< 30 分鐘寫程式〉

撰寫影像處理程式 Java 篇

文：J.J. (井民全)
Email: mqJing@msn.com
Homepage: http://debut.cis.nctu.edu.tw/~ching

上一期,我們介紹了如何使用 C# 這個程式語言在 .Net Framework 的環境下，實作簡單的影像處理程式。 現在我們使用 Java 程式語言來實作類似的程式，讓你知道 Java 也能輕易地做到一樣的事情， 毫不遜色!!

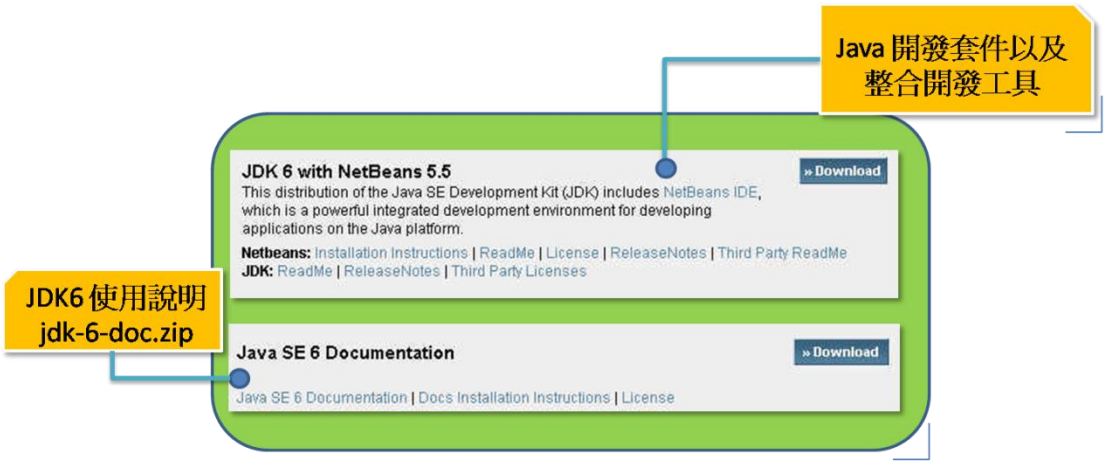
使用 Java 語言來開發程式， 首先要做的事情就是決定使用是哪一家的開發工具。 目前比較知名的工具有 JBuilder [1]， Eclipse[2] 以及 NetBeans[3]。 每一種開發工具都很好用而且極具特色。 其中 NetBeans 因爲與 JDK (Java Development ToolKit) 綁在一起， 使得 NetBeans 成爲許多新手第一次使用 Java 的開發環境， 這樣的方式造就了 NetBeans 逐漸成爲最多新手使用的 Java 整合式開發工具。 不例外的我們這次教學使用的工具就是使用 NetBeans。 最新版本爲 5.5。

比較值得一提的是， 從過去的 Turbo C 一直到現在的 JBuilder， 有人用慣了 Borland 老牌子的產品後就黏上去了。 之後無論其他的開發環境有多方便，也覺得不順手， 即使應公司要求而使用其他的軟體開發工具， 也要將除錯熱鍵改成 F7 和 F8， 像這種重度依賴使用特定公司軟體的行爲， 我稱爲軟體使用慣性或者是軟體使用黏性。 因爲這樣的慣性與黏性使得在 Eclipse 與 NetBeans 上也存在著許多死忠的跟隨者。

下載 Java 開發工具組

首先要介紹的是 Sun Developer Network http://java.sun.com 這個網站。 Sun Developer Network 幾乎是所有 Java 開發者都要朝聖的地方， 這個地方不僅提供了 Java 開發工具組 (Java Development Toolkit， JDK) 的原始下載地點， 也提供了 Java 語言相關發展的最新消息以及龐大的相關知識。 身爲一個 Java 程式設計師， 你一定要知道這個網站。

從 Java Developer Network 右邊的 Popular Downloads 中點選 Java SE，進入下載頁面。 在這個頁面中我們將下載 Java 開發工具組(JDK 6)搭配 NetBeans 5.5 以及 Java SE 6 文件。 選項如下圖所示:

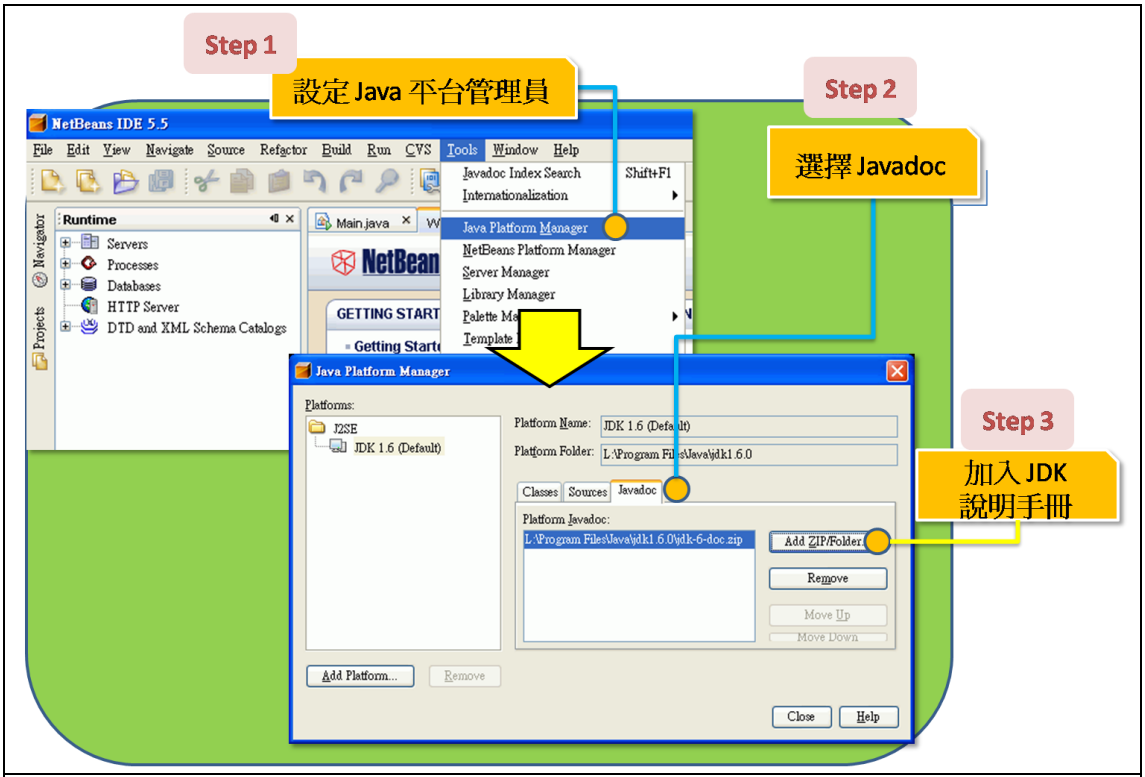


圖說: 下載我們需要的開發工具以及 JDK6 使用說明書

安裝 NetBeans 與設定說明文件

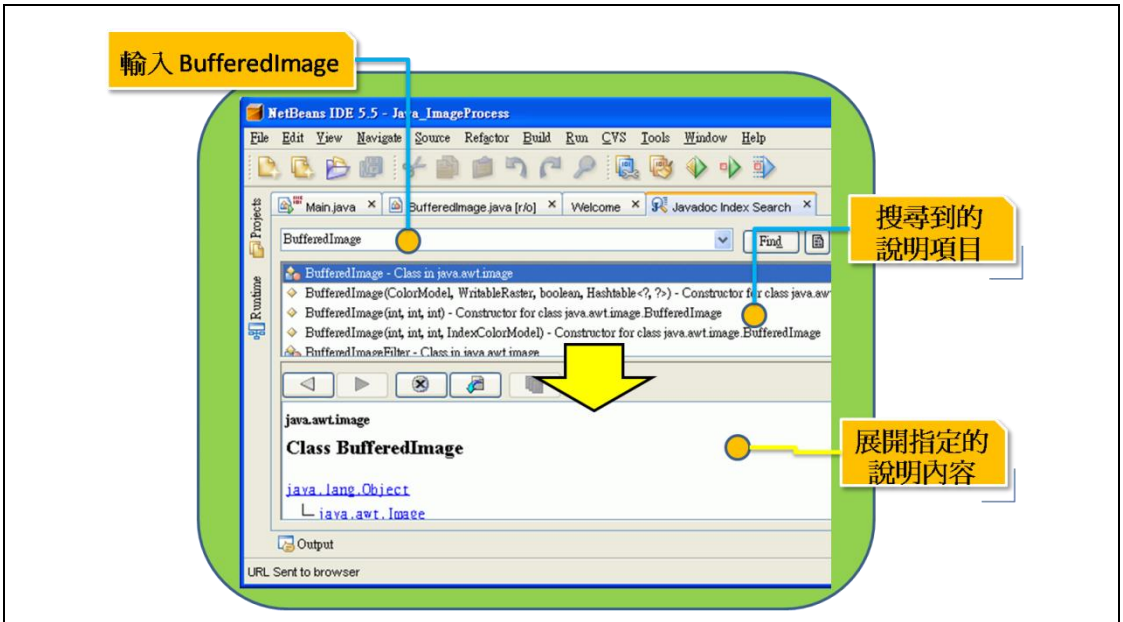
下載回來後， 執行 jdk-6-nb-5_5-win.exe， 經過幾分鐘 JDK6 以及整合開發工具 NetBeans 就會安裝進你的電腦中。 接下來， 我們就要設定 JDK6 使用說明書了， 這樣才不會因爲忘記某些類別或方法的使用方式， 而手忙腳亂。 在過去有多人就是因爲 JBuilder 的 Help 好用， 而使用它來當作主要的 Java 程式發展環境。 由此可知說明文件的索引與搜尋工具， 對程式設計師而言， 是非常重要的。

現在 NetBeans 做的也一樣好， 只要簡單地設定說明文件的位置， 我們就可以直接在 Tools -> Javadoc Index Search 中直接搜尋我們想要知道的類別使用方法。 其中詳細的設定流程， 如下圖所示。



圖說: 在 NetBeans 開發環境中設定 JDK 使用說明書

測試一下， 在 JavaDoc Index Search 中搜尋 BufferedImage 類別， 看看會不會出現說明資訊。



圖說: 使用 JavaDoc Index Search 搜尋 BufferedImage 類別

建立新的專案

跟其他程式語言的開發環境下一樣， 如果想在 NetBeans 中開發程式， 首

先要建立專案，在 NetBeans 中建立最陽春的專案可以分三個步驟來完成: Step 1: 選擇 File ->New Project Step 2: 在專案分類面版上， 選擇 General 分類下的 Java Applications， 我們這樣的設定是要告訴 NetBeans， 這個專案要建立的是一個能在使用者機器上運行的獨立程序而非其他的專案。 如你所猜想的， NetBeans 也可以開發在瀏覽器下運行的小程序，這種程序稱爲 Java Applet，不過這不是我們今天要討論的重點， 我們的重點是放在如何撰寫高效率的影像處理程式。 Step 3: 接著就是選擇我們想要的專案名稱與專案目錄， 設定完成後最後按 Finish 即完成建製專案的必要步驟了。
詳細的建立專案過程， 如下圖所示。

圖說: 建立新的專案

載入第一張影像

如果你有看上一期雜誌的內容， 你會知道其實我們在處理影像的時候， 是把影像視爲一個方形的陣列， 每個陣列的點是由 R，G，B (紅， 綠， 藍) 三種顏色所組成的。 在 .Net Framework 裡面， 要載入一張影像使用的指令是 Image 類別的 From File 方法， *image = Image.FromFile(...)* 來載入影像， 那麼在 Java 下， 有沒有甚麼程式庫可以幫我們載入圖檔呢? javax.imageio 提供了一系列有關 Java 影像輸入輸出的程式碼， 可以幫助我們處理載入圖檔的工作。
ImageIO 提供了常用的影像格式， 如 JPEG， PNG， BMP 和 GIF 等圖檔的存取功能。 如果我們想要載入一張影像， 使用下面的範例就會傳回一個

BufferedImage 影像物件回來。 而今天我們主要使用的類別就是 BufferedImage。一般來說,只要是輸入輸出處理相關的動作， 都必須要加入一些處理例外情況的程式碼， 以防萬一 指定的檔案不存在等， 這種情況發生。 所以我們通常會在適當的地方加入 try 與 catch 指令， 攔截例外訊息。 詳細寫法如下:

讀取圖檔:

```
try{
    BufferedImage image=ImageIO.read(new File("c:\\ 火焰超人.JPG"));
}catch(Exception e){
    // 在這裡寫出例外處理程式
}
```

顯示一張影像

爲了方便起見， 我們建立一個方法來顯示剛剛建立的 BufferedImage 圖形物件 image。 整個思考的邏輯是:

1. 這個影像必須要放在視窗中， 我們使用下面的程式碼完成

```
JFrame f = new JFrame("");
```

2. 若影像超過實際螢幕大小時， 我們希望加入捲軸讓使用者可以捲動。 這裡我們使用 JScrollPane 類別來處理。

```
JScrollPane scrollPane = new JScrollPane(new JLabel(new ImageIcon(image)));
f.getContentPane().add(scrollPane);
f.pack();
```

在上面的程式碼的行爲是利用 JScrollPane 物件把 image 包起來， 並且加到視窗 f 中。

3. 當使用者按下視窗由上角的 x 鍵， 我們希望的行爲是關閉這個視窗。 面對這樣的要求， 我們使用 setDefaultCloseOperation 指定當使用者按下 x 鍵時的程式行爲。

```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

ImageComponent 類別

由上面的講解， 我們已經有足夠的知識， 建立一個專門處理影像的類別， 以

方便將來的程式處理。 整個類別的架構應該是長的這個樣子。

ImageComponent 類別架構:

```
class ImageComponent implements Runnable{
    // …
    ImageComponent(String aFilename){
        // 載入圖形的程式碼， 放在這裡 !!
    }
    public void Show(){
        // 要求系統以 “event-dispatching” 執行緒,
        // 執行目前物件中 run 方法內的程式碼
        SwingUtilities.invokeLater(this);
    }
    public void run(){
        // 建立視窗並且顯示圖形的程式碼， 放在這裡 !!
    }
}
```

在把程式填進去之前， 請注意一下上面 Show 方法的內容. 你會覺得奇怪爲什麼我們要用 SwingUtilities.invokeLater(this) 這個指令 ?

這是因爲 Java 的程式庫裡面有一個很大的 API (Application Programming Interface)叫做 Swing API， 專門提供輕巧、靈活、易用的元件讓我們使用。 這其中包含了我們用來建立視窗顯示影像的元件 JFrame。 然而爲了符合輕巧、簡單的設計理念， 所有 Swing 元件的事件處理與繪製， 都是交由一個稱爲 ” event-dispatching”的執行緒來統一負責執行。 一般來說， 我們的程式是由主執行緒所執行， 但是若要牽涉到實體化 Swing 元件， 例如: 畫出元件等要求。 則必須使用 “event- dispatching” 執行緒來進行處理。

所以當我們使用 Swing API 中的 JFrame 類別來建立並且顯示一個視窗時， 這是一個實體化的動作， 根據上面提到的規則， 這段程式碼必須要交由 “event-dispatching” 執行緒來處理。 而下達的指令就是 SwingUtilities.invokeLater 方法。

執行緒

我們預期操作 ImageComponent 的方式應該是如此。

```
String Filename="c:\\wen.bmp";
```

```
ImageComponent image =new ImageComponent(Filename); //  載入影像
image.Show(); //  顯示影像
```

執行的結果， 如下:

圖說:使用 JFrame 顯示的影像

載入影像與顯示圖形的完整程式碼。

```
class ImageComponent implements Runnable{
        String Filename; //  目前的檔名
        BufferedImage image; //  目前的影像
        JFrame f; //  目前的視窗
        public ImageComponent(String aFilename){
                Filename=aFilename;
                image=LoadImage(Filename);
        }
        //  載入一張影像
        public static BufferedImage LoadImage(String Filename){
                BufferedImage image;
                try{
                        image=ImageIO.read(new File(Filename));
                }catch(Exception e){
                        javax.swing.JOptionPane.showMessageDialog(null,
                                "載入圖檔錯誤: "+Filename);
                        image=null;
                }
                return image;
        }
```

```
    // 顯示一張影像
    public void Show(){
        SwingUtilities.invokeLater(this);
    }
    // 實體化影像物件的方法
    public void run(){
        f = new JFrame("");

        // Step 1: 若影像超過螢幕， 則加入捲軸
        JScrollPane scrollPane = new JScrollPane(
                new JLabel(new ImageIcon(image)));
        f.getContentPane().add(scrollPane);
        f.pack();

        // Step 2: 設定點選 x 表示關閉視窗
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Step 3: 加入視窗標題
        f.setTitle(Filename+" "+image.getWidth()+
                " x "+image.getHeight());

        // Step 4: 設定視窗顯示在螢幕中央
        f.setLocationRelativeTo(null);
        // Step 5: 顯示出視窗
        f.setVisible(true);
    }
}
```

把彩色影像轉換成黑白影像

上一期我們提到製作黑白影像的訣竅， 就是把影像中的每個像點的值， 設定成灰階值。 灰階值的計算可以是取得該點的 R，G，B 三原色的值加起來在除以 3。

```
int gray=(r+g+b)/3;
```

所以如果要進行轉換黑白影像的工作， 最簡單的方法就是把每個點的顏色全部設定爲他們自己的灰階值。 在 Java 程式中， 我們可以用 BufferedImage 的 GetRGB 方法取得指定位置的 RGB 值， 計算完灰階值後， 利用 SetRGB 方法把資料回填進去。 詳細的作法如下

灰階化彩色影像方法

```
public void doGray(){
        int Height=image.getHeight();
        int Width=image.getWidth();
        for(int y=0;y<Height;y++){
                for(int x=0;x<Width;x++){
                        int rgb=image.getRGB(x，y);
                        int r=(rgb&0x00ff0000)>>16; // 取得紅色的資料
                        int g=(rgb&0x0000ff00)>>8; // 取得綠色資料
                        int b=rgb&0x000000ff; // 取得藍色資料
                        int gray=(r+g+b)/3; // 計算灰階值
                        rgb=(0xff000000|(gray<<16)|(gray<<8)|gray);
                        image.setRGB(x，y，rgb);
                }
        }
}
```

加入功能選項

我們希望在視窗上方加入一個選單， 讓使用者選擇要執行的影像處理運算，該怎麼做呢? 很簡單， 只要使用 JMenuBar， JMenu 以及 JMenuItem 這三個類別就能輕鬆的加入選單。 例如我們把下面的程式碼， 加到建構視窗的 run 方法中。

ImageComponent 類別選單建立部分:

```
// 建立主選單
JMenuBar menuBar = new JMenuBar();

// 建立 File 以及 Operator 選項
JMenu File_Menu=new JMenu("File");
JMenu Operator_Menu = new JMenu("Operator");

// 把 File 與 Operator 加到主選單上
menuBar.add(File_Menu);
menuBar.add(Operator_Menu);

// 把 [轉成黑白影像] 按鈕 加到 Operator 選單上
JMenuItem item = new JMenuItem("轉成黑白影像");
```

```
Operator_Menu.add(item);
f.setJMenuBar(menuBar);
```

加入事件處理功能

我們希望當使用者按下[轉成黑白影像] 按鈕時， 執行彩色轉黑白的動作。 這樣的要求， 在 Java 程式語言中只要讓我們的 ImageComponent 類別實作 ActionListener 介面， 並且向[轉成黑白影像] 按鈕註冊即可達成。

程式的骨架應該長成下面這個樣子: 我們使用 implements 語法， 定義 ImageComponent 類別實作 ActionListener 介面。 利用 addActionListener(this) 指令把目前物件向 item 按鈕註冊。 這兩個動作完成後， 當使用者按下選單[轉成黑白影像] 按鈕時， 會呼叫目前物件的 actionPerformed 方法執行。

事件處理程式碼:

```
class ImageComponent implements ActionListener， Runnable {
      // 向 [轉成黑白影像] 按鈕註冊
      // 告訴它，被按下時， 呼叫目前物件的
      //  actoionPerformed 方法
      void run(){
            // …
            JMenuItem item = new JMenuItem("轉成黑白影像");
            item.addActionListener(this);
      }
      public void actionPerformed(ActionEvent e) {
           // 把彩色轉黑白的程式碼， 放在這裡 !!
           if(e.getActionCommand().equals("轉成黑白影像")){
                 doGray();
                 f.repaint();
            }
       }
}
```

事實上， 從程式碼中可以看出來我們的 ImageComponent 同時也實作了另一個介面 Runnable。 在前面，我們提到建立與顯示視窗的程式碼， 是利用 invokeLater 方法， 而間接被呼叫的。

下達指令 SwingUtilities.invokeLater(this)的意思是: 讓專門處理 java swing 繪

圖元件的執行緒 (thread)，來執行繪圖的動作。 而非使用目前執行的 main thread 來執行繪圖。

當一些必要的事件被執行完畢後， 再執行自己物件的 run 方法， 我們則在 run 方法中撰寫建構視窗的程式碼。 爲了讓這樣的機制能夠運行， 我們的 ImageComponent 必須要實作 Runnable 介面。

加入主選單的結果， 顯示如下

圖說:使用 JMenuBar 建立的選單

下圖是執行轉成黑白影像的的結果

圖說: 彩色轉成黑白影像

檔案選擇對話框

有的朋友會希望能夠跳出一個檔案選擇的對話框， 讓使用者選擇要處理的圖形。上一期我們已經看到了 C# 的作法， 現在我們來看看 Java 的作法， 是否一樣簡單。

檔案選擇對話框:

```
// 開啓新的圖檔
```

```
public void Open(){
        FileDialog fd = new FileDialog(f， "Open..."， FileDialog.LOAD);
        fd.setVisible(true);
        if(fd!=null){
              Filename=fd.getDirectory() +
                        System.getProperty("file.separator").charAt(0)
                        + fd.getFile();
              // 開啓檔案的程式， 就放在這裡!!
        }
}
```

因爲 Java 的執行檔是跨平台的。 也就是你編譯好的執行檔可以直接拿到 Linux 以及 Windows 或者是 Mac 都可以正確的執行， 其實只要你的平台有實作 Java Virtual Machine(JVM)， 就可以執行我們的 Java 程式。 Java 的設計者必須考量程序可能在不同系統上執行的可能性， 我們要盡量避免讓程式相依於某個特定的系統。 例如: Windows 作業系統中， 目錄的符號爲”\”， 但是在 Linux 作業系統的目錄符號卻是反方向的 ”/”。

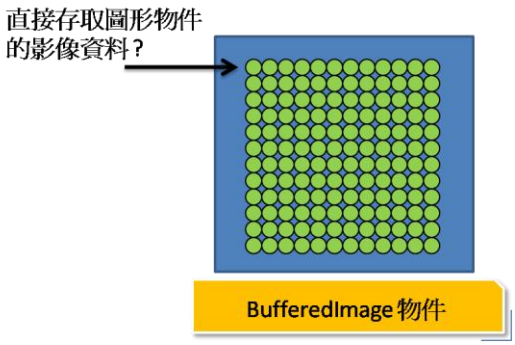
我們使用 System.getProperty("file.separator") 這指令來解決這個系統差異的問題。 當我們的程序在某一個平台上執行時， file.separator 系統屬性會告訴我們正確的目錄符號。 使得程式依然可以在不同的平台上順利的運行。

執行的結果如下:

圖說: 檔案選擇對話視窗

比較有效率的程式碼

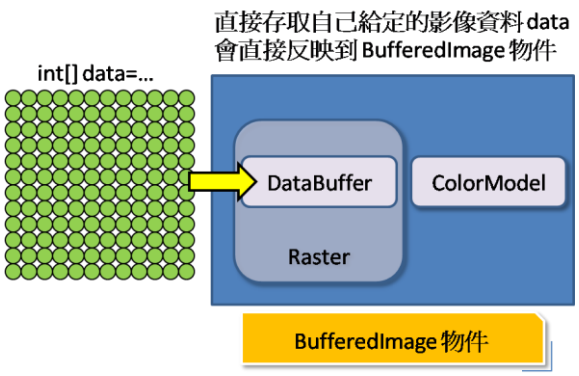
由 doGray 方法中， 我們觀察程式碼， 這個方法不斷地呼叫 GetRGB 與 SetRGB 這兩個 BufferedImage 的方法進行像點的灰階化。 我們都知道呼叫函式(方法)是很慢的動作。 尤其是不斷的呼叫 GetRGB 與 SetRGB， 更讓影像處理的效率變慢了。 因此這兩個方法都有提供陣列批次給定影像資料的版本， 可以讓我們把影像資料批次的讀到陣列中。 然後運用比較快速的陣列索引， 進行影像資料的存取動作。 但是有沒有辦法如下圖所示， 讓我們直接改 BufferedImage 中的影像原始資料來達到灰階化的目的呢?



圖說: 直接存取圖形物件的影像資料示意圖

據我所知要達到這樣的要求， 除了要對 BufferedImage 的組成物件有深入的瞭解外， 還要根據不同的像點儲存模式(有的影像檔格式儲存以 24 bit 存一個像點，有的格式使用 32 bit)， 撰寫不同的處理程式。 我的結論是很難。

但是有一種簡單的方法， 可以輕易的達到這樣的要求， 作法是利用 BufferedImage 的 GetRGB 陣列版本複製一份整數陣列的影像資料到一維陣列 data 中，然後根據這個陣列的內容建立一個新的 BufferedImage 取代舊的物件。因爲 data 是我們所建立的， 所以我們可以直接存取 data 陣列的內容。 因爲 BufferedImage 是根據 data 所建立的， 所以直接更改 data 的內容將直接反映到 BufferedImage 物件的影像內容。 整個的架構如下圖所示。 這樣做的好處是我們將不需要再使用 SetRGB 的方式把資料寫回去 BufferedImage 物件中，BufferedImage 本身的影像資料就是參考我們的陣列 data。



圖說: 使用自己給定的資料建立 BufferedImage 物件

詳細的程式作法如下:

```
image=LoadImage(Filename);
Height=image.getHeight();
Width=image.getWidth();
 // 複製影像資料到 data 陣列中
data=image.getRGB(0，0，Width，Height，null，0，Width);

// 利用整數資料， 建立新的 BufferedImage
image=CreateBufferedImage_Direct(data，Height，Width);
```

CreateBufferedImage_Direct 方法的實作如下:

```
// 建立 BufferedImage    (int[]陣列版本)   (非常快)
// 方法:
//        直接建立 BufferedImage， 把 int[] 陣列內容當作資料來源
// 優點:
//        只要改變 rgbData 的內容，
//       直接會反應到 BufferedImage
public static BufferedImage
CreateBufferedImage_Direct(int[] rgbData，int Height，int Width){
      // Step 1: 建立 DataBuffer 物件, 把資料封裝起來
      DataBuffer db = new DataBufferInt(rgbData， Height*Width);
      WritableRaster raster = Raster.createPackedRaster(db， Width，
                              Height， Width，
                              new int[] {0xff0000， 0xff00， 0xff}，
                              null);
      // Step 2: 建立 ColorModel 物件, 用來向系統解釋
      //          我們的資料與顏色的關係
      ColorModel cm = new DirectColorModel(24， 0xff0000，
                                           0xff00， 0xff);
      // Step 3: 最後得到 BufferedImage, 這個影像物件
      //          將與我們的資料陣列同步改變內容
      return new BufferedImage(cm， raster， false， null);
}
```

於是之前的彩色轉黑白的程式， 現在可以改寫爲高效率版本的函式。 你可以把下面的程式與上面 doGray 方法作比較， 你將會發現， 其實只是把 GetRGB 與

SetRGB 拿掉了改成單純而且快速的 1 維陣列存取。

灰階化彩色影像高效率方法:

```
public void doGray_Speed(){
        for(int y=0;y<Height;y++){
                for(int x=0;x<Width;x++){
                        int offset=y*Width+x;
                        int rgb=data[offset];
                        int r=(rgb&0x00ff0000)>>16;
                        int g=(rgb&0x0000ff00)>>8;
                        int b=rgb&0x000000ff;
                        int gray=(r+g+b)/3;
                        rgb=(0xff000000|(gray<<16)|(gray<<8)|gray);
                        data[offset]=rgb;
                }
        }
}
```

如果你使用 System.currentTimeMillis() 等計算時間函數來評估函式執行時間，你會發現新的程式比舊的版本要快上數倍。

計算高耗時函式執行時間的模版

```
long start=System.currentTimeMillis(); // 計算處理時間 (起始)
高耗時函式();
long end=System.currentTimeMillis(); // 計算處理時間 (結束)
long time=(end-start); // 計算處理時間
javax.swing.JOptionPane.showMessageDialog(null，"執行時間= "+time+" ms");
```

改裝 ImageComponent

把 ImageComponent 類別改裝成高效率版本一點也不困難。 只要在建立 BufferedImage 時， 重新的使用整數資料的方式建立自己的 BufferedImage 取代就可以了。 馬上看範例， 修改的部分非常短， 但是得到的好處卻很多!!

ImageComponent 類別效率修改版

```
class ImageComponent implements Runnable， ActionListener{
    // 影像原始資料 (1 維整數陣列)
      int[] data;
```

```
    String Filename; BufferedImage image;
    JFrame f;    int Height，Width;

    public ImageComponent(String aFilename){
        Filename=aFilename;
        image=LoadImage(Filename);
        Height=image.getHeight();
        Width=image.getWidth();

        // == 修改的部分 ==
        // 取出影像資料
        data=image.getRGB(0，0，Width，Height，null，0，Width);
        image=CreateBufferedImage_Direct(data，Height，Width);
        // == 結束修改 ==
    }
    // 後面的程式全部不用修改，
    // 如果要做影像處理， 直接針對 data 做即可.
}
```

簡單的影像處理

紅色濾鏡

上期我們使用 C# 實作紅色濾鏡， 這次我們使用同樣的方法應用在 Java 的應用程式中。 複習一下， 紅色濾鏡的實作就是只保留像點的紅色資料， 其他顏色資料保持爲 0。 其他顏色濾鏡方法依此類推， 就不在贅述了。 下面是紅色濾鏡的 Java 程式作法。

紅色濾鏡處理方法

```
public void RedFilter(){
    for(int y=0;y<Height;y++){
        for(int x=0;x<Width;x++){
            int offset=y*Width+x;
            int rgb=data[offset];
            int r=(rgb&0x00ff0000)>>16;
            rgb=(0xff000000|(r<<16));
            data[offset]=rgb;
        }
     }
```

```
}
```

反轉顏色

同樣的顏色反轉基本原理， 即是把原來的像點彩色資料反轉過來。 詳細的作法如下:

反轉顏色處理方法

```
public void ColorInverse(){
            for(int y=0;y<Height;y++){
                for(int x=0;x<Width;x++){
                    int offset=y*Width+x;
                    int rgb=255-data[offset]; //  反轉顏色
                    int r=(rgb&0x00ff0000)>>16;
                    int g=(rgb&0x0000ff00)>>8;
                    int b=rgb&0x000000ff;
                    rgb=(0xff000000|(r<<16)|(g<<8)|b);
                    data[offset]=rgb;
                }
            }
    }
```

有的時候， 使用者可能會希望把處理後的結果用另外一個視窗顯示， 藉以比較處理前後之間的差別。 所以我們的思考如下:

Step 1: 利用目前的影像資料 data 陣列， 建立一個新的 ImageComponent 物件。

Step 2: 呼叫新物件的影像處理函式， 最後顯示出來

即可完成。

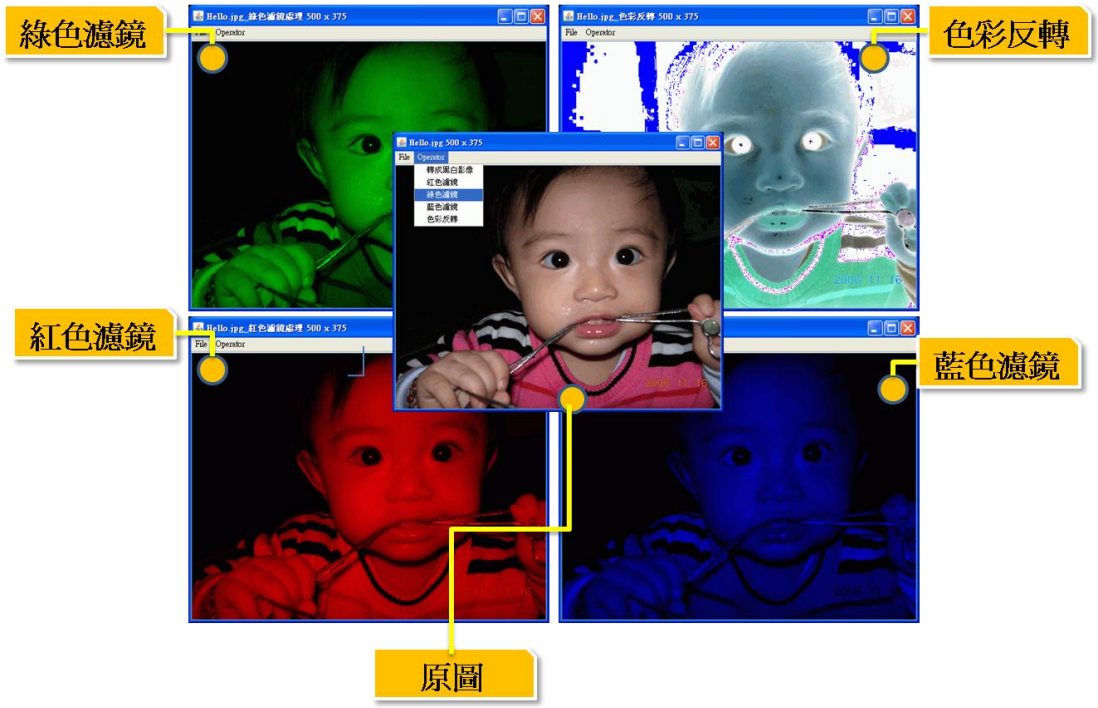
紅色濾鏡處理建立視窗方法

```
public    int[] RedFilter_newWindow(){
// Step 1:
            int[]newdata=new int[data.length];
            System.arraycopy(data，0，newdata，0，data.length);
            ImageComponent grayImage=new ImageComponent(newdata，Height，
Width，"紅色濾鏡處理");

// Step 2:
            grayImage.RedFilter(); //  比較快的方法
```

```
            grayImage.Show();
            return newdata;
}
```

最後將完成處理的資料回傳給使用者。 下面是這次影像處理教學範例程式的主程式功能展示。



圖說: 影像處理主程式功能範例

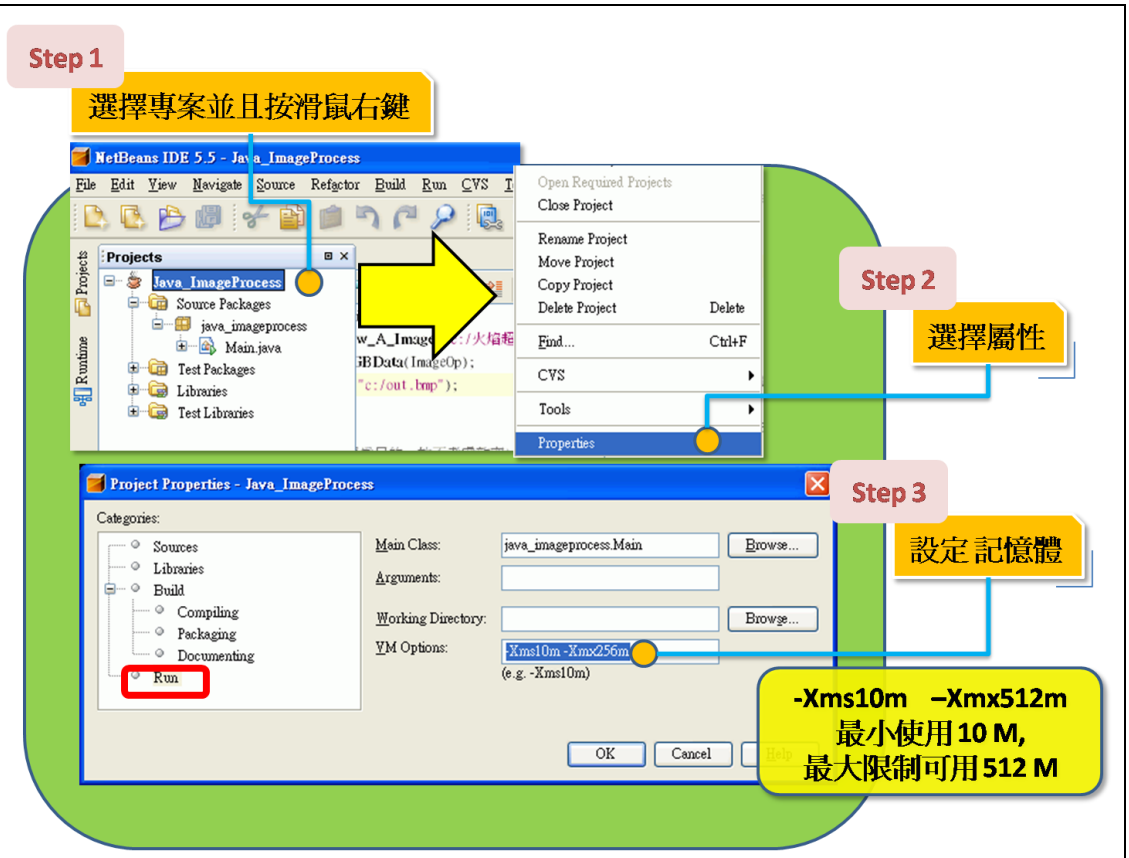
大圖測試 – 處理記憶體不足的問題

一個真正的影像處理軟體， 必須要能夠處理大圖， 所以我們特別建立一個大小爲 3264 x 2448 的圖檔， 進行影像處理。 很快的我們會發現程序在執行時會發生例外!!

OutOfMemoryError 例外訊息

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

發生這個例外的主要原因在於， Java Virtual Machine (JVM)預設的記憶體限定使用量， 不足以讓我們的程式配置更多的記憶體空間來處理大圖。 所以造成這個問題。這個問題可以藉由設定 JVM 的參數來擴大程序可以使用的記憶體容量。我們可以利用 –Xms 參數來指定至少使用的記憶體數量 與 -Xmx 參數來限制程序最多可以配置的記憶體數量。 在 NetBeans 開發環境中， 增加記憶體的設定步驟， 如下所示。



圖說:在 NetBeans 開發環境中增加記憶體使用空間

載入大圖的測試結果

圖說: 載入的大圖測試

影像存檔

Java 的 ImageIO 類別提供了 write 方法， 可以讓我們將 BufferedImage 元件存檔。 下面列出各種常用的圖檔格式存檔範例

```
public void Save(){
        try{
                ImageIO.write(image，"jpeg"，new File("c:\\Result.jpg"));
                ImageIO.write(image，"bmp"，new File("c:\\Result.bmp"));
                ImageIO.write(image，"png"，new File("c:\\Result.png"));
                 ImageIO.write(image，"gif"，new File("c:\\Result.gif"));
           }catch(Exception e){
                javax.swing.JOptionPane.showMessageDialog(null，
                        "存圖錯誤: "+SaveFilename);
                image=null;
           }
     }
```

結論

參考資料:

[1] JBuilder 網址: http://www.codegear.com/tabid/102/Default.aspx

[2] Eclipse 網址: http://www.eclipse.org/

[3] NetBeans 網址: http://www.netbeans.org/