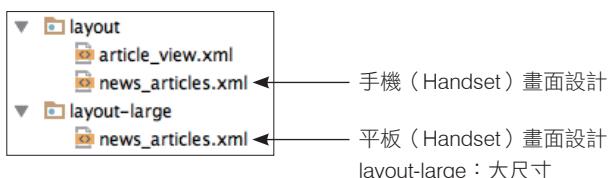
Fragment UI 與 Java 程式對應關係圖

UI 界面部署與設計

Android 專案環境下的呈現：



Packages 專案環境下的呈現：



手机 XML 佈局檔

CH12_04_Flexible/res/layout/news_articles.xml

01	<FrameLayout
02	xmlns:android="http://schemas.android.com/apk/res/android"
03	android:id="@+id/fragment_container"
04	android:layout_width="match_parent"
05	android:layout_height="match_parent" />

CH12_04_Flexible/res/layout/article_view.xml

```
01 <TextView  
02     xmlns:android="http://schemas.android.com/apk/res/android"  
03     android:id="@+id/article"  
04     android:layout_width="match_parent"  
05     android:layout_height="match_parent"  
06     android:padding="16dp"  
07     android:textSize="18sp" />
```

👉 平板 XML 佈局檔

CH12_04_Flexible/res/layout-large/new_articles.xml

```
01 <LinearLayout  
02     xmlns:android="http://schemas.android.com/apk/res/android"  
03     android:orientation="horizontal"  
04     android:baselineAligned="false" ←—— 關閉 baseline 對齊  
05     android:layout_width="match_parent"  
06     android:layout_height="match_parent">  
07  
08     <fragment android:name="com.ch12.HeadlinesFragment"  
09         android:id="@+id/headlines_fragment"  
10         android:layout_weight="1"  
11         android:layout_width="0dp"  
12         android:layout_height="match_parent" />  
13  
14     <fragment android:name="com.ch12.ArticleFragment"  
15         android:id="@+id/article_fragment"  
16         android:layout_weight="2"  
17         android:layout_width="0dp"  
18         android:layout_height="match_parent" />  
19  
20 </LinearLayout>
```

👉 重點說明：

在 `<LinearLayout>` 標籤中的子組件若有使用 `android:layout-weight` 屬性，可以於 `<LinearLayout>` 標籤屬性中設定 `android:baselineAligned="false"` 關閉子組件的基線 (baseline) 對齊，因為在線性佈局 `Linearlayout` 中預設所有子組件都會被排列對齊到本文的基線位置，但若是依權重來佈局子組件則應關閉此設置以提高佈局時的效率，參考的 XML 結構如下所示：

```
<linearLayout
    android:baselineAligned="false"
    ...
    >
<fragment
    android:layout-weight=" 1"
    ...
    />
<fragment
    android:layout-weight=" 2"
    ...
    />
</linearLayout>
```

👉 範例程式 (CH12_04_Flexible) 相關 Java API

android.support.v4.app.ListFragment;

就如同 `ListActivity` 一樣 `ListFragment` 也是一種 `ListView` 的載體，可以顯示條列項目並提供點選事件。由範例程式 (CH12_04_Flexible) 可知：

`HeadlinesFragment.java` 是一個 `ListFragment`，宣告如下：

```
public class HeadlinesFragment extends ListFragment { ←—— 支援 ListView 的 Fragment
}
```

而 `ArticleFragment.java` 是一個標準的 `Fragment`，宣告如下：

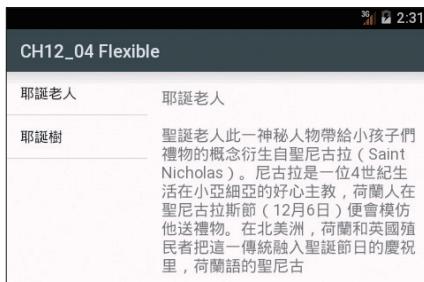
```
public class ArticleFragment extends Fragment {
}
```

範例程式 (CH12_04_Flexible)：手機與平板彈性佈局設計

執行結果：



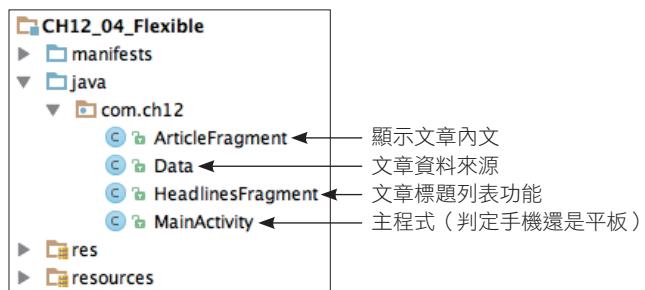
手機執行畫面



平板執行畫面

重點程式碼：

程式碼部署與功能說明：



主程式

```
CH12_04_Flexible/com.ch12.MainActivity.java
...
09     ...
10    @Override
11    public void onCreate(Bundle savedInstanceState) {
12        super.onCreate(savedInstanceState);
13        setContentView(R.layout.news_articles);
14
15        FragmentManager fm = getSupportFragmentManager();
16        fm.popBackStack(null, FragmentManager.POP_BACK_STACK_INCLUSIVE);
17
18        if (findViewById(R.id.fragment_container) != null) {
19            HeadlinesFragment headlinsFragment = new HeadlinesFragment();
20            getSupportFragmentManager().beginTransaction()
21                .add(R.id.fragment_container, headlinsFragment).commit();
22        }
23    }
...
```

重點說明：

程式第 14~15 行先清除所有退線堆疊中的資料保持執行此程式的潔淨。

程式第 17 行判斷當下得到的 R.layout.news_articles 是否含有 R.id.fragment_container 藉以判定執行到的是 layout\news_articles 還是 layout-large\news_articles 就可以知道使用者是使用手機還是平板來執行此程式，請參考下圖：

```
1 <FrameLayout
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/fragment_container" ← 只有手機的佈局
4     android:layout_width="match_parent"
5     android:layout_height="match_parent" />
```

手機：layout\news_articles.xml

```

1 <LinearLayout
2     xmlns:android="http://schemas.android.com/apk/res/android"
3         android:orientation="horizontal"
4         android:baselineAligned="false"
5         android:layout_width="match_parent"
6         android:layout_height="match_parent">
7
8     <fragment android:name="com.ch12.HeadlinesFragment"
9             android:id="@+id/headlines_fragment"
10            android:layout_weight="1"
11            android:layout_width="0dp"
12            android:layout_height="match_parent" />
13
14     <fragment android:name="com.ch12.ArticleFragment"
15             android:id="@+id/article_fragment"
16             android:layout_weight="2"
17             android:layout_width="0dp"
18             android:layout_height="match_parent" />
19
20 </LinearLayout>

```

平板 : layout-large\news_articles.xml

文章資料來源

CH12_04_Flexible/com.ch12.Data.java

```

.. ...
03 public class Data {
04
05     static String[] Headlines = {
06         " 耶誕老人 ",
07         " 耶誕樹 "
08     };
09
10    static String[] Articles = {
11        " 耶誕老人 \n\n 聖誕老人此一神秘人物帶著給小孩子們禮物的概念衍生自 ... 。",
12        " 耶誕樹 \n\n 聖誕樹是聖誕節慶祝中最有名的傳統之一。通常人們在聖 ... 。"
13    };
14 }

```

重點說明 :

Data.Headlines 資料是給用於列表顯示的 **HeadlinesFragment.java** 使用。

Data.Articles 資料是給用於文章內容顯示的 **ArticleFragment.java** 使用。

 文章標題列表 Fragment

```
CH12_04_Flexible/com.ch12.HeadlinesFragment.java
...
25     ...
26     public class HeadlinesFragment extends ListFragment {
27
28         @Override
29         public void onCreate(Bundle savedInstanceState) {
30             super.onCreate(savedInstanceState);
31             setListAdapter(new ArrayAdapter<String>(getActivity(),
32                 android.R.layout.simple_list_item_1, Data.Headlines));
33         }
34
35         @Override
36         public void onStart() {
37             super.onStart();
38
39             if (getFragmentManager()
40                 .findFragmentById(R.id.article_fragment) != null) {
41                 getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
42             }
43
44         @Override
45         public void onListItemClick(ListView l, View v, int position, long id) {
46             // highlighted
47             getListView().setItemChecked(position, true);
48
49             ArticleFragment articleFragment = (ArticleFragment) getActivity()
50                 .getSupportFragmentManager()
51                 .findFragmentById(R.id.article_fragment);
52
53             if (articleFragment == null) {
54                 articleFragment = new ArticleFragment();           設置參數 position 讓
55                 Bundle args = new Bundle();                      articleFragment 根據 position
56                 args.putInt("position", position);               內容值顯示對應的文章內容
57                 articleFragment.setArguments(args);              ←
58                 FragmentTransaction transaction = getActivity()
59                     .getSupportFragmentManager().beginTransaction();
60
61                 transaction.replace(R.id.fragment_container, articleFragment);
62                 transaction.addToBackStack(null);
63
64                 transaction.commit();
65             }
```

```
66     } else {
67         articleFragment.updateArticleView(position);
68     }
69 }
70 }
```

👉 重點說明：

程式第 25 行由於 HeadlinesFragment 是一個支援 ListView 的 Fragment 所以需繼承 ListFragment。

程式第 30、31 行針對 ListFragment 進行內容適配器的設定，資料來源請參考 Data.java。

程式第 38~41 行於 onStart() 中設定 ListFragment 為單選點擊。

程式第 47 行設定 High Light 顯示模式，使用者點擊 ListFragment 項目後該項目會呈現反白效果讓使用者清楚明白所點擊的項目。

程式第 53 行判斷執行此 ListFragment 的當下是否有 articleFragment 物件（由程式第 49~51 宣告而來），若 articleFragment == null 表示是手機執行此程式反之則為平板執行此程式，請參考下圖：

```
1 <FrameLayout
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/fragment_container"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent" />
```

手機 : layout\news_articles.xml

```
1 <LinearLayout
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="horizontal"
4     android:baselineAligned="false"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent">
7
8     <fragment android:name="com.ch12.HeadlinesFragment"
9         android:id="@+id/headlines_fragment"
10        android:layout_weight="1"
11        android:layout_width="0dp"
12        android:layout_height="match_parent" />
13
14     <fragment android:name="com.ch12.ArticleFragment"
15         android:id="@+id/article_fragment" ←
16        android:layout_weight="2"
17        android:layout_width="0dp"
18        android:layout_height="match_parent" />
19
20 </LinearLayout>
```

只有平板的佈局
UI 才有

平板 : layout-large\news_articles.xml

程式第 54~64 行此為手機環境下執行邏輯。調用 `transaction.replace(R.id.fragment_container, articleFragment);` 讓 `articleFragment` 部署到 `R.id.fragment_container` 容器，值得注意的是程式第 62 行 `transaction.addToBackStack(null)` 的調用將 `fragment` 加入到退線堆疊中好讓使用者得以透過返回鍵回朔之前所點選過的文章。

程式第 67 行此為平板環境下執行邏輯。因為此環境已有 `articleFragment` 物件可以使用，所以直接調用 `updateArticleView()` 方法並根據 `position` 更換指定文章內容即可。

文章內容顯示 Fragment

CH12_04_Flexible/com.ch12.ArticleFragment.java

```
..  
10    ...  
11  
12    @Override  
13    public View onCreateView(LayoutInflater inflater,  
14        ViewGroup container, Bundle savedInstanceState) {  
15        return inflater.inflate(R.layout.article_view, container, false);  
16    }  
17  
18    @Override  
19    public void onStart() {  
20        super.onStart();  
21        Bundle args = getArguments();  
22        if (args != null) {  
23            updateArticleView(args.getInt("position"));  
24        }  
25    }  
26  
27    public void updateArticleView(int position) {  
28        TextView article = (TextView) getActivity()  
29            .findViewById(R.id.article);  
30        article.setText(Data.Articles[position]);  
31    }  
32  
33 }
```

重點說明：

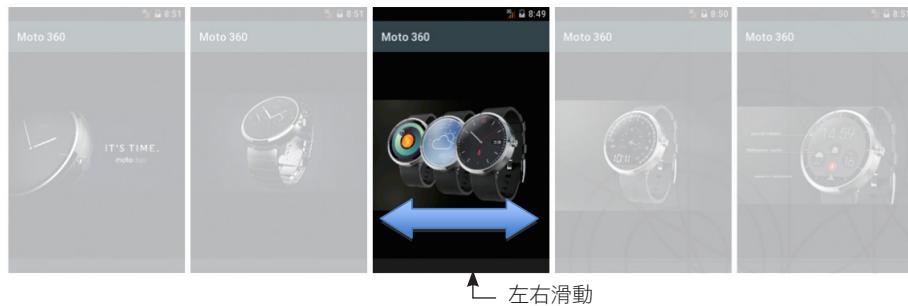
程式第 19~24 行於 `onStart()` 中接取參數 `position` 並調用 `updateArticleView()` 方法來更新文章內容。

程式第 27~31 行根據 `position` 參數更新文章內容。

其他更多在 `fragment` 上有彈性的設計包含 `Tablet` 與 `Handset` 互搭界面設計與應用請參考 <http://developer.android.com/training/basics/fragments/fragment-ui.html> (搜尋關鍵字：Building a Flexible UI)。

12-5 分頁指示器 ViewPagerIndicator 與傳值

`ViewPager` 是 Android 內建提供的分頁瀏覽功能，它的特性就是資訊內容以頁 (page) 為單位，好讓使用者可以左右滑動螢幕來導覽訊息。

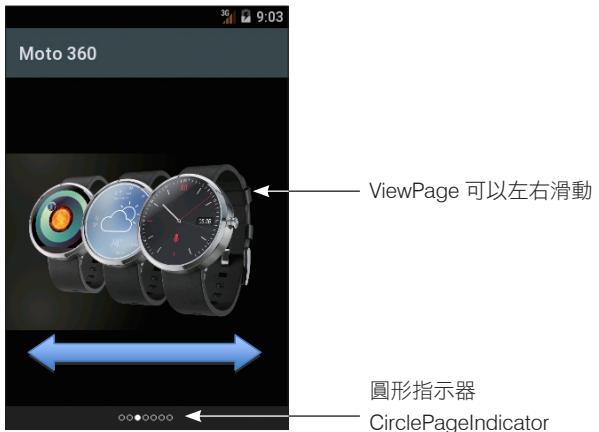


圓形指示器 (CirclePageIndicator) 執行範例圖

不過它有個小缺點，就是使用者往往在瀏覽時因為左右滑動而容易失去導覽順序與方向，不曉得目前到底是看到第幾頁？以及還有幾頁？索性有位工程師名叫 **JakeWharton** 解決此問題，除了可以跟 Android 內建的 `PageView` 完美結合外（不用改之前所撰寫的程式碼）並且還把它開源造福許多 Android 開發者，這個第三方解決方案就是 `ViewPagerIndicator`。

- 官方網站：<http://viewpagerindicator.com/>。
- 下載網址：<https://github.com/JakeWharton/Android-ViewPagerIndicator>。
- JakeWharton 其他開源：<https://github.com/JakeWharton>。

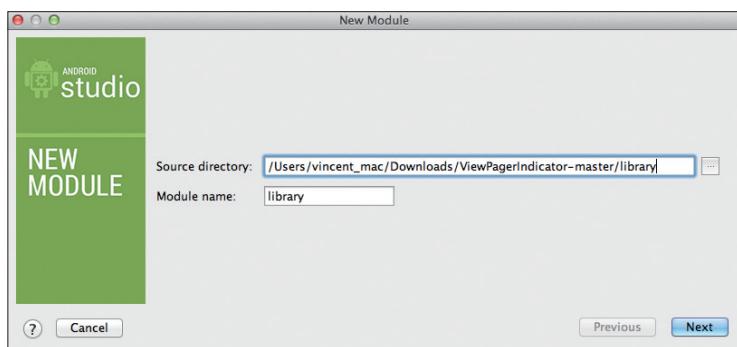
下圖是一個 ViewPagerIndicator 圓形指示器 (CirclePageIndicator) 執行範例圖：



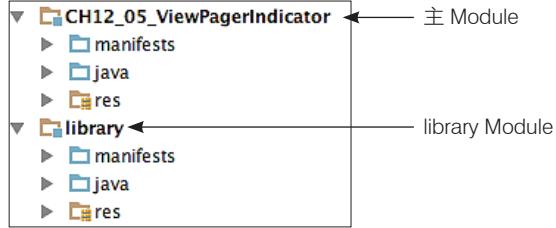
有了指示器，使用者就可以很清楚知道目前瀏覽進度與所在位置加強 App 貼心設計。以下就以範例 CH12_05_ViewPagerAdapter 來說明實作方式與步驟：

範例程式（CH12_05_ViewPagerAdapter）ViewPagerIndicator 第三方資源 套件下載與安裝

- 下載 ViewPagerIndicator-master.zip > 解壓。
- 下載網址：<https://github.com/JakeWharton/Android-ViewPagerIndicator>。
- 解壓縮：解壓縮下載檔 Android-ViewPagerIndicator-master.zip。
- 於 Android Studio 中匯入 Module...。
- File > New > Import Module...。
- 匯入路徑：/ViewPagerIndicator-master/library。



修改範例程式 (CH12_05_ViewPagerAdapter) build.gradle

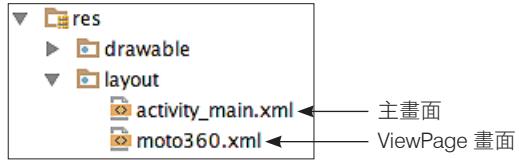


修改 build.gradle (Module : CH12_05_ViewPagerAdapterIndicator)

```
22 dependencies {  
23     compile fileTree(dir: 'libs', include: ['*.jar'])  
24     compile 'com.android.support:appcompat-v7:22.1.1'  
25     compile project(':library')  
26 }
```

將 library Module 汇入編譯

UI 界面部署與設計



res\layout\main.xml

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
2     xmlns:tools="http://schemas.android.com/tools"  
3     android:layout_width="fill_parent"  
4     android:layout_height="fill_parent"  
5     android:orientation="vertical"  
6     tools:context=".MainFragmentActivity" >  
7  
8     <android.support.v4.view.ViewPager  
9         android:id="@+id/pager"  
10        android:layout_width="fill_parent"  
11        android:layout_height="0dp"  
12        android:layout_weight="1" />  
13  
14     <com.viewpagerindicator.CirclePageIndicator  
15         android:id="@+id/indicator"  
16         android:background="#FF555555"  
17         android:layout_width="fill_parent"  
18         android:layout_height="wrap_content"  
19         android:padding="10dp" />  
20  
21 </LinearLayout>
```

res\layout\moto360.xml

```
1 <LinearLayout  
2     xmlns:android="http://schemas.android.com/apk/res/android"  
3     android:orientation="vertical"  
4     android:layout_width="fill_parent"  
5     android:layout_height="fill_parent"  
6     android:background="#FF000000" >  
7  
8     <ImageView  
9         android:id="@+id/imageView1"  
10        android:layout_width="wrap_content"  
11        android:layout_height="wrap_content"  
12        android:src="@drawable/moto360_1" />  
13  
14 </LinearLayout>
```

👉 範例程式（CH12_05_ViewPagerAdapter）相關 Java API

```
public class android.support.v4.view.ViewPager extends android.view.ViewGroup
```

`ViewPager` 是一個界面佈局管理器，提供左右滑動功能瀏覽頁面資訊，並提供 `PagerAdapter` 專屬適配器來產生多樣的圖文資訊。

```
public class CirclePageIndicator extends View implements PageIndicator
```

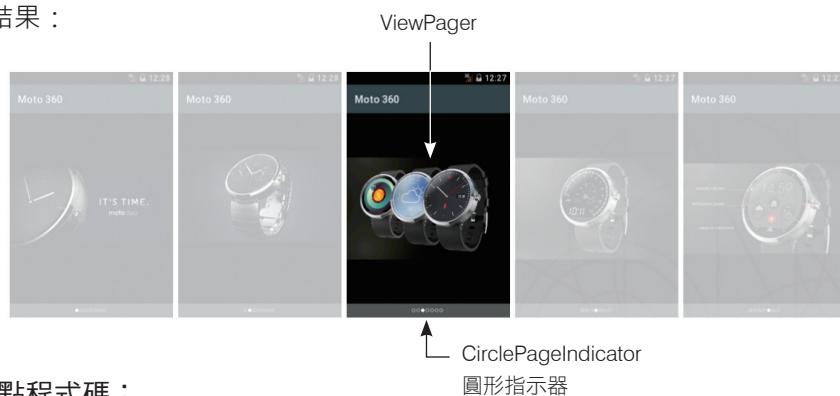
`CirclePageIndicator` 是 `Android-ViewPagerIndicator` 第三方資源套件所提供的圓形指示器讓使用者可以知道目前翻閱 `Page` 的進度。

```
public void setViewPager(ViewPager view);
```

`Android-ViewPagerIndicator` 第三方資源套件針對每一個指示器（例如：`CirclePageIndicator`）都提供了 `setViewPager()` 方法讓 `ViewPager` 可以輕易置入並自動產生指示功能。

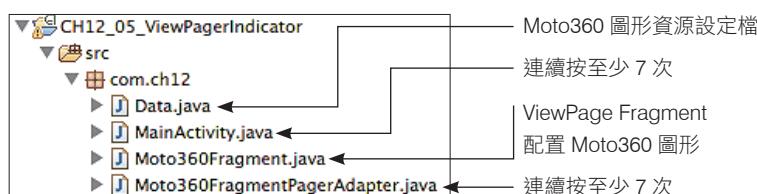
👉 範例程式（CH12_05_ViewPagerAdapter）：建立含有指示器的 PageView

執行結果：



👉 重點程式碼：

程式碼部署與功能說明：



CH12_05_ViewPagerAdapterIndicator/com.ch12.MainActivity.java

```
..  
09     ...  
10     public class MainActivity extends FragmentActivity {  
11         private Moto360FragmentPagerAdapter mAdapter;  
12         private ViewPager mPager;  
13         private PageIndicator mIndicator;  
14  
15     @Override  
16     protected void onCreate(Bundle savedInstanceState) {  
17         super.onCreate(savedInstanceState);  
18         setContentView(R.layout.main);  
19  
20         mAdapter = new Moto360FragmentPagerAdapter( ←—— ViewPager 適配器  
21             getSupportFragmentManager(), Data.MOTO360);  
22  
23         mPager = (ViewPager)findViewById(R.id.pager);  
24         mPager.setAdapter(mAdapter); ↑ 建立 ViewPager  
25  
26         mIndicator = (CirclePageIndicator)findViewById(R.id.indicator);  
27  
28         mIndicator.setViewPager(mPager); ↑ 建立圓形指示器  
29     }  
30 }
```

↑ 圓形指示器加入至 ViewPager

👉 重點說明：

程式第 26 行建立圓形指示器。

程式第 28 行調用 `setViewPager()` 方法將圓形指示器加入至 `ViewPager`。

CH12_05_ViewPagerAdapterIndicator/com.ch12.Moto360Fragment.java

```
..  
10     ...  
11     public class Moto360Fragment extends Fragment {  
12         public Moto360Fragment() {  
13         }  
14  
15     @Override  
16     public View onCreateView(LayoutInflater inflater,
```

```
18     ViewGroup container, Bundle savedInstanceState) {  
19         int resId = getArguments().getInt("resId");  
20         View layout = inflater.inflate(R.layout.moto360, null);  
21         ImageView imageView1 = (ImageView) layout ← 取得圖像資源參數  
22             .findViewById(R.id.imageView1);  
23         imageView1.setImageResource(resId);  
24  
25         return layout;  
26     }  
27  
28 }
```

👉 重點說明：

程式第 19 取得圖像資源參數 `resId`。

程式第 23 將圖像資源 `resId` 設定到指定 `ImageView` 當中。

CH12_05_ViewPagerAdapter/com.ch12. Moto360FragmentPagerAdapter.java

```
..  
07     ...  
08  
09     class Moto360FragmentPagerAdapter extends FragmentPagerAdapter {  
10  
11         private int[] imageResIds;  
12  
13         public Moto360FragmentPagerAdapter(FragmentManager fm,  
14             int[] imageResIds) {  
15             super(fm);  
16             this.imageResIds = imageResIds;  
17         }  
18         @Override  
19         public Fragment getItem(int position) {  
20             int resId = imageResIds[position];  
21             Fragment fragment = new Moto360Fragment();  
22             Bundle bundle = new Bundle();  
23             bundle.putInt("resId", resId);  
24             fragment.setArguments(bundle); ← 將指定圖像資源傳入參數  
25             return fragment;          (Fragment 傳值)  
26         }
```

```
27  
28     @Override  
29     public int getCount() {  
30         return imageResIds.length;  
31     }  
32 }  
33 }
```

重點說明：

程式第 18~26 行於 `getItem()` 方法中根據 `position` 取得對應的圖像資源並建立 `Moto360Fragment` 實例，以便讓 `Moto360Fragment` 得以將指定圖像顯示。

`Fragment` 的出現讓 `Android` 在 `UI` 佈局思維上產生重大的變化，要創造出高彈性與好維護的 `UI` 介面與元件重複使用，就必須仰賴工程師對於 `Fragment` 的特性是否熟悉與到位。

注意根據 `Fragment` 官方 API 建議，`Fragment` 傳遞參數不建議使用傳統建構式方式傳遞，而要使用 `setArguments(Bundle)` / `getArguments()` 來設定傳遞與取得參數。以便該參數值得以在 `Fragment` 生命週期中持續維護。避免若手機翻轉時因生命週期重跑導致參數遺失。