程式人 月刊雜誌

Programmer

012378

捐發票愛心條碼

讀書做善事、寫書做公益 – 歡迎程式人認養專欄或捐出您的網誌

參考價： NT 50 元，如果您喜歡本雜誌，請將書款捐贈公益團體

羅慧夫顱顏基金會 彰化銀行 (009) 帳號： 5234-01-41778-800

程式人雜誌

2014 年 2 月號

本期焦點：Lua 程式語言

程式人雜誌

前言

編輯小語

在本期的「程式人雜誌」中，聚焦的主題是「Lua 程式語言」！

筆者是因為「開放電腦計畫」需要一個極為簡單的程式語言，才開始對 Lua 產生興趣，並研究 Lua 的原始碼的。

由於 Lua 的語法非常簡單，因此很適合當成「編譯器、解譯器、如何設計一個程式語言？」等研究的入門磚，所以筆者企圖利用這種方式闡述「開放電腦計畫」中的軟體面向，用 Lua 做為「開放電腦計畫」的主要程式語言。

當然、本期不只有 Lua 的相關文章，還有更精彩的 Arduino, VB, OpenNI, C 語言等內容，希望讀者會喜歡這期的「程式人雜誌」！

----（程式人雜誌編輯 - 陳鍾誠）

授權聲明

程式人短訊

程式短訊：Lua 語言簡介

Lua 是一個輕量級的程式語言，該語言是由巴西的 Computer Graphics Technology Group (Tecgraf) 這家公司的員工 Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes 等人所創造的。

讀者可能會感到驚訝，為何會有一個來自巴西的程式語言呢？關於這件事，其實是有歷史典故的。很多技術的創新，其實都來自於某個歷史意外，Lua 語言的發明也是如此。

在 1977 年到 1992 年之間，巴西對電腦軟硬體進行了嚴格的貿易管制保護措施，很多需要電腦軟硬體的公司無法買到國外的軟硬體，於是 Tecgraf 這家主攻「電腦繪圖與使用者介面」的公司就只好自行發展「程式語言與開發工具」，並且提供給客戶使用。

在設計出 Lua 之前，Tecgraf 其實已經創造過 SOL (Simple Object Language) 與 DEL (data-entry language) 這兩個程式語言，但這兩個語言並沒有甚麼「流程控制」的語法，於是他們吸取了 SOL 與 DEL 的設計經驗，然後再參考了 「Modula 語言的 控制結構 (if, while, repeat/until)」，以及「CLU 語言的 (多重指定、多傳回值) 等特性」，還有 「SNOBOL、AWK 中的字典 (associative arrays) 概念」，最後創造出了一種語法非常簡易的語言，那就是 Lua。

為何叫做 Lua 呢？Lua 一詞是葡萄牙語中「Luna」（月亮）的意思，因為其前身 SOL 語言的葡萄牙語意義為「太陽」，所以下一個語言就自然叫做「月亮」囉！

Lua 的語法非常的簡單，其 BNF 語法寫起來只有一頁，因此可以說是世界上最簡單的語言之一 (雖然 LISP 的語法比 Lua 更簡單，其 BNF 應該只要兩三行就寫完了，但是寫起來卻很不容易閱讀；而 Lua 則是容易讀容易寫，但是 BNF 語法就比 LISP 稍長了一些)。

由於這種簡易的特性，後來 Lua 被廣泛使用在各種遊戲引擎裏做為「腳本語言」(Script Language) 使用，列表如下：

- 2D 遊戲引擎 : Corona SDK、Love2D、Agen、Blitwizard、Cocos2d-x、EGSL、Grail、MOAI。
- 2.5D 遊戲引擎 : Lavgine。
- 3D 遊戲引擎 : Baja、Glint 3d、Irrlicht、Leadwerks、Spring RTS、Luxinia、Polycode、Cryengine。

另外、Lua 也常被各式各樣需要腳本語言的工具軟體使用，像是 Damn Small Linux、Wikipedia template engine、Wireshark、Nginx、LuaTeX、Adobe Photoshop Lightroom 等軟體裏都有內嵌 Lua 腳本語言。

筆者之所以對 Lua 感興趣，主要是因為我們正在進行「開放電腦計畫」，該計劃採用 KISS (Keep It Simple and Stupid) 的極度簡化原則，希望讓整台電腦的軟硬體設計都達到最簡易的程度。

當我看到 Lua 的 BNF 語法時，感覺非常驚訝。因為我發現自己不可能設計得比 Lua 更好了，於是我決定要在「開放電腦計畫」中以 Lua 做為主要的程式語言，並開始研究 Lua 的編譯器與解譯引擎。

筆者學習 Lua 的主要目的是為了創造開放電腦計畫中的編譯器與解譯器。官方版 Lua 預設的編譯器是用 C 語言寫成的，不過也有其他人寫出用 Lua 寫的 Lua 解譯器與編譯器，以下是相關資訊的網址：

- http://lua-users.org/wiki/LuaInterpreterInLua
- http://lua-users.org/wiki/LuaCompilerInLua

另外、也有像 LuaJIT 與 LLVM-Lua 這樣的立即編譯引擎 (Just in Time Compiler)，或許對「開放電腦計畫」也是有用的，以下是其網址：

- http://luajit.org/
- https://code.google.com/p/llvm-lua/

而且、在 LuaWiki 這個網站中，也有不少關於 Lua 原始碼與實作的研究，您可以從下列網址找到：

- http://lua-users.org/wiki/LuaSource
- http://lua-users.org/wiki/LuaImplementations
- http://lua-users.org/wiki/LuaGrammar

當我開始做這件事情時，竟發現自己深深的被 Lua 迷住，我簡直愛死這個月亮了！

如果您用過 JavaScript，應該會覺得這個語言怎麼這麼簡單又強大。但是當您看過 Lua 之後，可能會覺得 JavaScript 還是有點複雜，因為 Lua 感覺就像是「脫光光」的 JavaScript，沒有任何一點累贅的語法啊！

參考文獻

- Wikipedia:Lua
- TecGraf -- http://lac-rio.com/tecgraf
- http://stackoverflow.com/questions/5053134/what-is-a-good-game-engine-that-uses-lua

程式短訊：Lua 的程式範例

本文的 Lua 程式乃是在 MS. Windows 的 MinGW 環境下執行的結果。

檔案：hello.lua

```
print('Hello World!')
```

執行結果：

```
ccc@ccc-PC /c/lua
$ lua hello.lua
Hello World!
```

範例：註解語法

```
-- A comment in Lua starts with a double-hyphen and runs to the end of the line.

--[[ Multi-line strings & comments
     are adorned with double square brackets. ]]

--[=[ Comments like this can have other --[[comments]] nested. ]=]
```

範例：迴圈語法

```
--condition = true

while condition do
  --statements
end

repeat
  --statements
until condition

for i = first,last,delta do     --delta may be negative, allowing the for loop to cou
nt down or up
  print(i)
end
```

```
for key, value in pairs(_G) do
  print(key, value)
end
```

檔案：fact.lua -- 計算階層 n!

```
function factorial(n)
  local x = 1
  for i = 2,n do
    x = x * i
  end
  return x
end

print("factorial(5)="..factorial(5));
```

執行結果：

```
ccc@ccc-PC /c/lua
$ lua fact.lua
factorial(5)=120
```

檔案：obj.lua -- 物件

```
function Point(x, y)        -- "Point" object constructor
  return { x = x, y = y }   -- Creates and returns a new object (table)
end

array = { Point(10, 20), Point(30, 40), Point(50, 60) }   -- Creates array of points

for i = 1,3 do
  print("point("..array[i].x..","..array[i].y..")")
end
```

執行結果：

```
ccc@ccc-PC /c/lua
$ lua obj.lua
point(10,20)
point(30,40)
point(50,60)
```

另外、Lua 不只可以直接執行，也可以先編譯成中間碼 (bytecode) 之後再執行，如下所示：

```
ccc@ccc-PC /c/lua
$ luac -o obj.lo obj.lua
```

```
ccc@ccc-PC /c/lua
$ lua obj.lo
point(10,20)
point(30,40)
point(50,60)
```

參考文獻

- Wikipedia:Lua

程式短訊：Lua 的 BNF 語法

看完上述的 Lua 程式範例之後，讓我們來看看 Lua 程式語言的語法吧！

Lua 的語法非常精簡，可以列印在單一頁面中，以下是 Lua 的 BNF 語法。

```
    chunk ::= block

    block ::= {stat} [retstat]

    stat ::=  ';' |
         varlist '=' explist |
         functioncall |
         label |
         break |
         goto Name |
         do block end |
         while exp do block end |
         repeat block until exp |
         if exp then block {elseif exp then block} [else block] end |
         for Name '=' exp ',' exp [ ',' exp] do block end |
         for namelist in explist do block end |
         function funcname funcbody |
         local function Name funcbody |
         local namelist [ '=' explist]

    retstat ::= return [explist] [ ';' ]

    label ::= '::' Name '::'

    funcname ::= Name { '.' Name} [ ':' Name]

    varlist ::= var { ',' var}
```

```
    var ::=  Name | prefixexp ‘[’ exp ‘]’ | prefixexp ‘.’ Name

    namelist ::= Name { ‘,’ Name}

    explist ::= exp { ‘,’ exp}

    exp ::=  nil | false | true | Number | String | ‘...’ | functiondef |
         prefixexp | tableconstructor | exp binop exp | unop exp

    prefixexp ::= var | functioncall | ‘(’ exp ‘)’

    functioncall ::=  prefixexp args | prefixexp ‘:’ Name args

    args ::=  ‘(’ [explist] ‘)’ | tableconstructor | String

    functiondef ::= function funcbody

    funcbody ::= ‘(’ [parlist] ‘)’ block end

    parlist ::= namelist [ ‘,’ ‘...’ ] | ‘...’

    tableconstructor ::= ‘{’ [fieldlist] ‘}’

    fieldlist ::= field {fieldsep field} [fieldsep]

    field ::= ‘[’ exp ‘]’ ‘=’ exp | Name ‘=’ exp | exp

    fieldsep ::= ‘,’ | ‘;’

    binop ::= ‘+’ | ‘-’ | ‘*’ | ‘/’ | ‘^’ | ‘%’ | ‘..’ |
          ‘<’ | ‘<=’ | ‘>’ | ‘>=’ | ‘==’ | ‘~=’ |
         and | or

    unop ::= ‘-’ | not | ‘#’
```

以上的 Lua 語法中總有 23 個非終端項目，去除空行不算的話只有 40 行的 BNF 語法，而且高階語言該有的特性 Lua 都不會少，是一個非常完整的程式語言。

對於那些想學習「編譯器」與「設計程式語言」的程式人而言，我相信研究 Lua 會是一件很美好的事情，因為「簡單就是美」啊！

參考文獻

- Wikipedia:Lua
- Lua 5.2 Reference Manual -- http://www.lua.org/manual/5.2/manual.html

程式短訊：如何在 Windows 中用 MinGW 建置 Lua ？

為了學習 Lua 的「編譯器與解譯器」的設計方式，筆者必須要自行建置 Lua 的環境，於是找到 Lua 官網的下載點如下：

- http://www.lua.org/download.html

筆者下載時為 5.2.3 版，但是從官網下載下列網址的壓縮檔後，發現檔案似乎已經損壞，因此無法解開。

- http://www.lua.org/ftp/lua-5.2.3.tar.gz

後來筆者乾脆到所有版本的下載點中：

- http://www.lua.org/ftp/

下載那個所有版本一次打包的檔案如下：

- http://www.lua.org/ftp/lua-all.tar.gz

結果就可以順利的解壓縮了！

於是我將解壓縮後的 lua-5.2.3 版之內容，放到 c:\lua 資料夾中。

接著我參考下列網址的說明：

- http://stackoverflow.com/questions/16641826/how-do-i-build-lua-for-windows-using-mingw-and-msys

在 MinGW 的環境中，使用 make PLAT=mingw 指令編譯 Lua 環境，結果得到下列錯誤訊息：

```
ccc@ccc-PC /c/lua
$ make PLAT=mingw
cc -Wall -O2   -c -o lapi.o lapi.c
make: cc: Command not found
make: *** [lapi.o] Error 127
```

看來是 make 的 cc 指令沒有定義，於是我在 Makefile 最前面一行加上 CC=gcc ，存檔後然後再下一次 make PLAT=mingw 指令，結果就可以了：

```
ccc@ccc-PC /c/lua
$ make PLAT=mingw
gcc -Wall -O2   -c -o lapi.o lapi.c
gcc -Wall -O2   -c -o lcode.o lcode.c
gcc -Wall -O2   -c -o lctype.o lctype.c
gcc -Wall -O2   -c -o ldebug.o ldebug.c
gcc -Wall -O2   -c -o ldo.o ldo.c
gcc -Wall -O2   -c -o ldump.o ldump.c
gcc -Wall -O2   -c -o lfunc.o lfunc.c
gcc -Wall -O2   -c -o lgc.o lgc.c
gcc -Wall -O2   -c -o llex.o llex.c
gcc -Wall -O2   -c -o lmem.o lmem.c
gcc -Wall -O2   -c -o lobject.o lobject.c
gcc -Wall -O2   -c -o lopcodes.o lopcodes.c
gcc -Wall -O2   -c -o lparser.o lparser.c
gcc -Wall -O2   -c -o lstate.o lstate.c
gcc -Wall -O2   -c -o lstring.o lstring.c
gcc -Wall -O2   -c -o ltable.o ltable.c
```

```
gcc -Wall -O2   -c -o ltm.o ltm.c
gcc -Wall -O2   -c -o lundump.o lundump.c
gcc -Wall -O2   -c -o lvm.o lvm.c
gcc -Wall -O2   -c -o lzio.o lzio.c
gcc -Wall -O2   -c -o lua.o lua.c
gcc -Wall -O2   -c -o lauxlib.o lauxlib.c
gcc -Wall -O2   -c -o lbaselib.o lbaselib.c
gcc -Wall -O2   -c -o lbitlib.o lbitlib.c
gcc -Wall -O2   -c -o lcorolib.o lcorolib.c
gcc -Wall -O2   -c -o ldblib.o ldblib.c
gcc -Wall -O2   -c -o liolib.o liolib.c
gcc -Wall -O2   -c -o lmathlib.o lmathlib.c
gcc -Wall -O2   -c -o loslib.o loslib.c
gcc -Wall -O2   -c -o lstrlib.o lstrlib.c
gcc -Wall -O2   -c -o ltablib.o ltablib.c
gcc -Wall -O2   -c -o loadlib.o loadlib.c
gcc -Wall -O2   -c -o linit.o linit.c
gcc -o lua lapi.o lcode.o lctype.o ldebug.o ldo.o ldump.o lfunc.o lgc.o llex.o l
mem.o lobject.o lopcodes.o lparser.o lstate.o lstring.o ltable.o ltm.o lundump.o
 lvm.o lzio.o lua.o lauxlib.o lbaselib.o lbitlib.o lcorolib.o ldblib.o liolib.o
lmathlib.o loslib.o lstrlib.o ltablib.o loadlib.o linit.o -lm
gcc -Wall -O2   -c -o luac.o luac.c
gcc -o luac lapi.o lcode.o lctype.o ldebug.o ldo.o ldump.o lfunc.o lgc.o llex.o
lmem.o lobject.o lopcodes.o lparser.o lstate.o lstring.o ltable.o ltm.o lundump.
o lvm.o lzio.o luac.o lauxlib.o -lm
./lua test.lua
Hello from Lua 5.2
./luac -l test.lua

main <test.lua:0,0> (6 instructions at 005CE140)
0+ params, 3 slots, 1 upvalue, 0 locals, 3 constants, 0 functions
        1       [1]     GETTABUP        0 0 -1  ; _ENV "print"
        2       [1]     LOADK           1 -2    ; "Hello from "
        3       [1]     GETTABUP        2 0 -3  ; _ENV "_VERSION"
        4       [1]     CONCAT          1 1 2
        5       [1]     CALL            0 2 1
        6       [1]     RETURN          0 1
```

看來是編譯成功了，編譯完成之後我們可以看到 ls 指令列出的檔案如下，裏面有 lua.exe 與 luac.exe 等兩個檔案，這分別是 Lua 的「解譯器」與「編譯器」。

```
ccc@ccc-PC /c/lua
$ ls
Makefile    lcorolib.c  ldump.o   llimits.h   loslib.c   ltable.h   luac.o
lapi.c      lcorolib.o  lfunc.c   lmathlib.c  loslib.o   ltable.o   luac.out
```

```
lapi.h      lctype.c    lfunc.h     lmathlib.o  lparser.c   ltablib.c   luaconf.h
lapi.o      lctype.h    lfunc.o     lmem.c      lparser.h   ltablib.o   lualib.h
lauxlib.c   lctype.o    lgc.c       lmem.h      lparser.o   ltm.c       lundump.c
lauxlib.h   ldblib.c    lgc.h       lmem.o      lstate.c    ltm.h       lundump.h
lauxlib.o   ldblib.o    lgc.o       loadlib.c   lstate.h    ltm.o       lundump.o
lbaselib.c  ldebug.c    linit.c     loadlib.o   lstate.o    lua.c       lvm.c
lbaselib.o  ldebug.h    linit.o     lobject.c   lstring.c   lua.exe     lvm.h
lbitlib.c   ldebug.o    liolib.c    lobject.h   lstring.h   lua.h       lvm.o
lbitlib.o   ldo.c       liolib.o    lobject.o   lstring.o   lua.hpp     lzio.c
lcode.c     ldo.h       llex.c      lopcodes.c  lstrlib.c   lua.o       lzio.h
lcode.h     ldo.o       llex.h      lopcodes.h  lstrlib.o   luac.c      lzio.o
lcode.o     ldump.c     llex.o      lopcodes.o  ltable.c    luac.exe    test.lua
```

其中的 lua.c 是解譯器 lua.exe 的主程式，而 luac.c 則是編譯器 luac.exe 的主程式。

參考文獻

- Wikipedia:Lua
- Lua 5.2 Reference Manual -- http://www.lua.org/manual/5.2/manual.html
- http://stackoverflow.com/questions/16641826/how-do-i-build-lua-for-windows-using-mingw-and-msys

人物速寫

Lua 的創造者 -- 巴西的 Tecgraf 組織與 Roberto Ierusalimschy 教授

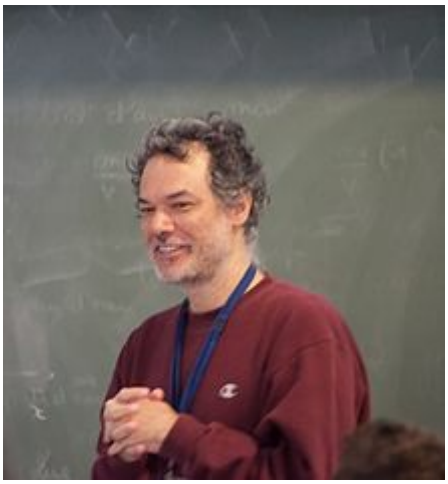
Lua 是由巴西「里約熱內盧」的 Pontifical Catholic 大學 (簡稱 PUC-Rio) 的一個大型實驗室 Tecgraf 所創造的，該創造團隊的核心人物是 Roberto Ierusalimschy 教授，以下是 Roberto 教授與 PUC-Rio 的連結。

- Roberto Ierusalimschy
- Pontifical Catholic University of Rio de Janeiro, Brazil.

Tecgraf 創立於 1987 年五月，與巴西很多公司有合作案。Tecgraf 的研究工作主要是為位電腦圖學與互動程式提供軟體工具，像是繪圖終端機、印表機的驅動程式等等。

由於 1977 年到 1992 年之間巴西對電腦軟硬體進行了嚴格的貿易管制保護措施，因此有很強的動機必須要開發自己的軟硬體。於是 Tecgraf 開發了一系列的語言，包含 DEL, SOL 與 Lua，其中 DEL 與 SOL 可以說是 Lua 的生父與生母，因為 Lua 繼承了不少他們的語法。

由於巴西的官方語言為葡萄牙語，因此很多關於 Lua, Roberto Ierusalimschy 與 Tecgraf 的資料都是用葡萄牙語寫的，這讓筆者在蒐集這些資訊時感到有點困難。



圖、Roberto Ierusalimschy 教授

還好、在 2006 年時，Computer World 雜誌曾經專訪 Roberto 教授，以下是該專訪的連結。

- Computerworld Interview with Roberto Ierusalimschy on Lua

在專訪中 Roberto 教授提到他曾在 2006 年 ACM History of Programming Languages 的一篇論文中提到 Lua 語言的創造過程，以下是該論文的連結。

- The Evolution of Lua (PDF), Roberto Ierusalimschy, Luiz Henrique, Waldemar Celes.

Lua 經常被使用在遊戲引擎中，例如魔獸世界 (World of Warcraft) 就內嵌了 Lua 語言。

另外、Roberto 教授在 SlideShare 上有篇投影片，也很直得參考！

- A brief history of Lua - Roberto Ierusalimschy (PUC Rio).

Roberto 教授在 Lua 上的經驗，或許可以給台灣的學術界一個啟示，不知何時才會有「來自台灣的程式語言」被發明出來，並且廣為世人使用呢？

參考文獻

- http://en.wikipedia.org/wiki/Roberto_Ierusalimschy

影音頻道

看影片學 Lua 程式設計

Lua 的語法很簡單，執行環境也很簡單，如果您用的是 Linux，應該從 Lua 官網上下載 建置一下就可以了，以下是官網所提供的建置方法：

```
curl -R -O http://www.lua.org/ftp/lua-5.2.3.tar.gz
tar zxf lua-5.2.3.tar.gz
cd lua-5.2.3
make linux test
```

如果您用 MS. Windows ，那麼就可以安裝 Lua for Windows，以下是其網址：

- http://luaforwindows.luaforge.net/

下載安裝後，您會發現在「開始/所有程式」裏有個 Lua 的資料夾，裏面有「iLua, Lua Command Line, Lua Examples, QuickLuaTour」等項目，建議您看看「QuickLuaTour」，它會帶領你快速的熟悉 Lua 的語法與範例。

接著您可以直接起動命令列，然後用任何的編輯器，像是「Notepad++」等，開始寫一些簡單的程式，然後直接用 lua 〈程式名稱〉去執行您的程式即可。以下是筆者執行幾個 Lua 程式的過程：

```
D:\Dropbox\Public\pmag\201402\code>lua hello.lua
Hello World!

D:\Dropbox\Public\pmag\201402\code>lua fact.lua
factorial(5)=120

D:\Dropbox\Public\pmag\201402\code>lua obj.lua
10,20
30,40
50,60

D:\Dropbox\Public\pmag\201402\code>lua obj.lua
point(10,20)
point(30,40)
point(50,60)
```

您可以以看看下列 Lua 的影片 ，以便瞭解 Lua 的程式寫法：

影片	連結
Lua Tutorial #1: Introduction and Setup	http://youtu.be/dHURyRLMOK0
Lua Tutorial #2: Hello World	http://youtu.be/aSxoOCn6Y4E
Lua Tutorial #3: Variables and User Input	http://youtu.be/ClThmOGuMi4
Lua Tutorial #4: Basic Mathematics	http://youtu.be/jQ40M1DObl4

Lua Tutorial #5: If and Else	http://youtu.be/vlJftHgeByg

當然、多寫多看，應該是學習程式的不二法門。學程式與學習游泳一樣，只有下水開始扭動身體，才有機會真正學會游泳，也只有真正開始上機寫程式，才有可能真正學會寫程式。對於 Lua 、當然也是如此！

Lua 引擎的解析與使用（作者：陳鍾誠）

本文解析的是 Lua 5.2.3 的原始碼，筆者採用 MinGW 在 MS. Windows 7 上進行編譯測試。

基本型態

Lua 是一種弱型態 (Weakly Typed) 的腳本語言 (Script Language)，就像 JavaScript/Python/Ruby/Perl 一樣，您不需像在 C/C++/Java 等語言中那樣宣告變數型態，但每個變數值其實都有個對應的型態，因為 Lua 會自動決定其型態。

Lua 中的基本型態如下：

型態	說明
string	字串：8 bit 字元、以 結尾。
number	數字：以 C 的 double 為預設型態，可代表正負整數與浮點數等。
nil	空值：類似 C 語言的 NULL。
boolean	布林：真或假。
function	函數：與 javascript 中的函數一樣，都是第一級變數，非次等公民。
userdata	使用者定義資料：任何 C 語言中的資料都可以儲存成 Lua 的 userdata 變數。
table	表格：使用字典 associative array 結構，採用 (key, value) 配對結構。
thread	線程 (執行緒)：執行的基本程式單元。

上述的 table 結構威力強大， userdata 也可封裝成類似 table 的 (key,value) 結構，這種結構可用來表示「樹 (Tree)、圖 (Graph)、甚至是 XML」等複雜結構。

Lua 的「物件導向」也是建構在 table 這樣的 (key, value) 結構之上的，依靠的是一種稱為 Meta-Methods 的概念。您可以將對應到物件裏的某欄位的 (key, value) 結構，也就是 (欄位、值) 的「值」改掉，就能完成像「抽換、繼承、多型、依賴性注射」之類的概念。

Lua 的函數採用傳參照 (call by reference) 的方式，但是卻不允許參數在函數內被修改，還好 Lua 的函數可以傳回多值結果，因此任何被修改的參數都可以透過多值回傳的方式傳回。

Lua 內部的運作方式

由於 Lua 強調可內嵌於其它應用程式的功能，因此允許隨時「解譯、載入、執行」，而且應用程式可以輕易的「捕捉錯誤」，甚至同時執行很多個 Lua 程式等等。

當您想載入 Lua 程式並執行時，通常會經過下列過程：

1. 透過 `luaL_newstate()` 創造出一份「狀態物件」(State)，這個狀態物件代表 Lua 語言。
2. 透過 `luaL_openlibs()` 載入原本的系統函數，或進一步註冊您自訂的 Lua 延伸函數。
3. 透過 `luaL_loadfile()` 載入 Lua 的程式碼，這些程式碼會被轉換成可被虛擬機執行的中間碼 (IR, bytecode)。
4. 透過 `lua_pcall()` 會呼叫虛擬機去執行 Lua 的中間碼 (bytecode)，若執行有錯就會傳回 1，否則會傳回 0。

在上述步驟中，luaL_loadfile() 這個函數會進行「掃描 lex、剖析 parse、編譯、compile」等動作，這些動作採用 「Pipes & Filters」的方式進行。

目前的 Lua 引擎並不採用像 lex 或 yacc 這樣的工具來產生剖析程式碼，而是採用手工打造的剖析器，來進行剖析與中間碼產生的動作。

當您呼叫 lua_pcall() 函數的時候，其實是呼叫虛擬機去執行 Lua 的中間碼 (bytecode)，虛擬機會解譯每個中間碼指令並執行對應的動作。

建置 Lua 解譯器 -- myLua

那麼、要怎麼用 C 呼叫並載入並執行一段「Lua 程式」呢？以下是 lua 原始碼中的解譯器 lua.c 裏的主程式，您可以用這種方法載入並執行 lua 程式。

```
int main (int argc, char **argv) {
  int status, result;
  lua_State *L = luaL_newstate();  /* 步驟1. create state */
  if (L == NULL) {
    l_message(argv[0], "cannot create state: not enough memory");
    return EXIT_FAILURE;
  }
  // 步驟2. 您可以透過註冊函數 pushfunction(L, &func) 的方式註冊延伸函數。
  /* call 'pmain' in protected mode */
  lua_pushcfunction(L, &pmain);
  lua_pushinteger(L, argc);  /* 1st argument */
  lua_pushlightuserdata(L, argv); /* 2nd argument */
  status = lua_pcall(L, 2, 1, 0);
  result = lua_toboolean(L, -1);  /* get result */
  finalreport(L, status);
  lua_close(L);
  return (result && status == LUA_OK) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

以下是一個簡化過的 Lua 解譯器程式：

檔案：myLua.c

```
#include <stdlib.h>
#include <stdio.h>

#include "lua.h"
#include "lauxlib.h"
#include "lualib.h"

void error(lua_State *L, char *msg){
    fprintf(stderr, "\nFATAL ERROR:\n  %s: %s\n\n", msg, lua_tostring(L, -1));
    exit(1);
}

int main(int argc, char *argv[])
{
    lua_State *L = luaL_newstate();                 // 1. 建立 State 狀態機
    luaL_openlibs(L);                               // 2. 載入 Lua 基本函式庫
```

```
    if (luaL_loadfile(L, argv[1]))                  // 3. 載入 argv[1] 指定的 Lua 程式檔 (還
沒執行)
      error(L, "luaL_loadfile() failed");

    if (lua_pcall(L, 0, 0, 0))                      // 4. 執行載入的 Lua 程式
      error(L, "lua_pcall() failed");

    lua_close(L);                                   // 5. 結束
    return 0;
}
```

然後我們修改 Lua 5.2.3 版原始碼的 Makefile 如下 (其中的 CC=gcc, MYOBJS=..., myLua: ..., myLuac:... 等段落是我們加入的)：

```
CC=gcc
OBJS0=lapi.o lcode.o lctype.o ldebug.o ldo.o ldump.o lfunc.o lgc.o llex.o lmem.o lobjec
t.o lopcodes.o lparser.o lstate.o lstring.o ltable.o ltm.o lundump.o lvm.o lzio.o
OBJS= $(OBJS0) lua.o lauxlib.o lbaselib.o lbitlib.o lcorolib.o ldblib.o liolib.o lmathl
ib.o loslib.o lstrlib.o ltablib.o loadlib.o linit.o
OBJS2= $(OBJS0) luac.o lauxlib.o

MYOBJS= $(OBJS0) myLua.o lauxlib.o lbaselib.o lbitlib.o lcorolib.o ldblib.o liolib.o lm
athlib.o loslib.o lstrlib.o ltablib.o loadlib.o linit.o
MYOBJS2= $(OBJS0) myLuac.o lauxlib.o


CFLAGS= -Wall -O2

T= lua

all:    $T luac
    ./$T test.lua
    ./luac -l test.lua

$T: $(OBJS)
    $(CC) -o $@ $(OBJS) -lm

luac:   $(OBJS2)
    $(CC) -o $@ $(OBJS2) -lm

myLua:  $(MYOBJS)
    $(CC) -o $@ $(MYOBJS) -lm

myLuac: $(MYOBJS2)
    $(CC) -o $@ $(MYOBJS2) -lm
```

```
clean:
    rm -f $T $(OBJS) $(OBJS2) core core.* luac.out luac

diff:
    diff ORIG . | grep -v ^Only > DIFFS
```

接著我們就可以進行編譯，以下是筆者的操作過程：

```
ccc@ccc-PC /c/lua
$ make myLua
gcc -Wall -O2   -c -o myLua.o myLua.c
gcc -o myLua lapi.o lcode.o lctype.o ldebug.o ldo.o ldump.o lfunc.o lgc.o llex.o
 lmem.o lobject.o lopcodes.o lparser.o lstate.o lstring.o ltable.o ltm.o lundump
.o lvm.o lzio.o myLua.o lauxlib.o lbaselib.o lbitlib.o lcorolib.o ldblib.o lioli
b.o lmathlib.o loslib.o lstrlib.o ltablib.o loadlib.o linit.o -lm

ccc@ccc-PC /c/lua
$ ls *.lua
fact.lua  hello.lua  obj.lua  test.lua

ccc@ccc-PC /c/lua
$ myLua hello.lua
Hello World!

ccc@ccc-PC /c/lua
$ myLua fact.lua
factorial(5)=120

ccc@ccc-PC /c/lua
$ myLua obj.lua
point(10,20)
point(30,40)
point(50,60)

ccc@ccc-PC /c/lua
$ myLua xxx.lua

FATAL ERROR:
  luaL_loadfile() failed: cannot open xxx.lua: No such file or directory
```

建置 Lua 編譯器 -- myLuac

由於 Lua 定義有中間碼格式，因此也可以將該格式輸出，變成 bytecode。

Lua 環境中的 luac 這個程式就是 Lua 編譯器，可以將 Lua 程式轉換成 bytecode ，但是若我們想在 C 語言裏寫程式將 Lua 程式轉為 bytecode 該怎麼做呢？以下是一個簡易的範例。

檔案：myLuac.c

```c
#include <stdlib.h>
#include <stdio.h>

#include "lua.h"
#include "lauxlib.h"
#include "lualib.h"
#include "lundump.h"
#include "lobject.h"
#include "lstate.h"

#define toproto(L,i) getproto(L->top+(i))

static int writer(lua_State* L, const void* p, size_t size, void* u)
{
 UNUSED(L);
 return (fwrite(p,size,1,(FILE*)u)!=1) && (size!=0);
}

void error(lua_State *L, char *msg){
    fprintf(stderr, "\nFATAL ERROR:\n  %s: %s\n\n", msg, lua_tostring(L, -1));
    exit(1);
}

int main(int argc, char *argv[])
{
    lua_State *L = luaL_newstate();                 // 1. 建立 State 狀態機

    if (luaL_loadfile(L, argv[1]))                  // 2. 載入 argv[1] 指定的 Lua 程式檔 (還沒執行)
      error(L, "luaL_loadfile() failed");

    int stripping=0;
    Proto* f = (Proto*) toproto(L,-1);
    FILE* D= fopen(argv[2],"wb");
    if (D != NULL) {
      lua_lock(L);
      luaU_dump(L,f,writer,D,stripping);            // 3. 輸出編譯後的檔案
      lua_unlock(L);
    }
    lua_close(L);                                   // 4. 結束
    return 0;
}
```

在上述的 Makefile 中，我們已經加入了 myLuac 的編譯指引，因此可以直接編譯了：

```
ccc@ccc-PC /c/lua
$ make myLuac
gcc -Wall -O2   -c -o myLuac.o myLuac.c
gcc -o myLuac lapi.o lcode.o lctype.o ldebug.o ldo.o ldump.o lfunc.o lgc.o llex.
o lmem.o lobject.o lopcodes.o lparser.o lstate.o lstring.o ltable.o ltm.o lundum
p.o lvm.o lzio.o myLuac.o lauxlib.o -lm

ccc@ccc-PC /c/lua
$ myLua hello.lua hello.lo
Hello World!

ccc@ccc-PC /c/lua
$ myLuac hello.lua hello.lo

ccc@ccc-PC /c/lua
$ myLua hello.lo
Hello World!

ccc@ccc-PC /c/lua
$ myLuac fact.lua fact.lo

ccc@ccc-PC /c/lua
$ myLua fact.lo
factorial(5)=120
```

上述執行過程的最後一個指令，是我們用 myLua 去執行 myLuac 編譯出來的 bytecode，所得到的結果。

您可以看到，myLua 除了執行像 fact.lua, hello.lua 的原始程式檔之外，也可以執行編譯後的 bytecode，像是上述的 fact.lo, hello.lo 等等中間檔。

結語

在本文中，我們介紹了 Lua 環境的基本架構，並修改建置檔 Makefile，建立了一個簡易的的 Lua 解譯器程式，透過這樣的方式，讀者應該已經掌握了 Lua 解譯器的建置、連結與執行方式，並且對 Lua 的運作過程有一定的瞭解了。

參考文獻

- Lua Source
- The Lua Architecture - Advanced Topics in Software Engineering (DOC), Mark Stroetzel Glasberg, Jim Bresler, Yongmin Kevin Cho.
- Calling Lua From a C Program (With Snippets)

Arduino入門教學(14) - 以 Amarino 連接 Android 與 Arduino（作者：Cooper Maa）

編輯說明：Amarino 是 Android meets Arduino 的縮寫， 可以用 Android 手機來控制 Arduino 裝置，以下連結有 Cooper Maa 對 Amarino 的投影片介紹。

- Getting Started with Amarino 簡報

這篇說明如何以 Amarino 通過藍芽連接 Android 與 Arduino，可作為 Amarino 入門的第一步

所需材料

- Android 手機一支
- Arduino x1
- bluetooth module x 1

Step 1：安裝 Amarino

到 http://www.amarino-toolkit.net/ 下載底下兩個 App 並安裝到 Android 手機上：

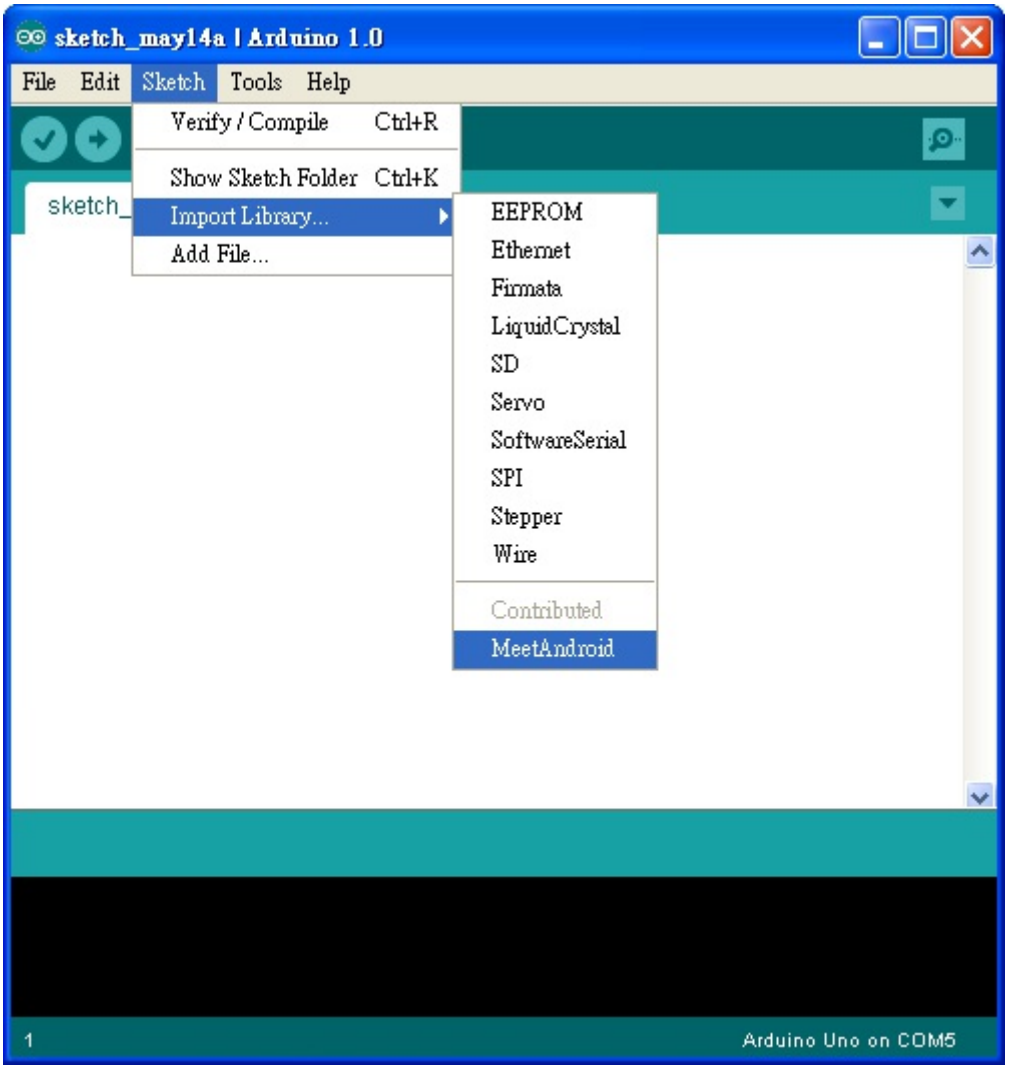
1. Amarino
2. Amarino Plug-in Bundle

Step 2: 安裝 Arduino IDE 與 MeetAndroid Library

如果你電腦上還沒有 Arduino IDE，請先到 http://arduino.cc/en/Main/Software 下載軟體，下載後解壓縮即可。

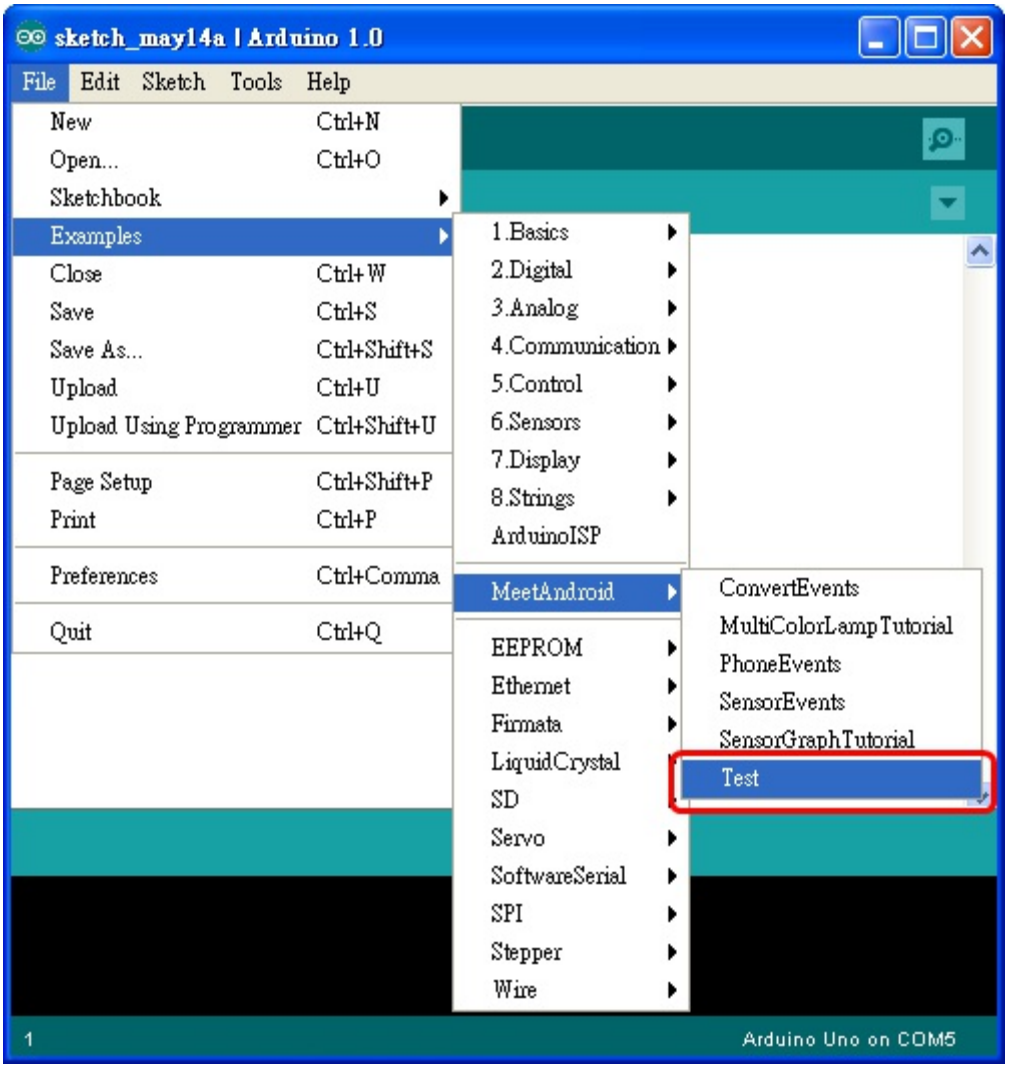
接著下載 MeetAndroid Library ，把 MeetAndroid 解到 Arduino IDE 安裝目錄的 libraries 資料夾下。

重新啟動 Arduino IDE，然後在 Sketch > Import Library 底下應該會看到 MeetAndroid，如下圖：

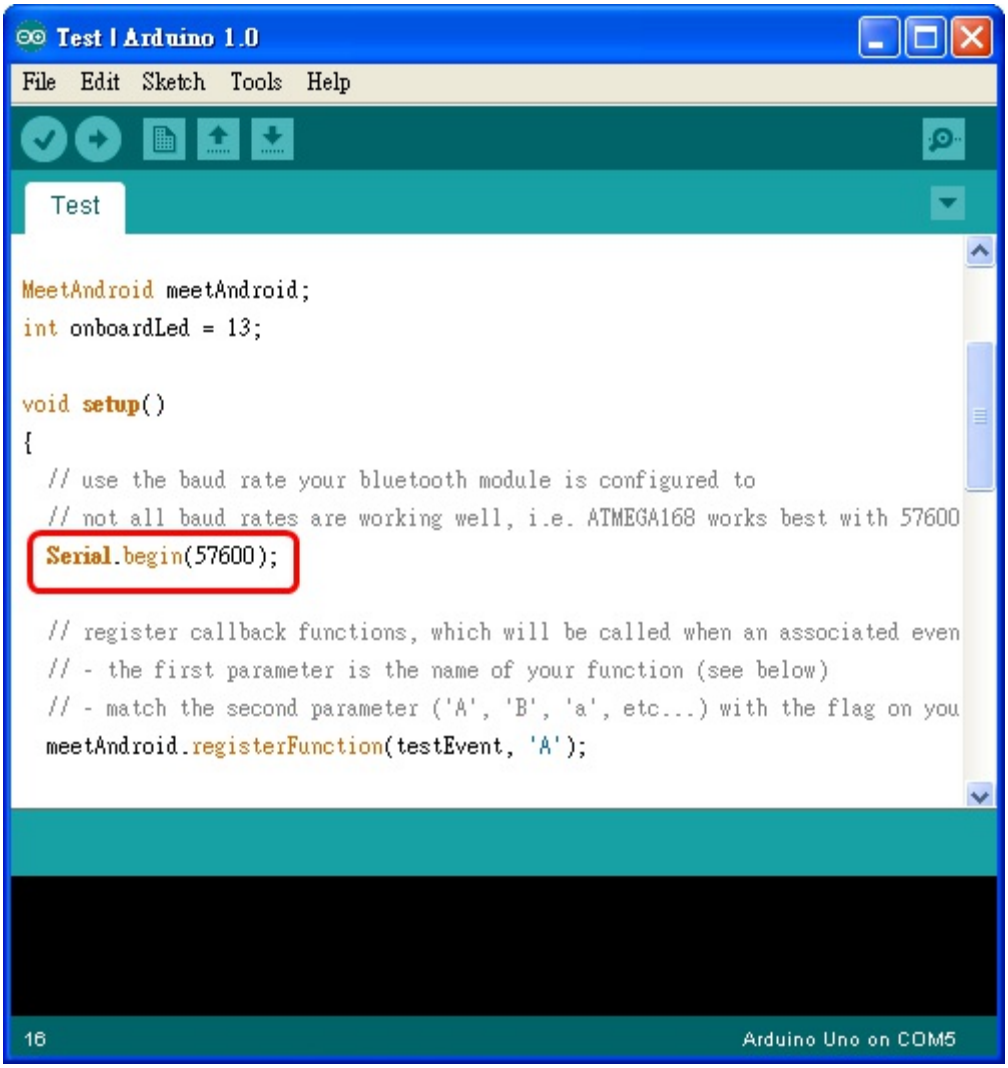


Step 3： 上傳 Test 程式

點 File > Examples > MeetAndroid > Test 打開 Test 程式：



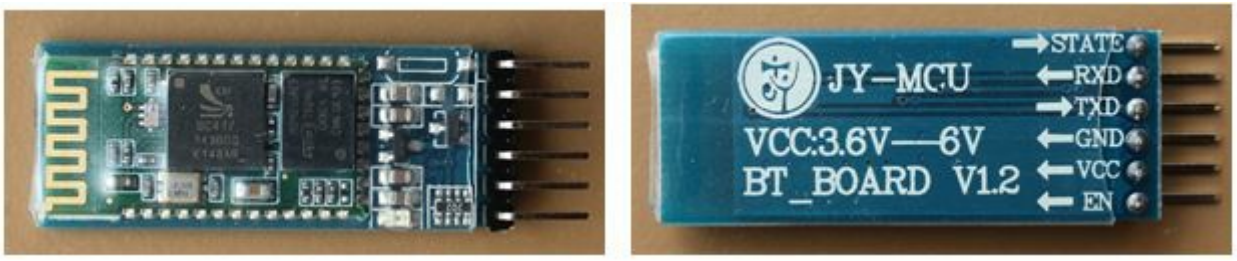
Test 程式所用的 baud rate 預設是 57600 bps，如果你的藍芽模組不是 57600 bps，請做適當的調整：



然後把程式上傳到 Arduino 板子上。

Step 4：連接藍芽模組

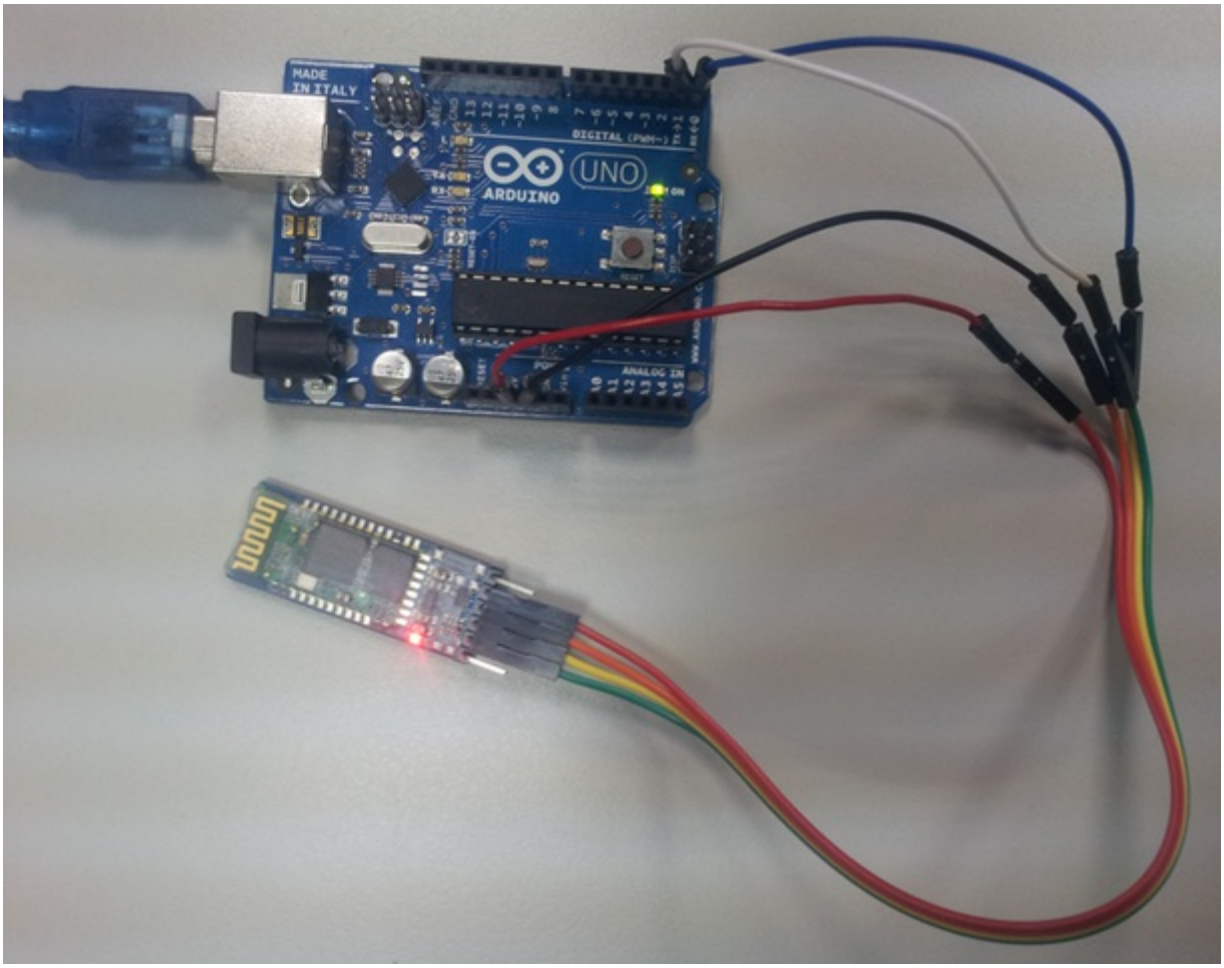
我用的是 廣州匯承信息科技 的 HC-0x 系列藍芽模組，下圖是 HC-0x 藍芽模組的外觀：



▲ HC-0x 藍芽模組 (圖左：正面圖，圖右：背面圖）

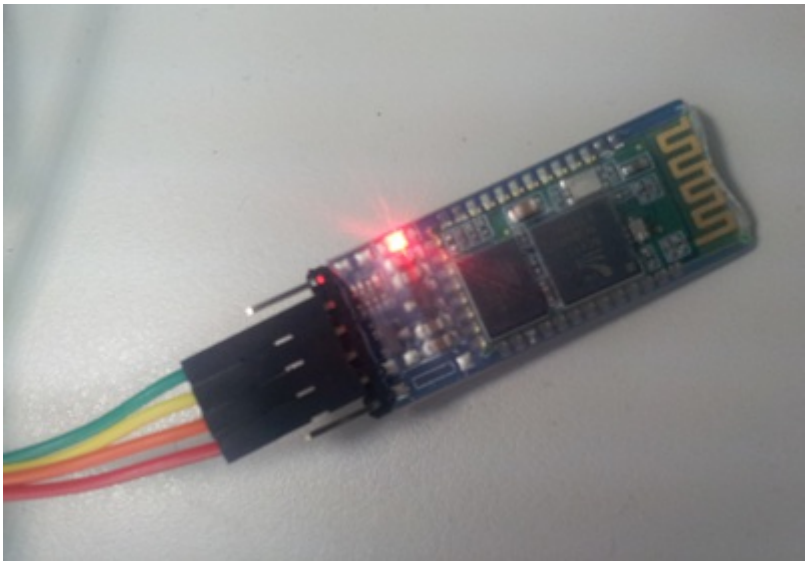
這個藍芽模組連接方法很簡單，只要照下表把 Arduino 和藍芽模組連接起來就好：

Arduino	藍芽模組	備註
5V	VCC	注意電源不可接錯
GND	GND	注意電源不可接錯
RXD	TXD	
TXD	RXD	



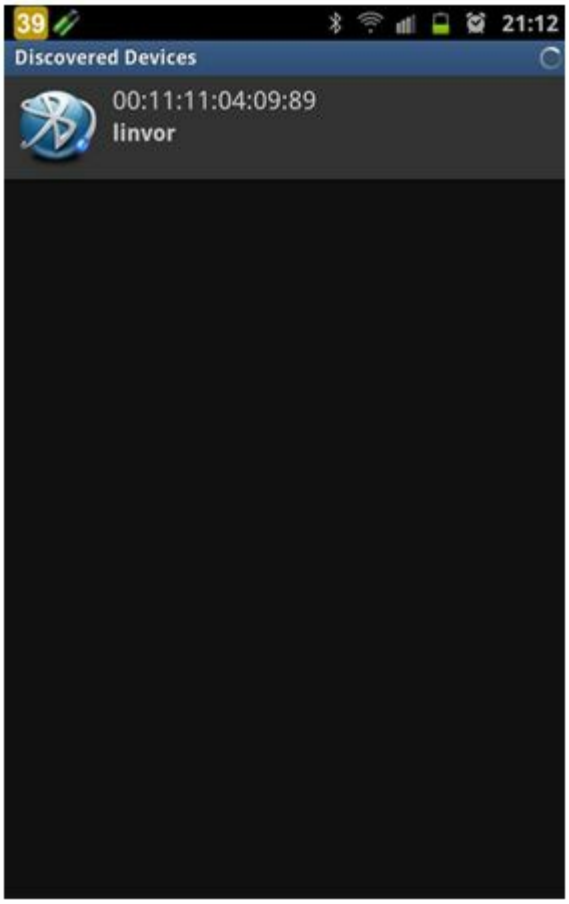
連接的時候有兩點要注意：第一是電源千萬不可接錯，不然藍芽模組可能會壞掉，第二是 Arduino 的 RXD 要接藍芽模組的 TXD，而 Arduino 的 TXD 要接藍芽模組的 RXD。

通電之後，藍芽模組上的 LED 會一直閃爍：

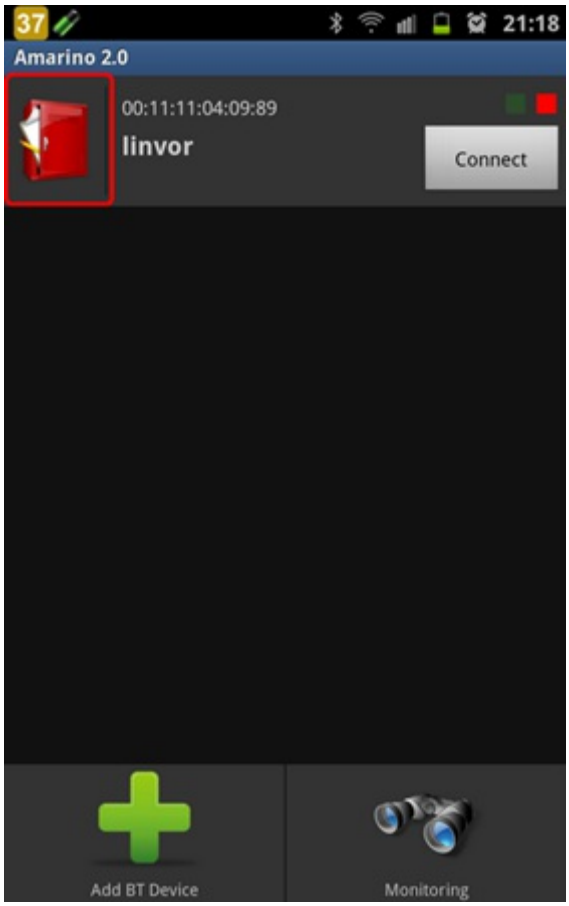


Step 5：測試連線

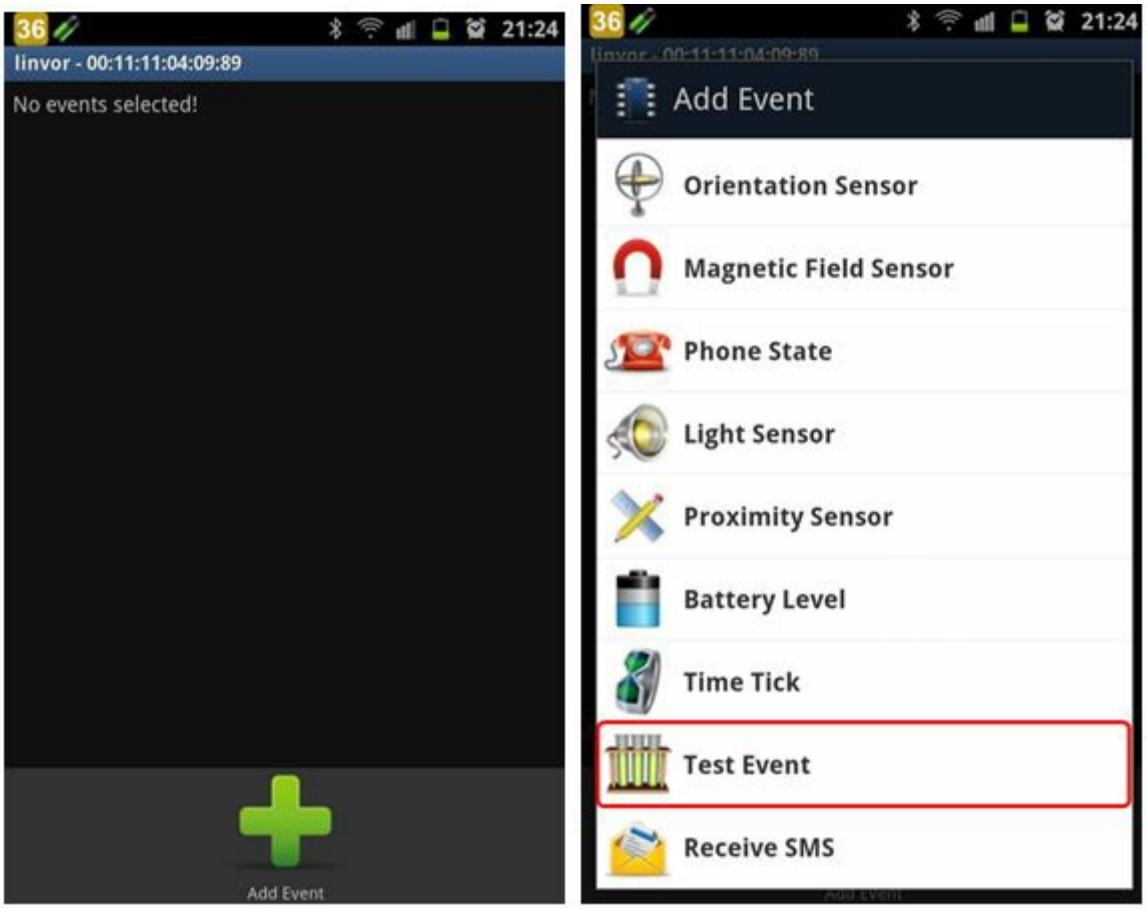
打開 Android 手機上的 Amarino App，點擊【Add BT Device】（如下圖左）搜尋藍芽設備，如果藍芽模組有正確安裝，便會出現在列表中（如下圖右）：



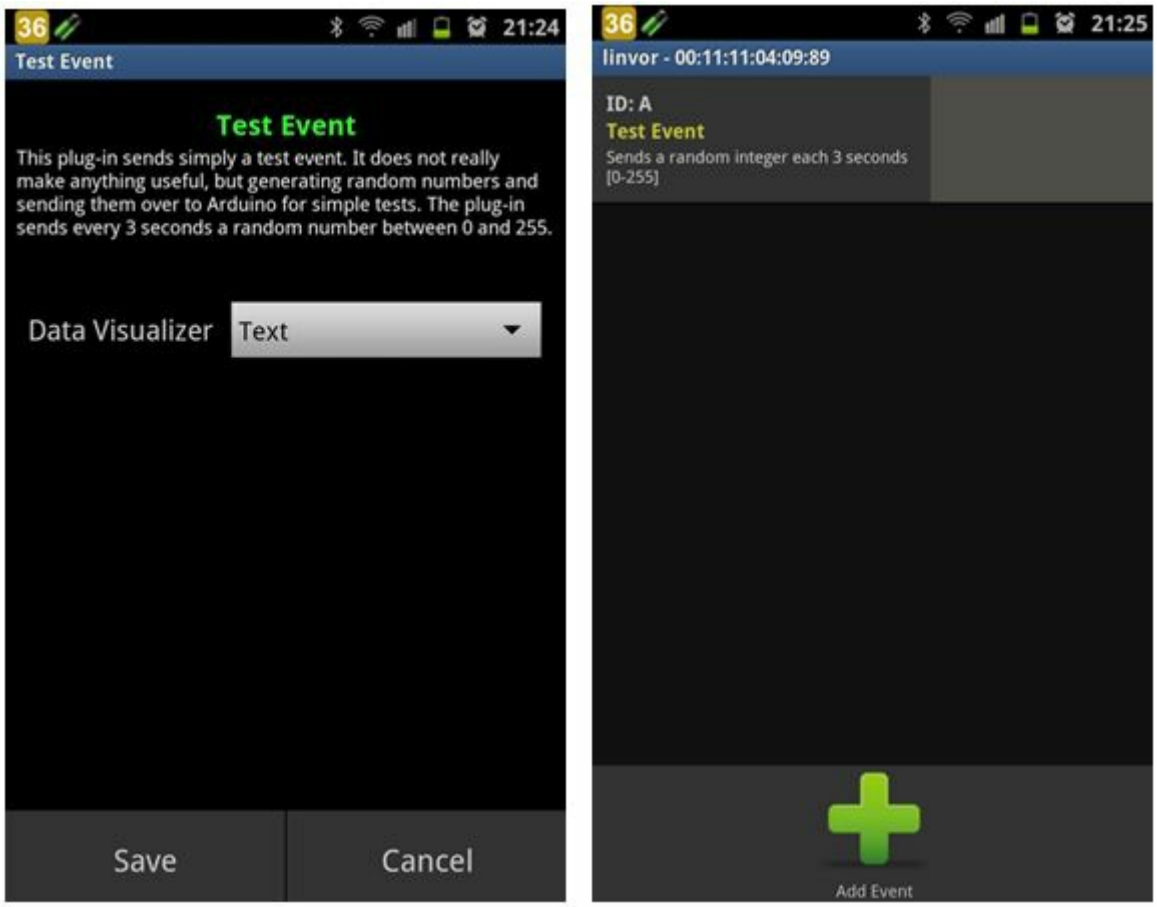
點選找到的藍芽設備，接著會進到設備列表：



點一下上圖紅框框起來的圖示，接著點【Add Event】（如下圖左）顯示可用的 Events，然後點選【Test Event】（如下圖右）：

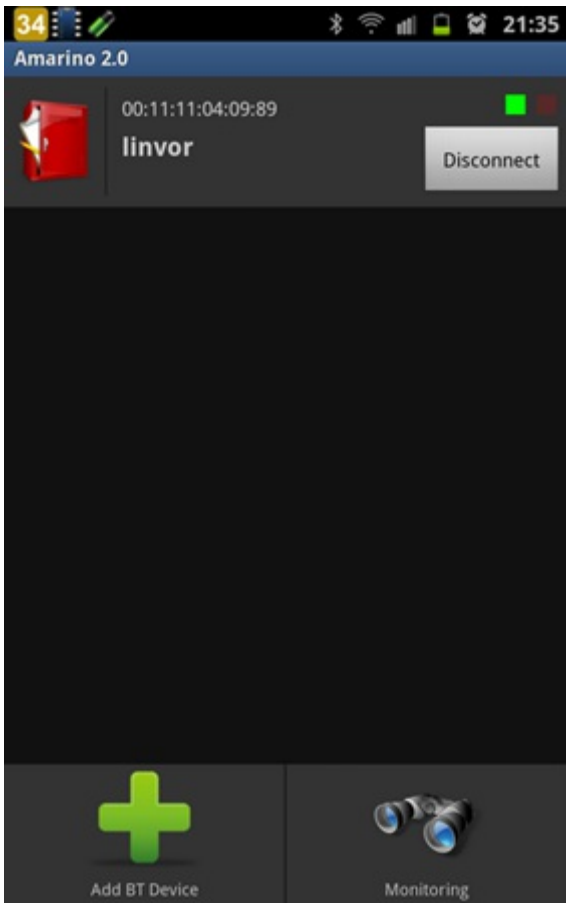


進入設定頁面後，直接點下面的【Save】鈕（如下圖左），此時藍芽設備已增加了一個 Test Event，如下圖右：

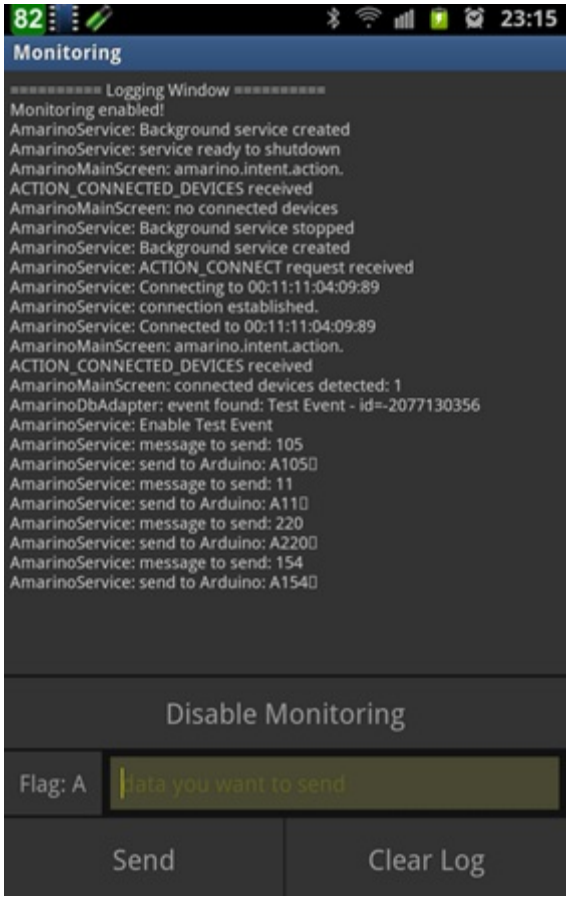


按下手機 Back 鍵回到設備列表畫面（如下圖左），點選【Connect】鈕跟 Arduino 連線，如果是第一次與此藍芽設備連線，Android 手機會出現「藍芽配對要求」的畫面，必須輸入配對密碼（一般為 0000 或 1234）才能與此藍芽設備連線（如下圖右）：

如果成功建立連線（如下圖），Amarino 便會每隔 3 秒發送一個數值介於 0 到 255 的 random number 給 Arduino，假如 Arduino 上的 LED 燈號（在 pin 13 上）會每隔 3 秒閃爍一下，那麼恭禧你，因為你的 Android 手機已經成功和 Arduino 建立通訊了。



如果想知道 Amarino 背景在做什麼，可以點選【Monitoring】鈕以觀察背後的通訊過程：



參考資料

- How to connect my phone to Arduino in 10 steps
- Android talking to Arduino

【本文作者為馬萬圳，原文網址為： http://coopermaa2nd.blogspot.tw/2012/06/amarino-android-arduino.html ，由陳鍾誠編輯後納入本雜誌】

C 語言秘技（3） - 快取記憶體的影響力實驗（作者：陳鍾誠）

前言

話說在周星馳「功夫」這部電影裏，火雲邪神接住子彈之後說道：「天下武功，無堅不破，唯快不破！」。

通常程式人之所以用 C 語言，主要原因有二，一是因為「快」、二是因為「指標在嵌入式系統上的用途」。

但是、同樣是用 C 語言，有些人的程式快如脫兔，而另一些人的程式卻慢如蝸牛，為何會有這樣的差異呢？

要能夠讓 C 語言快，必須瞭解「目標平台的計算機結構」，像是「管線、快取、記憶體管理、堆疊與堆積」等等，有時也要瞭解編譯器會如何編議你的程式。

在本文中，我們將利用一個「矩陣相乘」的範例，說明「快取」與「區域性」這兩個概念對程式速度的影響。

矩陣相乘速度評測

廢話不多說，讓我們直接來看這個「矩陣相乘」的測試程式，看完後再來分析為何會有很多倍的速度差異。

檔案：matrix.c

```
#include <stdio.h>
#include <time.h>

#define N    1000
```

```
#define TYPE int

TYPE A[N][N], B[N][N], C[N][N];

void init() {
  int i,j,k;
  for (i=0; i<N; i++)
    for (j=0; j<N; j++) {
      A[i][j] = i+j;
      B[i][j] = i+j;
      C[i][j] = 0;
    }
}

void mmul_ijk() {
  int i,j,k;
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      for (k=0; k<N; k++)
        C[i][j] += A[i][k] * B[k][j];
}

void mmul_ikj() {
  int i,j,k;
  for (i=0; i<N; i++)
    for (k=0; k<N; k++)
      for (j=0; j<N; j++)
        C[i][j] += A[i][k] * B[k][j];
}

void mmul_jki() {
  int i,j,k;
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      for (i=0; i<N; i++)
        C[i][j] += A[i][k] * B[k][j];
}

void run(void (*mmul)(), char *fname) {
  printf("========= %s ============\n", fname);
  time_t start, stop;
  init();
  start = time(NULL);
  printf("start=%d\n", start);
  mmul();
```

```
  stop  = time(NULL);
  printf("stop =%d\n", stop);
  printf("diff =%d\n", stop-start);
}

int main() {
  run(mmul_ijk, "mmul_ijk");
  run(mmul_ikj, "mmul_ikj");
  run(mmul_jki, "mmul_jki");
}
```

執行結果：

```
D:\c\cache>gcc -O0 matrix.c -o matrix

D:\c\cache>matrix
========= mmul_ijk ============
start=1388743938
stop =1388743953
diff =15
========= mmul_ikj ============
start=1388743953
stop =1388743958
diff =5
========= mmul_jki ============
start=1388743958
stop =1388743989
diff =31
```

您可以看到，mmul_ikj() 只花了 5 秒，比起 mmul_jki() 的 31 秒快上了六倍，究竟為何如此呢？

快取與區域性

在上述程式中，我們宣告了三組 1000*1000 的整數矩陣，每組大約耗用記憶體 1M 的整數大小，在筆者的電腦上，一個整數佔用 4byte，因此總共約耗用 3*4MB=12MB 的記憶體。

但是、筆者電腦的記憶體容量為 4G ，因此三個矩陣都可以完全放在記憶體中。

那麼、為何會有這麼大的速度差異呢？

根據「計算機結構的常識」推斷，原因應該在於快取記憶體，而要讓快取記憶體有效率的方式，在於增強程式的「區域性」(locality)。

筆者用 dxdiag 指令檢視處理器，發現是 AMD Athlon II X4 645 Processor, 3.1 GHZ 的處理器，根據維基百科的資料，該處理器的快取大小如下：

```
L1 cache: 64 kB + 64 kB (data + instructions) per core
L2 cache: 1024 kB
L3 cache: 無
```

因此 12MB 的資料無法完全放在 L2 cache 中，而 L1 cache 中所能放的資料量更少，所以在處理這個大小為 12MB 的矩陣相乘運算時，誰擁有

最好的區域性，誰應該就會有最快的速度。

讓我們先看看以下這個最常見的標準矩陣相乘寫法 mmul_ijk()，您可以看到他的區域性並不會很好，因為最裏層的 B[k][j] 每次 k 都會變動一步，而 B 是 1000*1000 的矩陣，因此相當於每次 B 都要跳上 1000 個整數。 (不過 A[i][k] 的區域性還不錯，C[i][j] 的區域性則很好)。

```
void mmul_ijk() {
  int i,j,k;
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      for (k=0; k<N; k++)
        C[i][j] += A[i][k] * B[k][j];
}
```

所以、執行時上述的標準寫法花了 14 秒，在三者當中效能排行第二。

接著讓我們看看最快的 mmul_ikj()，該程式原始碼如下：

```
void mmul_ikj() {
  int i,j,k;
  for (i=0; i<N; i++)
    for (k=0; k<N; k++)
      for (j=0; j<N; j++)
        C[i][j] += A[i][k] * B[k][j];
}
```

由於最內層變動者為 j，因此 B[k][j] 有很好的區域性，而 A[i][k] 並不太變動，C[i][j] 也有很好的區域性，因此 mul_ikj() 程式的速度區域性最好，所以速度也最快。

最後讓我們來看效能最差的 mmul_jki()，由於 i 在最內層，但每次 A[i][k] 與 C[i][j] 的 i 變動時，都要跳上 1000 格的整數，因此其區域性是最糟的，所以效能也是最差的。

```
void mmul_jki() {
  int i,j,k;
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      for (i=0; i<N; i++)
        C[i][j] += A[i][k] * B[k][j];
}
```

結語

雖然在此我們沒有詳細的討論快取記憶體的結構，但是光從區域性 (locality) 的角度來看，就能清楚的看出哪一個程式執行的最快，由於當今處理器的「快取與記憶體間的速度差異」越來越大 (一般來說可達 100 倍)，所以區域性越好的程式，通常速度也就會快上很多倍，這也是為何上述三個程式的表現差異如此之大的原因了。

因此、當您希望程式跑得快時，最好注意一下「區域性」結構是否良好，這很可能是決定程式效能的關鍵性因素。

參考文獻

- http://en.wikipedia.org/wiki/Athlon_II
- 深入理解计算机系统

資料型態認識—浮點數（single & double）（作者：研發養成所 Bridan）

簡介

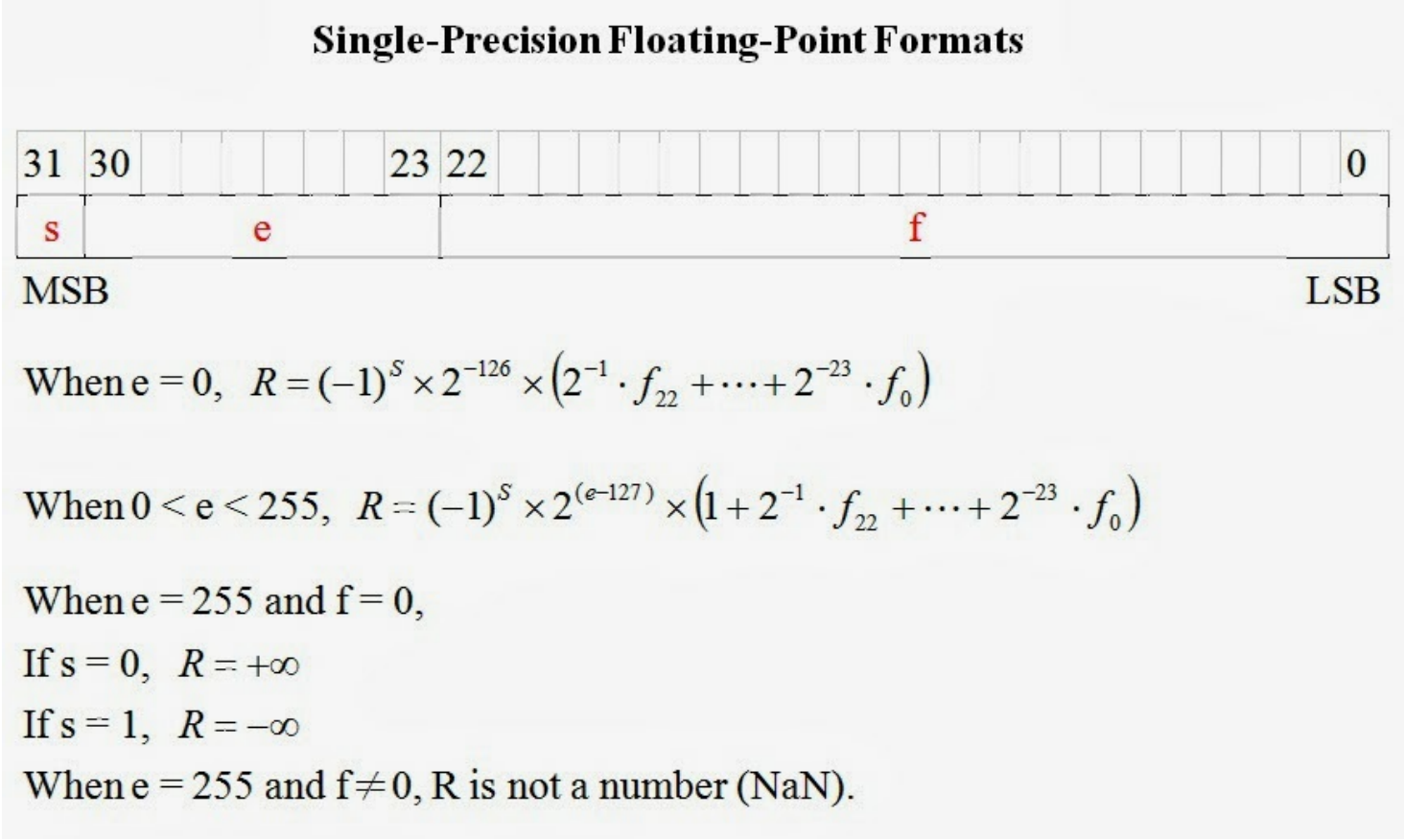
浮點數在電腦系統中是很重要的資料結構，很多程式設計者都知道有這樣的資料結構，但是清楚它的結構的人並不多，因為撰寫高階程式無需注意這些細節，只要會正確宣告變數型態，存在記憶體的資料不要有錯就好，況且高階語言的數學函數庫都很齊備，所以一般程式設計師根本碰不到這樣的問題。

常見二進制浮點數的宣告有兩種(IEEE-754,1985)： 單精準度 single (float)，32 bits，數值範圍 -3.4 x 1038 ～ +3.4 x 1038，最小的正數為 +1.4 x 10-45 雙精準度 double，64 bits，數值範圍 -1.80 x 10308 ～ +1.80 x 10308，最小的正數為 +4.94 x 10-324

標準請參見，

- IEEE-754,1985 : binary floating point arithmetic standard specified
- IEEE-854,1987: standard expanded to include decimal arithmetic

接下來說明單精準浮點數資料格式(32 bits)，當你弄懂後，雙精準數值(64 bits)自然就可以自己推導。



1. 1100 0001 1101 1100 0000 0000 0000 0000B

$$
\begin{aligned}
R &= -1 \times 2^{(131-127)} \times (1 + 2^{-1} + 2^{-3} + 2^{-4} + 2^{-5}) \\
&= -1 \times (2^4 + 2^3 + 2^1 + 2^0 + 2^{-1}) \\
&= -1 \times (16 + 8 + 2 + 1 + 0.5) \\
&= -27.5
\end{aligned}
$$

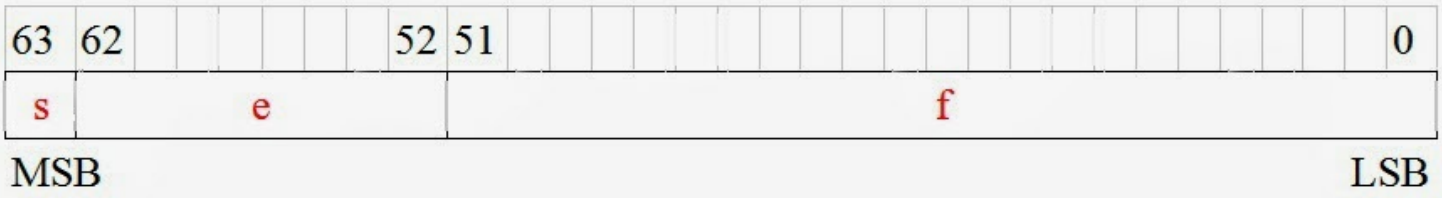
2. 0111 1111 0111 1111 1111 1111 1111 1111B

$$
\begin{aligned}
R_{\max} &= 2^{(254-127)} \times 2 \\
&= 2^{128} \\
&= 3.4 \times 10^{38}
\end{aligned}
$$

3. 0000 0000 0000 0000 0000 0000 0000 0001B

$$
\begin{aligned}
R_{\min} &= 2^{-126} \times 2^{-23} \\
&= 2^{-149} \\
&= 1.4 \times 10^{-45}
\end{aligned}
$$

Double-Precision Floating-Point Formats



When $e = 0$, $R = (-1)^S \times 2^{-1022} \times (2^{-1} \cdot f_{51} + \cdots + 2^{-52} \cdot f_0)$

When $0 < e < 1023$, $R = (-1)^S \times 2^{(e-1023)} \times (1 + 2^{-1} \cdot f_{51} + \cdots + 2^{-52} \cdot f_0)$

When $e = 1023$,

If $s = 0$, $R = +\infty$

If $s = 1$, $R = -\infty$

When $e = 1023$ and $f \neq 0$, R is not a number (NaN).

使用浮點數寫程式有兩種狀況要注意 ，

一、有效位數 — 當你把二進制浮點資料轉換成十進制數值，需要注意它的有效位數，例如 single 有效位數只有六位，為什麼只有六位？自己想一想。

二、尾數誤差 — 這是最常遇到的問題，如果引用數值經過計算或轉換後會產生尾數誤差，也就是十進制數值最後一位有效位數會誤差 ±1。例如，單精準 12.0000 會變成 12.0001，為什麼？如何解決？自己動腦想一想。

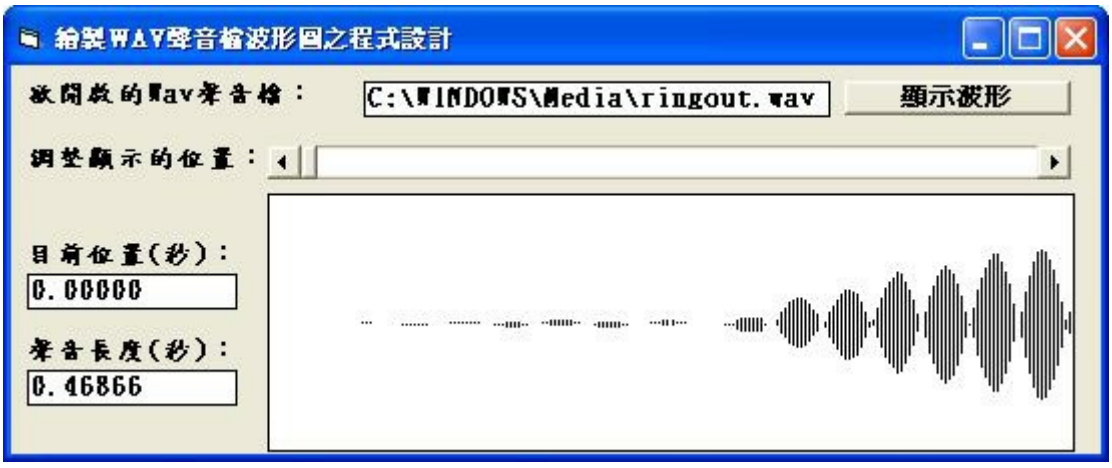
這些都是韌體工程師會遇到的數學難題。

(本文作者 研發養成所 Bridan，原文網址為 http://4rdp.blogspot.tw/2010/03/signle-double.html ，由陳鍾誠編輯後納入程式人雜誌)

[Visual Basic 6.0] 繪製WAV聲音檔波形圖之程式設計（作者：廖憲得 0xde）

[Visual Basic 6.0] 繪製WAV聲音檔波形圖之程式設計

當遇到 WAV、BMP 等檔案處理問題時，就必須使用二進位讀檔。




```
'# [Visual Basic 6.0] 繪製WAV聲音檔波形圖之程式設計
'# 0xDe

Dim WavByteArray(9999) As Byte
Dim DataE
Dim DataF
Private Sub Command1_Click()
Open Text1.Text For Binary As #1

Dim Inp As Byte

I = 0 ' 紀錄目前存放陣列數

Do While Not EOF(1)
    Get #1, , Inp
    WavByteArray(I) = Inp
    I = I + 1
Loop
    Close
' 因為題目說明說（表中數值為16進制數）所以將 16 進位轉換為 10 進位

' 判斷 00～03 Byte 是否為 "RIFF"
' 判斷 08～0E Byte 是否為 "WAVEfmt"
' 判斷 14～15 Byte 是否為 "0100"  表示PCM格式
' 判斷 16～17 Byte 是否為 "0100"  表示單聲道
' 判斷 22～23 Byte 是否為 "0800"  8位元
If ByteToData(&H0, &H3) = "RIFF" And ByteToData(&H8, &HE) = "WAVEfmt" And ByteData(&H1
4, &H17) = "1010" And ByteData(&H22, &H23) = "80" Then

' 取得 18～1B Byte 採樣頻率
' 取得 28～2B Byte 樣本數
DataF = 0 ' 頻率
DataE = 0 ' 樣本數
```

```
For I = 0 To 3
    DataF = DataF + 256 ^ I * WavByteArray(I + &H18)
    DataE = DataE + 256 ^ I * WavByteArray(I + &H28)
Next

Text3 = Format(DataE / DataF, "0.00000") ' 聲音長度
HScroll1.Min = &H2C ' 最小值為 資料開始
HScroll1.Max = HScroll1.Min + DataE  ' 最大值為 資料開始 + 聲音總樣本數

Else
    MsgBox "輸入的檔案名稱不是RIFF WAVEfmt PCM格式 8位元", 16, "輸入的檔案名稱：" & Text1
End If
End Sub

' Byte 轉換為 Data
Function ByteToData(ByteInp As Integer, ByteOut As Integer)
    Data = ""
For I = ByteInp To ByteOut
    Data = Data & Chr(WavByteArray(I))
Next I
    ByteToData = Data
End Function
' Byte 輸出
Function ByteData(ByteInp As Integer, ByteOut As Integer)
    Data = ""
For I = ByteInp To ByteOut
    Data = Data & WavByteArray(I)
Next I
    ByteData = Data
End Function

Private Sub HScroll1_Change()
        Picture1.Cls
        Picture1.Scale (0, 80)-(200, -80)
    For I = 0 To 200
        If HScroll1.Value + I <= HScroll1.Max Then
            Picture1.Line (I, (WavByteArray(HScroll1.Value + I) - 128))-(I, -(WavByteAr
ray(HScroll1.Value + I) - 128))
        Else
            Picture1.PSet (I, 0)
        End If
    Next I

    Text2.Text = Format((HScroll1.Value - 44) / DataF, "0.00000") ' 目前秒數
End Sub
```

- 原始碼下載： Visual Basic 6.0 繪製 Wav 波形.rar

【本文作者為「廖憲得」，原文網址為： http://www.dotblogs.com.tw/0xde/archive/2013/12/04/132528.aspx ，由陳鍾誠編輯後納入本雜誌】

開放電腦計畫（8） - 16 位元微控制器 MCU0 的中斷處理（作者：陳鍾誠）

在上一期當中，我們設計出了出了 MCU0 這顆 16 位元處理器，文章網址如下：

- 開放電腦計畫 (7) – 完整指令集的 16 位元處理器 MCU0s

雖然上述處理器的指令已經算是完整了，但是卻沒有加入中斷機制，本其中我們將說明中斷在 Verilog 中的處理方式。

MCU0 的中斷位元

現代的處理器通常會用中斷的方式來處理「輸出入請求」，或者進行「行程切換」， 在本節中我們將透過微控制器 MCU0 來示範中斷的處理方式。

在 MCU0 中，狀態暫存器裏有個中斷位元，用來代表目前是否正處於中斷狀態中，以下是其定義：

```
`define SW   R[2]      // 狀態暫存器
`define I    `SW[3]    // 是否中斷中
```

當 MCU0 處於中斷狀態時，就無法在接受任何的中斷請求，換言之、MCU0 採用不可重入的中斷機制。

MCU0 的中斷處理

當中斷 interrupt 發生時，會透過 irq 線路傳入中斷代號，此時 MCU0 會檢查是否已經處於中斷狀態， 如果是則忽略此一中斷請求。否則就會進入中斷狀態，將返回位址記錄到 LR 中，然後執行 PC = irq 指令 跳到中斷位址，開始執行中斷處理程式。其主要程式碼如下所示：

```
module cpu(input clock, input interrupt, input[2:0] irq);
...
  always @(posedge clock) begin // 在 clock 時脈的正邊緣時觸發
    IR = m[`PC];                // 指令擷取階段: IR=m[PC], 2 個 Byte 的記憶體
    pc0= `PC;                   // 儲存舊的 PC 值在 pc0 中。
    `PC = `PC+1;                // 擷取完成，PC 前進到下一個指令位址
    case (`OP)                  // 解碼、根據 OP 執行動作
      LD: `A = `M;              // LD C
      ...
      OP8: case (IR[11:8])      // OP8: 加長運算碼
        ...
        IRET: begin `PC = `LR; `I = 0; end          // IRET
    ...
      endcase
    endcase
    // 印出 PC, IR, SW, A 等暫存器值以供觀察
    $display("%4dns PC=%x IR=%x, SW=%x, A=%d SP=%x LR=%x", $stime, pc0, IR, `SW, `A, `S
P, `LR);
    if (!`I && interrupt) begin
      `I = 1;
      `LR = `PC;
```

```
      `PC = irq;
    end
  end
endmodule
```

完整的 MCU0 中斷測試程式

```
`define OP   IR[15:12] // 運算碼
`define C    IR[11:0]  // 常數欄位
`define SC8  $signed(IR[7:0]) // 常數欄位
`define C4   IR[3:0]   // 常數欄位
`define Ra   IR[7:4]   // Ra
`define Rb   IR[3:0]   // Rb
`define A    R[0]      // 累積器
`define LR   R[1]      // 狀態暫存器
`define SW   R[2]      // 狀態暫存器
`define SP   R[3]      // 堆疊暫存器
`define PC   R[4]      // 程式計數器
`define N    `SW[15]   // 負號旗標
`define Z    `SW[14]   // 零旗標
`define I    `SW[3]    // 是否中斷中
`define M    m[`C]     // 存取記憶體

module cpu(input clock, input interrupt, input[2:0] irq);
  parameter [3:0] LD=4'h0,ST=4'h1,ADD=4'h2,SUB=4'h3,MUL=4'h4,DIV=4'h5,AND=4'h6,OR=4'h7,
XOR=4'h8,CMP=4'h9,JMP=4'hA,JEQ=4'hB, JLT=4'hC, JLE=4'hD, JSUB=4'hE, OP8=4'hF;
  parameter [3:0] LDI=4'h0, MOV=4'h2, PUSH=4'h3, POP=4'h4, SHL=4'h5, SHR=4'h6, ADDI=4'h
7, SUBI=4'h8, NEG=4'h9, SWI=4'hA, NSW=4'hD, RET=4'hE, IRET=4'hF;
  reg [15:0] IR;    // 指令暫存器
  reg signed [15:0] R[0:4];
  reg signed [15:0] pc0;
  reg signed [15:0] m [0:4095]; // 內部的快取記憶體
  integer i;
  initial  // 初始化
  begin
    `PC = 0; // 將 PC 設為起動位址 0
    `SW = 0;
    $readmemh("mcu0i.hex", m);
    for (i=0; i < 32; i=i+2) begin
       $display("%x %x", i, m[i]);
    end
  end

  always @(posedge clock) begin // 在 clock 時脈的正邊緣時觸發
    IR = m[`PC];                // 指令擷取階段: IR=m[PC], 2 個 Byte 的記憶體
```

```
    pc0= `PC;                    // 儲存舊的 PC 值在 pc0 中。
    `PC = `PC+1;                 // 擷取完成，PC 前進到下一個指令位址
    case (`OP)                   // 解碼、根據 OP 執行動作
      LD: `A = `M;               // LD C
      ST: `M = `A;               // ST C
      ADD: `A = `A + `M;         // ADD C
      SUB: `A = `A - `M;         // SUB C
      MUL: `A = `A * `M;         // MUL C
      DIV: `A = `A / `M;         // DIV C
      AND: `A = `A & `M;         // AND C
      OR : `A = `A | `M;         // OR  C
      XOR: `A = `A ^ `M;         // XOR C
      CMP: begin `N=(`A < `M); `Z=(`A==`M); end // CMP C
      JMP: `PC = `C;             // JSUB C
      JEQ: if (`Z) `PC=`C;       // JEQ C
      JLT: if (`N) `PC=`C;       // JLT C
      JLE: if (`N || `Z) `PC=`C;// JLE C
      JSUB:begin `LR = `PC; `PC = `C; end // JSUB C
      OP8: case (IR[11:8])       // OP8: 加長運算碼
        LDI:  R[`Ra] = `C4;                          // LDI C
        ADDI: R[`Ra] = R[`Ra] + `C4;                 // ADDI C
        SUBI: R[`Ra] = R[`Ra] - `C4;                 // ADDI C
        MOV:  R[`Ra] = R[`Rb];                       // MOV Ra, Rb
        PUSH: begin `SP=`SP-1; m[`SP] = R[`Ra]; end // PUSH Ra
        POP:  begin R[`Ra] = m[`SP]; `SP=`SP+1; end // POP  Ra
        SHL:  R[`Ra] = R[`Ra] << `C4;                // SHL C
        SHR:  R[`Ra] = R[`Ra] >> `C4;                // SHR C
        SWI:  $display("SWI C8=%d A=%d", `SC8, `A); // SWI C
        NEG:  R[`Ra] = ~R[`Ra];                      // NEG Ra
        NSW:  begin `N=~`N; `Z=~`Z; end              // NSW  (negate N, Z)
        RET:  `PC = `LR;                             // RET
        IRET: begin `PC = `LR; `I = 0; end           // IRET
        default: $display("op8=%d , not defined!", IR[11:8]);
      endcase
    endcase
    // 印出 PC, IR, SW, A 等暫存器值以供觀察
    $display("%4dns PC=%x IR=%x, SW=%x, A=%d SP=%x LR=%x", $stime, pc0, IR, `SW, `A, `S
P, `LR);
    if (!`I && interrupt) begin
      `I = 1;
      `LR = `PC;
      `PC = irq;
    end
  end
endmodule
```

```
module main;                    // 測試程式開始
reg clock;                     // 時脈 clock 變數
reg interrupt;
reg [2:0] irq;

cpu cpux(clock, interrupt, irq);            // 宣告 cpu0mc 處理器

initial begin
  clock = 0;           // 一開始 clock 設定為 0
  interrupt = 0;
  irq = 2;
end
always #10 clock=~clock;    // 每隔 10ns 反相，時脈週期為 20ns

always #500 begin
  interrupt=1;
  #30;
  interrupt=0;
end

initial #4000 $finish;      // 停止測試。

endmodule
```

輸入機器碼與組合語言

```
A00F  // 00            JMP   RESET
A003  // 01            JMP   ERROR
A004  // 02            JMP   IRQ
A006  // 03    ERROR:  JMP   ERROR
F300  // 04    IRQ:    PUSH  A
F301  // 05            PUSH  LR
F302  // 06            PUSH  SW
001E  // 07            LD    TIMER
F701  // 08            ADDI  1
FA01  // 09            SWI   1
101E  // 0A            ST    TIMER
F402  // 0B            POP   SW
F401  // 0C            POP   LR
F400  // 0D            POP   A
FF00  // 0E            IRET
001F  // 0F    RESET:  LD    STACKEND
F230  // 10            MOV   SP, A
001C  // 11    LOOP:   LD    I
```

```
901D  // 12            CMP   N
B01A  // 13            JEQ   EXIT
F701  // 14            ADDI  1
101C  // 15            ST    I
001B  // 16            LD    SUM
201C  // 17            ADD   I
101B  // 18            ST    SUM
A011  // 19            JMP   LOOP
A01A  // 1A    EXIT:   JMP   EXIT
0000  // 1B    SUM:    WORD  0
0000  // 1C    I:      WORD  0
000A  // 1D    N:      WORD  10
0000  // 1E    TIMER:  WORD  0
007F  // 1F    STACKEND: WORD 127
```

執行結果

```
D:\Dropbox\Public\web\oc\code\mcu0>iverilog -o mcu0i mcu0i.v

D:\Dropbox\Public\web\oc\code\mcu0>vvp mcu0i
WARNING: mcu0i.v:29: $readmemh(mcu0i.hex): Not enough words in the file for the
requested range [0:4095].
00000000 a00f
00000002 a004
00000004 f300
00000006 f302
00000008 f701
0000000a 101e
0000000c f401
0000000e ff00
00000010 f230
00000012 901d
00000014 f701
00000016 001b
00000018 101b
0000001a a01a
0000001c 0000
0000001e 0000
  10ns PC=0000 IR=a00f, SW=0000, A=    x SP=xxxx LR=xxxx
  30ns PC=000f IR=001f, SW=0000, A=  127 SP=xxxx LR=xxxx
  50ns PC=0010 IR=f230, SW=0000, A=  127 SP=007f LR=xxxx
  70ns PC=0011 IR=001c, SW=0000, A=    0 SP=007f LR=xxxx
  90ns PC=0012 IR=901d, SW=8000, A=    0 SP=007f LR=xxxx
 110ns PC=0013 IR=b01a, SW=8000, A=    0 SP=007f LR=xxxx
 130ns PC=0014 IR=f701, SW=8000, A=    1 SP=007f LR=xxxx
```

```
 150ns PC=0015 IR=101c, SW=8000, A=    1 SP=007f LR=xxxx
 170ns PC=0016 IR=001b, SW=8000, A=    0 SP=007f LR=xxxx
 190ns PC=0017 IR=201c, SW=8000, A=    1 SP=007f LR=xxxx
 210ns PC=0018 IR=101b, SW=8000, A=    1 SP=007f LR=xxxx
 230ns PC=0019 IR=a011, SW=8000, A=    1 SP=007f LR=xxxx
 250ns PC=0011 IR=001c, SW=8000, A=    1 SP=007f LR=xxxx
 270ns PC=0012 IR=901d, SW=8000, A=    1 SP=007f LR=xxxx
 290ns PC=0013 IR=b01a, SW=8000, A=    1 SP=007f LR=xxxx
 310ns PC=0014 IR=f701, SW=8000, A=    2 SP=007f LR=xxxx
 330ns PC=0015 IR=101c, SW=8000, A=    2 SP=007f LR=xxxx
 350ns PC=0016 IR=001b, SW=8000, A=    1 SP=007f LR=xxxx
 370ns PC=0017 IR=201c, SW=8000, A=    3 SP=007f LR=xxxx
 390ns PC=0018 IR=101b, SW=8000, A=    3 SP=007f LR=xxxx
 410ns PC=0019 IR=a011, SW=8000, A=    3 SP=007f LR=xxxx
 430ns PC=0011 IR=001c, SW=8000, A=    2 SP=007f LR=xxxx
 450ns PC=0012 IR=901d, SW=8000, A=    2 SP=007f LR=xxxx
 470ns PC=0013 IR=b01a, SW=8000, A=    2 SP=007f LR=xxxx
 490ns PC=0014 IR=f701, SW=8000, A=    3 SP=007f LR=xxxx
 510ns PC=0015 IR=101c, SW=8000, A=    3 SP=007f LR=xxxx
 530ns PC=0002 IR=a004, SW=8008, A=    3 SP=007f LR=0016
 550ns PC=0004 IR=f300, SW=8008, A=    3 SP=007e LR=0016
 570ns PC=0005 IR=f301, SW=8008, A=    3 SP=007d LR=0016
 590ns PC=0006 IR=f302, SW=8008, A=    3 SP=007c LR=0016
 610ns PC=0007 IR=001e, SW=8008, A=    0 SP=007c LR=0016
 630ns PC=0008 IR=f701, SW=8008, A=    1 SP=007c LR=0016
SWI C8=   1 A=    1
 650ns PC=0009 IR=fa01, SW=8008, A=    1 SP=007c LR=0016
 670ns PC=000a IR=101e, SW=8008, A=    1 SP=007c LR=0016
 690ns PC=000b IR=f402, SW=8008, A=    3 SP=007d LR=0016
 710ns PC=000c IR=f401, SW=8008, A=    3 SP=007e LR=0016
 730ns PC=000d IR=f400, SW=8008, A=    3 SP=007f LR=0016
 750ns PC=000e IR=ff00, SW=8000, A=    3 SP=007f LR=0016
 770ns PC=0016 IR=001b, SW=8000, A=    3 SP=007f LR=0016
 790ns PC=0017 IR=201c, SW=8000, A=    6 SP=007f LR=0016
 810ns PC=0018 IR=101b, SW=8000, A=    6 SP=007f LR=0016
 830ns PC=0019 IR=a011, SW=8000, A=    6 SP=007f LR=0016
 850ns PC=0011 IR=001c, SW=8000, A=    3 SP=007f LR=0016
 870ns PC=0012 IR=901d, SW=8000, A=    3 SP=007f LR=0016
 890ns PC=0013 IR=b01a, SW=8000, A=    3 SP=007f LR=0016
 910ns PC=0014 IR=f701, SW=8000, A=    4 SP=007f LR=0016
 930ns PC=0015 IR=101c, SW=8000, A=    4 SP=007f LR=0016
 950ns PC=0016 IR=001b, SW=8000, A=    6 SP=007f LR=0016
 970ns PC=0017 IR=201c, SW=8000, A=   10 SP=007f LR=0016
 990ns PC=0018 IR=101b, SW=8000, A=   10 SP=007f LR=0016
1010ns PC=0019 IR=a011, SW=8000, A=   10 SP=007f LR=0016
```

```
1030ns PC=0011 IR=001c, SW=8000, A=    4 SP=007f LR=0016
1050ns PC=0002 IR=a004, SW=8008, A=    4 SP=007f LR=0012
1070ns PC=0004 IR=f300, SW=8008, A=    4 SP=007e LR=0012
1090ns PC=0005 IR=f301, SW=8008, A=    4 SP=007d LR=0012
1110ns PC=0006 IR=f302, SW=8008, A=    4 SP=007c LR=0012
1130ns PC=0007 IR=001e, SW=8008, A=    1 SP=007c LR=0012
1150ns PC=0008 IR=f701, SW=8008, A=    2 SP=007c LR=0012
SWI C8=   1 A=    2
1170ns PC=0009 IR=fa01, SW=8008, A=    2 SP=007c LR=0012
1190ns PC=000a IR=101e, SW=8008, A=    2 SP=007c LR=0012
1210ns PC=000b IR=f402, SW=8008, A=    4 SP=007d LR=0012
1230ns PC=000c IR=f401, SW=8008, A=    4 SP=007e LR=0012
1250ns PC=000d IR=f400, SW=8008, A=    4 SP=007f LR=0012
1270ns PC=000e IR=ff00, SW=8000, A=    4 SP=007f LR=0012
1290ns PC=0012 IR=901d, SW=8000, A=    4 SP=007f LR=0012
1310ns PC=0013 IR=b01a, SW=8000, A=    4 SP=007f LR=0012
1330ns PC=0014 IR=f701, SW=8000, A=    5 SP=007f LR=0012
1350ns PC=0015 IR=101c, SW=8000, A=    5 SP=007f LR=0012
1370ns PC=0016 IR=001b, SW=8000, A=   10 SP=007f LR=0012
1390ns PC=0017 IR=201c, SW=8000, A=   15 SP=007f LR=0012
1410ns PC=0018 IR=101b, SW=8000, A=   15 SP=007f LR=0012
1430ns PC=0019 IR=a011, SW=8000, A=   15 SP=007f LR=0012
1450ns PC=0011 IR=001c, SW=8000, A=    5 SP=007f LR=0012
1470ns PC=0012 IR=901d, SW=8000, A=    5 SP=007f LR=0012
1490ns PC=0013 IR=b01a, SW=8000, A=    5 SP=007f LR=0012
1510ns PC=0014 IR=f701, SW=8000, A=    6 SP=007f LR=0012
1530ns PC=0015 IR=101c, SW=8000, A=    6 SP=007f LR=0012
1550ns PC=0016 IR=001b, SW=8000, A=   15 SP=007f LR=0012
1570ns PC=0017 IR=201c, SW=8000, A=   21 SP=007f LR=0012
1590ns PC=0002 IR=a004, SW=8008, A=   21 SP=007f LR=0018
1610ns PC=0004 IR=f300, SW=8008, A=   21 SP=007e LR=0018
1630ns PC=0005 IR=f301, SW=8008, A=   21 SP=007d LR=0018
1650ns PC=0006 IR=f302, SW=8008, A=   21 SP=007c LR=0018
1670ns PC=0007 IR=001e, SW=8008, A=    2 SP=007c LR=0018
1690ns PC=0008 IR=f701, SW=8008, A=    3 SP=007c LR=0018
SWI C8=   1 A=    3
1710ns PC=0009 IR=fa01, SW=8008, A=    3 SP=007c LR=0018
1730ns PC=000a IR=101e, SW=8008, A=    3 SP=007c LR=0018
1750ns PC=000b IR=f402, SW=8008, A=   21 SP=007d LR=0018
1770ns PC=000c IR=f401, SW=8008, A=   21 SP=007e LR=0018
1790ns PC=000d IR=f400, SW=8008, A=   21 SP=007f LR=0018
1810ns PC=000e IR=ff00, SW=8000, A=   21 SP=007f LR=0018
1830ns PC=0018 IR=101b, SW=8000, A=   21 SP=007f LR=0018
1850ns PC=0019 IR=a011, SW=8000, A=   21 SP=007f LR=0018
1870ns PC=0011 IR=001c, SW=8000, A=    6 SP=007f LR=0018
```

```
1890ns PC=0012 IR=901d, SW=8000, A=    6 SP=007f LR=0018
1910ns PC=0013 IR=b01a, SW=8000, A=    6 SP=007f LR=0018
1930ns PC=0014 IR=f701, SW=8000, A=    7 SP=007f LR=0018
1950ns PC=0015 IR=101c, SW=8000, A=    7 SP=007f LR=0018
1970ns PC=0016 IR=001b, SW=8000, A=   21 SP=007f LR=0018
1990ns PC=0017 IR=201c, SW=8000, A=   28 SP=007f LR=0018
2010ns PC=0018 IR=101b, SW=8000, A=   28 SP=007f LR=0018
2030ns PC=0019 IR=a011, SW=8000, A=   28 SP=007f LR=0018
2050ns PC=0011 IR=001c, SW=8000, A=    7 SP=007f LR=0018
2070ns PC=0012 IR=901d, SW=8000, A=    7 SP=007f LR=0018
2090ns PC=0013 IR=b01a, SW=8000, A=    7 SP=007f LR=0018
2110ns PC=0002 IR=a004, SW=8008, A=    7 SP=007f LR=0014
2130ns PC=0004 IR=f300, SW=8008, A=    7 SP=007e LR=0014
2150ns PC=0005 IR=f301, SW=8008, A=    7 SP=007d LR=0014
2170ns PC=0006 IR=f302, SW=8008, A=    7 SP=007c LR=0014
2190ns PC=0007 IR=001e, SW=8008, A=    3 SP=007c LR=0014
2210ns PC=0008 IR=f701, SW=8008, A=    4 SP=007c LR=0014
SWI C8=   1 A=    4
2230ns PC=0009 IR=fa01, SW=8008, A=    4 SP=007c LR=0014
2250ns PC=000a IR=101e, SW=8008, A=    4 SP=007c LR=0014
2270ns PC=000b IR=f402, SW=8008, A=    7 SP=007d LR=0014
2290ns PC=000c IR=f401, SW=8008, A=    7 SP=007e LR=0014
2310ns PC=000d IR=f400, SW=8008, A=    7 SP=007f LR=0014
2330ns PC=000e IR=ff00, SW=8000, A=    7 SP=007f LR=0014
2350ns PC=0014 IR=f701, SW=8000, A=    8 SP=007f LR=0014
2370ns PC=0015 IR=101c, SW=8000, A=    8 SP=007f LR=0014
2390ns PC=0016 IR=001b, SW=8000, A=   28 SP=007f LR=0014
2410ns PC=0017 IR=201c, SW=8000, A=   36 SP=007f LR=0014
2430ns PC=0018 IR=101b, SW=8000, A=   36 SP=007f LR=0014
2450ns PC=0019 IR=a011, SW=8000, A=   36 SP=007f LR=0014
2470ns PC=0011 IR=001c, SW=8000, A=    8 SP=007f LR=0014
2490ns PC=0012 IR=901d, SW=8000, A=    8 SP=007f LR=0014
2510ns PC=0013 IR=b01a, SW=8000, A=    8 SP=007f LR=0014
2530ns PC=0014 IR=f701, SW=8000, A=    9 SP=007f LR=0014
2550ns PC=0015 IR=101c, SW=8000, A=    9 SP=007f LR=0014
2570ns PC=0016 IR=001b, SW=8000, A=   36 SP=007f LR=0014
2590ns PC=0017 IR=201c, SW=8000, A=   45 SP=007f LR=0014
2610ns PC=0018 IR=101b, SW=8000, A=   45 SP=007f LR=0014
2630ns PC=0019 IR=a011, SW=8000, A=   45 SP=007f LR=0014
2650ns PC=0002 IR=a004, SW=8008, A=   45 SP=007f LR=0011
2670ns PC=0004 IR=f300, SW=8008, A=   45 SP=007e LR=0011
2690ns PC=0005 IR=f301, SW=8008, A=   45 SP=007d LR=0011
2710ns PC=0006 IR=f302, SW=8008, A=   45 SP=007c LR=0011
2730ns PC=0007 IR=001e, SW=8008, A=    4 SP=007c LR=0011
2750ns PC=0008 IR=f701, SW=8008, A=    5 SP=007c LR=0011
```

```
SWI C8=   1 A=    5
2770ns PC=0009 IR=fa01, SW=8008, A=    5 SP=007c LR=0011
2790ns PC=000a IR=101e, SW=8008, A=    5 SP=007c LR=0011
2810ns PC=000b IR=f402, SW=8008, A=   45 SP=007d LR=0011
2830ns PC=000c IR=f401, SW=8008, A=   45 SP=007e LR=0011
2850ns PC=000d IR=f400, SW=8008, A=   45 SP=007f LR=0011
2870ns PC=000e IR=ff00, SW=8000, A=   45 SP=007f LR=0011
2890ns PC=0011 IR=001c, SW=8000, A=    9 SP=007f LR=0011
2910ns PC=0012 IR=901d, SW=8000, A=    9 SP=007f LR=0011
2930ns PC=0013 IR=b01a, SW=8000, A=    9 SP=007f LR=0011
2950ns PC=0014 IR=f701, SW=8000, A=   10 SP=007f LR=0011
2970ns PC=0015 IR=101c, SW=8000, A=   10 SP=007f LR=0011
2990ns PC=0016 IR=001b, SW=8000, A=   45 SP=007f LR=0011
3010ns PC=0017 IR=201c, SW=8000, A=   55 SP=007f LR=0011
3030ns PC=0018 IR=101b, SW=8000, A=   55 SP=007f LR=0011
3050ns PC=0019 IR=a011, SW=8000, A=   55 SP=007f LR=0011
3070ns PC=0011 IR=001c, SW=8000, A=   10 SP=007f LR=0011
3090ns PC=0012 IR=901d, SW=4000, A=   10 SP=007f LR=0011
3110ns PC=0013 IR=b01a, SW=4000, A=   10 SP=007f LR=0011
3130ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=0011
3150ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=0011
3170ns PC=0002 IR=a004, SW=4008, A=   10 SP=007f LR=001a
3190ns PC=0004 IR=f300, SW=4008, A=   10 SP=007e LR=001a
3210ns PC=0005 IR=f301, SW=4008, A=   10 SP=007d LR=001a
3230ns PC=0006 IR=f302, SW=4008, A=   10 SP=007c LR=001a
3250ns PC=0007 IR=001e, SW=4008, A=    5 SP=007c LR=001a
3270ns PC=0008 IR=f701, SW=4008, A=    6 SP=007c LR=001a
SWI C8=   1 A=    6
3290ns PC=0009 IR=fa01, SW=4008, A=    6 SP=007c LR=001a
3310ns PC=000a IR=101e, SW=4008, A=    6 SP=007c LR=001a
3330ns PC=000b IR=f402, SW=4008, A=   10 SP=007d LR=001a
3350ns PC=000c IR=f401, SW=4008, A=   10 SP=007e LR=001a
3370ns PC=000d IR=f400, SW=4008, A=   10 SP=007f LR=001a
3390ns PC=000e IR=ff00, SW=4000, A=   10 SP=007f LR=001a
3410ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3430ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3450ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3470ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3490ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3510ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3530ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3550ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3570ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3590ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3610ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
```

```
3630ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3650ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3670ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3690ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3710ns PC=0002 IR=a004, SW=4008, A=   10 SP=007f LR=001a
3730ns PC=0004 IR=f300, SW=4008, A=   10 SP=007e LR=001a
3750ns PC=0005 IR=f301, SW=4008, A=   10 SP=007d LR=001a
3770ns PC=0006 IR=f302, SW=4008, A=   10 SP=007c LR=001a
3790ns PC=0007 IR=001e, SW=4008, A=    6 SP=007c LR=001a
3810ns PC=0008 IR=f701, SW=4008, A=    7 SP=007c LR=001a
SWI C8=   1 A=    7
3830ns PC=0009 IR=fa01, SW=4008, A=    7 SP=007c LR=001a
3850ns PC=000a IR=101e, SW=4008, A=    7 SP=007c LR=001a
3870ns PC=000b IR=f402, SW=4008, A=   10 SP=007d LR=001a
3890ns PC=000c IR=f401, SW=4008, A=   10 SP=007e LR=001a
3910ns PC=000d IR=f400, SW=4008, A=   10 SP=007f LR=001a
3930ns PC=000e IR=ff00, SW=4000, A=   10 SP=007f LR=001a
3950ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3970ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
3990ns PC=001a IR=a01a, SW=4000, A=   10 SP=007f LR=001a
```

您可以看到上述執行過程中，每隔一小段時間就會印出 SWI 指令，那就是下列中斷處理常式的 SWI 1 這個指令所印出來的， 當 2 號中斷發生時，程式會跳到 02 位址的指令，也就是 JMP IRQ，接著跳到 04 位址的 IRQ 標記上，開始執行中斷常式， 於是在 09 的 SWI 1 這行當中，我們會利用 SWI: $display(″SWI C8=%d A=%d″,SC8,A)；這行程式印出累積器 A 的值 以便觀察到中斷的發生，以下是測試程式中斷部份得機器碼與組合語言。

```
A00F  // 00           JMP   RESET
A003  // 01           JMP   ERROR
A004  // 02           JMP   IRQ
A006  // 03    ERROR: JMP   ERROR
F300  // 04    IRQ:   PUSH  A
F301  // 05           PUSH  LR
F302  // 06           PUSH  SW
001E  // 07           LD    TIMER
F701  // 08           ADDI  1
FA01  // 09           SWI   1
101E  // 0A           ST    TIMER
F402  // 0B           POP   SW
F401  // 0C           POP   LR
F400  // 0D           POP   A
FF00  // 0E           IRET
...
```

以上片段中 IRQ 到 IRET 指令之間為中斷常式，每次中斷時該常式就會將 TIMER 加一，因此就可以用來計算中斷發生的次數， 雖然這樣的程式並沒有太大的用途，但還蠻適合用來說明中斷原理的。

結語

中斷向量裏通常存放一堆跳躍指令，在中斷的時候可以可以透過這些指令跳到中斷處理程式， 以便判別中斷類型並進行對應的處理，後續我們將介紹如何使用中斷機制進行輸出入處理 與行程切換的動作。

講題分享 - 用 R 進行中文 text Mining（作者:陳嘉葳@Taiwan R User Group）

本文使用的分析方法，目前僅能在Windows上測試成功。

簡介

現今網路上有大量文字資料,例如 ptt, facebook, 或 mobile01等討論網站上都有大量文字留言, 由於這些資料繁多雜亂, 我們可藉由文字探勘技術萃取出有用的訊息, 讓人們有效率掌握這些網路文字所提供的訊息。而R語言是一款非常適合資料分析的工具,有一系列文字探勘的套件可供使用,本文將簡單介紹中文文字探勘套件的使用方法。

安裝需要工具

我的環境: Windows 7 + R 版本 2.15.3 + RStudio 0.98.484

安裝以下套件

```
install.packages("rJava")
install.packages("Rwordseg", repos="http://R-Forge.R-project.org")
install.packages("tm")
install.packages("tmcn", repos="http://R-Forge.R-project.org", type="source")
install.packages("wordcloud")
install.packages("XML")
install.packages("RCurl")
```

Windows上安裝rJava的注意事項:

- 將jvm.dll加到環境變數PATH之中
- 注意java的版本(32-bit or 64-bit)必須要和R一致 (Windows binary)之後再安裝，以避免

抓取ptt笨版文章列表

我們以分析ptt笨版文章為範例, 首先到ptt web 版抓取文章的url連結。 我們利用 RCurl套件的 getURL和 XML套件的 htmlParseApply 抓取笨版文章列表的 html網頁, 再用xpathSApply 去擷取出所有文章的url連結並儲存。 或是可以到已下載ptt笨版文章

```
library(XML)
library(RCurl)

data <- list()

for( i in 1058:1118){
  tmp <- paste(i, '.html', sep='')
  url <- paste('www.ptt.cc/bbs/StupidClown/index', tmp, sep='')
  html <- htmlParse(getURL(url))
  url.list <- xpathSApply(html, "//div[@class='title']/a[@href]", xmlAttrs)
  data <- rbind(data, paste('www.ptt.cc', url.list, sep=''))
}
```

```
data <- unlist(data)
```

下載列表中的笨版文章之後，接著才開始利用所有文章的url連結去抓所有文章的html網頁, 並用xpathSApply去解析出文章的內容並儲存。

```
getdoc <- function(line){
  start <- regexpr('www', line)[1]
  end <- regexpr('html', line)[1]

  if(start != -1 & end != -1){
    url <- substr(line, start, end+3)
    html <- htmlParse(getURL(url), encoding='UTF-8')
    doc <- xpathSApply(html, "//div[@id='main-content']", xmlValue)
    name <- strsplit(url, '/')[[1]][4]
    write(doc, gsub('html', 'txt', name))
  }
}
```

開始下載文章

先用 setwd(填入資料夾位置) 這指令, 選定文件下載位置

或是用 getwd() 確定目前的工作資料夾位置

```
sapply(data, getdoc)
```

開始文字處理

首先載入以下text mining 套件

```
library(tm)
library(tmcn)
```

```
## # tmcn Version: 0.1-2
```

```
library(Rwordseg)
```

```
## Loading required package: rJava
## # Version: 0.2-1
```

匯入剛才抓完的笨版文章，doc 是儲存下載ptt文章的資料夾, 這些文章變成我們分析的語料庫。

```
d.corpus <- Corpus(DirSource("doc"), list(language = NA))
```

進行數據清理

1 清除標點符號, 數字

```
d.corpus <- tm_map(d.corpus, removePunctuation)
d.corpus <- tm_map(d.corpus, removeNumbers)
```

2 清除大小寫英文與數字

```
d.corpus <- tm_map(d.corpus, function(word) {
    gsub("[A-Za-z0-9]", "", word)
})
```

進行中文斷詞

首先，由於ptt有自己獨特的詞彙，例如發文不附沒圖、沒圖沒真相...等等，因此我們另外安裝 ptt 常用詞彙來協助斷詞。ptt常用詞彙可以從搜狗細胞辭庫下載 。

```
words <- readLines("http://wubi.sogou.com/dict/download_txt.php?id=9182")
words <- toTrad(words)
insertWords(words)
```

接著，我們利用我們 Rwordseg套件裡的segmentCN來進行斷詞。Rwordseg是李艦所撰寫的R套件，利用rJava去連結java分詞工具ansj來進行斷詞。 另外，斷詞後的詞彙有詞性，例如動詞、名詞、形容詞、介係詞等等，我們只挑出名詞來進行分析。

```
d.corpus <- tm_map(d.corpus[1:100], segmentCN, nature = TRUE)
d.corpus <- tm_map(d.corpus, function(sentence) {
    noun <- lapply(sentence, function(w) {
        w[names(w) == "n"]
    })
    unlist(noun)
})
d.corpus <- Corpus(VectorSource(d.corpus))
```

接著進行清除停用字符，停用字符指的是一些文章中常見的單字，但卻無法提供我們資訊的冗字。例如有些、以及、因此...等等字詞。

```
myStopWords <- c(stopwordsCN(), "編輯", "時間", "標題", "發信", "實業", "作者")
d.corpus <- tm_map(d.corpus, removeWords, myStopWords)
```

我們可以看看有哪些停用字符，這裡抽出前20個停用字符來看

```
head(myStopWords, 20)
```

```
##  [1] "第二"     "一番"     "一直"     "一<U+4E2A>" "一些"     "<U+8BB8>多"
##  [7] "种"       "有的是"   "也就是<U+8BF4>" "末"       "啊"       "阿"
## [13] "哎"       "哎呀"     "哎<U+54DF>" "唉"       "俺"       "俺<U+4EEC>"
## [19] "按"       "按照"
```

建立 TermDocumentMatrix

中文斷詞結束後，我們用矩陣將結果儲存起來。TermDocumentMatrix 指的是關鍵字為列，文件是行的矩陣。儲存的數字是關鍵字在這些文件中出現的次數。其中 wordLengths=c(2,Inf) 表示我們挑至少兩個字的詞。

```
tdm <- TermDocumentMatrix(d.corpus, control = list(wordLengths = c(2, Inf)))
```

我們可以看看TermDocumentMatrix裡面，前兩篇文章的前10個關鍵字

```
inspect(tdm[1:10, 1:2])
```

```
## A term-document matrix (10 terms, 2 documents)
##
## Non-/sparse entries: 3/17
## Sparsity           : 85%
## Maximal term length: 3
## Weighting          : term frequency (tf)
##
##         Docs
## Terms    M.1384834727.A.0CB.txt M.1384838698.A.957.txt
##   一生                        0                      0
##   一家                        1                      0
##   一家子                      0                      0
##   一線                        0                      0
##   人士                        0                      0
##   人才                        0                      0
##   人中                        0                      0
##   人文                        0                      0
##   人民                        0                      1
##   人生                        0                      2
```

畫出關鍵字詞雲

我們利用 wordcloud 套件，畫出在所有文件中出現次數超過10次的名詞。出現次數越多，字體就會呈現越大。

```
library(wordcloud)
```

```
m1 <- as.matrix(tdm)
v <- sort(rowSums(m1), decreasing = TRUE)
d <- data.frame(word = names(v), freq = v)
wordcloud(d$word, d$freq, min.freq = 10, random.order = F, ordered.colors = F,
    colors = rainbow(length(row.names(m1))))
```

plot of chunk unnamed-chunk-13

這邊看到百合出現次數非常多，原因是因為一位叫小百合的網友所發的po文被鄉民推爆，留言當中也重複出現小百合的緣故。

尋找關鍵字之間的關聯

當我們利用wordcloud 知道哪些關鍵字常出現後，接著可能想知道哪些字與它有關聯。 因此我們先建出DocumentTermMatrix，這是以文件名稱為列，關鍵字為欄的矩陣。

```
d.dtm <- DocumentTermMatrix(d.corpus, control = list(wordLengths = c(2, Inf)))
inspect(d.dtm[1:10, 1:2])
```

```
## A document-term matrix (10 documents, 2 terms)
##
## Non-/sparse entries: 1/19
## Sparsity           : 95%
## Maximal term length: 2
## Weighting          : term frequency (tf)
##
##                        Terms
## Docs                    一生 一家
##   M.1384834727.A.0CB.txt    0    1
##   M.1384838698.A.957.txt    0    0
##   M.1384840050.A.414.txt    0    0
##   M.1384840304.A.EF5.txt    0    0
##   M.1384842495.A.5B8.txt    0    0
##   M.1384842609.A.A5B.txt    0    0
##   M.1384847473.A.6A5.txt    0    0
##   M.1384847771.A.729.txt    0    0
##   M.1384848469.A.AD8.txt    0    0
##   M.1384849471.A.B71.txt    0    0
```

可以用 findFreqTerms 看看在所有文件裡出現30次以上的關鍵字有哪些。

```
findFreqTerms(d.dtm, 30)
```

```
## [1] "同學" "百合" "老闆" "朋友" "東西" "時候" "閃光" "問卷" "結果"
```

再來可以用 findAssocs 找出最常與"同學"關程度0.5以上的關鍵字。

```
findAssocs(d.dtm, "同學", 0.5)
```

```
## 同學.原因
##      0.56
```

結尾

以上我們介紹了如何將中文文章進行清理、斷詞等處理，最後轉換成矩陣，再進行一些簡單分析與繪圖。本文介紹之操作，還可以繼續進行關鍵字分群、主題模型、情緒分析等進階應用，有興趣繼續深入, 可以自行搜尋 tm, tmcn, Rwordseg 這三個套件，可以發現許多資源自行學習。

作者

- 陳嘉葳 : 國立高雄大學資管所, Kaohsiung useR! Meetup, Taiwan R User Group

參考資料

- Rwordseg
- 中文文字資料探勘 (英國Mango Solutions上海分公司資深顧問李艦)
- MLDM Monday - 如何用 R 作中文斷詞
- using-the-rjava-package-on-win7-64-bit-with-r

雜誌訊息

讀者訂閱

程式人雜誌是一個結合「開放原始碼與公益捐款活動」的雜誌，簡稱「開放公益雜誌」。開放公益雜誌本著「讀書做善事、寫書做公益」的精神，我們非常歡迎程式人認養專欄、或者捐出您的網誌，如果您願意成為本雜誌的專欄作家，請加入 程式人雜誌社團 一同共襄盛舉。

我們透過發行這本雜誌，希望讓大家可以讀到想讀的書，學到想學的技術，同時也讓寫作的朋友的作品能產生良好價值 － 那就是讓讀者根據雜誌的價值捐款給慈善團體。 讀雜誌做公益也不需要有壓力，您不需要每讀一本就急著去捐款，您可以讀了十本再捐，或者使用固定的月捐款方式，當成是雜誌訂閱費，或者是季捐款、一年捐一次等都 OK！甚至是單純當個讀者我們也都很歡迎！

本雜誌每期參考價：NT 50 元，如果您喜歡本雜誌，請將書款捐贈公益團體。例如可捐贈給「羅慧夫顱顏基金會 彰化銀行(009) 帳號：5234-01-41778-800」。(若匯款要加註可用「程式人雜誌」五個字)

投稿須知

給專欄寫作者： 做公益不需要有壓力。如果您願意撰寫專欄，您可以輕鬆的寫，如果當月的稿件出不來，我們會安排其他稿件上場。

給網誌捐贈者： 如果您沒時間寫專欄或投稿，沒關係，只要將您的網誌以 [創作共用的「姓名標示、非商業性、相同方式分享」授權] 並通知我們，我們會自動從中選取需要的文章進行編輯，放入適當的雜誌當中出刊。

給文章投稿者： 程式人雜誌非常歡迎您加入作者的行列，如果您想撰寫任何文章或投稿，請用 markdown 或 LibreOffice 編輯好您的稿件，並於每個月 25 日前投稿到程式人雜誌社團 的檔案區，我們會盡可能將稿件編入隔月1號出版程式人雜誌當中，也歡迎您到社團中與我們一同討論。

如果您要投稿給程式人雜誌，我們最希望的格式是採用 markdown 的格式撰寫，然後將所有檔按壓縮為 zip 上傳到社團檔案區給我們， 如您想學習 markdown 的撰寫出版方式，可以參考 看影片學 markdown 編輯出版流程 一文。

如果您無法採用 markdown 的方式撰寫，也可以直接給我們您的稿件，像是 MS. Word 的 doc 檔或 LibreOffice 的 odt 檔都可以，我們 會將這些稿件改寫為 markdown 之後編入雜誌當中。

參與編輯

您也可以擔任程式人雜誌的編輯，甚至創造一個全新的公益雜誌，我們誠摯的邀請您加入「開放公益出版」的行列，如果您想擔任編輯或創造新雜誌，也歡迎到 程式人雜誌社團 來與我們討論相關事宜。

公益資訊

公益團體	聯絡資訊	服務對象	捐款帳號
財團法人羅慧夫顱顏基金會	http://www.nncf.org/ lynn@nncf.org 02-27190408分機 232	顱顏患者 (如唇顎裂、小耳症或其他罕見顱顏缺陷）	銀行：009彰化銀行民生分行 帳號：5234-01-41778-800
社團法人台灣省兒童少年成長協會	http://www.cyga.org/ cyga99@gmail.com 04-23058005	單親、隔代教養.弱勢及一般家庭之兒童青少年	銀行：新光銀行 戶名：台灣省兒童少年成長協會 帳號：103-0912-10-000212-0