

跨平台視窗程式設計-使用 Ruby 與 GTK+

摘要

Write Once, Run Anywhere, 一直以來都是開發人員的美夢。Java 提出了這個賣點, 但是 VM 的速度太慢以及 Java 複雜的物件架構與囉唆的寫法, 使得開發人員總是望之卻步。在本文中介紹了另種方式, 使用直譯式腳本語言 Ruby, 與 GTK+ 所提供的 GUI 環境, 來快速的開發跨平台的視窗程式設計。

Abstract

For a long time, "Write Once, Run Anywhere" is always the dream for software developers. The primitive feature of Java satisfies this dream, but the dream accompanies the performance issues caused by virtual machine. And sharp curve of Java programming learning let software developers can't get a quick start of Java. In this article, we introduce another interpreter-based program language "Ruby" and its GTK+ development environment to develop the cross-platform windows program rapidly.

關鍵字 (Keywords)

- 使用者圖形介面 (GUI)
- 跨平台 (cross-platform)
- GTK+
- Ruby

1. GTK+

GTK+最初是為發展 GIMP (GNU Image Manipulation Program) 此著名圖形處理軟體而誕生的成果一稱為“The GIMP Toolkit”。如今 GTK+被大量的使用在各種的應用程式之中, 並且也使用在 GNU 的“GNOME”桌面環境之中。

GTK+是一個用來建立 GUI 的跨平台開發工具。GTK+提供了完整的視窗元件 (widgets), 適合各種大小規模的程式來使用。

GTK+是 GNU 計畫中的一個自由軟體。授權方式為 GNU LGPL, 可讓所有的開發者來使用, 亦能用來發展私有軟體, 同時不用收取任何的授權或版權費用。

GTK+由下列 3 個函式庫所組成:

- Glib 是個組成 GTK+與 GNOME 的低階核心函式庫。
- Pango 負責產生與描繪字型。
- ATK 提供了易操作性的輔助使用界面。

除了 C/C++ 之外, GTK+也提供了與其他語言的界面。如 Perl、Python 與本文提到的 Ruby。另外還提供了可以使用例如 Glade 這類的 GUI 產生工具, 可加速開發程式的速度。

GTK+可執行在以下的環境之上:

- Unix 系列
- GTK+ for Win32
- Gtk MacOS X
- GTK+ on DirectFB
- GTK+ for BeOS

2. Ruby

Ruby 是由松本行弘先生“Matz” (Matsumoto Yukihiro) 於 1993 年開始研發的語言。它是個具有物件導向的直譯式 script 語言, 可以用來快速的開發物件導向的程式。

Ruby 具有以下的特性:

- 語法簡單
- 例外處理

- 物件導向
- 垃圾收集器 (garbage collector)
- 動態載入
- 多執行緒
- 跨平台 (UNIX, DOS, Windows 95/98/Me/NT/2000/XP, MacOS, BeOS, OS/2)
- 可與多種 GUI 函式庫結合 (Tk, GTK+, QT, Motif 等等)

3.Ruby/GTK

由於 Ruby 與 GTK+ 都具有跨平台的特性，所以我們選擇這兩者來開發程式。以下以一個簡單的 Hello World 程式當成範例，並同時與原先 GTK+ 所提供的 C 語言版本的 HelloWorld 範例來作比較。

hello.c

```
01 include <gtk/gtk.h>
02 int main( int argc, char *argv[] )
03 {
04     GtkWidget *window;
05     GtkWidget *button;
06     gtk_init (&argc, &argv);
07     window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
08     button = gtk_button_new_with_label ("Hello World");
09     gtk_container_add (GTK_CONTAINER (window), button);
10     gtk_widget_show (button);
11     gtk_widget_show (window);
12     gtk_main ();
13     return(0);
14 }
```

hello.rb

```
01 require 'gtk2'
02 window = Gtk::Window.new(Gtk::WINDOW_TOPLEVEL)
03 button = Gtk::Button.new('Hello World')
04 window.add button
05 button.show
06 window.show
07 Gtk.main
```

程式說明：

hello.rb:

01 行載入 ruby-gtk 函式庫

02 行產生一個 window，這裡可以注意到 ruby 並不需要事先宣告變數

03 行產生一個 button，與上一行類似使用物件導向的類別.new 方法來產生一個新的物件

04 行在 window 上放上 button，使用 Window 類別的方法 add，參數為 button

05 行顯示 button，使用 Button 類別的方法 show

06 行顯示 window，使用 Window 類別的方法 show

07 行進入 GTK+ 主迴圈

使用 Ruby/GTK 是不是看起來簡單又方便呢？

編譯執行的方法：

hello.c：

C 語言版本的 hello.c，必須在系統上先安裝 c 的編譯器如 gcc，以及開發 GTK+ 相關的函式庫和標頭檔。

在 linux 底下使用以下的方式來編譯與執行：

```
$ gcc -o hello hello.c `pkg-config --cflags --libs gtk+-2.0`
$ ./hello
```

hello.rb

Ruby 版本的 hello.rb, 由於 Ruby 是直譯式的 Script 語言, 所以只需要事先安裝 Ruby 與 GTK+的執行環境與函式庫, 通常都可包裝成安裝程式, 直接安裝即可執行。

```
$ruby ./hello.rb
```

4.GTK+的事件驅動與 *callback* 函數

事實上當執行上述的 hello 或 hello.rb 時, 就算使用者把視窗關掉, 程式依然還在執行, 沒有結束, 只能用 ctrl+c 來中斷它。這是因為程式最後執行到 GTK.main 的事件處理迴圈之中, 只有當它執行了 GTK.main_quit 函數之後, 才會從 GTK.main 中離開。

這裡所指的事件包含了使用者的輸入, 滑鼠的移動或視窗大小的改變或需要重繪。詳細的資料請參考 GTK+的官方網站。在使用 C 語言來撰寫 GTK+程式時, 需要寫許多繁雜且超長的函數宣告。接著看看使用 Ruby/GTK 的狀況 :

helloworld.rb

```
01 #!/usr/bin/env ruby
  =begin
    helloworld.rb - Ruby/GTK first sample script.
    Copyright (c) 2002,2003 Ruby-GNOME2 Project Team
    This program is licenced under the same licence as Ruby-GNOME2.
    $Id: helloworld.rb,v 1.4 2003/02/01 16:46:22 mutoh Exp $
  =end
02 require 'gtk2'
03 Gtk.init
04 button = Gtk::Button.new("Hello World")
05 button.signal_connect("clicked") {
06   puts "Hello World"
07 }
08 window = Gtk::Window.new
09 window.signal_connect("delete_event") {
10   puts "delete event occurred"
11   #true #此行是註解
12   false
13 }
14 window.signal_connect("destroy") {
15   puts "destroy event occurred"
16   Gtk.main_quit
17 }
18 window.border_width = 10
19 window.add(button)
20 window.show_all
21 Gtk.main
```

程式說明 :

helloworld.rb :

02行-08行在“=begin“與“=end“之間為註解。

05行的 button.signal_connect("clicked") 表示當 button 在收到“clicked“信號時必須執行後面這個 {} 區塊中的程式碼, 印出 helloworld, (註: 區塊 block 在 Ruby 之中也是一個物件)。

09行-10行表示在當 window 收到一個當視窗被關掉, 由 window manager 送來的“delete_event“的信號時, 在標準輸出上印出一個訊息。

11行-12行在此區塊的最後一行, Ruby 會將區塊或函數的最後一行的值當成回傳值。回傳 true 的話表示這個事件的處理到此為止已經處理完畢, 而回傳 false 的話則系統會在之後再繼續產生後續的事件。此處的例子就是 window 這個物件的 destroy 事件。

14行-17行負責處理 window 的 destroy 事件。其中第 16 行的 Gtk.main_quit 函數, 會將程式由 GTK.main 迴圈中跳出, 而結束整個程式。

20行的 window.show_all, 表示將 window 這個容器上的所有物件都顯示出來。



5. 更深入了解的 **GTK+** 的事件驅動

```
% irb --simple-prompt
>> require 'gtk2'
=> true
>> b = Gtk::Button.new("hoge")
=> #<Gtk::Button:0x40a2a858 ptr=0x8237df8>
>> b.signal_connect("clicked") { puts "block1" }
=> 1 #此回傳值 1 為上行區塊的代號
>> b.signal_connect("clicked") { puts "block2" }
=> 2 #此回傳值 2 為上行區塊的代號
```

irb 是 Ruby 的交談式執行環境，在 >> 符號之後輸入 Ruby 敘述之後，會馬上顯示輸出結果與回傳值。在上面的程式之中，button b 的 "clicked" 事件會連結到兩個程式區塊，其內部代號分別為 1 與 2。

```
>> b.signal_emit("clicked")
block1
block2
=> nil
```

除了使用者本身與系統所發出的信號，物件也能夠使用 signal_emit 函數，自己發出特定的信號來觸發事件處理流程。

```
>> b.signal_handler_disconnect 1
=> #<Gtk::Button:0x40a2a858 ptr=0x8237df8>
>> b.signal_emit("clicked")
block2
=> nil
```

使用 signal_handler_disconnect 1 函數可以動態的刪除物件與事件處理區塊 1 的連結。

```
>> b.signal_handler_block 2
=> #<Gtk::Button:0x40a2a858 ptr=0x8237df8>
>> b.signal_emit("clicked")
=> nil
>> b.signal_handler_unblock 2
=> #<Gtk::Button:0x40a2a858 ptr=0x8237df8>
>> b.signal_emit("clicked")
block2
=> nil
```

另外還能夠使用 signal_handler_block 2 暫時的取消物件與事件處理區塊 2 的連結，接著使用 signal_handler_unblock 2 恢復。

```
def callback1(widget)
  puts "Hello - #{widget.label}(#{widget}) was pressed."
end
button1.signal_connect("clicked") do |w|
  callback1(w)
end
button2.signal_connect("clicked") do |w|
  callback1(w)
end
```

事件處理區塊的另種寫法，用 do |var| ...end。在 |var| 中的變數，代表前面觸發事件的物件。當成參數傳給 callback1 函數來使用。

6. **GTK+** 視窗元件的擺設方式

在以上的程式中，我們只有在視窗中放入一個按鈕，當 window 被調整大小時，button 會自動的縮放

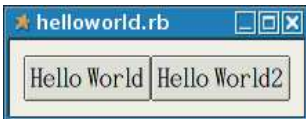
成與 window 同樣的大小。那如果我們要在一個 window 上擺放多個 widget 呢？當直接在產生一個 button 2 並且使用 window.add(button2)時，GTK+會顯示一個警告訊息告知 GtkWidget 此類別的容器之上只能擺置一個 widget。而在執行結果的畫面上也仍然只會出現第一個加入的 widget 也就是 button1。

GTK+提供了下列幾種的 box 容器處理 widget 的排列：

- HBox：將 widget 以水平的方式排列，並自動調成其大小。
- VBox：將 widget 以垂直的方式排列，並自動調成其大小。
- Fixed：將 widget 以固定座標的方式擺放，當視窗改變大小時並不會改變 widget 的大小。
- Table：將 widget 以表格的方式排列，並自動調成其大小。

使用的方式如下：

```
box1 = Gtk::HBox.new(false, 0)
window.add(box1)
box1.pack_start(button1, true, true, 0)
box1.pack_start(button2, true, true, 0)
window.show_all
```



```
fixedbox=Gtk::Fixed.new
window.add(fixedbox)
fixedbox.put(button1,0,0)
fixedbox.put(button2,50,100)
```



```
table=Gtk::Table.new(2,2,false)
table.attach(button1,0,2,0,1)
table.attach(button2,0,1,1,2)
table.attach(button3,1,2,1,2)
```

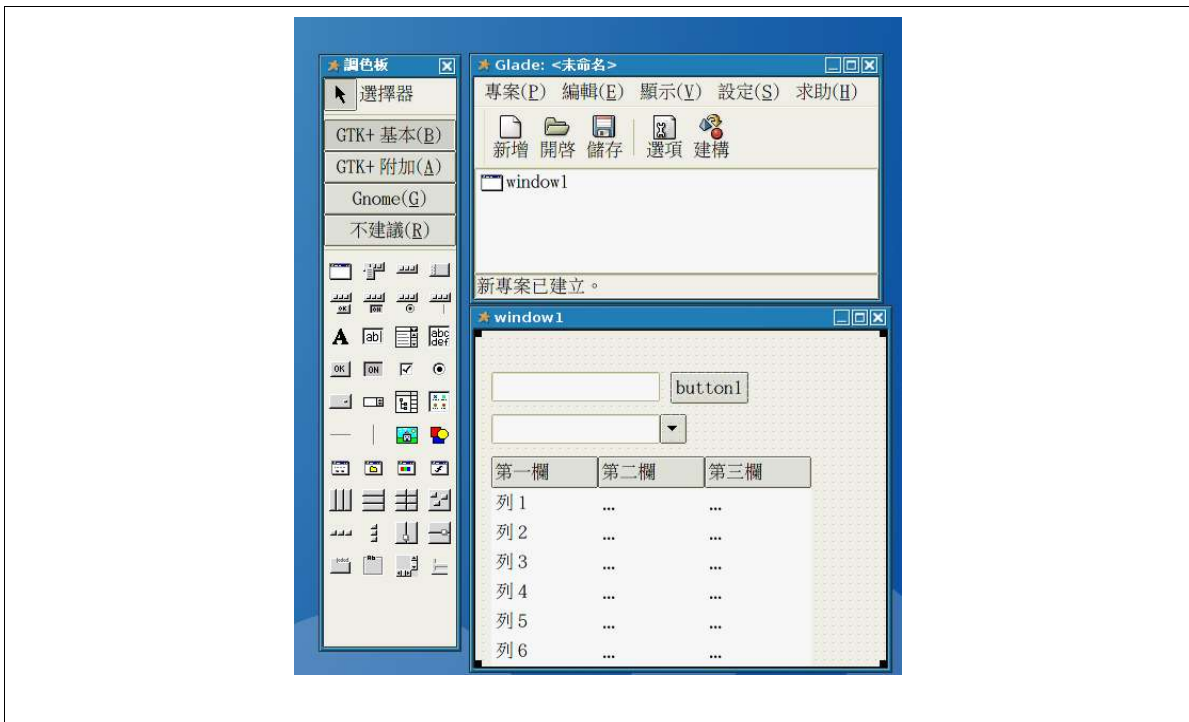


7.使用 Glade 來建立視窗介面

Glade 是一個用來建立 GTK+視窗介面的自由軟體。使用 Glade 可以免去手工建立與放置 widget 的麻煩，同時也可以建立 c 語言版本程式的框架，事後再補齊相對應的事件處理函數的內容實作。Glade 將產生的介面配置本身以 XML 格式儲存，並提供了 libglade2 函式庫可與多種語言連結。

Glade2 的執行畫面如下：

主畫面與工具列：



事件處理函數的框架：



8. 在 *Ruby* 中使用 *libglade2* 來載入事先設計的介面

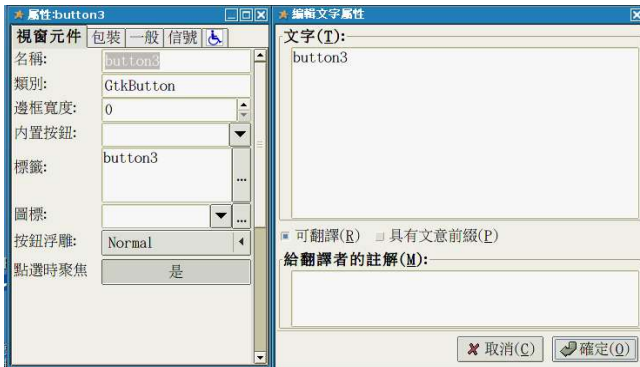
在使用 Glade2 設計好介面與程式框架之後，可以使用由 ruby-libglade2 套件中所提供的 ruby-glade-create-template 程式來將 .glade 檔案轉換成 Ruby 檔案。使用方法如下：

```
$ruby-glade-create-template yourgladefile.glade > yourapp.rb
$ruby yourapp.rb
```

不過當你按下視窗上的按鈕時，不會有任何的反應，接下來只要再補齊事件處理函數內部的實作即可。

9. 使用 *Ruby-Gettext* 與 *Ruby-libglade2* 來建立支援國際化(*internationalization-i18n*) 多國訊息的程式

在使用 glade2 來設計介面時，必須注意到 widget 的屬性是否如下圖所示的有勾選可翻譯此選項：



回到與剛剛產生的 `yourapp.glade` 和 `yourapp.rb` 的所在目錄，首先修改 `yourapp.rb` 中的

```
#PROG_NAME = "YOUR_APPLICATION_NAME"#  
#改成與檔名相同  
PROG_NAME = "yourapp"
```

執行以下的步驟來產生訊息檔案的範本：

```
rgettext yourapp.glade yourapp.rb -o yourapp.pot
```

執行以下的步驟來建立英文的訊息檔案

```
LANG=en_US msginit
```

會產生一個 `en_US.po` 檔案，將他轉成二元的 `mo` 檔案並且複製到 `/usr/share/locale/en_US/LC_MESSAGES/` 目錄之下：

```
rmsgfmt en_US.po -o yourapp.mo  
cp yourapp.mo /usr/share/locale/en_US/LC_MESSAGES/
```

執行以下的步驟來建立繁體中文的訊息檔案：

```
LANG=zh_TW msginit
```

會產生一個 `zh_TW.po` 檔案。修改 `zh_TW.po` 的內容（使用支援 `utf8` 的編輯器或事後再轉成 `utf8` 格式），再將他複製到 `/usr/share/locale/zh_TW/LC_MESSAGES/` 目錄之中：

```
zh_TW.po 的部份內容：  
#: yourapp.glade:49  
msgid "button3"  
msgstr "按鈕 3"  
  
轉成 utf8 格式：  
#iconv -f big5 zh_TW.po -t utf8 > zh_TW.utf8.po  
  
將 po 檔案轉成二元的 mo 檔案  
#rmsgfmt zh_TW.utf8.po -o yourapp.mo  
  
複製 mo 檔到中文語系目錄之中  
#cp yourapp.mo /usr/share/locale/zh_TW/LC_MESSAGES/
```

在不同的語系環境之下的執行狀況：

在英文語系之下執行：
#LANGUAGE=en_US ruby yourapp.rb



在繁體中文語系之下執行：
#LANGUAGE=zh_TW ruby yourapp.rb



10. 結語

GTK+的相關資源豐富，除了內建的 widget 之外，還可以配合其他的函式庫來發展更強大更複雜的軟體。例如：Ruby/GdkPixbuf 圖型處理函式庫，Ruby/GStreamer 可播放多媒體影音內容，Ruby/GtkMozEmbed 甚至可以將 mozilla 的核心-Gecko 放入 GTK+ 容器之中，輕輕鬆鬆的就可以寫出一個瀏覽器。Ruby 的語法簡潔，開發迅速。使用 Ruby/GTK 對於想要發展跨平台視窗程式設計的開發人員，是個不錯的選擇。

參考資料：

- [1] Ruby 官方網站：<http://www.ruby-lang.org>
- [2] GTK+官方網站：<http://www.gtk.org>
- [3] Ruby-Gnome2 網站：<http://ruby-gnome2.sourceforge.jp>
- [4] Glade2 網站：<http://glade.gnome.org>

作者資訊：

徐振華
現任職於工研院資通所網際網路系統技術部副工程師。
Email: sjh6537@itri.org.tw