第 2 章

Android 框架與應用類別的接合：卡榫(Hook)函數

2.1 前言

近十年來，應用框架搶走了作業系統(OS, Operation System)的的丰采，成爲軟體的主機板(Software Motherboard)角色，與硬體主機板互相輝映，形成現代電腦系統(包括手機)的主要軟硬體整合架構。君不見：

- 做軟體的 Microsoft 公司花了鉅額經費提供 Windows-based 的.NET 應用框架。
- 做硬體的新國眾公司也花了鉅大心力研發 OpenMOKO 應用框架。
- 做網路服務的 Google 公司也大力推廣 Android 應用框架。

當各種不同型態的公司不約而同地聚焦於應用框架之時，到底大家對應用框架的觀點和看法爲何呢? 觀點與看法會影響你的做法和行爲(包括你對它的用法)，而行爲會影響到最終的結果。爲了讓大家能釐清對 Android 應用框架的觀點，在我寫的第 1 本 Android 書(即<<Google Android 應用框架原理與程式設計 36 技>>)裡，我就以應用框架概念和機制爲出發點，用心闡述應用框架的角色和用法，以及其應用程式的寫法。

此外我也寫了一篇名爲：<<基類與愚公移山>>的文章(請看附錄 A)，提醒大家調整一下觀點，反思一下自己除了扮演愚公之外，或許還有更高尚角色也說不定。在該文章裡，也引用<<禮記.學記篇>>的智慧之言：「良弓之子，必學爲箕」。以中華文化來間接說明爲何上述各種不同型態的公司會不約而同地聚焦於應用框架。

言歸正傳，到底基於何種期待、目的或需求而對應用框架特別青睞呢？其中的關鍵在於：

1. 框住應用程式的『型』(Form)或架構(Architecture)。
2. 以及抓住應用程式的『控制點』(Control)。

其實，這兩點是一體的兩面。唯有抓住控制點才能有效框住應用程式的型。談到這裡，跑出了兩個問題：

- Why, 爲何需要去框住應用程式的型呢? 到底會有多少效益呢?

  關於這一點，我在另一篇文章：<<迎接 IT 第三波：移動時代>>文章(請

看附錄 B)裡，就以 Oracle 資料庫、微軟的 Windows 以及長榮航運公司的貨櫃爲例，仔細闡述其中的道理，請你閱讀之。

- How-to, 採用什麼技術或機制去發揮「框住」的力量呢?
  答案是：IoC (Inversion of Control)機制。這又稱爲「反向呼叫」機制，它讓框架裡的抽象(Abstract)類別能與應用程式裡的具體(Concrete)類別結合起來。

由於許多嵌入式軟體開發者以C語言爲主，而傳統的C語言又沒有提供物件導向(OO, Object-Oriented)機制，因此面對 Android 這種以 OO 技術爲基礎的應用框架時，無法體會 Android 應用框架的特性和角色，而單純地將它視爲比較進步的 OS。其實在一般商業業務溝通上，持著這個觀點是合宜的。但是對於軟硬體開發者來說，就必須對應用框架有更具建設性的觀點，才能獲得最佳的利益。也是基於這個理由，筆者努力撰寫了兩本書，分別從程式(Code)撰寫(即<<*Android 與物件導向技術*>>一書)和設計(Design)樣式(即本書)兩個層面，說明 Android 應用框架成爲萬眾矚目的焦點的背後意義，期盼你能深刻體會 Android 應用框架的特性和有建設性的用法。

因此本書是從設計層面來解釋 Android 在本章，將針對上述的第二項(即How-to)介紹 Android 框架是如何發揮其「框住」或「卡住」應用類別。其依賴 OO 技術裡的意義、特性和設計手藝。

首先，在本章裡，將從最基礎的技術談起，這項基本技術就是用來實現上述的「框住」機制問題。在建築界裡，自從商朝時期，就有了類似的問題，其解決之道就是：「卡榫」。在軟體業裡，其解決之道就是：掛鉤(Hook)函數。在本書裡，將採取「卡榫函數」做爲 Hook 函數的同義辭或譯詞。

卡榫函數是Android框架與應用類別的接合處。Android依賴類別繼承的可覆寫(Overridable)函數來做爲主要的卡榫函數。由於讓框架裡的抽象類別與應用類別能分合自如，是Android框架的核心目標，卡榫函數是認識Android應用框架的起點。

2.2 何謂卡榫(Hook)函數

所謂「卡榫」，就是用來接合兩個東西的接口。如果兩個東西於不同時間出現，則一方會預留虛空，給予另一邊於未來時刻能以實體來填補該空間，兩者虛實相依，就密合起來了。設計優良的卡榫，可以讓實體易於新陳代謝、抽換自如(Plug and Play, 俗稱 PnP)。卡榫函數對於 Android 是非常重要的，因爲 Android 是應用框架(Application Framework, 簡稱 AF)，它與應用程式(Application, 簡稱 AP)是虛與實的兩方，而且兩者是不同時間出現，通常是前者先出現，才能「框住」後者。因此，Android 必須藉由卡榫函數去預留虛空，讓後來的應用程式(AP)來填補該空間，兩者虛實相依，就密合起來了。

由於 Android 是物件導向(Object-Oriented)的應用框架(AF)，其卡榫函數主要是落實爲 C++或 Java 語言的可覆寫函數(Overridable Function)。例如 C++程式：

<<C++範例程式>>

```
//Ex02-01.cpp
#include <iostream>
using namespace std;

class SalesPerson{
    protected:
       double total_amt;
    public:
       SalesPerson( double a ) : total_amt(a) {}
       virtual double bonus() const =0;
    };
class SalesEngineer : public SalesPerson {
       public:
          SalesEngineer(double a):SalesPerson(a) {}
          virtual double bonus() const
             { return( total_amt * 0.008 + 500.0 ); }
    };
class SalesManager : public SalesPerson {
       public:
          SalesManager(double a):SalesPerson(a) {}
          virtual double bonus() const
             { return( total_amt * 0.008 + 2000.0 ); }
```

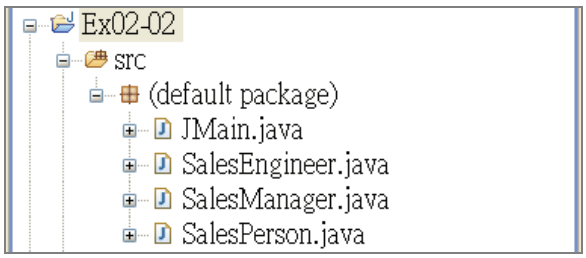
```
    };
void compute_bonus( const SalesPerson *ps )
    {   cout << "bonus:" << ps->bonus() << endl;   }

int main() {
        SalesManager   peter( 20000.0);
        SalesEngineer   alvin( 80000.0), lily( 50000.0 );
        compute_bonus( &peter );   compute_bonus( &alvin );
        compute_bonus( &lily );
        system("PAUSE");
        return EXIT_SUCCESS;
    }
```

其中，SalesPerson 類別的 bonus()虛擬函數就扮演了卡榫的角色，做爲父、子類別之間的接口。由於物件導向的 Android 應用框架非常依賴類別繼承(Class Inheritance)機制，所以像 bonus()這種銜接父、子類別之卡榫函數，在 Android 裡處處可見。再來看看 Java 的範例程式：

<<Java 範例程式>>

Step-1. 建立一個 Java 應用程式專案：Ex02-02。



Step-2. 撰寫各類別的程式碼。

// SalesPerson.java

```
public abstract class SalesPerson {
            protected double total_amt;

            public SalesPerson(double a)
                {   total_amt = a;  }
            public abstract double bonus();
}
```

```
// SalesEngineer.java
public class SalesEngineer extends SalesPerson {
        public SalesEngineer(double a)
                { super(a);    }
        public double bonus() {
                return total_amt * 0.008 + 500.0;
    }
}

// SalesManager.java
public class SalesManager extends SalesPerson {
        public SalesManager(double a)
                { super(a);    }
        public double bonus() {
                return total_amt * 0.008 + 2000.0;
    }
}

// JMain.java
public class JMain {
      static void compute_bonus( SalesPerson ps )
      {    System.out.println("bonus:" + ps.bonus());    }

 public static void main(String[] args) {
      SalesManager peter = new SalesManager(20000);
      SalesEngineer alvin = new SalesEngineer(80000);
      SalesEngineer lily = new SalesEngineer(50000);
       compute_bonus( peter );
       compute_bonus( alvin );
       compute_bonus( lily );
 }
}
```

這 SalesPerson 類別的 bonus()就是卡榫函數。當 SalesPerson 父類別被寫入 AF 裡時，這個虛空的卡榫接口就成爲 AF 與 AP 的銜接機制了。

2.3　Android 的 Hook 函數範例(一)

以上說明了 C++和 Java 如何實現卡榫函數，接下來，看看 Android 裡的卡榫函數之例，在 Android 的螢幕畫面上想呈現出 ListView 視窗時，通常會用到 Android 框架裡的 3 個類別：

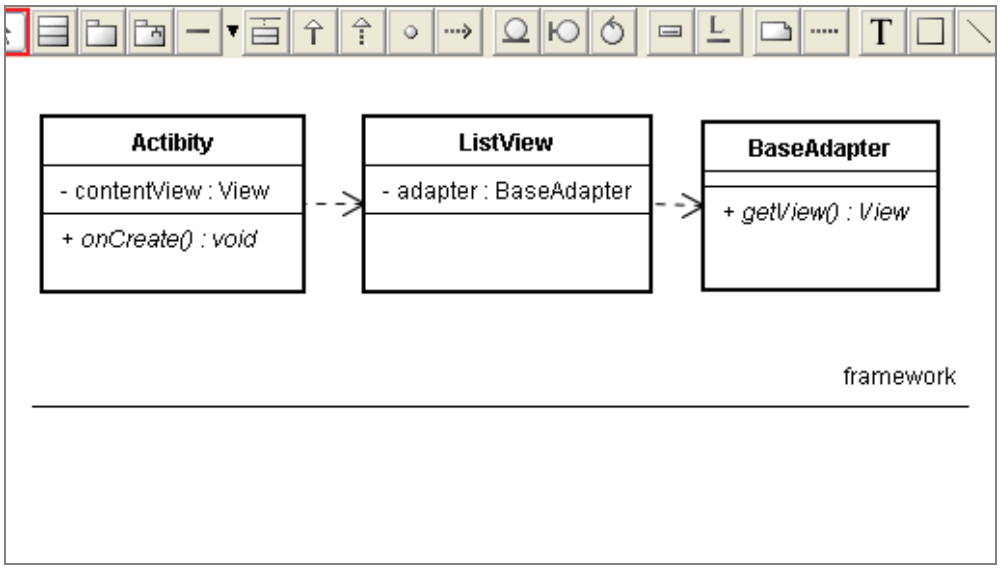


圖 2-1　Android 框架裡的一些抽象類別

這些抽象類別皆含有許多個卡榫函數，做爲與其子類別的銜接口。例如，在 Activity 類別裡，大家熟悉的就是它的 onCreate()卡榫函數，此外，Activity 還含有其他的卡榫函數。同樣地，BaseAdapter 類別裡也含有許多個卡榫函數，而 getView()就是其中之一。

前面說過，框架的抽象類別設計在先，之後才配上應用類別，所以在撰寫 Activity、ListView 和 BaseAdapter 等類別時，其開發者並不知道應用程式裡的 ListView 視窗的內容，所以預留了虛空給應用類別開發者去填補，如下圖：

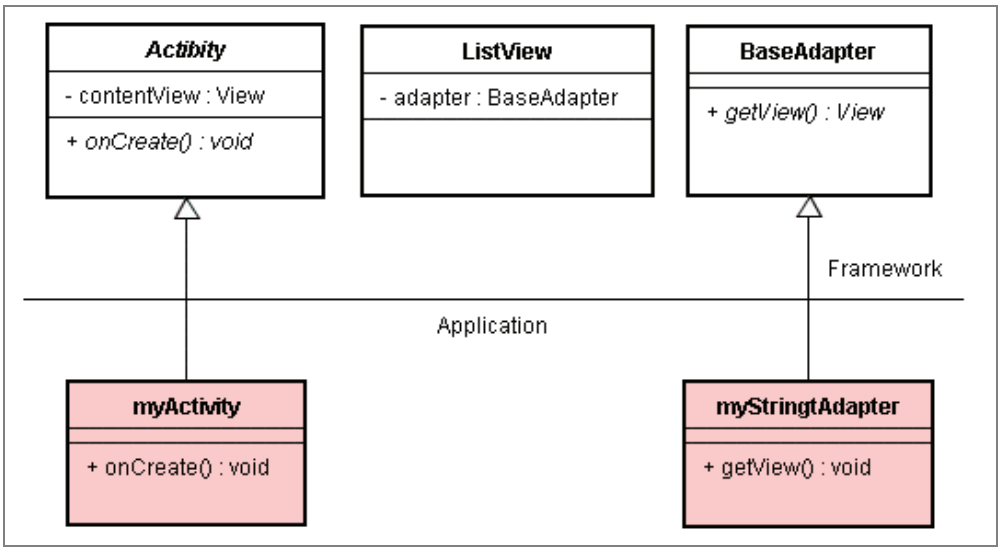


圖 2-2　Android 框架與應用程式的銜接口

當應用程式執行時，框架掌握整個控制權，呼叫 myActivity 子類別的 onCreate()函數，詢問應用類別開發者的見解：

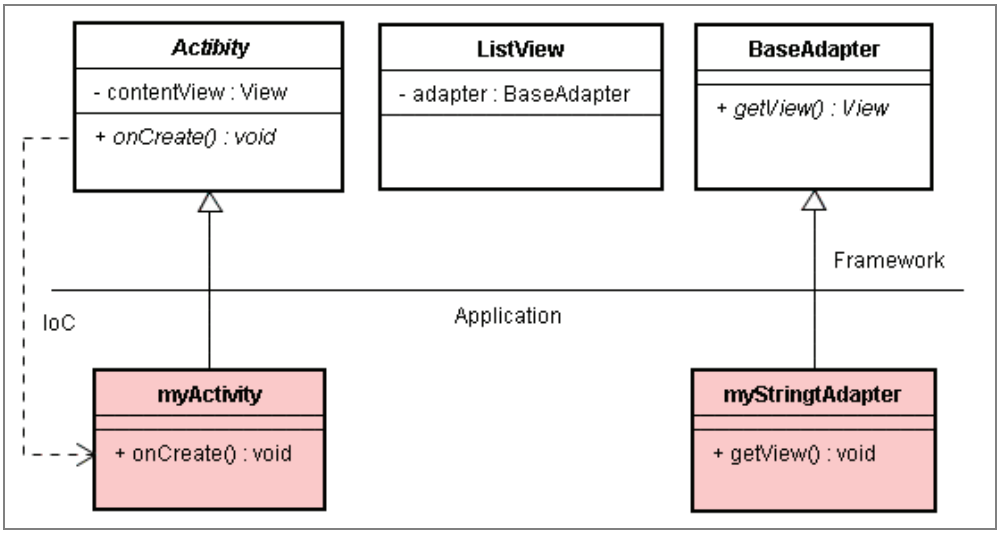


圖 2-3　卡槔函數實現了 IoC(一)

框架設計者與應用類別開發者，兩者在不同的時間及空間(即地點)合作。框架設計者在設計應用框架時並不知道應用程式的 ListView 視窗裡各個選項的長相(文字或是圖像等)，因爲這長相的決定是來自於應用程式的用戶。所以只有等到應用類別開發者動工之刻才會知道，而框架設計者並不知道。於是，框架設計師負責撰寫抽象類別(如 BaseAdapter)，而應用類別開發者負責撰寫子類別(如 myStringAdapter)。兩者跨時間和空間的智慧會合而成爲一支完整好用的手機應用軟體。例如，當 AP 開發者知道他的 UI 將顯示出單純的文字型 ListView 視窗時，就可以開始撰寫 myStringAdapter 子類別了。

這就是所謂的「反向呼叫」或稱爲「反向控制」(IoC：Inversion of Control)。由於在設計 Activity 類別時，有些關於最終客戶的特殊需求尚無法得知，只能留著虛空的卡榫函數(即 Activity 的 onCreate()函數)；等到其後的某一天，最終客戶的需求出現了，應用程式開發者將之寫入 myActivity 子類別(的 onCreate()函數裡。於是，當 Activity 類別進行反向呼叫到 myActivity 的 onCreate()函數時，myActivity 的『實』恰好填補了 Activity 所預留的『虛』空。框架設計者與應用類別開發者雙方的智慧也融合爲一了。如下圖：

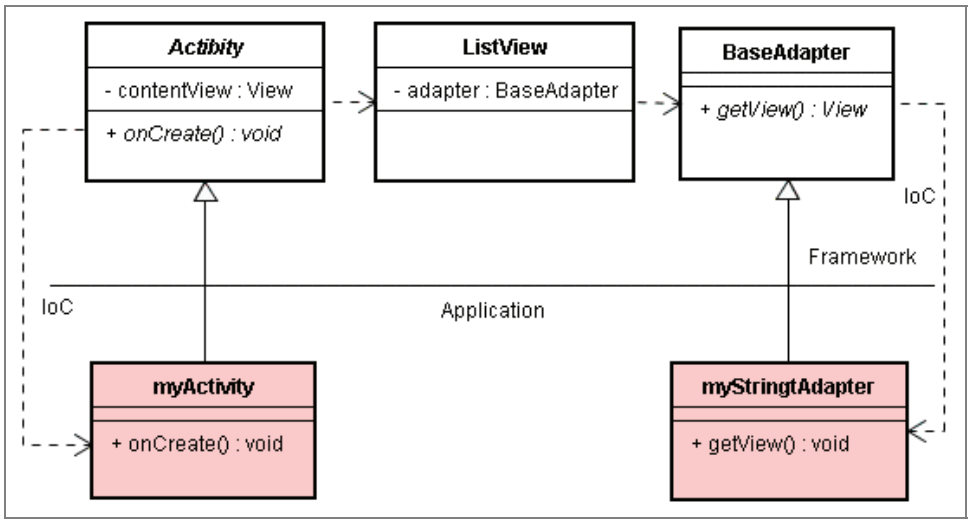


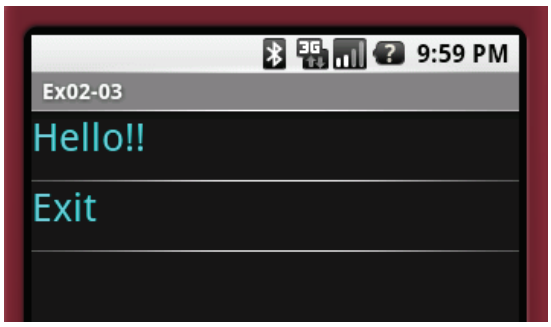
圖 2-4　Hook 函數實現了 IoC(二)

當 Activity 要求顯示 ListView 的內容時，ListView 就從 BaseAdapter 而反向

呼叫到 myStringAdapter 的 getView()等函數，而取得顯示的內容和長相了。Hook 函數實現了反向呼叫，促進了虛實相依。因此，它在框架裡佔有極爲核心的地位。於此，請看一個 Android 的範例 AP，它將實現上述的圖 2-4 之情境：

<<操作情境>>

此 AP 執行時，畫面上顯示出一個單純的文字 ListView 視窗，如下圖：



<<撰寫程式>>

Step-1. 建立一個 Android 程式專案：Ex02-03。

Step-2. 撰寫 BaseAdapter 的子類別：myStringAdapter。

```
// myStringAdapter.java
package com.misoo.pkzz;
import java.util.ArrayList;
import android.content.Context;
import android.graphics.Color;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

public class myStringAdapter extends BaseAdapter {
    private Context ctx;
```

```
    private ArrayList<String> coll;

    public myStringAdapter(Context context) {
            ctx = context;
            coll    = new ArrayList<String>();
            coll.add("Hello!!");    coll.add("Exit");    }
    @Override public int getCount()    {    return coll.size();    }
    @Override public Object getItem(int position) {    return coll.get(position);    }
    @Override public long getItemId(int position) {       return position;    }
    @Override public View getView(int position, View convertView, ViewGroup parent) {
        TextView tv;
        if (convertView == null)        tv = new TextView(ctx);
        else                            tv = (TextView)convertView;
        tv.setTextColor(Color.CYAN);        tv.setHeight(45);
        tv.setTextSize(26);                 tv.setText(coll.get(position));
        return tv;
}}
```

Step-3. 撰寫 Activity 的子類別：myActivity。

```
// myActivity.java
package com.misoo.pkzz;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

public class myActivity extends Activity implements OnItemClickListener {
    @Override public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
        ListView lv = new ListView(this);
        lv.setAdapter(new myStringAdapter(this));     lv.setOnItemClickListener(this);
        setContentView(lv);
      }
    public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
          if(arg2 == 0)          setTitle(((TextView)arg1).getText());
          else if(arg2 == 1)    finish();
}}
```

<<說明>>

當此程式執行時，框架先藉由 Activity 的卡榫函數：onCreate()反向呼叫了 myActivity 類別的 onCreate()函數。其內之指令：

```
ListView lv = new ListView(this);
lv.setAdapter(new myStringAdapter(this));
……..
setContentView(lv);
```

這誕生了 ListView 和 myStringAdapter 物件，並建立如下關係：

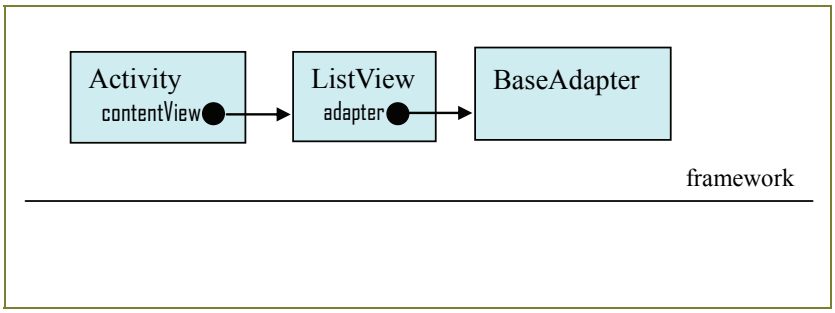


圖 2-5　框架定義了抽象類別間之關係

此外，執行 setContentView(lv)指令時，還有一項動作是：藉由 BaseAdapter 的 4 個 Hook 函數：

★ getCount()　　★ getItem()
★ getItemId()　　★ getView()

而反向呼叫了 myStringAdapter 類別的：

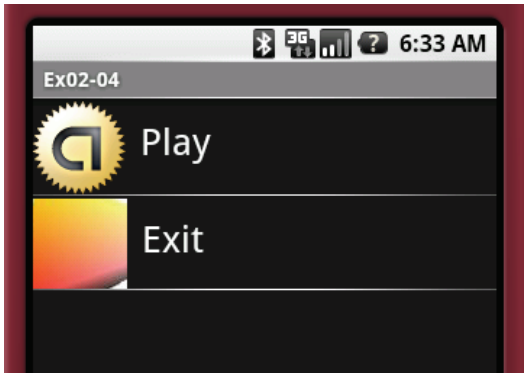
- getCount()函數而得知此 ListView 含有 N 個細項。
- 呼叫 N 次 getItem()函數，每次取得 Coll 裡的一個物件內容。
- 呼叫 N 次 getItemId()函數，詢問該物件將出現在畫面上 ListView 裡的第幾個細項。
- 呼叫 N 次 getView()函數，每次取得一個 View 物件，將其內容顯示於畫面上的 ListView 裡。

2.4　Android 的卡榫函數範例(二)

上述畫面上 ListView 裡的長相是簡單文字而已。你也能依樣畫葫蘆，撰寫一個長相較美觀的 ListView，其範例程式如下：

<<操作情境>>

此 AP 執行時，畫面上顯示出一個較美觀的 ListView 視窗，如下圖：



<<撰寫程式>>

Step-1. 建立一個 Android 程式專案：Ex02-04。

Step-2.　撰寫 Row 類別。

```
// Row.java
public class Row {
        private String text;
        private int resid;

        Row(String tx, int id){ text = tx;    resid = id;  }
        public int getDwId() {    return resid;    }
        public String getText() {    return text;    }
```

```
    public boolean isSelectable() {   return true;   }
}
```

Step-3. 撰寫 BaseAdapter 的子類別：myAdapter。

```
// myAdapter.java
package com.misoo.ex02;
import java.util.ArrayList;
import android.content.Context;
import android.graphics.Color;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;

public class myAdapter extends BaseAdapter {
    private final int WC = ViewGroup.LayoutParams.WRAP_CONTENT;
    private Context ctx;
    private ArrayList<Row> coll;

    public myAdapter(Context context, ArrayList<Row> list) {
            ctx = context;   coll = list;   }
    public int getCount() {   return coll.size();   }
    public Object getItem(int position) {   return coll.get(position);   }
    public long getItemId(int position) {   return position;    }

    public View getView(int position, View convertView, ViewGroup parent) {
        LinearLayout layout = new LinearLayout(ctx);
        layout.setOrientation(LinearLayout.HORIZONTAL);
        Row rw = (Row)coll.get(position);
        ImageView iv = new ImageView(ctx);
        iv.setImageResource(rw.getDwId());
        iv.setPadding(0, 2, 5, 0);
        LinearLayout.LayoutParams param
                    = new LinearLayout.LayoutParams(WC, WC);
        layout.addView(iv, param);
        TextView txv = new TextView(ctx);       txv.setText(rw.getText());
        txv.setTextColor(Color.WHITE);          txv.setTextSize(26);
        LinearLayout.LayoutParams param2
            = new LinearLayout.LayoutParams(WC, WC);
        param2.leftMargin = 5;     param2.topMargin = 13;
        layout.addView(txv, param2);
        return layout;
```

```
}}
```

Step-3. 撰寫 Activity 的子類別：myActivity。

```
// myActivity.java
package com.misoo.ex02;
import java.util.ArrayList;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.AdapterView.OnItemClickListener;

public class myActivity extends Activity implements OnItemClickListener {
        private ArrayList<Row> coll;
        @Override public void onCreate(Bundle icicle) {
             super.onCreate(icicle);
             setContentView(R.layout.main);
             coll = new ArrayList<Row>();
             coll.add(new Row("Play",R.drawable.icon) );
             coll.add(new Row("Exit", R.drawable.icon2));
             ListView lv = new ListView(this);
             lv.setAdapter(new myAdapter(this, coll));
             lv.setOnItemClickListener(this);
             setContentView(lv);
          }
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
               if(arg2 == 0) {
                      Row rw = (Row)coll.get(arg2);
                      setTitle(rw.getText());  }
               else if(arg2 == 1) finish();
}}
```

<<說明>>

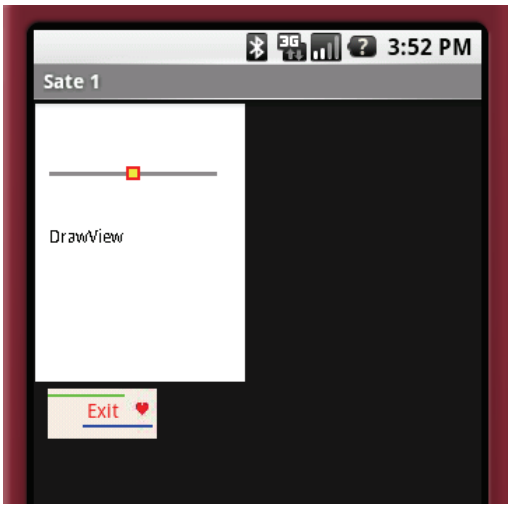
Android 框架提供了 BaseAdapter 抽象類別，並設計了 getView()等卡榫函數。於是，框架就基於 BaseAdapter 類別的卡榫函數來與應用類別 myAdapter 搭配起來了。此範例裡的 myAdapter 類別與上一個範例的 myStringAdapter 類別屬於不同的應用程式，皆能與框架裡的 BaseAdapter 抽象類別相結合，就發揮了卡榫函數的功能了。

2.5 Android 的卡榫函數範例(三)

在上一節裡，已經從 Android 的 BaseAdapter 抽象類別看到 getView()等卡榫函數了。由於 Hook 函數是框架的基本機制，所以在 Android 框架裡，處處可以見到它的身影。例如，下述之範例程式：

<<操作情境>>

此 AP 執行時，畫面上顯示出一個 2D 圖形，如下：



如果按下<Exit>按鈕，就結束了。

<<撰寫程式>>

Step-1. 建立一個 Android 程式專案：Ex02-05。

Step-2.　撰寫 View 的子類別：DrawView 類別。

// DrawView.java

```
package com.misoo.ppxx;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.view.View;

public class DrawView extends View{
        private Paint    paint= new Paint();

        DrawView(Context ctx) {    super(ctx);    }
        @Override protected void onDraw(Canvas canvas) {
              super.onDraw(canvas);
              int line_x = 10;   int line_y = 50;
              canvas.drawColor(Color.WHITE);
              paint.setColor(Color.GRAY);
              paint.setStrokeWidth(3);
              canvas.drawLine(line_x, line_y, line_x+120, line_y, paint);
              paint.setColor(Color.BLACK);
              paint.setStrokeWidth(2);

              canvas.drawText("DrawView", line_x, line_y + 50, paint);
              int pos = 70;
              paint.setColor(Color.RED);
              canvas.drawRect(pos-5, line_y - 5, pos+5, line_y + 5, paint);
              paint.setColor(Color.YELLOW);
              canvas.drawRect(pos-3, line_y - 3, pos+3, line_y + 3, paint);
 }}
```

Step-3.　撰寫 Button 的子類別：exitButton。

// exitButton.java

```
package com.misoo.ppxx;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.widget.Button;

public class exitButton extends Button{
```

```
    public exitButton(Context ctx){
            super(ctx);
            super.setText("Exit");
            super.setTextColor(Color.RED);
            super.setBackgroundResource(R.drawable.heart);
    }
    @Override protected void onDraw(Canvas canvas) {
            Paint paint= new Paint();
            paint.setStrokeWidth(2);
            paint.setColor(Color.GREEN);
            canvas.drawLine(0, 5, 55, 5, paint);
            paint.setColor(Color.BLUE);
            canvas.drawLine(25, 27, 75, 27, paint);
            super.onDraw(canvas);
}}
```

Step-3. 撰寫 Activity 的子類別：myActivity。

```
// myActivity.java
package com.misoo.ppxx;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.RadioGroup;

public class myActivity extends Activity implements OnClickListener {
    private final int WC = RadioGroup.LayoutParams.WRAP_CONTENT;
    @Override protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            goto_state_1();
        }
    private void goto_state_1()
       {   setTitle("Sate 1");   show_layout_01();   }
    private void goto_state_2()
       {   setTitle("Sate 2");   finish();   }
    private void show_layout_01(){
        LinearLayout layout_01 = new LinearLayout(this);
        layout_01.setOrientation(LinearLayout.VERTICAL);
        LinearLayout.LayoutParams param =
                new LinearLayout.LayoutParams(150, 200);
```

```
            param.leftMargin = 1;    param.topMargin = 3;
            DrawView gv = new DrawView(this);
            layout_01.addView(gv,param);

            Button btn = new exitButton(this);
            btn.setOnClickListener(this);
            LinearLayout.LayoutParams param2 =
                    new LinearLayout.LayoutParams(WC, WC);
            param2.leftMargin = 10;      param2.topMargin = 5;
            layout_01.addView(btn,param2);
            setContentView(layout_01);
      }
       public void onClick(View v) {    goto_state_2();       }
}
```

<<說明>>

在 Android 的抽象類別 View 裡，提供了 onDraw()卡榫函數，用來結合 DrawView 應用類別，如下圖：

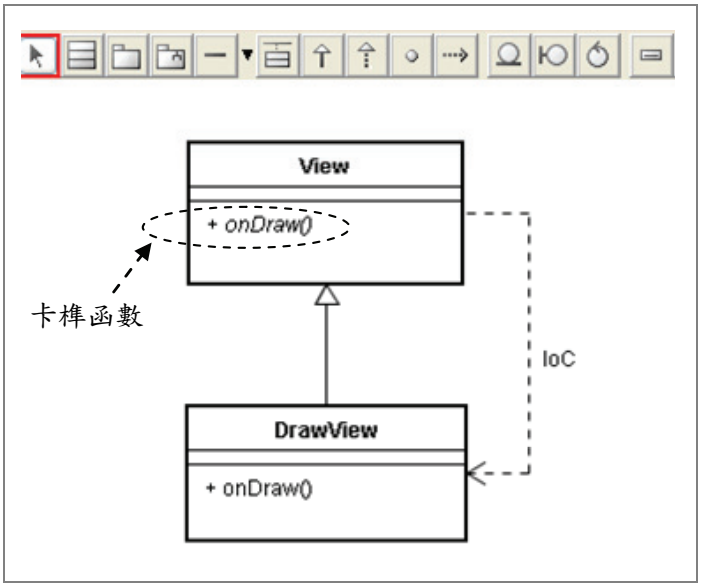


圖2-6　View抽象類別的onDraw()卡榫函數

2.6 卡榫函數與樣式之關係

---- 以 Factory Method 和 Template Method 樣式爲例

卡榫(Hook)函數是父、子類別之間的卡榫接口處。在應用框架(AF)裡有許多抽象(不變)的父、子類別，需要卡榫函數來銜接；此外，在應用程式(AP)裡也有各式各樣的父、子類別，它們也藉由卡榫函數來接合。所以處處可見到卡榫函數的身影。當然也常見父類別(如上節範例裡的 Button 及 View 父類別)是位於 AF 裡，而其子類別是位於(如上節範例裡的 exitButton 及 DrawView 子類別)是位於 AP 裡，則卡榫函數就扮演框架與應用類別之間的卡榫功能了。

Android 常用到的設計樣式通常都涉及多個類別之間的合作模式，也涵蓋父、子類別之間的合作及溝通情形，所以卡榫函數是框架設計樣式裡的重要機制，因此在本章裡就得先介紹卡榫函數，充分理解其特性和涵意，才能領悟各種設計樣式，並進而活用 Android 框架。例如，最簡單的 Factory Method 樣式：

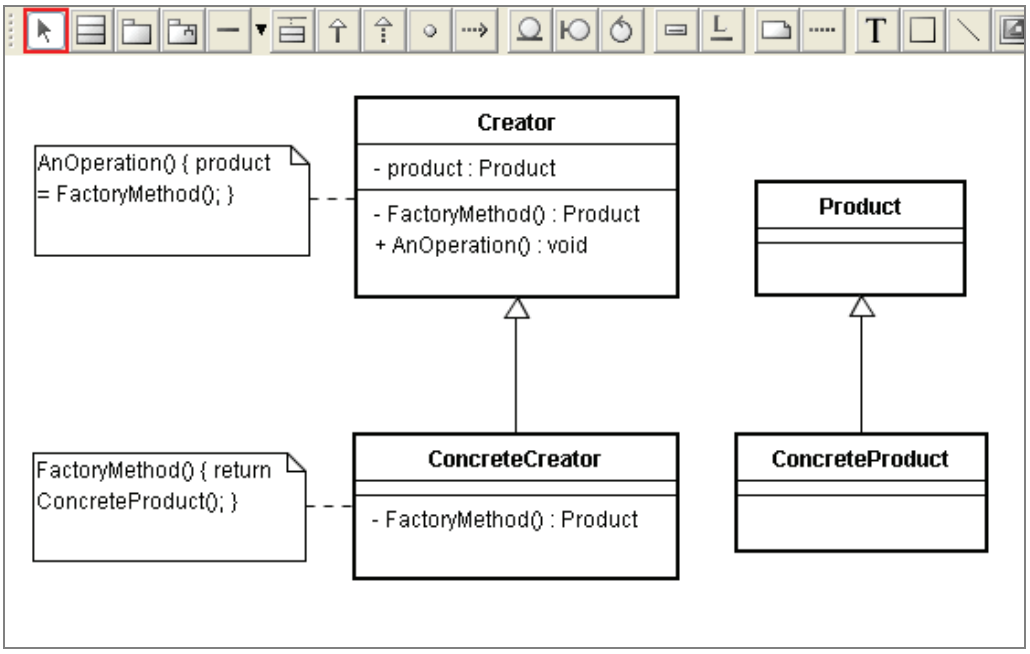


圖 2-7 Factory Method 樣式圖[GoF]

[附註]　GoF 樣式是指 Design Patterns: Elements of Reusable Object-Oriented Software 一書裡所列舉的 23 個樣式(請參閱第 1 章)。

其中的 FactoryMethod()就是一種卡榫函數。即使是最簡單的 Android 程式，都運用了上圖的 Factory Method 樣式，而其幕後的台柱就是卡榫函數。例如下述之範例程式 Ex02-06：

<<操作情境>>

此 AP 執行時，畫面上顯示出 2D 圖形，如下：

<<撰寫程式>>

Step-1. 建立一個 Android 程式專案：Ex02-05。

Step-2.　撰寫 View 的子類別：GraphicView 類別。

```
// GraphicView.java
package com.misoo.pkcc;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.view.View;

public class GraphicView extends View{
        private Paint paint= new Paint();

        GraphicView(Context ctx) {      super(ctx);        }
        @Override protected void onDraw(Canvas canvas) {
            int line_x = 10;      int line_y = 50;
            canvas.drawColor(Color.WHITE);
            paint.setColor(Color.GRAY);
            paint.setStrokeWidth(3);
            canvas.drawLine(line_x, line_y, line_x+120, line_y, paint);
             paint.setColor(Color.BLACK);
            paint.setStrokeWidth(2);
            canvas.drawText("這是GraphicView繪圖區", line_x, line_y + 50, paint);
            int pos = 70;
            paint.setColor(Color.RED);
            canvas.drawRect(pos-5, line_y - 5, pos+5, line_y + 5, paint);
            paint.setColor(Color.YELLOW);
            canvas.drawRect(pos-3, line_y - 3, pos+3, line_y + 3, paint);
}}
```

Step-3. 撰寫 Activity 的子類別：GraphicActivity。

```
// GraphicActivity.java
package com.misoo.pkcc;
import android.app.Activity;
import android.os.Bundle;

public class GraphicActivity extends Activity {
       private GraphicView gv = null;
       @Override public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
               gv = new GraphicView(this);
                setContentView(gv);
}}
```

<<說明>>

此 GraphicActivity 類別裡的 onCreate()函數就是定義於 Activity 父類別裡的卡榫函數。如下圖所示：

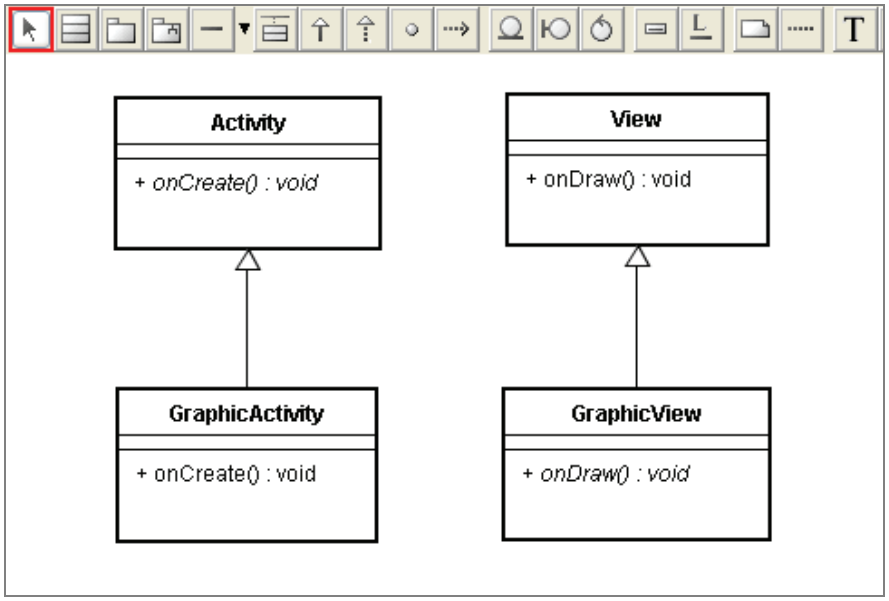


圖 2-8　Factory Method 樣式依賴於卡榫函數

當你仔細觀察這 onCreate()函數裡的指令：

```
void onCreate(Bundle savedInstanceState) {
    ……
   gv = new GraphicView(this);
    ……
}
```

你會發現了 Factory Method 樣式的身影了。原來，在 Activity 與 GraphicActivity 的父、子類別繼承機制幕後，運用了 Factory Method 樣式。而這個 onCreate()就是卡榫函數的角色，它負責誕生 GraphicView 物件之任務。由於 onCreate()也負有誕生應用類別物件之責任，所以稱爲 Factory Method。

除了上述的 Factory Method 樣式之外，其它樣式也大多使用到卡榫函數，例

如 GoF 的 Template Method 樣式：

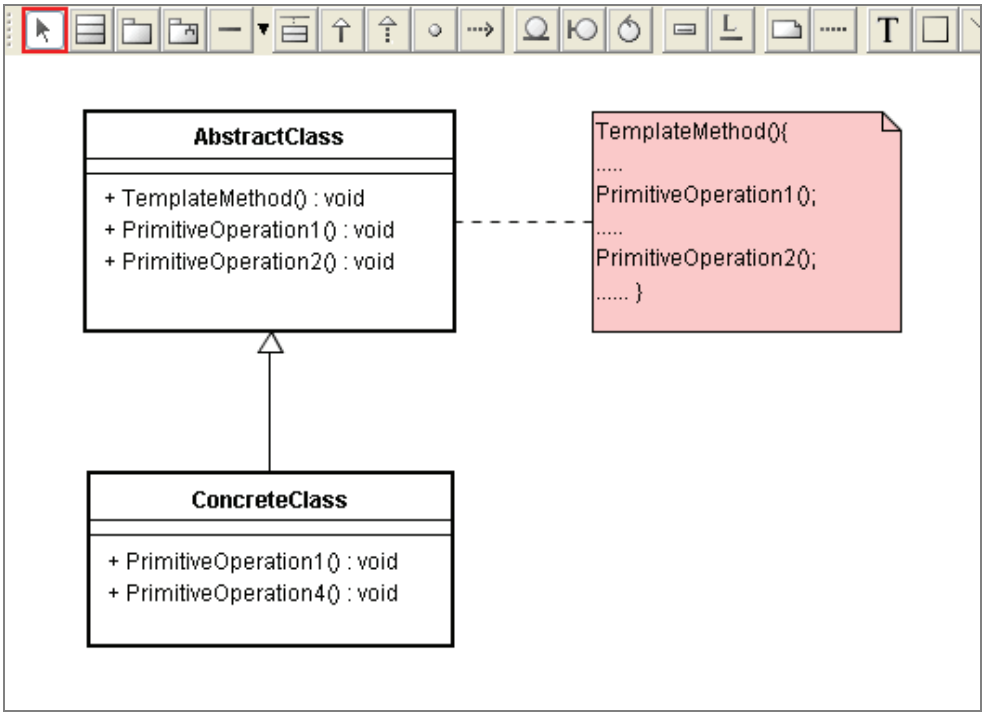


圖 2-9　Template Method 樣式[GoF]

其中的 PrimitiveOperation()就是一種卡榫函數。總而言之，樣式是框架設計專家慣用的手藝，蘊含了深入的設計理念，如此透過樣式來解說 Android 框架的設計理念是精通 Android 平台架構的不二法門。有了 Android 這個頂級的實際框架作爲樣式應用情境，讓人們更易於領悟設計樣式之真諦。於是我們對設計樣式與應用框架兩者之認識將同步成長，且相得益彰。而在這些設計樣式及框架的幕後，常需要依賴卡榫函數來實現父、子類別之間的銜接，讓樣式的優良設計技藝得以實現，因而能設計出優越的應用框架。◆