程式人 月刊雜誌

Programmer

CC BY SA

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

前言

編輯小語

在本期的「程式人雜誌」中，我們引入了一個比較奇怪的主題：「鍵盤革命」，這個主題有點「政治性」。

雖然可能有些「程式人」的朋友們會不喜歡，不過由於 2013/08/03 凱達格蘭大道的 25 萬人集會，讓我們決定要引入這樣一個 與「程式」距離較遠的主題。

如果讀者對這個主題比較不喜歡的話，可以選擇從「程式人文集」的技術類文章開始閱讀，跳過前半部的「鍵盤革命」主題。

但是、「鍵盤革命」的主題其實也有一些與程式人的關連，像是今年在 COSCUP 開源人年會的 g0v 組織，開源之父 Richard Stallman 等， 都是這個主題的相關人物，不排斥政治主題的程式人，可能會有興趣也說不定。

另外、由於小編已經從「中斷」(Interrupted) 的度假狀態，回復到「可工作」的 Ready 狀態，因此中斷一期的「R、Verilog、開放電腦計畫」 等主題，都將回到正常狀態，繼續刊登了。

----（程式人雜誌編輯 - 陳鍾誠）

授權聲明

本雜誌採用 創作共用：姓名標示、相同方式分享 授權，若您想要修改本書產生衍生著作時，至少應該遵守下列授權條件：

1. 標示原作者姓名
2. 採用 創作共用：姓名標示、相同方式分享 的方式公開衍生著作。

另外、當本雜誌中有文章或素材並非採用 姓名標示、相同方式分享 時，將會在該文章或素材後面標示其授權，此時該文章將以該標示的方式授權釋出，請修改者注意這些授權標示，以避免產生侵權糾紛。

例如有些文章可能不希望被作為「商業性使用」，此時就可能會採用創作共用：姓名標示、非商業性、相同方式分享 的授權，此時您就不應當將該文章用於商業用途上。

最後、懇請勿移除公益捐贈的相關描述，以便讓愛心得以持續散播！

程式人介紹

資訊政治人：**Stallman, Lessig, Swartz** 與 **Assange**

看到上述的人名，很多讀者勢必心中會產生一大堆問號？他們是誰？這些人之間有甚麼關連呢？

程式人雜誌曾經介紹過這四位當中的兩位，也就是 Stallman 與 Swartz，甚至也曾經談到過 Lessig ， 但是還沒有介紹過 Assange，讓我們稍微介紹一下這些人的重要事蹟。

- Stallman：開放原始碼之父–Richard Stallman, 是著名的程式人, GNU 組織的創建者，GPL 授權的制定者。
- Lessig：勞倫斯·雷席格, 哈佛大學憲法學教授，創作共用 (Creative Commons) 授權的創造者。
- Swartz：Markdown 與 RSS 的創造者-Aaron Swartz, 2010 年從 MIT 校園內大量論文後被起訴，然後不堪檢察官威脅，在 2013 年 1 月 11 日時自殺身亡。
- Assange：維基解密組織的創建者與核心人物。

筆者之所以將這些人寫在一起，原因在於這些人是「鍵盤革命」這一個概念當中的重要人物，他們都利用「電腦、程式、網路」等工具，與當權者進行對抗。

Stallman 因為不滿程式碼被商業公司封閉起來，而創造了 GNU 組織與 GPL 這個開放原始碼的程式授權，以便強制商業公司開放原始碼。 Lessig 延續 Stallman 的想法，將開放原始碼的概念引入一般著作與網頁，創造出了 Creative Commons 授權。而 Swartz 則為了讓資訊的流通更容易， 創造出 Markdown 格式與 RSS 訂閱技術，並且因為想把學術論文開放而遭到起訴，最後更因此而自殺。

至於 Assange，則因為將那些政府機密大量上網，成為各國政府的頭號公敵，並因「性犯罪」的罪名而被瑞典通輯，因此後來向厄瓜多駐倫敦大使館尋求政治庇護，目前英國還企圖與「厄瓜多」談判將 Assange 引渡到瑞典受審當中。

以上這些人與事件，都涉及到「資訊揭露」的政治學，有些人用合法的方式企圖揭露資訊 (像是 Stallman, Lessig)，而有些人則用目前法律不允許的手段去揭露資訊 (像是 Swartz 與 Assange)。

18 世紀工業革命 (第二波) 的結果，造成工業取代農業，因此政治上從封建莊園制度轉向了議會政治，權力的核心從封建領主轉向了資本家。

在今日 21 世紀的資訊革命 (第三波) 當中，權力又將如何移轉，這個世界會如何變化呢？

未來、網路與資訊科技會帶領我們建立怎麼樣的一個「政治結構」呢？這是筆者很想知道，但卻也還在探索當中的一個重要關注項目啊！

參考文獻

- 開放原始碼之父–Richard Stallman, 程式人雜誌 2013 年 3 月號
- Markdown 與 RSS 的創造者-Aaron Swartz, 程式人雜誌 2013 年 4 月號
- http://en.wikipedia.org/wiki/Lawrence_Lessig
- 維基百科：勞倫斯·雷席格
- http://en.wikipedia.org/wiki/Julian_Assange
- 維基百科：朱利安·阿桑奇
- 英國厄瓜多爾或商談阿桑奇去留問題, 作者：亞太日報綜合報導 發稿地：香港 時間：2013-6-03

- http://en.wikipedia.org/wiki/Alvin_Toffler
- 第三波 (The Third Wave), 作者：艾文・托佛勒 (Alvin Toffler), 譯者：黃明堅, 時報出版社, 1994年06月20日.
- 第三波 - 農業、工業與資訊文明, 陳鍾誠的網站。

【本文由陳鍾誠取材並修改自維基百科，原本寫得很長，但是後來決定簡化，若您想看那個長的版本，我們也有留著，請點選 這裏】

程式人短訊

資訊政治：台灣「鍵盤革命」的歷史與近況

前言

最近台灣的「程式人」很多都投入了政治運動，企圖用程式改變政治，因此今年的開源人年會 COSCUP 上也出現了好幾場的「政治相關」演講。

這個現象或許可以用「鍵盤革命」、「網路政治」、「程式政治學」等詞彙描述。

莫拉克風災

從 2009 年的莫拉克風災開始，台灣的程式人就開始運用專業發揮了不小的影響力，以下是當年的一則 PTT 報導：

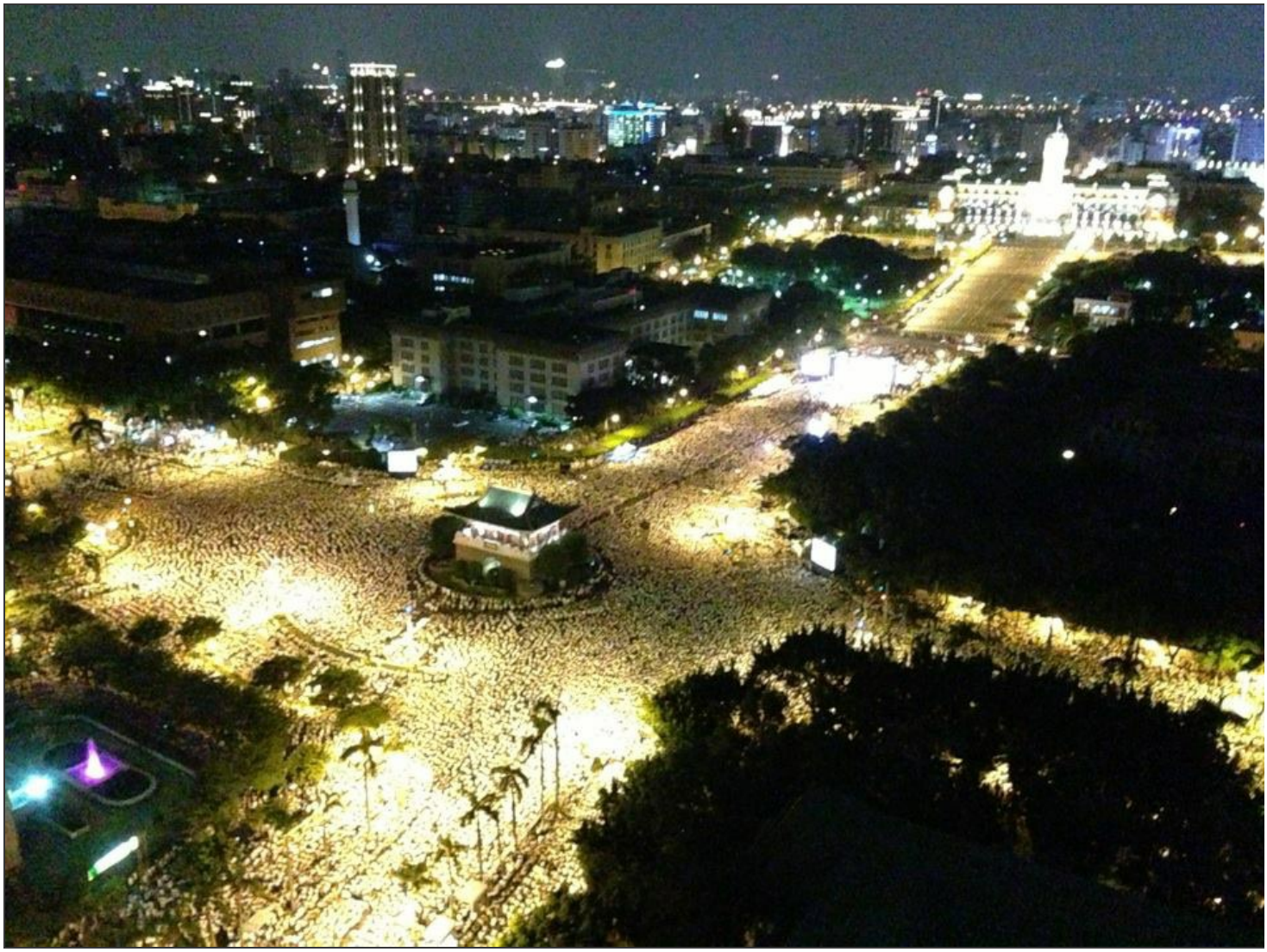
- PTT 精華區 - 閱讀文章 (Emergency) -- http://www.ptt.cc/man/Emergency/D8B1/M.1252037298.A.654.html

莫拉克風災是我第一次看到「網友們」展現政治力量的事件，雖然這件事情很不政治，但是基於孫中山的名言：「管理眾人之事便是政治」， 我們看到當政府失能的時候，網友們透過網路進行救災，彌補掉政府的無能，讓資訊得以快速流通的的「政治事件」。

洪仲秋被虐死引發 **25** 萬人上凱達格蘭大道

然後，在 2013 年，我們看到了一次「重要的網路政治事件」，那就是最近因「洪仲秋被虐死一案」，透過 公民 1985 行動聯盟 這個網路組織的協調，引發了 2013/8/3 日 25 萬人上凱達格蘭大道包圍總統府，卻又盡可能排除政黨介入的情況下，仍然表現出高度的組織能力，以及保持了良好秩序的事件。

以下的空拍圖片為這個事件留下了一個重要的見證，這是「鍵盤革命」的重要里程碑，也是台灣「網路世代」開始參與政治運動的一個明證。



圖、photo by Jack0000

- 圖片來源：親愛的芭樂人類學家專欄, 25萬人上街抗議真的有用嗎？, 2013/08/05 公民運動 政治 洪仲丘 社會運動
  - http://guavanthropology.tw/article/5006

結語

雖然政治令人覺得很黑暗，也讓很多人討厭，但是如果透過「程式專業」與「網路社群」，有機會能讓這個世界變得更好的話， 我相信還是有許多程式人願意付出時間來讓世界變得更美好的！

參考文獻

- 工程師的鍵盤革命：拆政府，原地重建, inside 網站。
- PTT 精華區 - 閱讀文章 (Emergency)
- 維基百科：中度颱風莫拉克
- 維基百科：八八水災

【本文由陳鍾誠取材並修改自維基百科，原本寫得很長，但是後來決定簡化，若您想看那個長的版本，我們也有留著，請點選 這裏】

資訊政治：海盜黨簡介

海盜黨是近年來在政治領域上異軍突起的一個政治團體，而德國海盜黨則是各國海盜黨當中發展得最好的一個，以下是海盜黨的一些相關資料，請讀者參考：

- 維基百科, 海盜黨國際
- 維基百科, 德國海盜黨
- Wikipedia:Pirate_Party_Germany
- 「遙控民主」新實驗，讓百年大黨取經的七歲駭客政黨 - 專訪海盜黨主席（上）

第一個海盜黨是由 李卡德·法克明炎於2006年1月1日成立於瑞典的組織，稱為 Piratpartiet。該組織認為現時的版權制度已經過時，危害到人們傳播知識的權力。 2006年海盜黨成為了瑞典無議會席位政黨之中最大的一個，後來奧地利、丹麥、芬蘭、德國、愛爾蘭、荷蘭、波蘭和西班牙也先後成立了海盜黨。

海盜黨的主張主要有三個：一是改革版權法，二是廢除專利，三是尊重私隱。他們認為現在的版權制度已經過時，很多企業利用版權限制知識發放， 侷限了很多創造性工作。所以於網路上分享電影、音樂等等行為不應該被視為違法。

2009年 4月 17 日，4名 海盜灣經營者被瑞典法庭判處 1 年有期徒刑之後，盜版黨黨員人數暴增到 2萬8千多人，躍身為瑞典第三大黨，並於2009年歐洲議會 選舉中在瑞典獲得了 7.1% 的選票，在歐洲議會中將擁有1個席位。

2011 年，德國海盜黨針對柏林州議會選舉推出 15 位候選人，全上。其後幾次其他的州議會選舉，德國海盜黨也獲得 7-8% 的得票率，取得席次。

海盜黨的一些網路民主的實施方法可以參考以下的文章：

- 德國海盜黨的流動式民主

我們將上述文章的重點摘要如下：

- 海盜黨的多數的決策都是先在網路上經過許多人的討論，例如用 PiratePad、聊天室、wiki 和郵件論壇來協作。
- 投票時，每人一票。但是，因為並非大家都能夠詳閱政策內容，所以系統允許成員可以委託他人代為投票。委託範圍可以是所有提案、某個主題的提案、或是某特定提案。而被委派投票的成員，還可以把這些票、包括自己的一票，再度委託給他人投票！
- Liquid Feedback 這種以信任為基礎的模式，類似信譽系統（reputation system），只不過成員的參與所換到的不是點數，而是選票。理論上，這種投票鏈可能會把票集中到人緣好的菁英或獨裁者身上，但好在有個防錯機制，可以在任何時候取消委託投票，有野心的海盜也得認真才能成事。Bormuth 說，我們讓有戰鬥力的人可以成事，但也讓大家有權控制這些人。

雖然海盜黨在很多國家都成立了，但是在台灣卻被政府禁止成立，請參考以下新聞。

- 學者籌組海盜黨 內政部不准中央社中央社 – 2012年2月26日 上午11:37

不過還是有人在網路上成立了這樣的社團，像是 Google Plus 上就有下列社團：

- [台灣海盜黨(推廣處)] -- https://plus.google.com/103500053191876104832/posts

結語

雖然德國的國情與台灣不同，不過在網路時代，各國的做法都可以很透明的被大家所參考模仿，然後經過嘗試後找到一個比較適合自己國家的實施方式， 運用網路來改變政治結構啊！

【本文由陳鍾誠取材並修改自維基百科】

資訊政治：**g0v** 政府零時差組織

高嘉良所發起的 g0v 組織，是一個企圖用程式改變政治的組織，有興趣的人可以參考一下他們的網站。

- 零時政府 -- http://g0v.tw/

g0v 做了很多有趣的「政治性」程式，以下是 g0v 的專案列表：

- g0v 專案列表 -- http://hack.g0v.tw/project

您可以看到其中有琳瑯滿目的專案，大部分是與資訊揭露有關的，像是：

- 社會運動資訊平台 -- http://www.movement-itw.com/
- 公務員出國考察追蹤網 -- http://hack.g0v.tw/abroadplay/nOWhTpJPKx7
- 中央政府總預算 -- http://budget.g0v.tw/
- 律師幫幫我 -- http://i64885.tw/
- 鄉民關心你 -- http://hack.g0v.tw/kuansim/nvw5cFwVmab
- 台灣法規的API -- http://laweasyread.herokuapp.com/
- 福利請聽 -- http://listening.g0v.tw/
- 新聞小幫手 -- http://newshelper.g0v.tw
- 政誌 -- http://fact.g0v.tw/
- 政府公開通訊錄 -- https://github.com/g0v/addressbook

最近我與 g0v 創辦人高嘉良連絡時，發現他們正在關注「柏林海盜黨」所釋出的一個實驗性開放原始碼軟體 liquidfeedback，網址如下：

- http://liquidfeedback.org/

而且他們正在進行一個工作，就是將 liquidfeedback 修改為中文版，並且嘗試用這種方式改變台灣的政治環境，您可以從 github 上下載這個專案。

- https://github.com/g0v/liquid_feedback_frontend

當然、並不是只有 g0v 在進行「用程式改造社會」的活動，另外像 Code for Tomorrow 也是一個具有類似想法的台灣程式團體。

- Code for Tomorrow -- http://codefortomorrow.org/

而在美國也有像 Code for America 這樣的組織，企圖用程式讓美國社會變得更好。

- Code for America -- http://codeforamerica.org/

筆者覺得、如果真的能用「程式讓世界變得更美好」，那真的是一件非常有意義的事情啊！希望 g0v 能夠有更多好的想法， 並發展出更多改善社會的程式，讓我們的社會能夠變得更美好啊！

【本文由陳鍾誠取材撰寫】

程式人頻道

看影片瞭解「鍵盤革命」的「資訊政治學」

現在、讓我們透過影片來看看，台灣與國際上對鍵盤革命的一些看法與想法，以便讓大家能夠進一步的思考：「資訊技術對政治領域，會產生甚麼樣的影響呢」？

國際上的鍵盤革命

- TED 影片 - Clay Shirky：網際網路（將來）如何改變政府
  - 從 git 版本管理系統看政治文化改變的可能性。
- TED 影片 - Lessig 談法律如何箝制創造力
  - Lessig 是 Creative Commons 創作共用的創造者。
- TED 影片 - Julian Assange：維基解密存在的必要性
  - Julian Assange 是維基解密的主要人物。
- TED 影片 - Rick Falkvinge: I am a pirate
  - Rick Falkvinge 是海盜黨的核心人物。
- YouTube - Aaron Swartz: The Documentary - Teaser
  - 紀念為鍵盤革命而死的 -- Aaron Swartz (RSS 與 Markdown 的創造者)

台灣的鍵盤革命

- YouTube -- 2013/08/03 公民1985 公民覺醒 最後演說(完整)
  - 2013 年 8 月 3 日因為洪仲秋被軍方虐死一案，引發 25 萬人上凱達格蘭大道抗議，會場中令人動容的演說。
- YouTube -- g0v 黑客松 - 寫程式改造社會 / clkao (Taiwan WebConf)
  - 高嘉良創立政府零時差組織 g0v 的原因，可以參考這個影片。
- YouTube -- Hack Everything, Including Society - 雨蒼
  - 雨蒼在 COSCUP 2013 對於如何用網路改變社會的演講。
- YouTube -- 蒐證雲/evi.tw - 翟本喬 和沛科技總經
  - 翟本喬在 COSCUP 2013 的演講，因洪仲秋案讓他們想到要用 APP 錄音蒐證。

參考文獻

- 工程師的鍵盤革命：拆政府，原地重建 http://www.inside.com.tw/2013/08/05/coscup-2013-coders-keyboard-revolution

【本文由陳鍾誠撰寫】

程式人討論區

網路與政治：網友們可以形成一個「虛擬國家」嗎？

受到 2013/8/3 時 25 萬人上凱達格蘭大道這件事的激勵，讓我想到能否號召網友們，形成一個沒有實體領土，只有虛擬領土的「網路國家」呢？

透過這個國家，我們可以制定法律、提出政策、甚至發動資訊戰。

而且、這些網民仍然是某些實體國家（例如台灣）的國民，因此仍然具有該國的投票權，所以就可以透過「網路公投」決定政策後， 要求政黨或立委認養這些政策並簽下契約，以換取將這些網民的票投給他們的「交易」。

我認為這種方式或許能讓網民可以有效的影響政府，並且跨過「公投法門檻過高」的問題。

於是我發了以下的訊息在「程式人雜誌的討論區」以及自己的 facebook 上，並引發了一些討論：

- 問題：我想我又瘋了，今年初我辦了一本新的雜誌，現在居然想成立一個新的國家
    - 網址： https://www.facebook.com/groups/programmerMagazine/permalink/690675390949180/

我之所以會認為應該用「虛擬國家」的概念，而非採用像「海盜黨」這樣的「黨」的概念，或許原因之一是「黨」這個中文字 其實隱含了非常糟糕的負面意義，但是在英文中的 Party 卻沒有這種意思。

還有一個原因是，我想法中的這種網路組織，其實更像是一個國家，因為這個組織可以擁有「人民、土地、政府、主權」這四種形成國家的要素，只不過其領土 乃是在網路上的虛擬領土，而非真實世界的領土而已，有興趣的讀者可以參考以下兩篇文章。

- 網路世代所需要的制度與法律 -- 虛擬國家的概念
- 網路公民國 -- 一個虛擬國家的憲法，需要您一起來制定

那麼、虛擬國家的憲法應該是什麼樣的呢？以下是一個我構思中的範例：

1. 任何人都可以經由明文宣誓的方式，成為網路公民國 (以下簡稱本國) 的公民 (以下簡稱網民)。
2. 網民除了本國之外，還可以自由參加任何組織或實體國家，本國並無任何禁止「雙重國籍」的規定。
3. 網民有上網之自由，任何組礙或限制網民上網的行為，都違反本憲法之精神。
4. 網民有集會結社之基本權利，任何損害此一權利的組織，都將視為本國之敵人，本國有封鎖該組織的權利。
5. 網民的任何作品，只要不附加「著作權宣告」，就被預設視為「公共領域」之作品，任何網民都擁有合法修改、複製、散布之權利。
6. 網民可以下載任何未被身分認證機制所保護的內容，而不被控以侵犯著作權之自由。 ....

這樣的法律其實意在保障網民們的自由，並且同時尊重創作者的權力，但是將著作權的預設值反轉過來，從 CopyRight 「版權所有」的 All Rights Reserved 轉化為「版權所無」的 No Rights Reserved 的情況，這讓網路自由可以得到法律基礎。

這種手法與 GNU 的 GPL 有些不同，GPL 是透過法律來挑戰法律，但「網路虛擬國家」則直接透過「建國」與「制憲」挑戰傳統的政治結構。

「虛擬國家」或許不會擁有土地，但是卻擁有「網路空間上的領土」，可以透過「封鎖」、「不提供資訊」與實

體國家進行對抗，網民們 也有可能透過像「鍵盤戰」的方式，進行某種形式的「虛擬戰爭」。

更重要的是，「虛擬國家」沒有禁止「雙重國籍」的規定，網民們在實體世界的國家裏，還是有投票權的。因此、「虛擬國家」可以透過 「公投」制定政策，試圖影響某個實體國家的政策，讓這些國家的政治可以更好。

舉例而言，如果台灣的「網民」們聯合起來，形成一個「虛擬國家」，例如叫做「網路公民國」(簡稱網國)，就可以先在網路上制定政策 並且透過「公投」表決，決定建議國民將選票投給「認養」這個政策的「立委」或「政黨」，然後告訴「國民黨」與「民進黨」這個遊戲規則， 用「虛擬國家」的政策影響甚至左右「實體國家」的政策。

於是、這個虛擬國家成了一個「國中之國」，而且可以發揮強大的政治影響力，這就是我對「虛擬國家」運作方式的初步想法。

後記：我們甚至連國歌都寫好了，直接採用悲慘世界的革命之歌，配上自己填的歌詞，有興趣的朋友可以點選下列網址。

- 網路公民國之歌

不過可惜的是，由於這個理想尚未得到真實國家的認同，筆者怕因為將自己唱國歌的錄音上網而被告，所以現在這首國歌只能私下唱， 因為這首歌的商業權還在華納公司的手上啊！請參考下列文章。

- 《你敢有聽著咱的歌》是誰的（江雅綺）

這也正是我們為何在上述憲法中要加入那些 CopyLeft 版權條款的原因啊！

當然、您也可以直接宣誓加入這個虛擬國家，只要在 facebook 上按一下加入就行了。

- 「網路公民國」 -- https://www.facebook.com/groups/netcountry/

【本文由陳鍾誠撰寫】

程式人文集

Arduino入門教學**(9) –** 在 **2x16 LCD** 上顯示 **"Hello World"** 訊息 **(**作者：**Cooper Maa)**

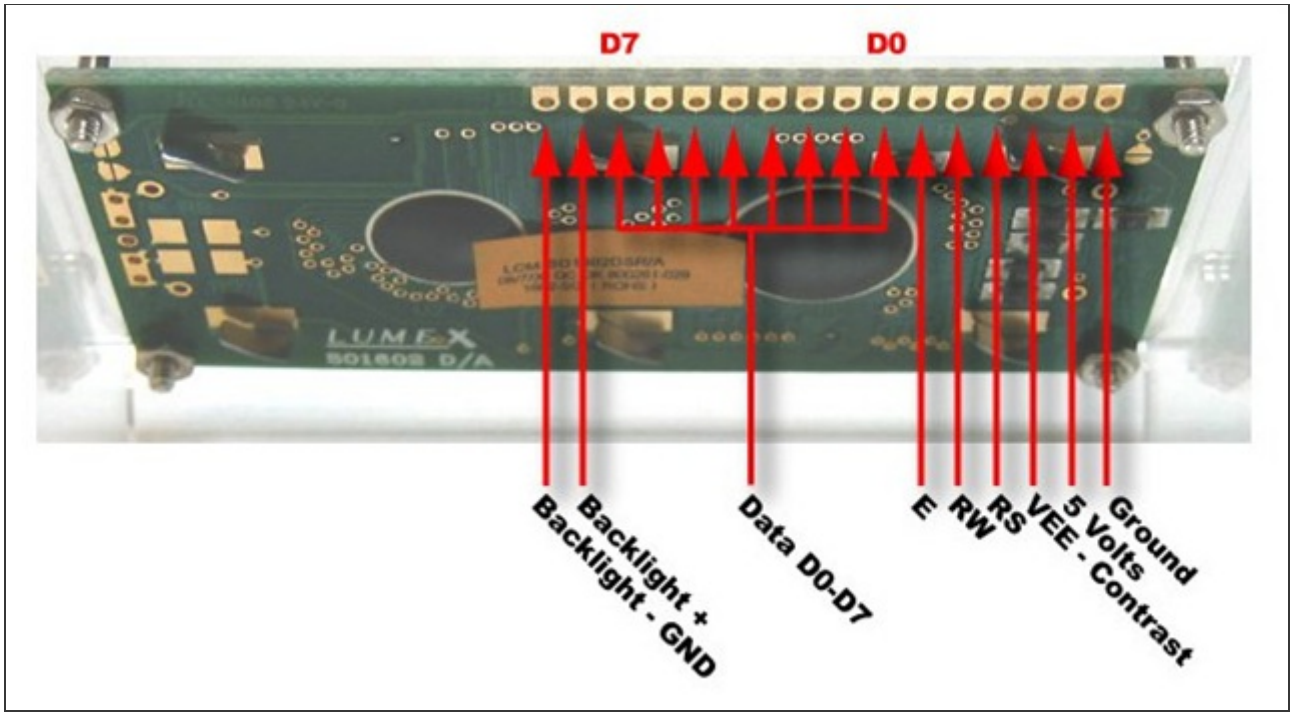
實驗目的

練習使用 HD44780 相容的文字型 LCD(Liquid crystal display)，在 2x16 LCD 上顯示 "Hello World" 訊息。

2x16 LCD 簡介

圖片來源: arduino.cc

HD44780 相容的 2x16 LCD 可以顯示兩行訊息，每行 16 個字元，它可以顯示英文字母、希臘字母、標點符號以及數學符號，除了顯示訊息外，它還有其它功能，包括訊息捲動(往左和往右捲動)、顯示游標和 LED背光等。



圖片來源: LCD 101

LCD 總共有 14 支接腳，如果內建背光的話是 16 支，這些腳位的功能整理於下表:

腳位編號	名稱	說明
1	Vss	接地 (0V)
2	Vdd	電源 (+5V)
3	Vo 或稱 Vee	對比(0-5V), 可接一顆 1k 電阻，或利可變電阻調整適當的對比
4	RS	Register Select: 1: D0 – D7 當作資料解釋 0: D0 – D7 當作指令解釋
5	R/W	Read/Write mode: 1: 從 LCD 讀取資料 0: 寫資料到 LCD, 因為很少從 LCD 這端讀取資料，可將此腳位接地以節省 I/O 腳位。
6	E	Enable
7	D0	Bit 0 LSB
8	D1	Bit 1
9	D2	Bit 2
10	D3	Bit 3
11	D4	Bit 4

12	D5	Bit 5
13	D6	Bit 6
14	D7	Bit 7 MSB
15	A+	背光(串接 330R 電阻到電源)
16	K-	背光(GND)

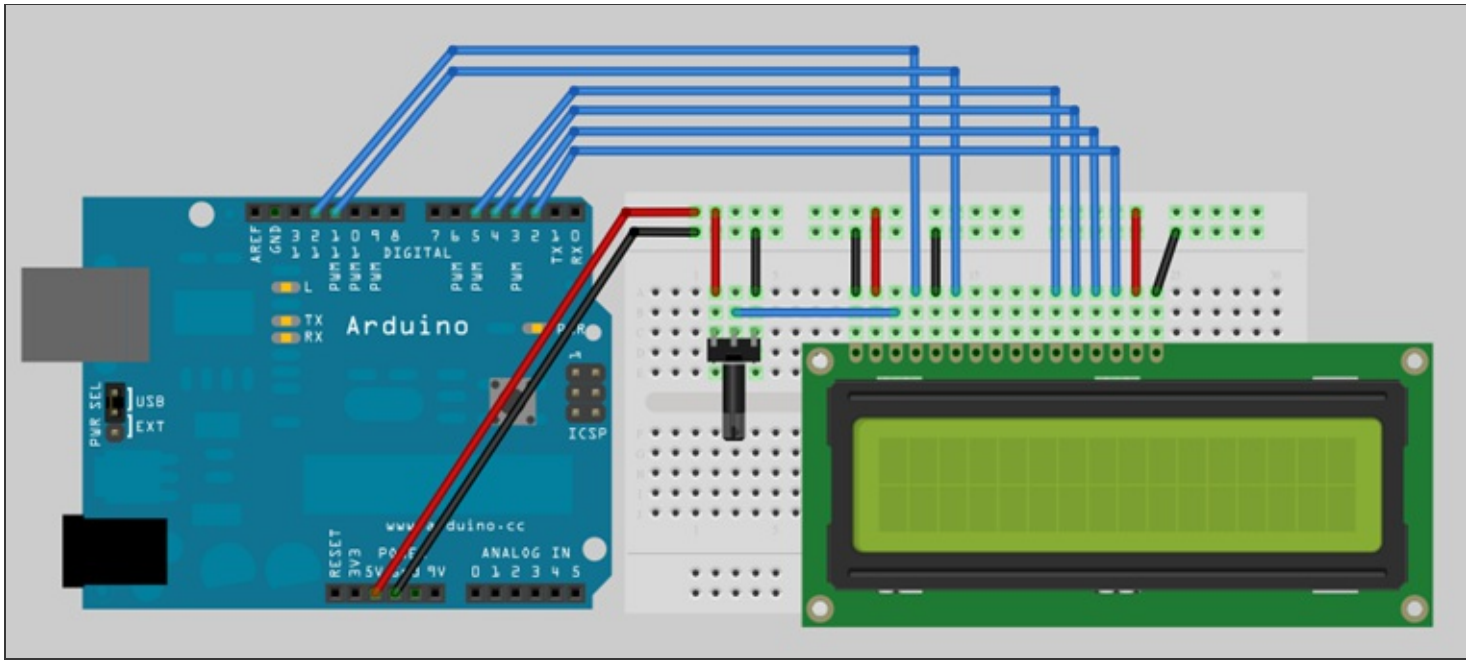
日立 HD44780 相容的 LCD 有 4-bit 和 8-bit 兩種使用模式，使用 4-bit 模式主要的好處是節省 I/O 腳位，通訊的時候只會用到 4 個高位元 (D4-D7)，D0-D3 這四支腳位可以不用接。每個送到 LCD 的資料會被分成兩次傳送 – 先送 4 個高位元，然後才送 4 個低位元。

材料

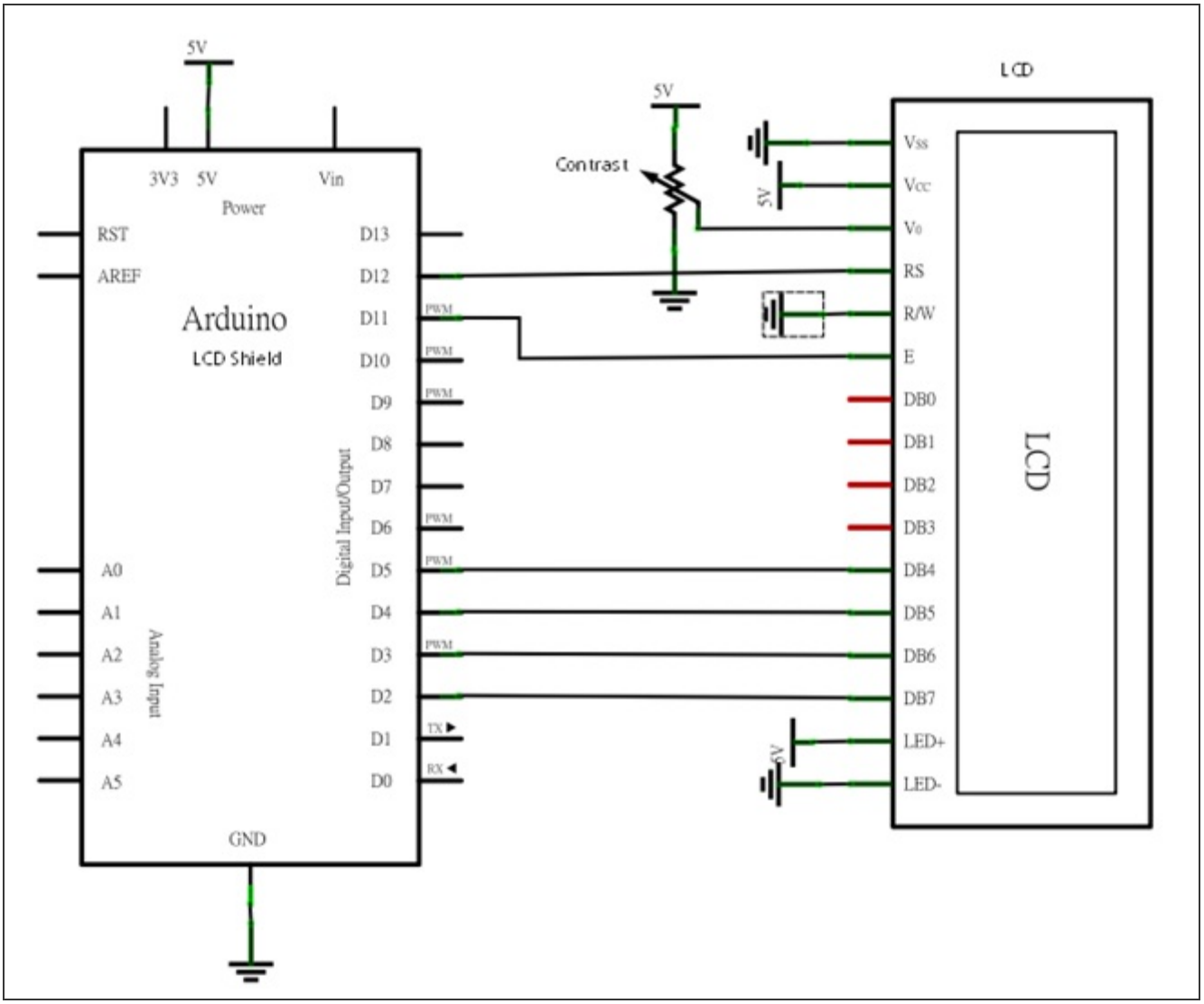
- 麵包板 x 1
- Arduino 主板 x 1
- HD44780 相容 LCD x 1 (本文所用的為 14 pin 無背光 LCD)
- 旋轉式可變電阻 x 1
- 單心線 x N

接線

- 將 LCD 的 RS, Enable, D4, D5, D6, D7 依序接到 12, 11, 5, 4, 3, 2 等腳位上
- 將 LCD 的 Vss 及 R/W 接到 GND，Vdd 接到 +5V
- 可變電阻中間腳位接到 LCD 的 Vo，剩下的兩支腳位，一支接到 5V，另外一支接到 GND (註：也可以不使用可變電阻，只要在 LCD 的 Vo 上串接一顆 1k ohm 電阻連到 GND 即可)



電路圖



程式碼

要在 LCD 上顯示訊息，會涉及初始化 LCD 、下指令以及傳送資料給 LCD 等工作，Arduino LiquidCrystal Library 已經把這些工作簡化了，所以你不需要知道這些低階的指令。底下的程式在 2x16 LCD 上第一行顯示 "hello, world!" 訊息，並在第二行不斷更新 Arduino 重開之後經過的秒數，使用的是 4-bit 模式(HelloWorld.pde):

```
/* Lab9 - 在 2x16 LCD 上顯示 "Hello World" 訊息
 The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * 10K Potentiometer:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

 This example code is in the public domain.
```

```
 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */

// 引用 LiquidCrystal Library
#include <LiquidCrystal.h>

// 建立 LiquidCrystal 的變數 lcd
//              LCD 接腳:  rs, enable, d4, d5, d6, d7
//     對應到 Arduino 接腳:  12,     11,  5,  4,  3,  2
LiquidCrystal lcd(12,  11,  5,  4,  3,  2);

void setup() {
  // 設定 LCD 的行列數目 (2 x 16)
  lcd.begin(16,  2);

  // 列印 "Hello World" 訊息到 LCD 上
  lcd.print("hello, world!");
}

void loop() {
  // 將游標設到 column 0, line 1
  // (注意: line 1 是第二行(row)&#65292;因為是從 0 開始數起):
  lcd.setCursor(0,  1);

  // 列印 Arduino 重開之後經過的秒數
  lcd.print(millis()/1000);
}
```

如果訊息沒有顯示在螢幕上，或者是模糊朦朧的，首先應該做的事是檢查可變電阻以調整對比。

註：這支是 Arduino 內建的範例程式，點選 File > Examples > LiquidCrystal > HelloWorld 就可以找到。

範例照片／影片

- Arduino 筆記 -- Lab9 在 2x16 LCD 上顯示 "Hello World" 訊息 -- http://youtu.be/jUbNR54mfgM

動動腦

1. 接一顆光敏電阻或可變電阻，然後寫一支程式讀取光敏電阻或可變電阻的讀值，並將讀到的數值顯示在 LCD 上。
2. 寫一支 SerialLCD 程式，程式的邏輯是： 接受來自 Serial Port 的資料(從 PC 或筆電端送出)，然後把資料顯示在 LCD 上。
3. 承上題，把 SerialLCD 變成一個網路型的 LCD，讓電腦透過網路就可以把資料丟到 LCD 上顯示。

延伸閱讀

- How to control a HD44780-based Character LCD
- ladyada – Wiring up a character LCD to Arduino
- LCD 101
- Scroll : scroll text left and right.
- Autoscroll : shift text right and left.

【本文作者為馬萬圳，原文網址為： http://coopermaa2nd.blogspot.tw/2010/12/arduino-lab9-2x16-lcd-world.html ，由陳鍾誠編輯後納入本雜誌】

JavaScript (9) – Google 的語音合成 API 之使用 (作者：陳鍾誠)

簡介

我還記得在 1996 年時，我到中研院許老師的實驗室當助理，做了兩年自然語言處理的相關程式， 學習到其中主要較成熟的技術像是「注音轉國字」(對應產品為自然輸入法)，然後這兩年整個實驗室 還試圖去研究一些較不成熟，但卻具有挑戰性的技術，特別是「自然語言理解」領域的一些嘗試，例如 有位同事就發展出了一個有趣的程式，當您輸入小學課本中的數學問題時，該程式可以輸出該問題的解答。

自然語言技術一直是筆者相當關心的領域，雖然筆者的關注比較偏向「文字」部分，在去年於金門大學 教授「計算語言學」這門課時，我就將這些相關的技術寫成了一本書，放在筆者的 github 當中，您可以 從以下的網址找到這本書。

- 陳鍾誠的教科書 -- http://ccckmit.github.io/home/
- 語言處理技術 -- http://ccckmit.github.io/BookLanguageProcessing/

但是，自然語言處理可以說是一門既深奧又困難的領域，雖然研究了很久，但筆者一直還沒有去觸碰 「語音合成」與「語音辨識」這兩個領域的主題，對這種「語音領域」的東西可以說是既期待又怕受傷害啊！

不過、現在由於「微軟」與 Google 等軟體大廠，都已經投入了非常多的資源在研究這些先進的領域， 並且製作出了足以商品化的功能，因此我們只要善用他們所釋出的 API，就可以輕鬆的應用這些功能了。

在本文中，我們將介紹如何使用 HTML+JavaScript 技術，去使用 Google 所提供的「語音辨識與合成」的服務。

語音合成的 Google 服務

Google 的語音合成服務非常容易使用，因為您只要利用 Google 翻譯中的 TTS (Text to Speech) 功能，將 文字放到以下網址中的 {query} 欄位中，然後設定正確的語言欄 {lang}，就可以取得這句話的語音檔了。

```
http://translate.google.com/translate_tts?ie=utf-8&tl={lang}&q={query}
```

舉例而言，假如您想讓瀏覽器說出 Text to speech 這句英文，只要將 {lang} 設為 en，然後將 {query} 設為 Text to speech 即可，您可以點選下列網址聽到 Google 所合成的語音。

- http://translate.google.com/translate_tts?ie=utf-8&tl=en&q=Text to speech

如果您想讓瀏覽器說中文，那麼 {lang} 欄位就必須要設定為 zh，例如您可以點選下列網址廳到「語音合成」這句話。

- http://translate.google.com/translate_tts?ie=utf-8&tl=zh&q=語音合成

一但瞭解了 Google TTS 功能的使用方式之後，您就可以很容易的在網頁中嵌入這樣的「功能元件」了。

在 HTML 5.0 當中有個特殊的標記， 可以在網頁中嵌入語音，那就是 <audio src="..."> 標記，您只要在 src 欄位中 填入正確的語音檔網址，該標記就可以用來控制語音的播放，如果您在 <audio> 當中加上 controls 這個屬性，那麼 畫面上就會出現一個像錄音機的控制面版，讓你自行操控語音的「播放與暫停」等功能。

舉例而言，以下是一個很簡單的程式，其中有兩個「語音控制項」，一個按下後可以播放「你好、這是谷歌的語音合成測試！」 這句中文，另一個按下後可以播放「Hi! This is the text to speech function of Google.」這句英文。

圖、簡單的語音合成範例

您可以透過這樣的控制項，控制「播放、暫停、調整音量」等等功能，上述畫面的 HTML 原始程式碼如下：

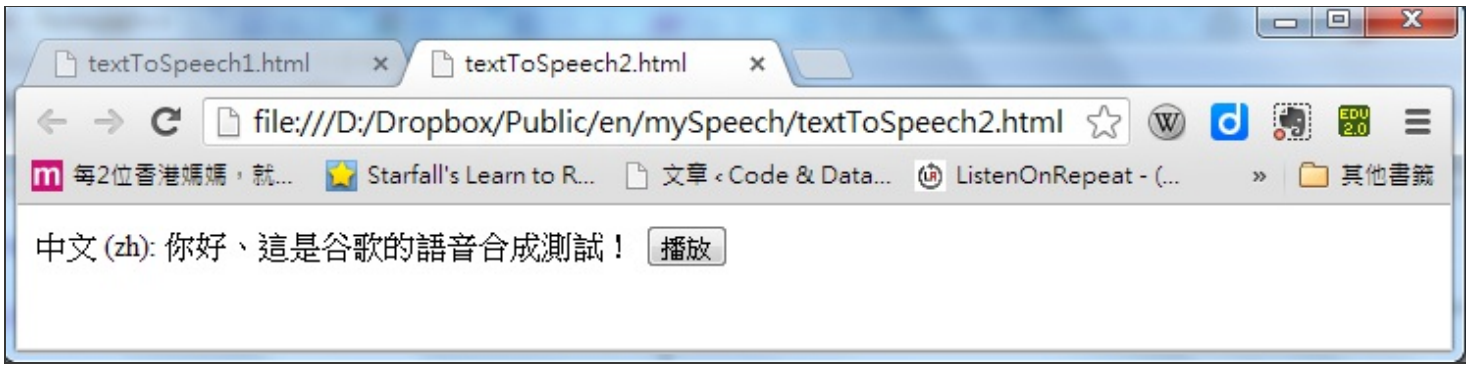
檔案：textToSpeech1.html 網址：https://dl.dropboxusercontent.com/u/101584453/pmag/201309/code/textToSpeech1.html

```
<html>
<head><meta charset="utf-8" /></head>
<body>
  <audio controls src="http://translate.google.com/translate_tts?ie=utf-8&tl=zh&q=你好、這是谷歌的語音合
  </audio>
  中文 (zh): 你好、這是谷歌的語音合成測試！ <BR/>
  <audio controls src="http://translate.google.com/translate_tts?ie=utf-8&tl=en&q=Hi! This is the text to spe
  </audio>
  英文 (en): Hi! This is the text to speech function of Google. <BR/>
</body>
</html>
```

說明：必須注意的一點是，目前並非每種瀏覽器都有支援 <audio> 標記，而且下節當中所使用的 JavaScript 程式也不見得都能正常運作， 像是在 IE 9.0 當中就似乎不支援 audio 標記，但是在 Firefox 22.0 與 Google Chrome 28.0 當中就支援了 audio 標記，而且 以下程式均能正常運作。

使用 **JavaScript** 控制 **audio** 元件

有時，我們不希望出現預設的語音控制項，但是卻又希望在某些按鈕被按下時能夠合成語音，這時候我們就需要撰寫一些 JavaScript 程式來控制 audio 元件的播放行為，這時我們可以呼叫 audio 元件的 play() 函數，以進行播放動作。舉例而言，以下程式就改用了按鈕控制



圖、按鈕按下後會播放合成的語音

檔案：textToSpeech2.html 網址：https://dl.dropboxusercontent.com/u/101584453/pmag/201309/code/textToSpeech2.html

```
<html>
<head><meta charset="utf-8" /></head>
<body>
  <audio id="audio1" src="http://translate.google.com/translate_tts?ie=utf-8&tl=zh&q=你好、這是谷歌的語
  </audio>
  中文 (zh): 你好、這是谷歌的語音合成測試!
  <button onclick="document.getElementById('audio1').play();">播放</button>
</body>
</html>
```

當然、上述的程式用途不大，因為播放的語句是固定的，如果我們想讓網頁能播放使用者輸入的內容，就必須要動態的在 audio 元件的 src 欄位當中，塞入使用者所輸入的文字，以下程式示範了這樣的功能。

檔案：textToSpeech3.html 網址：https://dl.dropboxusercontent.com/u/101584453/pmag/201309/code/textToSpeech3.html

```
<html>
<head><meta charset="utf-8" /></head>
<body>
<script>
 function playAudio(id, lang, text) {
  var audio = document.getElementById(id); // 取得 audio 控制項
    audio. src = "http://translate.google.com/translate_tts?ie=utf-8&tl="+lang+"&q="+text; // 設定語音為 g
  audio.addEventListener('ended', function(){ this.currentTime = 0; }, false); // 當播放完畢，強制
  audio.play(); // 播放語音。
```

```
  }
</script>
  <audio id="audio1"></audio>
  <textarea id="text" rows=10 cols=60>
  你好、這是谷歌的語音合成測試!
  我們呼叫谷歌翻譯的 TTS API 去合成語音。
  </textarea><BR/>
  <button onclick="playAudio('audio1', 'zh', document.getElementById('text').value);">播放</button>
</body>
</html>
```

在上面的程式中，我們在 button 「播放」按鈕按下時，會呼叫下列指令去播放 text 這個 textarea 中的文字。

```
playAudio('audio1', 'zh', document.getElementById('text').value);
```

而在 playAudio() 函數中，我們用下列指令設定 audio 控制項的語音網址為 Google TTS 的網址，然後播放，如下程式碼所示：

```
    var audio = document.getElementById(id); // 取得 audio 控制項
    audio.src = "http://translate.google.com/translate_tts?ie=utf-8&tl="+lang+"&q="+text; // 設定語音為 g
    audio.play(); // 播放語音。
```

原本其實只要上述三行就夠了，但是由於在 Chrome 當中，audio 控制項在呼叫完 play() 之後，似乎並不會自動回到開頭， 導致第二次的無法播放聲音 (因為已經在最後了)，因此才需要加入下列這行，強制 play() 函數在播放完成之後回到開頭， 以便在下一次播放時能聽得到聲音。

```
    audio.addEventListener('ended', function(){ this.currentTime = 0; }, false); // 當播放完畢，強制
```

結語

現在、我們已經講解完整個 Google 語音合成 API 的使用方式了，筆者覺得 Google 的這種設計方式其實很棒，讓我們可以 很容易的在任何網頁中加入 Text to Speech 的功能，而且不需要安裝任何的軟體。

不過、筆者發現目前的Google 語音合成服務還有幾個小問題，例如在中文的模式下，如果夾雜英文的時候，Google TTS 會用 逐字的方式念初英文。舉例而言，假如我們想讓 Google TTS 念初下列文章。

> Hello, 你好！

則 Google TTS 所念出的語句，將會變成下列情況：

> H. e. l. l. o, 你好！

這聽起來很怪！但可惜的是，筆者還不知道有沒有甚麼方法可以讓 Google TTS 平順的念出中文中所夾雜的英文，如果有人知道 也請告訴我！

不過如果將 {lang} 欄位設定為 en (英文模式)，那麼 Google TTS 就可以平順的念出英文，而且品質還不錯。所以目前如果要 做中英夾雜的發音，可能要用 JavaScript 自行將中英文切割，然後利用類似下列方法，自行切換 {lang} 欄位，以便能順利的念出 中英夾雜的句子，只是這樣真的很不方便就是了。

```
audio.addEventListener('ended', function() { ..... } );
```

在下期當中，我們將會繼續探討有關 Google 語音服務的主題，不過不再是「語音合成」 (Text to Speech) 了，而是「語音辨識」 (Speech to Text)。

這是一個在技術上更困難的主題，不過幸運的是，Google 已經幫我們完成了這些程式，我們只要懂得如何用 JavaScript 呼叫就行了。

參考文獻

- Google Text-To-Speech (TTS)
- The Unofficial Google Text-To-Speech API

R 統計軟體(6) – 迴歸分析 (作者：陳鍾誠)

簡介

在本系列文章的前兩篇當中，我們說明了如何用 R 軟體來進行估計與檢定，特別是有關平均值的估計與檢定。

- R 統計軟體(4) – 平均值的估計與檢定
- R 統計軟體(5) – 再探檢定

這種估計通常是對於某個算式結果的「點估計」與「區間估計」，被估計的對象是一個點。

但是、如果我們想要找尋的是，兩個以上變數之間的「運算式」關係，那麼就不能只用「估計」了，而必須採用「迴歸分析」的方法。

迴歸分析是在尋找這些變數之間的關係，通常是尋找「線性關係」，舉例而言，假如我們認為 y 與 x 之間具有線性關係，也就是 y 是 x 的線性函數，那麼我們可以將兩者之間的關係寫成 y= a + b * x ，其中 a 與 b 都是某個未知的常數。

當我們取得了很多組 (x,y) 的樣本 (x1, y1) (x2, y2) ..... (xk, yk) 時，我們就可以透過迴歸分析來尋找出這些未知的常數， 進而建立變數之間的線性方程關係式。

R 軟體中的 lm() 函數

在 R 軟體當中，用來做迴歸分析的是 lm() 函數，其函數原型如下：

- lm(formula, data, subset, weights, na.action, method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...)

通常，我們只要用到 formula 與 data 兩個參數就可以進行迴歸運算了，舉例而言，假如我們有 25 個樣本 xy = (x1, y1) (x2, y2) .... (x25, y25)，那麼我們就 可以用下列 lm 函數找出 x, y 之間的線性關係式。

- lm(y~x, xy)

當然、如果自變數不只一個，例如我們想尋找的是 y = a + b1 * x1 + b2 * x2 的話，那麼就可以用下列函數去計算出 a, b1, b2 等係數，以建立迴歸模型。

- lm(y~x1+x2, xy)

單一自變數的迴歸分析：完全線性，無誤差值

現在、就讓我們用 R 軟體來示範「迴歸分析」的做法，

```
> x = sample(1:10, 25, replace=TRUE) # 從 1 到 10 之中可重複的隨機抽出 25 個樣本
> x
 [1]  5  7  8  4  3  2  3  4  5  4  7  7  2  4  2 10  7  3  3  2  7  5 10  7
[25] 10
> y = 1+3*x    # 用這些 x 樣本透過線性關係式產生 y 樣本，這是完美的線性關係，完全沒有誤差。
> y
 [1] 16 22 25 13 10  7 10 13 16 13 22 22  7 13  7 31 22 10 10  7 22 16 31 22
[25] 31
> plot(x,y)  # 畫出 (x,y) 的圖形，您會發現所有點都分布在一條斜率為 3 的斜線上
> xy = data.frame(x, y)  # 讓 (x,y) 的配對形成 frame 變數，這樣才能做為 lm(formula, data) 中的 data 參
> xy  # 印出 xy frame 變數
    x  y
1   5 16
2   7 22
3   8 25
4   4 13
5   3 10
6   2  7
7   3 10
8   4 13
9   5 16
10  4 13
11  7 22
12  7 22
13  2  7
14  4 13
15  2  7
16 10 31
17  7 22
18  3 10
19  3 10
20  2  7
21  7 22
```

```
22  5 16
23 10 31
24  7 22
25 10 31
>
> model = lm(y~x, data=xy) # 開始作線性迴歸分析
> model    # 顯示分析結果，發現 截距 intercept 為  1, 且 x 的係數為 3，也就是 y=1+3*x，正確找出我們產

Call:
lm(formula = y ~ x, data = xy)

Coefficients:
(Intercept)            x
          1            3
```

單一自變數的迴歸分析：有誤差值

上述的範例雖然很完美，但是卻很不真實，因為在機率統計的世界中，通常有很多難以捕捉的「隨機性誤差」，反應在樣本上面。

現在、就讓我們再度進行一次迴歸分析，只不過這次我們將加入一些常態分布的誤差值進去。

```
> x = sample(1:10, 25, replace=TRUE)  # 從 1 到 10 之中可重複的隨機抽出 25 個樣本
> x
 [1]  5  7  8  4  3  2  3  4  5  4  7  7  2  4  2 10  7  3  3  2  7  5 10  7
[25] 10
> y = 1 + 3*x + rnorm(25, mean=0, sd=1) # 用這些 x 樣本透過線性關係式產生 y 樣本，其中的誤差是用
> xy = data.frame(x, y)  # 讓 (x,y) 的配對形成 frame 變數，這樣才能做為 lm(formula, data) 中的 data 參
> xy
    x         y
1   5 15.936440
2   7 22.382565
3   8 25.872976
4   4 11.879862
5   3 10.283478
6   2  7.259466
7   3 10.487880
8   4 12.330273
9   5 15.735540
10  4 11.933706
11  7 23.185950
12  7 20.830941
13  2  7.162297
14  4 13.798160
```

```
15  2  6.868275
16 10 33.310490
17  7 22.403416
18  3 10.481201
19  3 11.122462
20  2  7.646084
21  7 22.467235
22  5 14.943285
23 10 32.170245
24  7 22.300601
25 10 32.522192
> model2 = lm(y~x, xy, x=T) # 開始作線性迴歸分析
> model2 # 顯示分析結果，發現 截距 intercept 為  0.5345, 且 x 的係數為 3.1447，也就是 y=0.5345+3.144

Call:
lm(formula = y ~ x, data = xy, x = T)

Coefficients:
(Intercept)            x
     0.5345       3.1447

> model2$x
   (Intercept)  x
1            1  5
2            1  7
3            1  8
4            1  4
5            1  3
6            1  2
7            1  3
8            1  4
9            1  5
10           1  4
11           1  7
12           1  7
13           1  2
14           1  4
15           1  2
16           1 10
17           1  7
18           1  3
19           1  3
20           1  2
```

```
21           1  7
22           1  5
23           1 10
24           1  7
25           1 10
attr(,"assign")
[1] 0 1
>
```

兩組自變數的迴歸分析：完全線性，無誤差值

當然、我們不只可以做單一自變數的迴歸，也可以做多組自變數的迴歸，以下讓我們用 R 軟體來示範 y=a + b1 * x1 + b2 * x2 迴歸式的分析。

```
> x1 = sample(1:10, 25, replace=TRUE) # 產生第一個自變數的 25 個樣本值
> x2 = sample(1:8, 25, replace=TRUE) # 產生第二個自變數的 25 個樣本值
> y = 5 + 3 * x1 - 2 * x2 # 用這些 (x1, x2) 樣本透過線性關係式產生 y 樣本，這是完美的線性關係，完全沒
> x1
 [1]  8  8  8  2  6  3  4  1  5  4  2  1  6  4  2  4  1  5  7  2  9  2 10  4
[25]  5
> x2
 [1] 7 7 1 8 5 5 5 2 6 8 5 7 4 6 8 5 6 8 2 5 7 2 7 6 5
> y
 [1] 15 15 27 -5 13  4  7  4  8  1  1 -6 15  5 -5  7 -4  4 22  1 18  7 21  5
[25] 10
> yx12 = data.frame(y, x1, x2)  # 讓 (y, x1, x2) 的配對形成 frame 變數，這樣才能做為 lm(formula, dat
> yx12.model = lm(y~x1+x2, yx12) # 開始作線性迴歸分析
> yx12.model # 顯示分析結果，發現 截距 intercept 為  5, 且 x1 的係數為 3，x2 的係數為 -2 也就是 y=5+3

Call:
lm(formula = y ~ x1 + x2, data = yx12)

Coefficients:
(Intercept)           x1           x2
          5            3           -2

>
```

兩組自變數的迴歸分析：有誤差值

同樣的，對於兩組或多組自變數的情況，我們也可以加入「隨機誤差值」，來讓整個資料集更有真實感，以下是我們的「資料產生」與「迴歸分析」的過程。

```
> x1 = sample(1:10, 25, replace=TRUE) # 產生第一個自變數的 25 個樣本值
> x2 = sample(1:8, 25, replace=TRUE) # 產生第二個自變數的 25 個樣本值
> y2 = 5 + 3*x1-2*x2 + rnorm(25, mean=0, sd=5)
> y2x12 = data.frame(y2, x1, x2) # 讓 (y, x1, x2) 的配對形成 frame 變數，這樣才能做為 lm(formula, d
> y2x12
          y2 x1 x2
1  10.2069412  8  7
2  11.5760467  8  7
3  24.8724883  8  1
4  -3.4406110  2  8
5   9.0650415  6  5
6   8.2621227  3  5
7  18.7755635  4  5
8  -5.1753518  1  2
9  14.1795708  5  6
10 -2.9588236  4  8
11  4.4931402  2  5
12 -9.1706740  1  7
13 15.7826413  6  4
14 11.1684672  4  6
15 -4.2108325  2  8
16 14.0557877  4  5
17  2.9787818  1  6
18  0.2277253  5  8
19 31.3466157  7  2
20 11.2311146  2  5
21 17.9397316  9  7
22  6.1773147  2  2
23 17.5177323 10  7
24  1.1189083  4  6
25 15.5696626  5  5
> y2x12.model = lm(y~ x1+x2, y2x12) # 開始作線性迴歸分析
> y2x12.model # 顯示分析結果，發現 截距 intercept 為  5.315, 且 x1 的係數為 2.886，x2 的係數為 -1.997

Call:
lm(formula = y ~ x1 + x2, data = y2x12)

Coefficients:
(Intercept)           x1           x2
      5.315        2.886       -1.997

>
```

結語

透過上述的實驗，我們可以發現在沒有誤差的情況下，線性迴歸函數 lm() 都可以找出正確的模型，得到正確的「截距」與「係數值」， 而在有隨機誤差的情況下，線性迴歸函數 lm() 雖然沒有辦法完全環原正確的模型，但是也找到還算不錯的結果，這正是「迴歸分析」 這個工具的威力之所在阿！

參考文獻

- R 統計軟體(4) – 平均值的估計與檢定
- R 統計軟體(5) – 再探檢定
- 陳鍾誠的網站/免費電子書/R 統計軟體 -- http://ccckmit.wikidot.com/r:main
- 陳鍾誠的網站/免費電子書/機率與統計 (使用 R 軟體) -- http://ccckmit.wikidot.com/st:main

Verilog (3) – 組合邏輯電路 (作者：陳鍾誠)

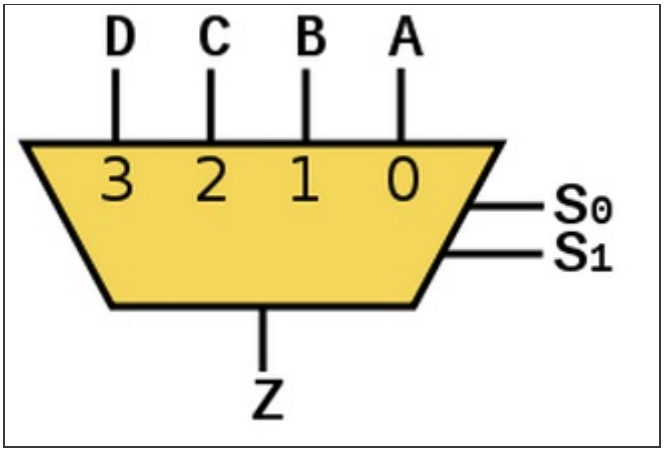
在數位電路當中，邏輯電路通常被分為兩類，一類是沒有「回饋線路」(No feedback) 的組合邏輯電路 (Combinatorial Logic)， 另一類是有「回饋線路」的循序邏輯電路 (Sequential Logic)。

組合邏輯的線路只是將輸入訊號轉換成輸出訊號，像是加法器、多工器等都是組合邏輯電路的範例，由於中間不會暫存，因此無法記憶位元。 而循序邏輯由於有回饋線路，所以可以製作出像 Flip-Flop，Latch 等記憶單元，可以記憶位元。

在本文中，我們將先專注在組合邏輯上，看看如何用基本的閘級寫法，寫出像多工器、加法器、減法器等組成 CPU 的基礎 電路元件。

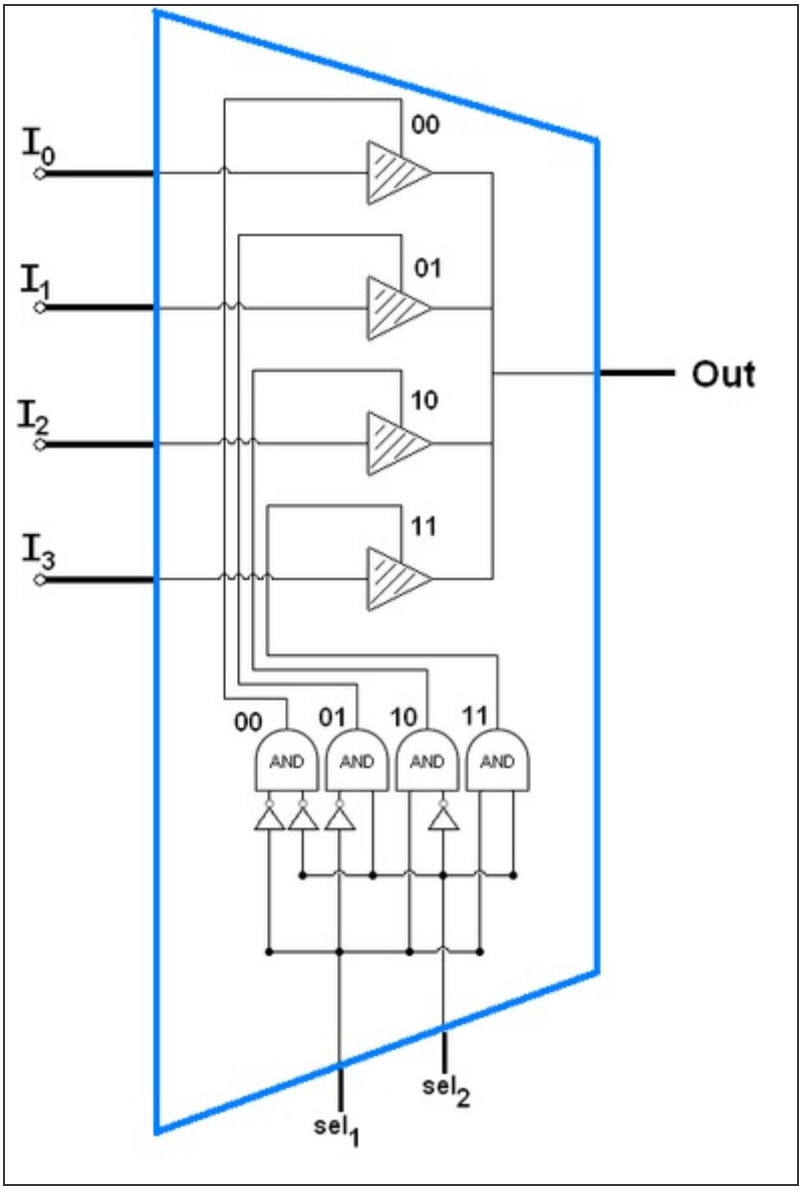
多工器

如果您曾經用硬接線的方式設計過 CPU，那就會發現「控制單元」主要就是一堆多工器的連接。多工器可以從很多組輸入資料中 選擇一組輸出，以下是一個四選一多工器的方塊圖。



圖、4 選 1 多工器

4 選 1 多工器的內部電路結構如下：



圖、4 選 1 多工器的內部電路

接著、就讓我們來看一個完整的 Verilog 的 4 選 1 的多工器程式，由於 Verilog 支援像 Case 這樣的高階語法，因此在實作時 可以不需要採用細部的接線方式，只要使用 case 語句就可以輕易完成多工器的設計。

檔案：mux4.v

```
module mux4(input[1:0]  select, input[3:0] d, output reg q );
always @( select or d )
begin
   case( select )
       0 : q = d[0];
       1 : q = d[1];
       2 : q = d[2];
       3 : q = d[3];
   endcase
end
endmodule
```

```
module main;
reg [3:0] d;
reg [1:0] s;
wire q;

mux4 DUT (s, d, q);

initial
begin
  s = 0;
  d = 4'b0110;
end

always #50 begin
  s=s+1;
  $monitor("%4dns monitor: s=%d d=%d q=%d", $stime, s, d, q);
end

initial #1000 $finish;

endmodule
```

執行結果

```
D:\ccc101\icarus>iverilog mux4.v -o mux4

D:\ccc101\icarus>vvp mux4
  50ns monitor: s=1 d= 6 q=1
 100ns monitor: s=2 d= 6 q=1
 150ns monitor: s=3 d= 6 q=0
 200ns monitor: s=0 d= 6 q=0
 250ns monitor: s=1 d= 6 q=1
 300ns monitor: s=2 d= 6 q=1
 350ns monitor: s=3 d= 6 q=0
 400ns monitor: s=0 d= 6 q=0
 450ns monitor: s=1 d= 6 q=1
 500ns monitor: s=2 d= 6 q=1
 550ns monitor: s=3 d= 6 q=0
 600ns monitor: s=0 d= 6 q=0
 650ns monitor: s=1 d= 6 q=1
 700ns monitor: s=2 d= 6 q=1
 750ns monitor: s=3 d= 6 q=0
 800ns monitor: s=0 d= 6 q=0
 850ns monitor: s=1 d= 6 q=1
```

```
 900ns monitor: s=2 d= 6 q=1
 950ns monitor: s=3 d= 6 q=0
1000ns monitor: s=0 d= 6 q=0
```

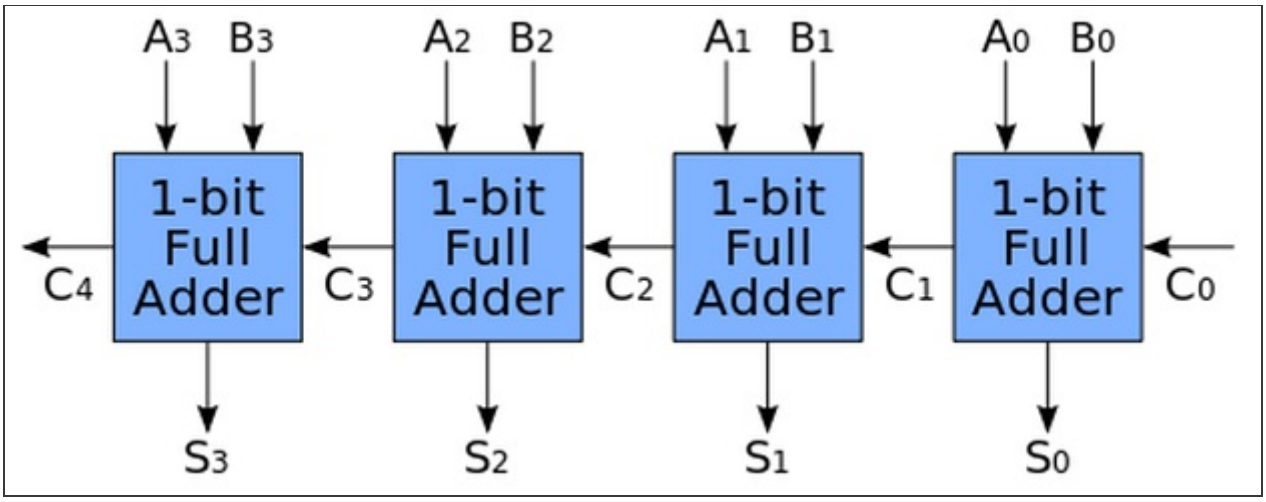
您可以看到在上述範例中，輸入資料 6 的二進位是 0110，如下所示：

```
位置 s  3 2 1 0
位元 d  0 1 1 0
```

因此當 s=0 時會輸出 0, s=1 時會輸出 1, s=2 時會輸出 1, s=3 時會輸出 0，這就是上述輸出結果的意義。

加法器

接著、讓我們用先前已經示範過的全加器範例，一個一個連接成四位元的加法器，電路圖如下所示



圖、用 4 個全加器組成 4 位元加法器

上圖寫成 Verilog 就變成以下 adder4 模組的程式內容。

```
module adder4(input signed [3:0] a, input signed [3:0] b, input c_in, output signed [3:0] sum, ou
wire [3:0] c;

fulladder fa1(a[0],b[0], c_in, sum[0], c[1]) ;
fulladder fa2(a[1],b[1], c[1], sum[1], c[2]) ;
fulladder fa3(a[2],b[2], c[2], sum[2], c[3]) ;
fulladder fa4(a[3],b[3], c[3], sum[3], c_out) ;

endmodule
```

以下是完整的 4 位元加法器之 Verilog 程式。

檔案：adder4.v

```
module fulladder (input a, b, c_in, output sum, c_out);
```

```
wire s1, c1, c2;

xor g1(s1, a, b);
xor g2(sum, s1, c_in);
and g3(c1, a,b);
and g4(c2, s1, c_in) ;
xor g5(c_out, c2, c1) ;

endmodule

module adder4(input signed [3:0] a, input signed [3:0] b, input c_in, output signed [3:0] sum, ou
wire [3:0] c;

fulladder fa1(a[0],b[0], c_in, sum[0], c[1]) ;
fulladder fa2(a[1],b[1], c[1], sum[1], c[2]) ;
fulladder fa3(a[2],b[2], c[2], sum[2], c[3]) ;
fulladder fa4(a[3],b[3], c[3], sum[3], c_out) ;

endmodule

module main;
reg signed [3:0] a;
reg signed [3:0] b;
wire signed [3:0] sum;
wire c_out;

adder4 DUT (a, b, 1'b0, sum, c_out);

initial
begin
  a = 4'b0101;
  b = 4'b0000;
end

always #50 begin
  b=b+1;
  $monitor("%dns monitor: a=%d b=%d sum=%d", $stime, a, b, sum);
end

initial #2000 $finish;

endmodule
```

執行結果

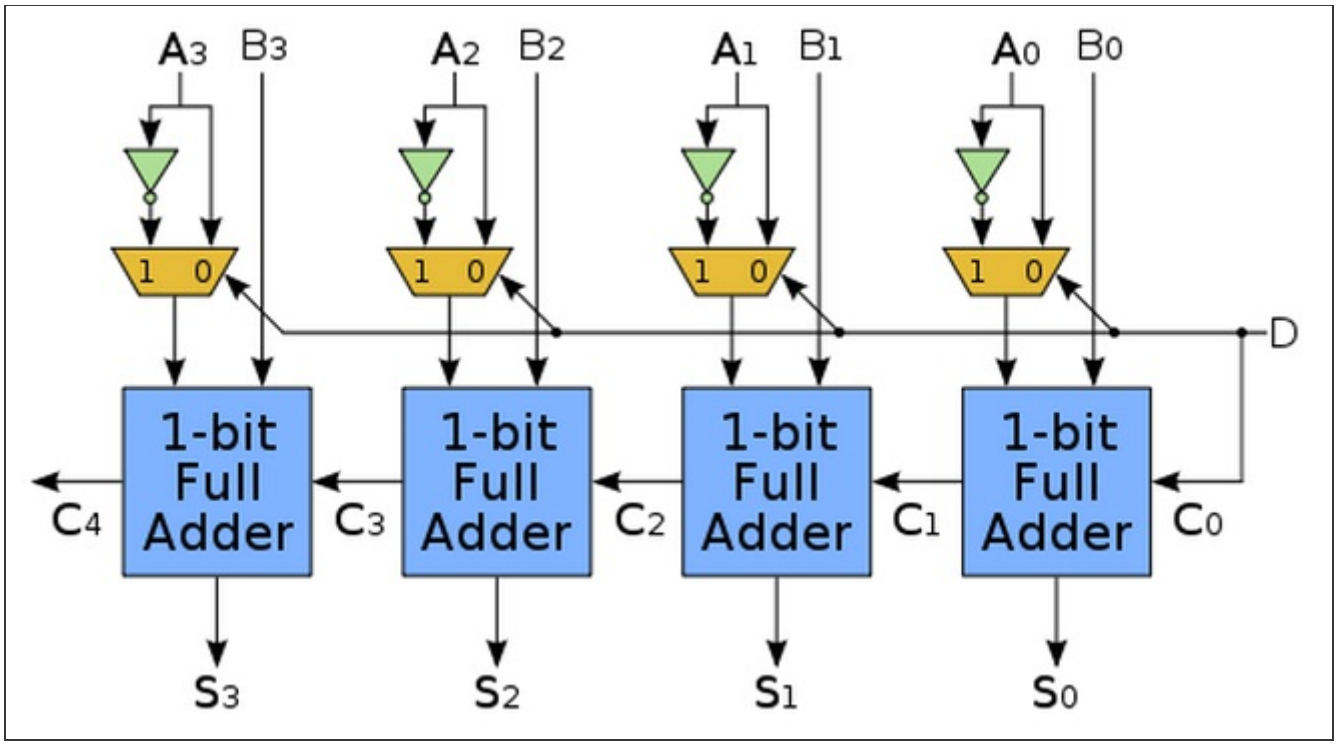
```
D:\ccc101\icarus\ccc>iverilog -o sadd4 sadd4.v

D:\ccc101\icarus\ccc>vvp sadd4
        50ns monitor: a= 5 b= 1 sum= 6
       100ns monitor: a= 5 b= 2 sum= 7
       150ns monitor: a= 5 b= 3 sum=-8
       200ns monitor: a= 5 b= 4 sum=-7
       250ns monitor: a= 5 b= 5 sum=-6
       300ns monitor: a= 5 b= 6 sum=-5
       350ns monitor: a= 5 b= 7 sum=-4
       400ns monitor: a= 5 b=-8 sum=-3
       450ns monitor: a= 5 b=-7 sum=-2
       500ns monitor: a= 5 b=-6 sum=-1
       550ns monitor: a= 5 b=-5 sum= 0
       600ns monitor: a= 5 b=-4 sum= 1
       650ns monitor: a= 5 b=-3 sum= 2
       700ns monitor: a= 5 b=-2 sum= 3
       750ns monitor: a= 5 b=-1 sum= 4
       800ns monitor: a= 5 b= 0 sum= 5
       850ns monitor: a= 5 b= 1 sum= 6
       900ns monitor: a= 5 b= 2 sum= 7
       950ns monitor: a= 5 b= 3 sum=-8
      1000ns monitor: a= 5 b= 4 sum=-7
      1050ns monitor: a= 5 b= 5 sum=-6
      1100ns monitor: a= 5 b= 6 sum=-5
      1150ns monitor: a= 5 b= 7 sum=-4
      1200ns monitor: a= 5 b=-8 sum=-3
      1250ns monitor: a= 5 b=-7 sum=-2
      1300ns monitor: a= 5 b=-6 sum=-1
      1350ns monitor: a= 5 b=-5 sum= 0
      1400ns monitor: a= 5 b=-4 sum= 1
      1450ns monitor: a= 5 b=-3 sum= 2
      1500ns monitor: a= 5 b=-2 sum= 3
      1550ns monitor: a= 5 b=-1 sum= 4
      1600ns monitor: a= 5 b= 0 sum= 5
      1650ns monitor: a= 5 b= 1 sum= 6
      1700ns monitor: a= 5 b= 2 sum= 7
      1750ns monitor: a= 5 b= 3 sum=-8
      1800ns monitor: a= 5 b= 4 sum=-7
      1850ns monitor: a= 5 b= 5 sum=-6
      1900ns monitor: a= 5 b= 6 sum=-5
      1950ns monitor: a= 5 b= 7 sum=-4
      2000ns monitor: a= 5 b=-8 sum=-3
```

在上述執行結果中，您可以看到在沒有溢位的情況下，sum = a+b，但是一但加總值超過 7 之後，那就會變成負值，這也正是有號二補數表示法 溢位時會產生的結果。

加減器

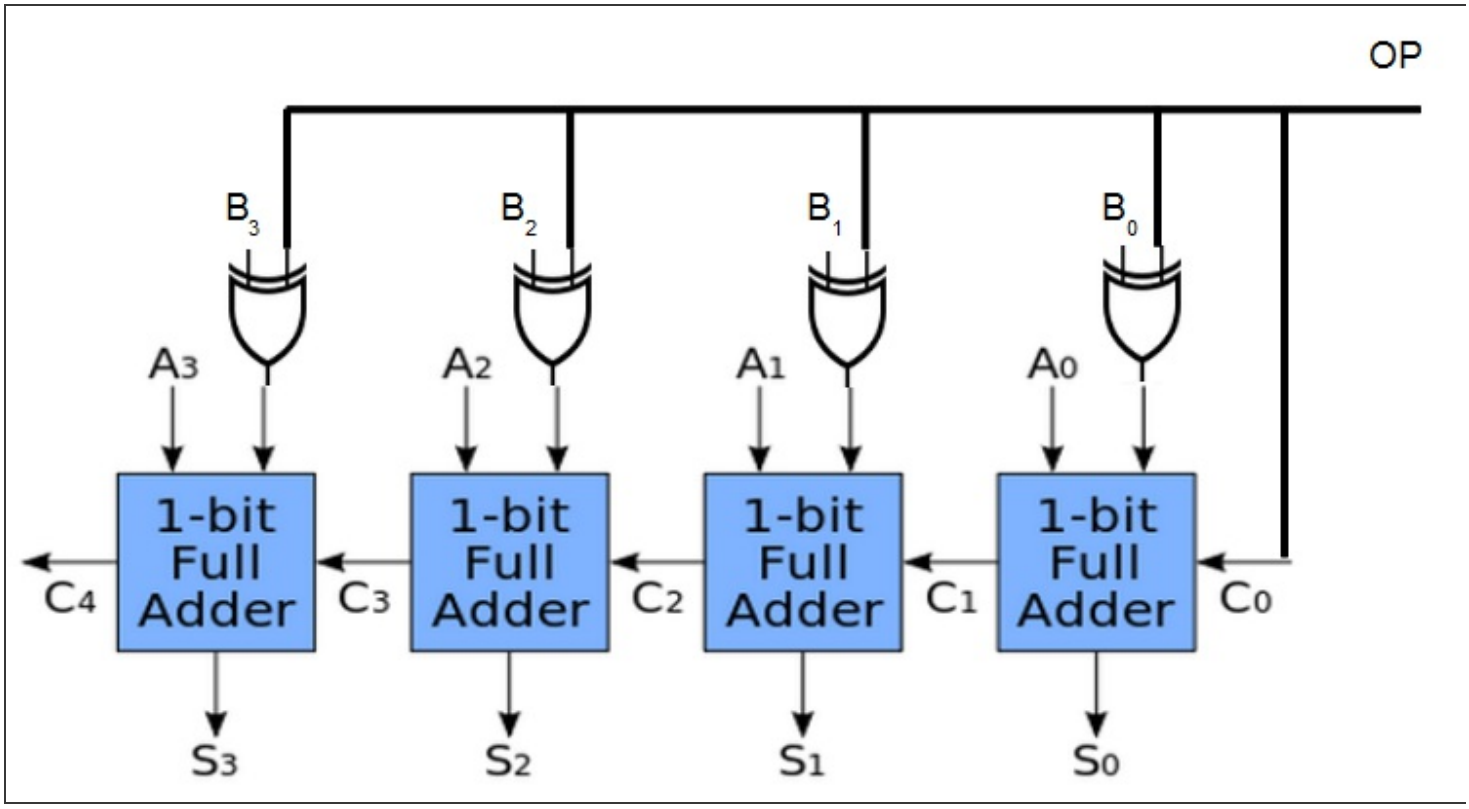
接著、我們只要把上面的加法器，加上一組控制的互斥或閘，並控制輸入進位與否，就可以成為加減器了，這是因為我們採用了二補數的關係。

二補數讓我們可以很容易的延伸加法器電路就能做出減法器。我們可以在運算元 B 之前加上 2 選 1 多工器或 XOR 閘來控制 B 是否應該取補數，並且 運用 OP 控制線路來進行控制，以下是採用 2 選 1 多工器的電路做法圖。



圖、採用 2 選 1 多工器控制的加減器電路

另一種更簡單的做法是採用 XOR 閘去控制 B 是否要取補數，如下圖所示：



圖、採用 XOR 控制的加減器電路

清楚了電路圖的布局之後，讓我們來看看如何用 Verilog 實做加減器吧！關鍵部分的程式如下所示，這個模組就對應到上述的「採用 XOR 控制的加減器電路」之圖形。

```
module addSub4(input op, input signed [3:0] a, input signed [3:0] b,
               output signed [3:0] sum, output c_out);

wire [3:0] bop;

xor4 x1(b, {op,op,op,op}, bop);
adder4 a1(a, bop, op, sum, c_out);

endmodule
```

接著讓我們來看看完整的加減器程式與測試結果。

檔案：addsub4.v

```
module fulladder (input a, b, c_in, output sum, c_out);
wire s1, c1, c2;

xor g1(s1, a, b);
xor g2(sum, s1, c_in);
and g3(c1, a,b);
and g4(c2, s1, c_in) ;
xor g5(c_out, c2, c1) ;
```

```
endmodule

module adder4(input signed [3:0] a, input signed [3:0] b, input c_in,
              output signed [3:0] sum, output c_out);
wire [3:0] c;

fulladder fa1(a[0],b[0], c_in, sum[0], c[1]) ;
fulladder fa2(a[1],b[1], c[1], sum[1], c[2]) ;
fulladder fa3(a[2],b[2], c[2], sum[2], c[3]) ;
fulladder fa4(a[3],b[3], c[3], sum[3], c_out) ;

endmodule

module xor4(input [3:0] a, input [3:0] b, output [3:0] y);
  assign y = a ^ b;
endmodule

module addSub4(input op, input signed [3:0] a, input signed [3:0] b,
               output signed [3:0] sum, output c_out);

wire [3:0] bop;

xor4 x1(b, {op,op,op,op}, bop);
adder4 a1(a, bop, op, sum, c_out);

endmodule

module main;
reg signed [3:0] a;
reg signed [3:0] b;
wire signed [3:0] sum;
reg op;
wire c_out;

addSub4 DUT (op, a, b, sum, c_out);

initial
begin
  a = 4'b0101;
  b = 4'b0000;
  op = 1'b0;
end
```

```
always #50 begin
  op=op+1;
  $monitor("%dns monitor: op=%d a=%d b=%d sum=%d", $stime, op, a, b, sum);
end

always #100 begin
  b=b+1;
end

initial #2000 $finish;

endmodule
```

執行結果：

```
D:\ccc101\icarus\ccc>iverilog -o addSub4 addSub4.v

D:\ccc101\icarus\ccc>vvp addSub4
        50ns monitor: op=1 a= 5 b= 0 sum= 5
       100ns monitor: op=0 a= 5 b= 1 sum= 6
       150ns monitor: op=1 a= 5 b= 1 sum= 4
       200ns monitor: op=0 a= 5 b= 2 sum= 7
       250ns monitor: op=1 a= 5 b= 2 sum= 3
       300ns monitor: op=0 a= 5 b= 3 sum=-8
       350ns monitor: op=1 a= 5 b= 3 sum= 2
       400ns monitor: op=0 a= 5 b= 4 sum=-7
       450ns monitor: op=1 a= 5 b= 4 sum= 1
       500ns monitor: op=0 a= 5 b= 5 sum=-6
       550ns monitor: op=1 a= 5 b= 5 sum= 0
       600ns monitor: op=0 a= 5 b= 6 sum=-5
       650ns monitor: op=1 a= 5 b= 6 sum=-1
       700ns monitor: op=0 a= 5 b= 7 sum=-4
       750ns monitor: op=1 a= 5 b= 7 sum=-2
       800ns monitor: op=0 a= 5 b=-8 sum=-3
       850ns monitor: op=1 a= 5 b=-8 sum=-3
       900ns monitor: op=0 a= 5 b=-7 sum=-2
       950ns monitor: op=1 a= 5 b=-7 sum=-4
      1000ns monitor: op=0 a= 5 b=-6 sum=-1
      1050ns monitor: op=1 a= 5 b=-6 sum=-5
      1100ns monitor: op=0 a= 5 b=-5 sum= 0
      1150ns monitor: op=1 a= 5 b=-5 sum=-6
      1200ns monitor: op=0 a= 5 b=-4 sum= 1
      1250ns monitor: op=1 a= 5 b=-4 sum=-7
      1300ns monitor: op=0 a= 5 b=-3 sum= 2
```

```
      1350ns monitor: op=1 a= 5 b=-3 sum=-8
      1400ns monitor: op=0 a= 5 b=-2 sum= 3
      1450ns monitor: op=1 a= 5 b=-2 sum= 7
      1500ns monitor: op=0 a= 5 b=-1 sum= 4
      1550ns monitor: op=1 a= 5 b=-1 sum= 6
      1600ns monitor: op=0 a= 5 b= 0 sum= 5
      1650ns monitor: op=1 a= 5 b= 0 sum= 5
      1700ns monitor: op=0 a= 5 b= 1 sum= 6
      1750ns monitor: op=1 a= 5 b= 1 sum= 4
      1800ns monitor: op=0 a= 5 b= 2 sum= 7
      1850ns monitor: op=1 a= 5 b= 2 sum= 3
      1900ns monitor: op=0 a= 5 b= 3 sum=-8
      1950ns monitor: op=1 a= 5 b= 3 sum= 2
      2000ns monitor: op=0 a= 5 b= 4 sum=-7
```

在上述結果中，您可以看到當 op=0 時，電路所作的是加法運算，例如：200ns monitor: op=0 a= 5 b= 2 sum= 7。而當 op=1 時， 電路所做的是減法運算，例如：250ns monitor: op=1 a= 5 b= 2 sum= 3。

結語

在本文中，我們大致將 CPU 設計當中最重要的組合邏輯電路，也就是「多工器、加法器與減法器」的設計原理說明完畢了，希望透過 Verilog 的實作方式， 能讓讀者更瞭解數位電路的設計原理，並且為接下來所要介紹的「開放電腦計畫」進行鋪路的工作，以便讓讀者能夠具備用 Verilog 設計 CPU 的基礎， 這樣在後續幾期的開放電腦計畫文章中，讀者才比較容易讀懂 CPU 的 Verilog 程式之設計原理。

參考文獻

- 陳鍾誠的網站：Verilog 電路設計 -- 多工器
- 陳鍾誠的網站：Verilog 電路設計 -- 4 位元加法器
- 陳鍾誠的網站：Verilog 電路設計 -- 加減器
- Wikipedia:Adder
- Wikipedia:Adder–subtractor
- Wikipedia:Multiplexer

【本文由陳鍾誠取材 (主要為圖片) 並修改自維基百科】

開放電腦計畫 (3) – VM0 虛擬機：使用 JavaScript+Node.js 實作 (作者：陳鍾誠)

在前幾期中，我們介紹了 CPU0 處理器的指令集，以及組譯器的實作方式，文章網址如下：

- 開放電腦計畫 (1) – 整體架構與 CPU0 處理器
- 開放電腦計畫 (2) – AS0 組譯器：使用 JavaScript+Node.js 實作

在本文中，我們將接續前兩篇的內容，然後將焦點放在虛擬機 VM0 的實作上，說明一個最簡易的虛擬機是如何設計出來的。

組譯範例

首先、讓讀者回顧一下，在上一篇文章中，我們設計了一個組譯器，可以組譯像以下的組合語言程式。

組合語言：sum.as0

```
        LD      R1, sum       ; R1 = sum = 0
        LD      R2, i         ; R2 = i = 1
        LDI     R3, 10        ; R3 = 10
FOR:    CMP     R2, R3        ; if (R2 > R3)
        JGT     EXIT          ;   goto EXIT
        ADD     R1, R1, R2    ; R1 = R1 + R2 (sum = sum + i)
        ADDI    R2, R2, 1     ; R2 = R2 + 1  ( i  = i + 1)
        JMP     FOR           ; goto FOR
EXIT:   ST      R1, sum       ; sum = R1
        ST      R2, i         ; i = R2
        LD      R9, msgptr    ; R9= pointer(msg) = &msg
        SWI     3             ; SWI 3 : 印出 R9 (=&msg) 中的字串
        MOV     R9, R1        ; R9 = R1 = sum
        SWI     4             ; SWI 2 : 印出 R9 (=R1=sum) 中的整數
        RET                   ; return 返回上一層呼叫函數
i:      RESW    1             ; int i
sum:    WORD    0             ; int sum=0
msg:    BYTE    "1+...+10=", 0   ; char *msg = "sum="
msgptr: WORD    msg           ; char &msgptr = &msg
```

我們可以用 AS0 組譯器對這樣的 CPU0 組合語言進行組譯，以下是組譯過程與結果，會輸出機器碼到目的檔中。

```
D:\Dropbox\Public\oc\code>node as0 sum.as0 sum.ob0
...
...
=================SAVE OBJ FILE================

00 :  001F003C 002F0034 0830000A 10230000
10 :  2300000C 13112000 1B220001 26FFFFEC
20 :  011F001C 012F0014 009F001D 2A000003
30 :  12910000 2A000002 2C000000 00000000
40 :  00000000 73756D3D 00000000 44
```

接著、我們就可以用虛擬機 VM0 來執行這個目的檔，我們可以選擇用預設不傾印的方式，得到以下的簡要執行結果。

虛擬機執行過程 (不傾印)

```
D:\oc\code>node vm0 sum.ob0
1+...+10=55
```

也可以用加上 -d 參數的方式，傾印每一個指令的執行過程，如下所示：

虛擬機執行過程 (詳細傾印)

```
D:\oc\code>node vm0 sum.ob0 -d

00 :  001F003C 002F0034 0830000A 10230000
10 :  2300000C 13112000 1B220001 26FFFFEC
20 :  011F001C 012F0014 009F0022 2A000003
30 :  12910000 2A000004 2C000000 00000000
40 :  00000000 312B2E2E 2E2B3130 3D000000
50 :  0044
PC=0000 IR=001F003C SW=00000000 R[01]=0x00000000=0
PC=0004 IR=002F0034 SW=00000000 R[02]=0x00000000=0
PC=0008 IR=0830000A SW=00000000 R[03]=0x0000000A=10
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x00000000=0
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000001=1
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x00000001=1
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000002=2
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x00000003=3
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000003=3
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x00000006=6
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000004=4
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x0000000A=10
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000005=5
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x0000000F=15
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000006=6
```

```
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x00000015=21
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000007=7
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x0000001C=28
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000008=8
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x00000024=36
PC=0018 IR=1B220001 SW=80000000 R[02]=0x00000009=9
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=80000000 R[0C]=0x80000000=-2147483648
PC=0010 IR=2300000C SW=80000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=80000000 R[01]=0x0000002D=45
PC=0018 IR=1B220001 SW=80000000 R[02]=0x0000000A=10
PC=001C IR=26FFFFEC SW=80000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=40000000 R[0C]=0x40000000=1073741824
PC=0010 IR=2300000C SW=40000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=40000000 R[01]=0x00000037=55
PC=0018 IR=1B220001 SW=40000000 R[02]=0x0000000B=11
PC=001C IR=26FFFFEC SW=40000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=00000000 R[0C]=0x00000000=0
PC=0010 IR=2300000C SW=00000000 R[00]=0x00000000=0
m[0040]=55
PC=0020 IR=011F001C SW=00000000 R[01]=0x00000037=55
m[003C]=11
PC=0024 IR=012F0014 SW=00000000 R[02]=0x0000000B=11
PC=0028 IR=009F0022 SW=00000000 R[09]=0x00000044=68
1+...+10=PC=002C IR=2A000003 SW=00000000 R[00]=0x00000000=0
PC=0030 IR=12910000 SW=00000000 R[09]=0x00000037=55
55PC=0034 IR=2A000004 SW=00000000 R[00]=0x00000000=0
PC=0038 IR=2C000000 SW=00000000 R[00]=0x00000000=0
```

如果您詳細追蹤上述過程，就能更清楚的看出每個指令執行時，所造成的暫存器變化，舉例而言，您可以看到程式 在 PC=000C 到 PC=001C 之間循環了很多次，最後一次的循環印出下列內容。

```
PC=000C IR=10230000 SW=40000000 R[0C]=0x40000000=1073741824
PC=0010 IR=2300000C SW=40000000 R[00]=0x00000000=0
PC=0014 IR=13112000 SW=40000000 R[01]=0x00000037=55
PC=0018 IR=1B220001 SW=40000000 R[02]=0x0000000B=11
```

```
PC=001C IR=26FFFFEC SW=40000000 R[0F]=0x0000000C=12
PC=000C IR=10230000 SW=00000000 R[0C]=0x00000000=0
PC=0010 IR=2300000C SW=00000000 R[00]=0x00000000=0
m[0040]=55
```

其中得到 R[01]=0x00000037=55 的計算結果，正是整個程式計算 1+2+...+10=55 的結果。

VM0 虛擬機設計

接著、我們要來看看虛擬機 VM0 是如何設計的，但是在這之前，先讓我們看看虛擬機當中一個重要的資料結構， OpTable 指令表這個物件，其程式碼如下：

檔案：opTable.js

```
var c = require("./ccc");

var Op = function(line) {
    var tokens = line.split(/\s+/);
    this.name = tokens[0];
    this.id   = parseInt(tokens[1], 16);
    this.type = tokens[2];
}

var opTable = function(opList) {
  for (i in  opList) {
    var op = new Op(opList[i]);
    this[op.name] = op;
  }
}

opTable.prototype.ID = function(op) {
  return this[op].id;
}

opTable.prototype.dump=function() {
    for (key in this) {
      var op = this[key];
      if (typeof(op)!="function")
        c.log("%s %s %s", c.fill(' ', op.name, 8), c.hex(op.id, 2), op.type);
    }
}

module.exports = opTable;
```

然後、我們利用上述的 OpTable 模組，加入了 CPU0 的指令集之後，建出了 CPU0 這個代表處理器的模組， 程

式碼如下。

檔案：cpu0.js

```
var opTable = require("./optable");
var opList = [ "LD 00 L",  "ST  01 L",  "LDB 02 L",   "STB 03 L",  "LDR 04 L",
"STR 05 L",  "LBR 06 L",   "SBR 07 L",   "LDI 08 L",  "CMP 10 A",  "MOV 12 A",
"ADD 13 A",   "SUB 14 A",   "MUL 15 A",  "DIV 16 A",   "AND 18 A",  "OR  19 A",   "XOR 1A A",
"ADDI 1B A",  "ROL 1C A",   "ROR 1D A",  "SHL 1E A",  "SHR 1F A",
"JEQ 20 J",  "JNE 21 J",   "JLT 22 J",  "JGT 23 J",  "JLE 24 J",   "JGE 25 J",  "JMP 26 J",
"SWI 2A J",  "JSUB 2B J", "RET 2C J",  "PUSH 30 J",  "POP 31 J",   "PUSHB 32 J",
"POPB 33 J",  "RESW F0 D",  "RESB F1 D",  "WORD F2 D",  "BYTE F3 D"];

var cpu = { "opTable" : new opTable(opList) };

if (process.argv[2] == "-d")
    cpu.opTable.dump();

module.exports = cpu;
```

有了上述的兩個模組作為基礎，我們就可以開始撰寫虛擬機 VM0 了，以下是其原始程式碼。

檔案：vm0.js

```
var c = require("./ccc");
var cpu1 = require("./cpu0");
var Memory = require("./memory");

var isDump = process.argv[3] == "-d";

var IR = 16, PC = 15, LR = 14, SP = 13, SW = 12;
var ID = function(op) { return cpu1.opTable[op].id; }

var run = function(objFile) {
    R = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 13, -1, 0, 16];
    m = new Memory(1);
    m.load(objFile);
    if (isDump) m.dump();
    var stop = false;
    while (!stop) {                                          // 如果尚未結束
      var tpc = R[PC];
      R[0] = 0;                                              // R[0] 永遠為 0
      R[IR] = m.geti(R[PC]);                                 // 指令擷取，IR=[PC..PC+3]
      R[PC] += 4;                                            // 擷取完將 PC 加 4，指向下一個指令
```

```
      var op = c.bits(R[IR], 24, 31);                    // 取得 op 欄位，IR[24..31]
      var ra = c.bits(R[IR], 20, 23);                    // 取得 ra 欄位，IR[20..23]
      var rb = c.bits(R[IR], 16, 19);                    // 取得 rb 欄位，IR[16..19]
      var rc = c.bits(R[IR], 12, 15);                    // 取得 rc 欄位，IR[12..15]
      var c24= c.signbits(R[IR], 0,  23);                // 取得 24 位元的 cx
      var c16= c.signbits(R[IR], 0,  15);                // 取得 16 位元的 cx
      var c5 = c.bits(R[IR], 0,   4);                    // 取得 16 位元的 cx
      var addr  = R[rb]+c16;
      var raddr = R[rb]+R[rc];                           // 取得位址[Rb+Rc]
      var N  = c.bits(R[SW], 31, 31);
      var Z  = c.bits(R[SW], 30, 30);
//    c.log("IR=%s ra=%d rb=%d rc=%d c24=%s c16=%s addr=%s", c.hex(R[IR], 8), ra, rb, rc, c.hex(c2
      switch (op) {                                      // 根據op執行動作
        case ID("LD") : R[ra] = m.geti(addr); break;     // 處理 LD 指令
        case ID("ST") :                                  // 處理 ST 指令
          m.seti(addr, R[ra]);
          if (isDump) c.log("m[%s]=%s", c.hex(addr,4), m.geti(addr));
          break;
        case ID("LDB"): R[ra] = m.getb(addr); break;     // 處理 LDB 指令
        case ID("STB"): m.setb(addr, R[ra]); break;      // 處理 STB 指令
        case ID("LDR"): R[ra] = m.geti(raddr); break;    // 處理 LDR 指令
        case ID("STR"): m.seti(raddr, R[ra]); break;     // 處理 STR 指令
        case ID("LBR"): R[ra] = m.getb(raddr); break;    // 處理 LBR 指令
        case ID("SBR"): m.setb(raddr, R[ra]); break;     // 處理 SBR 指令
        case ID("LDI"): R[ra] = c16; break;              // 處理 LDI 指令
        case ID("CMP"): {                                // 處理 CMP指令，根據比較結果，設
          if (R[ra] > R[rb]) {                           // > : SW(N=0, Z=0)
            R[SW] &= 0x3FFFFFFF;                         // N=0, Z=0
          } else if (R[ra] < R[rb]) {                    // < : SW(N=1, Z=0, ....)
            R[SW] |= 0x80000000;                         // N=1;
            R[SW] &= 0xBFFFFFFF;                         // Z=0;
          } else {                                       // = : SW(N=0, Z=1)
            R[SW] &= 0x7FFFFFFF;                         // N=0;
            R[SW] |= 0x40000000;                         // Z=1;
          }
          ra = 12;
          break;
        }
        case ID("MOV"): R[ra] = R[rb]; break;            // 處理MOV指令
        case ID("ADD"): R[ra] = R[rb]+R[rc]; break;      // 處理ADD指令
        case ID("SUB"): R[ra] = R[rb]-R[rc]; break;      // 處理SUB指令
        case ID("MUL"): R[ra] = R[rb]*R[rc]; break;      // 處理MUL指令
        case ID("DIV"): R[ra] = R[rb]/R[rc]; break;      // 處理DIV指令
```

```
        case ID("AND"): R[ra] = R[rb]&R[rc]; break;           // 處理AND指令
        case ID("OR") : R[ra] = R[rb]|R[rc]; break;           // 處理OR指令
        case ID("XOR"): R[ra] = R[rb]^R[rc]; break;           // 處理XOR指令
        case ID("SHL"): R[ra] = R[rb]<<c5; break;             // 處理SHL指令
        case ID("SHR"): R[ra] = R[rb]>>c5; break;             // 處理SHR指令
        case ID("ADDI"):R[ra] = R[rb] + c16; break;           // 處理 ADDI 指令
        case ID("JEQ"): if (Z==1) R[PC] += c24; break;        // 處理JEQ指令 Z=1
        case ID("JNE"): if (Z==0) R[PC] += c24; break;        // 處理JNE指令 Z=0
        case ID("JLT"): if (N==1&&Z==0) R[PC] += c24; break; // 處理JLT指令 NZ=10
        case ID("JGT"): if (N==0&&Z==0) R[PC] += c24; break; // 處理JGT指令 NZ=00
        case ID("JLE"): if ((N==1&&Z==0)||(N==0&&Z==1)) R[PC]+=c24; break; // 處理JLE指令 NZ=1
        case ID("JGE"): if ((N==0&&Z==0)||(N==0&&Z==1)) R[PC]+=c24; break; // 處理JGE指令 NZ=
        case ID("JMP"): R[PC]+=c24; break;                    // 處理JMP指令
        case ID("SWI"):                                       // 處理SWI指令
            switch (c24) {
                case 3: c.printf("%s", m.getstr(R[9])); break;
                case 4: c.printf("%d", R[9]); break;
                default:
                    var emsg = c.format("SWI cx=%d not found!", c24);
                    c.error(emsg, null);
                    break;
            }
            break;
        case ID("JSUB"):R[LR] = R[PC]; R[PC]+=c24; break;     // 處理JSUB指令
        case ID("RET"): if (R[LR]<0) stop=true; else R[PC]=LR; break; // 處理RET指令
        case ID("PUSH"):R[SP]-=4; R[ra]=m.geti(addr); m.seti(R[SP], R[ra]); break; // 處理PUS
        case ID("POP"): R[ra] = m.geti(R[SP]); R[SP]+=4; break;      // 處理POP指令
        case ID("PUSHB"):R[SP]--; R[ra]=m.getb(addr); m.setb(R[SP], R[ra]); break; // 處理P
        case ID("POPB"):R[ra] = m.getb(R[SP]); R[SP]++; break;       // 處理POPB指令
        default: c.error("OP not found!", null);
      } // switch
      if (isDump)
        c.log("PC=%s IR=%s SW=%s R[%s]=0x%s=%d", // 印出 PC, IR, R[ra]暫存器的值，以利觀察
              c.hex(tpc,4), c.hex(R[IR],8), c.hex(R[SW],8), c.hex(ra,2), c.hex(R[ra], 8), R[
    } // while
}

run(process.argv[2]);
```

從上面的 VM0 虛擬機當中，您可以看到，假如不考慮執行速度的問題，那麼要撰寫一個虛擬機是非常容易的事情。 我們只要去忠實的模擬每一個指令所應該做的動作，就可以完成虛擬機的設計了。

讓我們稍微解釋一下上述 VM0 虛擬機的程式原理，請讀者將焦點先放在以下的程式片段中。

```
...
m = new Memory(1);
m.load(objFile);
var stop = false;
while (!stop) {                                          // 如果尚未結束
  ...
  R[IR] = m.geti(R[PC]);                                 // 指令擷取，IR=[PC..PC+3]
  R[PC] += 4;                                            // 擷取完將 PC 加 4，指向下一個指令
  var op = c.bits(R[IR], 24, 31);                        // 取得 op 欄位，IR[24..31]
  var ra = c.bits(R[IR], 20, 23);                        // 取得 ra 欄位，IR[20..23]
  var rb = c.bits(R[IR], 16, 19);                        // 取得 rb 欄位，IR[16..19]
  var rc = c.bits(R[IR], 12, 15);                        // 取得 rc 欄位，IR[12..15]
  var c24= c.signbits(R[IR], 0,  23);                    // 取得 24 位元的 cx
  var c16= c.signbits(R[IR], 0,  15);                    // 取得 16 位元的 cx
  var c5 = c.bits(R[IR], 0,   4);                        // 取得 16 位元的 cx
  var addr  = R[rb]+c16;
  var raddr = R[rb]+R[rc];                               // 取得位址[Rb+Rc]
  var N  = c.bits(R[SW], 31, 31);
  var Z  = c.bits(R[SW], 30, 30);
  switch (op) {                                          // 根據op執行動作
    case ID("LD") : R[ra] = m.geti(addr); break;         // 處理 LD 指令
    ...
    case ID("JMP"): R[PC]+=c24; break;                   // 處理JMP指令
    ...
    case ID("JSUB"):R[LR] = R[PC]; R[PC]+=c24; break;    // 處理JSUB指令
    ...
    case ID("RET"): if (R[LR]<0) stop=true; else R[PC]=LR; break; // 處理RET指令
    ...
```

首先我們用 m = new Memory(1) 建立一個空的記憶體，然後再用 m.load(objFile) 載入目的檔到記憶體中， 接著就開始進入 while (!stop) 起頭的指令解譯迴圈了，然後接著用 R[IR] = m.geti(R[PC]) 這個指令取出 程式計數暫存器 PC 所指到的記憶體內容 m[PC]，然後放到指令暫存器 IR 當中，接著就可以取出指令暫存器 IR 當中的欄位，像是指令碼 op、暫存器 ra, rb, rc 與常數部分 c24, c16, c5 等欄位。

然後就能對每個指令所應做的動作進行模擬，例如 LD 指令的功能是將記憶體位址 addr = R[rb]+c16 的內容 取出，放到編號 ra 的暫存器當中，因此只要用 R[ra] = m.geti(addr) 這樣一個函數呼叫，就可以完成模擬的動作了。

當然、有些模擬動作很簡單，可以用一兩個指令做完，像是 LD, ST, JMP 等都是如此，但有些動作就比較複雜，像是 JSUB, RET, PUSH, POP 等就要好幾個指令，最複雜的大概是 CMP 與 SWI 這兩個指令了，CMP 由於牽涉到比較動作 且需要設定 N, Z 等旗標，所以程式碼較長如下：

```
...
    case ID("CMP"): {                                    // 處理 CMP指令，根據比較結果，設
      if (R[ra] > R[rb]) {                               // > : SW(N=0, Z=0)
```

```
            R[SW] &= 0x3FFFFFFF;                    // N=0, Z=0
          } else if (R[ra] < R[rb]) {               // < : SW(N=1, Z=0, ....)
            R[SW] |= 0x80000000;                    // N=1;
            R[SW] &= 0xBFFFFFFF;                    // Z=0;
          } else {                                  // = : SW(N=0, Z=1)
            R[SW] &= 0x7FFFFFFF;                    // N=0;
            R[SW] |= 0x40000000;                    // Z=1;
          }
          ra = 12;
          break;
        }
...
```

而 SWI 則是軟體中斷，這個部分也可以不做任何事，不過如果要支援某些中斷函數的話，就可以在這個指令中進行模擬， 目前 SWI 指令處理的原始碼如下：

```
        case ID("SWI"):                             // 處理SWI指令
            switch (c24) {
                case 3: c.printf("%s", m.getstr(R[9])); break;
                case 4: c.printf("%d", R[9]); break;
                default:
                    var emsg = c.format("SWI cx=%d not found!", c24);
                    c.error(emsg, null);
                    break;
            }
            break;
```

目前我們支援兩個中斷處理呼叫，也就是 SWI 3 與 SWI 4。

其中的 SWI 4 會在螢幕上印出一個儲存在暫存器 R[9] 當中的整數，而 SWI 3 會在螢幕上印出一個記憶體當中的字串， 這個字串的起始位址也是儲存在暫存器 R[9] 當中的。

結語

透過 VM0，筆者希望能夠讓讀者清楚的瞭解虛擬機的設計方式，當然、VM0 是一個「跑得很慢」的虛擬機。

如果要讓虛擬機跑得很快，通常要搭配「立即編譯技術」(Just in Time Compiler, JIT) ，像是 Java 虛擬機 JVM 就是利用 JIT 才能夠讓 Java 程式跑得夠快。

另外、像是 VMWare、VirtualBox 等，則是在相同的 x86 架構下去執行的，因此重點變成「如何有效的繞過作業系統 的控管，讓機器碼在 CPU 上執行」的問題了。

在開放原始碼的領域，QEMU 是一個非常重要的虛擬機，其做法可以參考下列 QEMU 原作者 bellard 的論文：

- https://www.usenix.org/legacy/event/usenix05/tech/freenix/full_papers/bellard/bellard.pdf

摘要如下：

> The first step is to split each target CPU instruction into fewer simpler instructions called micro operations. Each micro operation is implemented by a small piece of C code. This small C source code is compiled by GCC to an object file. The micro operations are chosen so that their number is much smaller (typically a few hundreds) than all the combinations of instructions and operands of the target CPU. The translation from target CPU instructions to micro operations is done entirely with hand coded code. The source code is optimized for readability and compactness because the speed of this stage is less critical than in an interpreter.
>
> A compile time tool called dyngen uses the object file containing the micro operations as input to generate a dynamic code generator. This dynamic code generator is invoked at runtime to generate a complete host function which concatenates several micro operations.

筆者先前粗略的看了一下，原本以為「QEMU 則是機器法反編譯為 C 語言基本運算後，再度用 gcc 編譯 為機器碼，才能達到高速執行的目的」，但是仔細看又不是這樣，想想還是不要自己亂解釋好了，不過有高手 J 兄來信說明如下，原文附上：

> QEMU 採取的技術為 portable JIT，本質上是一種 template-based compilation，事先透過 TCG 做 code generation，使得 C-like template 得以在執行時期可對應到不同平台的 machine code，而執行時期沒有 gcc 的介入，我想這點該澄清。

像 bellard 這種高手寫的虛擬機，果然是又快又好啊！

VM0 與 QEMU 相比，速度上致少慢了幾十倍，不過程式碼絕對是簡單很多就是了。

在瞭解了 VM0 虛擬機之後，我們就要進入開放電腦計畫的另一個部分，CPU 的硬體設計了，這就不再是用 JavaScript 這樣 的高階語言去模擬 CPU 的行為了，而是直接用硬體描述語言 Verilog 來設計一顆 CPU。

當然、我們設計的 CPU 仍然是 CPU0，不過卻是用 Verilog 設計的 CPU0，可以被燒錄到 FPGA 上去執行，成為真正的硬體。

在這幾期的程式人雜誌中，我們已經開始介紹了 Verilog 語言，其目的也正是在為設計 Verilog 版的 CPU0 而鋪路， 有興趣的朋友們可以先看看這幾期的 Verilog 語言介紹，這樣就比較能在下期的「程式人雜誌」中，看懂如何用 Verilog 設計 CPU 了，以下是這幾期的 Verilog 相關文章列表。

- Verilog (1) – 以 Icarus 測試全加器
- Verilog (2) – 硬體語言的基礎
- Verilog (3) – 組合邏輯電路

當然、這短短幾篇文章無法完整的介紹 Verilog 的功能，另外再提供筆者先前的網誌給大家參考。

- 免費電子書：Verilog 電路設計, 陳鍾誠的網站。
- 計算機結構 -- 實務取向, 陳鍾誠的網站。

這學期，筆者也將開設一門「計算機結構」的課程，筆者最近努力為每一門課都寫一本電子書，這門課也不例外，而且 正是採用以 CPU0 為核心的寫法。雖然還沒完成，但裏面已經包含了上次開課時所做的教學錄影，或許讀者會有興趣， 在此一並提供給讀者參考。

- 電子書：計算機軟硬體結構, 開放電腦計畫 -- 陳鍾誠。

R 講題分享 – 利用 R 和 Shiny 製作網頁應用 (作者：Taiwan R User Group)

簡介Shiny

大家好，這篇文章要跟大家分享的是Rstudio這間公司在2012年釋出的R 套件: Shiny。

Shiny的設計目標，是希望讓不懂網頁技術的R使用者，可以用最短的時間，將他們的分析結果呈現在網站上和使用者互動。而且，透過Shiny，開發者只需要懂R的語法，就可以寫網頁了！（報告學長，完全不用學HTML，完全不用學javascript）

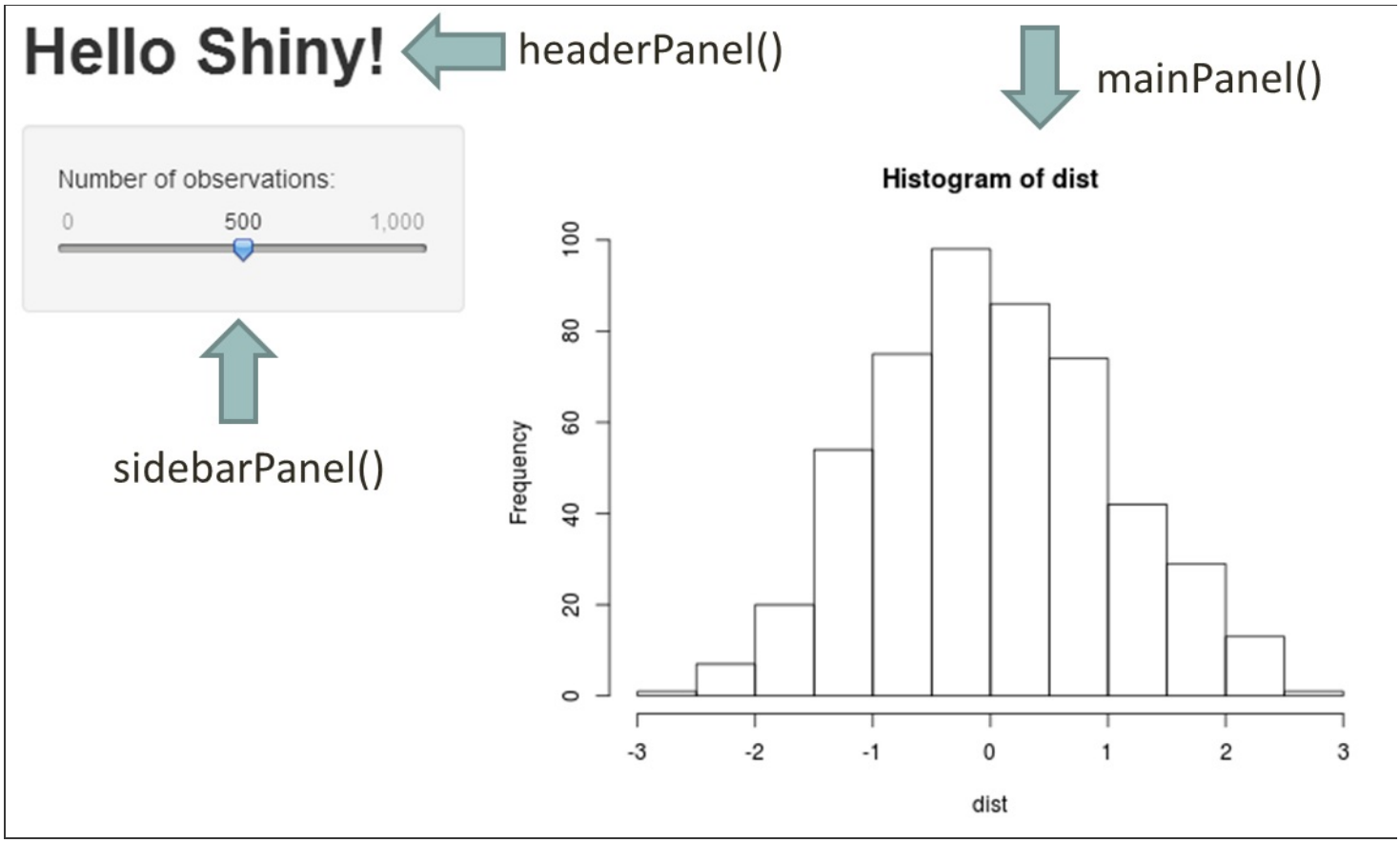
推出至今，在R社群內獲得廣大的迴響。在這篇文章中，我們將介紹近期最火紅的R套件: Shiny，並希望讀者能在閱讀後，獲得使用Shiny來建立網頁應用的能力，以更多元的方式呈現分析結果。

Hello Shiny

使用的第一步，當然是先到CRAN下載Shiny來安裝，並且試著執行範例(請一定要在本機上執行，在遠端伺服器上執行會出錯):

```
library(shiny)
runExample("01_hello")
```

執行後會自動打開預設瀏覽器，並看到如圖一顯示的網頁。讀者可以試著拉拉看左上角的拉條(slide bar)來調整數量，右邊的圖片會隨著更新！這就是Shiny所強調的互動式網頁應用。



圖一

Shiny也提供網頁模版供開發者使用。圖一中使用的模版pageWithSidebar，將網頁切割為以下三個部份： - 標題，也就是 **Hello Shiny!** - 控制面板(輸入)，左上角的灰色區塊 - 顯示面板(輸出)，右半邊的圖片

runExample("01_hello")的效果等同於在目錄下建立以下兩個檔案:

ui.R:

```
library(shiny)

# Define UI for application that plots random distributions
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Hello Shiny!"),

  # Sidebar with a slider input for number of observations
  sidebarPanel(
    sliderInput("obs",
                "Number of observations:",
                min = 1,
                max = 1000,
                value = 500)
  ),

  # Show a plot of the generated distribution
  mainPanel(
    plotOutput("distPlot")
  )
))
```

server.R:

```
library(shiny)

# Define server logic required to generate and plot a random distribution
shinyServer(function(input, output) {

    # Expression that generates a plot of the distribution. The expression is
    # wrapped in a call to renderPlot to indicate that:
    #
    # 1) It is 'reactive' and therefore should be automatically re-executed
    # when inputs change 2) Its output type is a plot
    output$distPlot <- renderPlot({
```

```
        # generate an rnorm distribution and plot it
        dist <- rnorm(input$obs)
        hist(dist)
    })
})
```

接著再執行:

```
runApp()
```

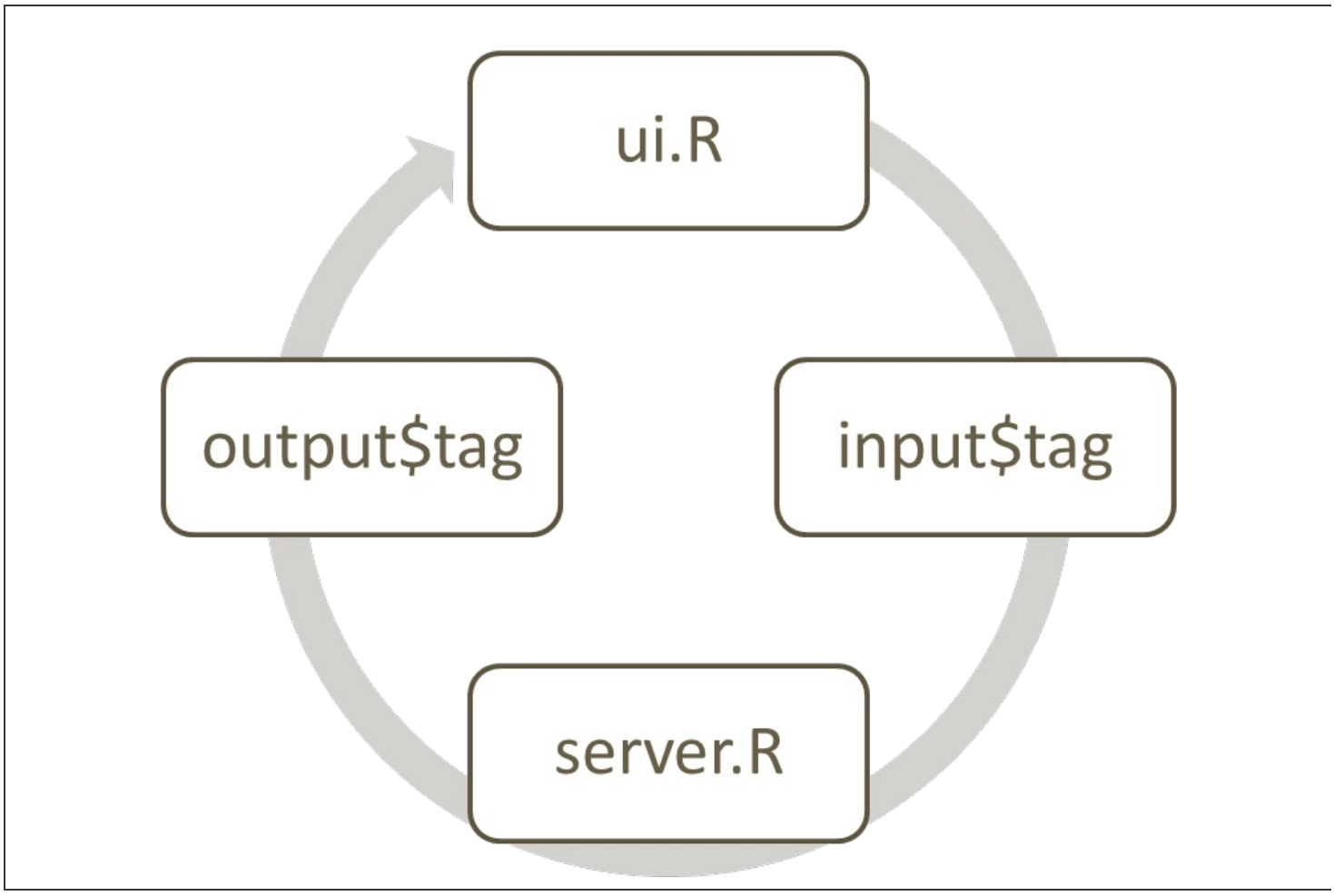
以下本文將以「Hello Shiny!」為例介紹Shiny的架構。

Shiny的架構

所有Shiny的網頁應用，都可以切割成兩個部份：

1. 使用者在瀏覽器看到的網頁，這部份的程式碼放在ui.R。實際上網頁的呈現，以及互動式的效果，都是由瀏覽器所執行。
2. 背景執行的R，這部份的程式碼放在server.R。這裡的R 是由開發者於佈署網頁應用時就啟動的，並靜靜地等待瀏覽器傳送參數。

瀏覽器和R 之間，則是透過網路來溝通。



圖二

當使用者打開瀏覽器，連接到Shiny應用程式時，R 會依照ui.R中的程式碼，產生出使用者看到的網頁內容，並在瀏覽器中顯示。

使用者在控制面板更改參數之後，瀏覽器就會將新的參數傳遞給背景執行的R，也就是server.R 。R 就會依照參數和server.R中的程式碼， 產生對應的物件。這物件可能是圖片、報表，甚至是其他的網頁元件。最後server.R會將物件回傳到瀏覽器，以ui.R定義的方式呈現給使用者。 這就是互動式網頁背後的原理。

ui.R

當使用者打開網頁之後，R 會將shinyUI的參數轉換成瀏覽器中的網頁元件。shinyUI的參數都對應到一個網頁元件，而這些物件的參數又分別對應到元件中的元件，形成一種巢狀結構。

第一個元件: pageWithSidebar提供了預設的模版，裏面包含三個參數:

```
str(pageWithSidebar)
```

```
## function (headerPanel, sidebarPanel, mainPanel)
```

- headerPanel對應到上圖中標題部份的網頁元件，也就是 **Hello Shiny!**
- sidebarPanel對應著控制面板(輸入)部份的網頁元件，也就是左上角的灰色區塊
- mainPanel對應著顯示面板(輸出)部份的網頁元件，也就是右半邊的圖片

這裡的headerPanel是比較簡單的，我們只要把要當標題的字串物件傳入，就可以設定網頁的標題了。

sidebarPenl和mainPanel中則可以再放入其他的網頁元件。

sidebarPanel

```
str(sidebarPanel)
```

```
## function (...)
```

sidebarPanel函數接受任意數量的參數，而每一個參數就是顯示在網頁上控制面板的元件。

Shiny提供了幾乎所有基本的網頁輸入(Input)元件。上圖中使用的sliderInput元件，是供使用者透過拉條(slide bar)來輸入數值參數。

```
str(sliderInput)
```

```
## function (inputId, label, min, max, value, step = NULL, round = FALSE,
##     format = "#,##0.#####", locale = "us", ticks = TRUE, animate = FALSE)
```

inputId非常重要，是瀏覽器和server.R溝通的依據。請讀者先記得，上圖範例中的sliderInput元件的inputId叫作"obs"，之後會在server.R中看到對應的input$obs。

其他的參數，則會依據不同的輸入元件而有不同的意義。請讀者再參閱Shiny的說明文件，在此就不一一解釋。

mainPanel

```
str(mainPanel)
```

```
## function (...)
```

mainPanel函數也接受任意數量的參數，而每一個參數就是在網頁上顯示面板的元件。

Shiny也提供數種不同的輸出元件，從基本的文字輸出，到圖、表、HTML元件和javascript。圖一中的plotOutput就是輸出圖片的元件。

```
str(plotOutput)
```

```
## function (outputId, width = "100%", height = "400px")
```

這裡的outputId非常重要，也是瀏覽器和server.R溝通的依據。請讀者先記得，上圖範例中的plotOutput元件的outputId叫作"distPlot"。之後會在server.R中看到對應的output$distPlot。

server.R

server.R定義了背景執行的R 如何處理瀏覽器傳遞來的參數，以及如何產生對應的R 物件。為了建立互動式的效果，server.R和一般R 的程式是不同的。

server.R 是一種依照事件驅動(event driven)的架構來執行的程式。當開發者在伺服器上啟動Shiny時，R會先執行shinyServer之前的程式碼，再進入等待狀態。

事件驅動(event driven)

shinyServer中的程式碼，會等使用者打開網頁，並待瀏覽器傳送參數給R 之後，R才會依照reactive或是renderXXX等函數中定義的方式來處理這些參數，並且產生對應的物件。R 會再回傳給瀏覽器，並依照ui.R所定義的方式呈現。R 會再回到等待狀態，直到收到下一組參數。

收到參數，就是一個「事件」，而所有的動作都是在收到事件後才會發生的。所以程式碼的執行是沒有一定的順序。這就是「事件驅動」架構的原理。

shinyServer

```
str(shinyServer)
```

```
## function (func)
```

shinyServer只有一個函數參數: func。它的長相必須是:

```
function(input, output) { ... }
```

input代表ui.R送給server.R的物件；output代表server.R送給ui.R的物件。

Shiny幫開發者處理input和output的傳遞，所以開發者只需要專注於設計ui.R呈現的功能，和server.R處理參數以及產生回應的功能即可。

訊息的傳遞

func的參數input就是將ui.R傳遞回來的參數打包好的R物件，開發者只要透過input$obs就可以讀取使用者於控制面板中，inputId為"obs"的元件內的參數。

func的參數output則是會被送回給ui.R的物件。而output$distPlot這個物件，則會由outputId為"distPlot"的網頁元件來處理。

生成回傳物件(應用的核心)

開發者利用如

```
output$distPlot <- renderPlot({...})
```

的方式定義產生回傳物件的動作。以Hello Shiny!為例:

```
output$distPlot <- renderPlot({
    dist <- rnorm(input$obs)
    hist(dist)
})
```

當偵測到input$obs改變之後，server.R就會依照input$obs的值重新繪製長條圖，再將圖片透過renderPlot函數，傳遞給瀏覽器。瀏覽器再透過ui.R中的定義，使用outputId為"distPlot"的網頁元件呈現給使用者。在Hello Shiny!的例子中，就是plotOutput("distPlot")所對應的網頁元件，也就是使用者在圖一右半邊看到的長條圖。

繪圖的功能置於renderPlot({...})是重要的，因為這樣才會在事件發生(接收到ui.R的參數)後，重新執行繪圖功能，並更新網頁上的圖。Shiny互動式的功能就是透過這個機制達成的。開發者必須要分清楚哪些程式碼是靜態的，只需要執行一次(例如固定資料的匯入)，哪些程式碼是動態的，必須要放入reactive或renderXXX之間。

細解Hello Shiny

對於整個Shiny的架構有初步的理解之後，我們再仔細的講解Shiny處理runExample("01_hello")的流程。

1. 執行shinyUI之前的程式碼
2. 將頁面區隔為三大區塊: headerPanel、sidebarPanel和mainPanel
3. 依照各Panel的參數顯示網頁元件，這時候mainPanel中的plotOutput("distPlot")還沒有物件可以呈獻。
4. sidebarPanel中的sliderInput傳送預設參數給server.R
5. 啟動server.R，執行shinyServer之前的程式碼
6. 執行renderPlot({...})之間的程式碼，也就是:

```
dist <- rnorm(input$obs)
hist(dist)
```

依照sliderInput設定的參數，產生常態分佈的樣本，並且繪製長條圖(histogram)。

7. 將產生的長條圖放入output$distPlot之中，回傳到瀏覽器
8. 瀏覽器將output$distPlot的物件呈現於plotOutput("distPlot")之中

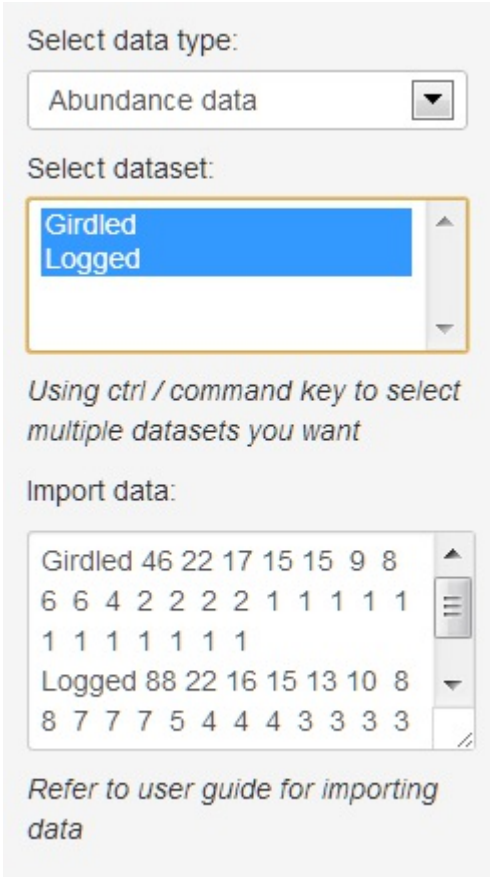
若使用者於瀏覽器更動sliderInput，那整個網頁應用就會從步驟4再跑一次，以實現互動式的效果。

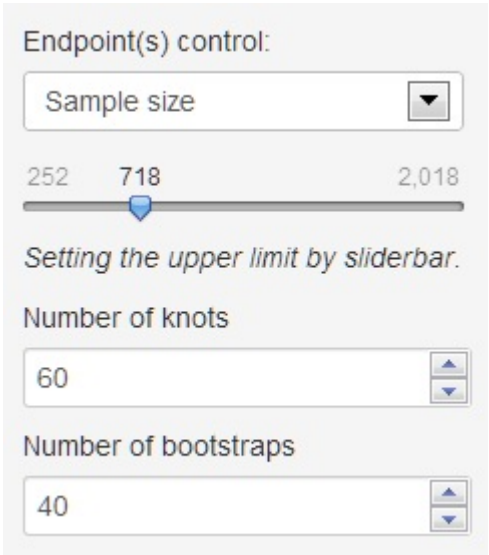
網頁元件範例

Shiny 中已經提供了各種基礎網頁元件。還有一個叫 **shinyExt** 的R 套件提供更多的元件。

本篇文章限於篇幅，無法一一講解，這部份只能請讀者依照自己的需求再去閱讀Shiny的套件說明。

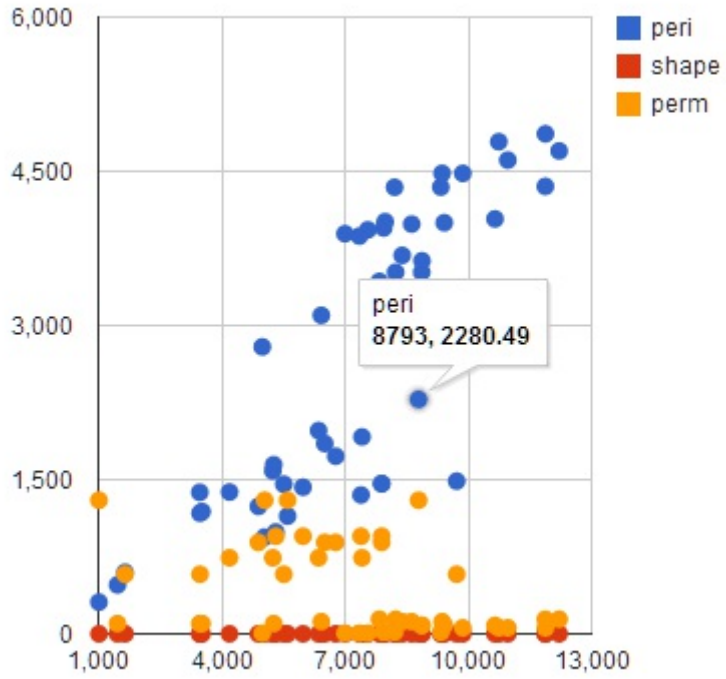
輸入元件範例

- 

- 


Choose CSV File
選擇檔案
example_2.png
Upload complete
Header
Separator
Comma
Semicolon
Tab
Quote
None
Double Quote
Single Quote

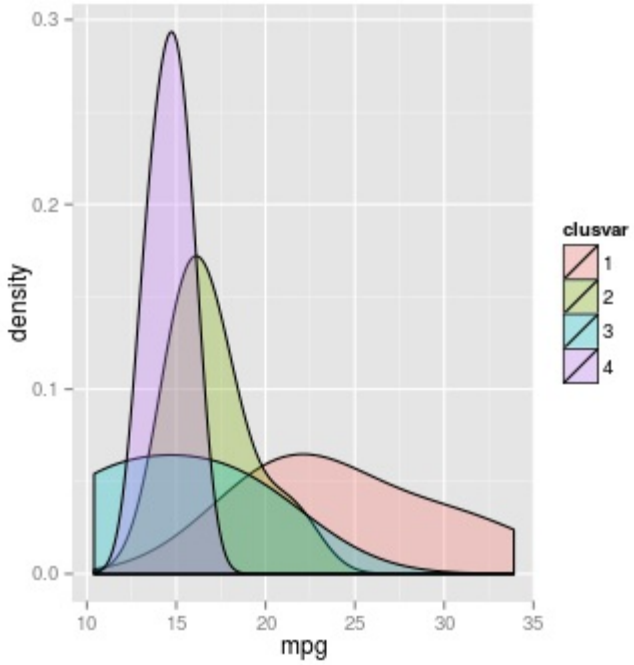
輸出元件範例


6,000
4,500
3,000
1,500
0
1,000
4,000
7,000
10,000
13,000
peri
shape
perm
peri
8793, 2280.49

```
Call:
glm(formula = formula, family = binomial(link = input$glm_linkfunc),

    data = dat)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
 -2.52   -0.23   -0.18    0.35    1.36

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)    15.97      11.02     1.4     0.15
mpg            -0.16       0.24    -0.7     0.50
cyl            -2.15       1.08    -2.0     0.05 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- rda | Save data

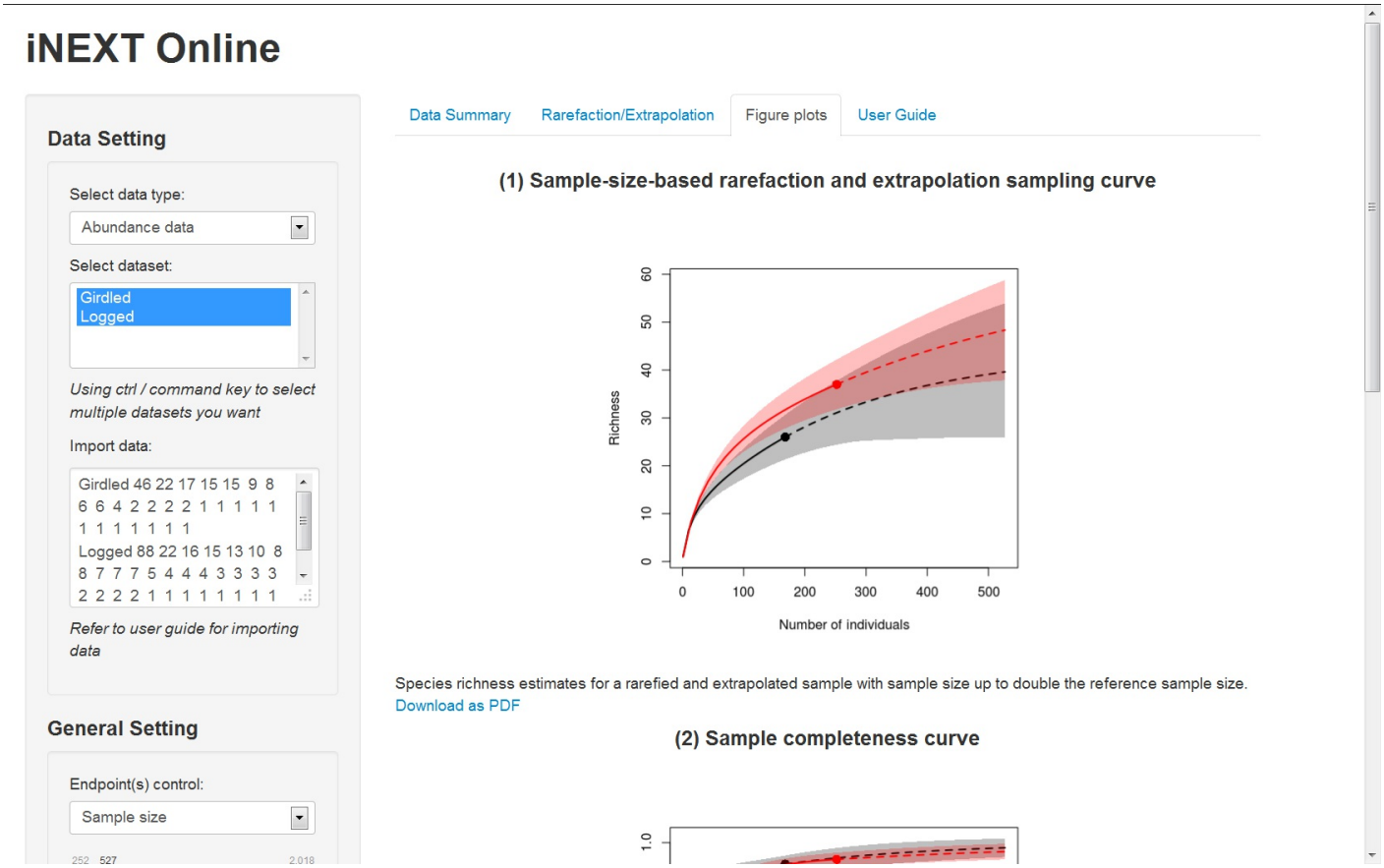
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	hclus4
Mazda RX4	21.00	6.00	160.00	110.00	3.90	2.62	16.46	0	1	4.00	4.00	1
Mazda RX4 Wag	21.00	6.00	160.00	110.00	3.90	2.88	17.02	0	1	4.00	4.00	1
Datsun 710	22.80	4.00	108.00	93.00	3.85	2.32	18.61	1	1	4.00	1.00	1
Hornet 4 Drive	21.40	6.00	258.00	110.00	3.08	3.21	19.44	1	0	3.00	1.00	1
Hornet Sportabout	18.70	8.00	360.00	175.00	3.15	3.44	17.02	0	0	3.00	2.00	2
Valiant	18.10	6.00	225.00	105.00	2.76	3.46	20.22	1	0	3.00	1.00	1
Duster 360	14.30	8.00	360.00	245.00	3.21	3.57	15.84	0	0	3.00	4.00	3
Merc 240D	24.40	4.00	146.70	62.00	3.69	3.19	20.00	1	0	4.00	2.00	4
Merc 230	22.80	4.00	140.80	95.00	3.92	3.15	22.90	1	0	4.00	2.00	1
Merc 280	19.20	6.00	167.60	123.00	3.92	3.44	18.30	1	0	4.00	4.00	1
Merc 280C	17.80	6.00	167.60	123.00	3.92	3.44	18.90	1	0	4.00	4.00	1
Merc 450SE	16.40	8.00	275.80	180.00	3.07	4.07	17.40	0	0	3.00	3.00	2
Merc 450SL	17.30	8.00	275.80	180.00	3.07	3.73	17.60	0	0	3.00	3.00	2
Merc 450SLC	15.20	8.00	275.80	180.00	3.07	3.78	18.00	0	0	3.00	3.00	2
Cadillac Fleetwood	10.40	8.00	472.00	205.00	2.93	5.25	17.98	0	0	3.00	4.00	3

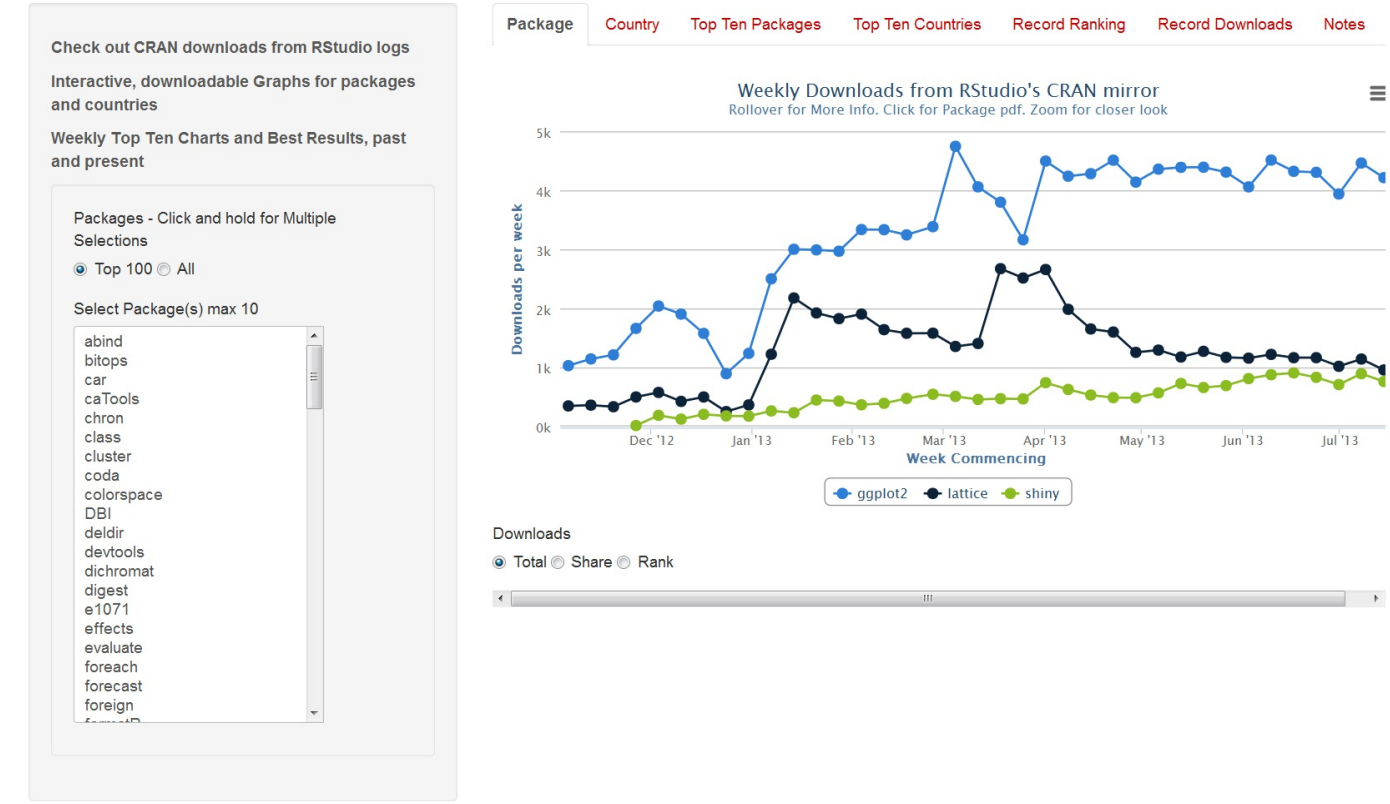


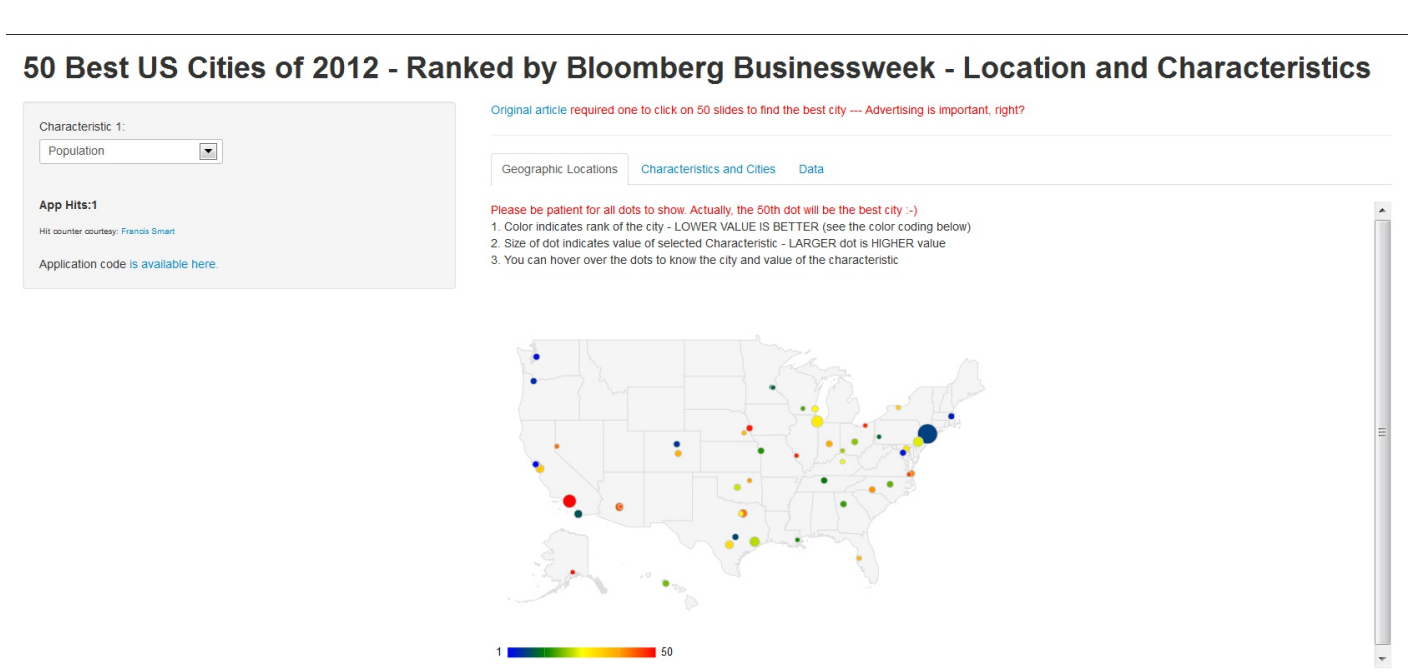
Shiny網頁應用範例

使用者可以到 http://www.rstudio.com/shiny/showcase/ 觀看其他開發者所開發的 Shiny 網頁應用。

如:

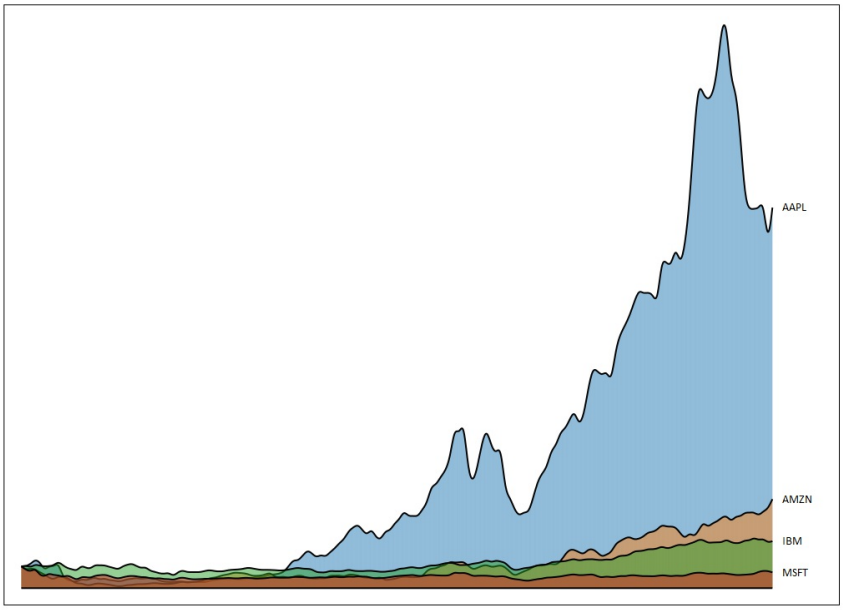



Check out CRAN downloads from RStudio logs
Interactive, downloadable Graphs for packages and countries
Weekly Top Ten Charts and Best Results, past and present
Packages - Click and hold for Multiple Selections
Top 100
All
Select Package(s) max 10
abind
bitops
car
caTools
chron
class
cluster
coda
colorspace
DBI
deldir
devtools
dichromat
digest
e1071
effects
evaluate
foreach
forecast
foreign
Package
Country
Top Ten Packages
Top Ten Countries
Record Ranking
Record Downloads
Notes
Weekly Downloads from RStudio's CRAN mirror
Rollover for More Info. Click for Package pdf. Zoom for closer look
Downloads per week
5k
4k
3k
2k
1k
0k
Dec '12
Jan '13
Feb '13
Mar '13
Apr '13
May '13
Jun '13
Jul '13
Week Commencing
ggplot2
lattice
shiny
Downloads
Total
Share
Rank


50 Best US Cities of 2012 - Ranked by Bloomberg Businessweek - Location and Characteristics
Characteristic 1:
Population
App Hits:1
Hit counter courtesy: Francis Smart
Application code is available here.
Original article required one to click on 50 slides to find the best city --- Advertising is important, right?
Geographic Locations
Characteristics and Cities
Data
Please be patient for all dots to show. Actually, the 50th dot will be the best city :-)
1. Color indicates rank of the city - LOWER VALUE IS BETTER (see the color coding below)
2. Size of dot indicates value of selected Characteristic - LARGER dot is HIGHER value
3. You can hover over the dots to know the city and value of the characteristic
1
50

d3 Showreel Example Using Data Supplied by R and Yahoo! Finance

All source available on Github



-

Radyant - Marketing analytics using Shiny View EDAT Regression Cluster Factor Conjoint

Datasets:
rock

Variables:
area {integer}
peri {numeric}
shape {numeric}
perm {numeric}

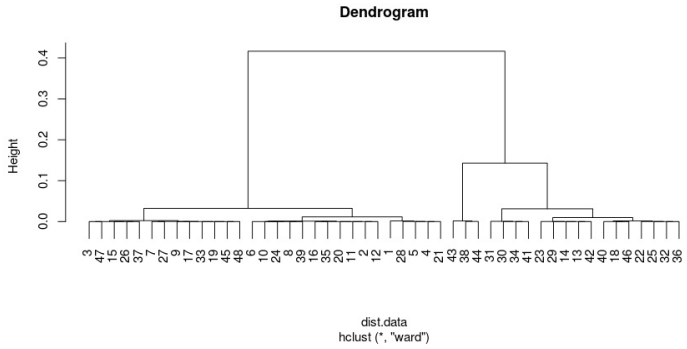
Distance measure:
sq.euclidian

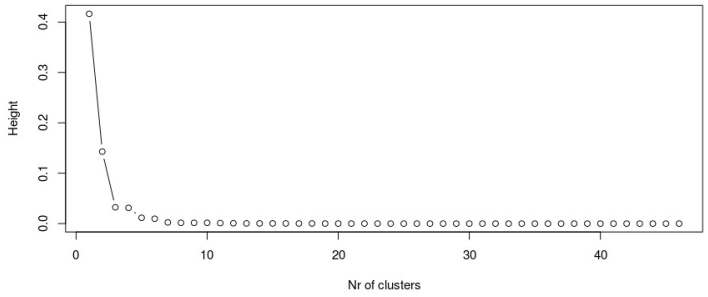
Method:
Ward's

Number of clusters
2

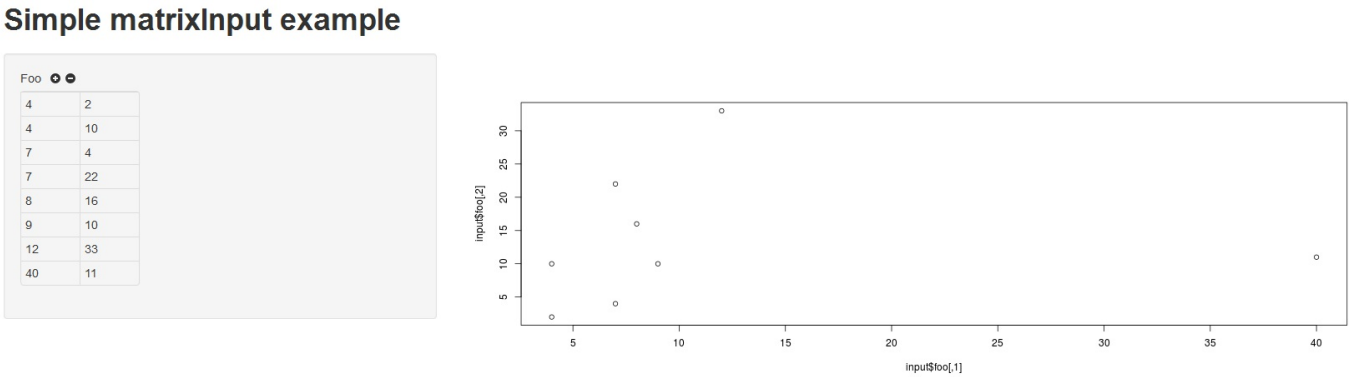
Save cluster membership

Summary Plots





-



-

開發建議

讀者如果想試著開發Shiny應用，我們有以下的tips供讀者參考:

- 先從 http://rstudio.github.io/shiny/tutorial/ 中找到符合需求的模版
- 把其他套件的載入，或資料源的設定，放置於shinyUI和shinyServer之前，如資料庫連線設定
- 先從ui.R開始建立使用者介面
- 網頁沒有回應，通常是ui.R有語法錯誤，runApp下方應該有錯誤訊息
- 在server.R中插入程式的核心演算法、分析方法。請務必理解server.R的架構，並且將功能正確的分為靜態的部份和動態的部份，才能將程式碼放置在正確的位置。靜態的部份放置於shinyServer之前，動態的部份會放在reactive，或對應的renderXXX裡
- 網頁有出現，但是mainPanel出現紅色錯誤訊息，代表server.R有錯誤
- 在server.R中插入browser()後，在runApp之後可以在執行相關事件時暫停在browser，開發者可以很方便的進行除錯

讀者如果對於基本的Shiny功能不滿意，可以在學習HTML/CSS/javascript後擴充Shiny網頁應用的功能。

佈署網頁應用

通常開發者撰寫出來的Shiny網頁應用可以依照以下方法來讓其他使用者使用:

直接傳遞相關檔案

開發者可以將撰寫好程式e-mail給使用者。

當使用者拿到相關檔案之後，仍需要有安裝R和Shiny，利用runApp來使用。

利用**Github**

開發者可以將撰寫好的ui.R和server.R放到github或gist。使用者仍需安裝R和Shiny，再利用runGist或runGithub來使用。

Rstudio的**Shiny**伺服器

開發者可以到 https://rstudio.wufoo.com/forms/shiny-server-beta-program/ 註冊，之後將撰寫好的ui.R和server.R上傳到Rstudio伺服器上。

使用者不需要安裝R，只需要打開瀏覽器後，連上相關網址就可以使用。

自行架設**Shiny**伺服器

請依照 https://github.com/rstudio/shiny-server 的說明建制Shiny伺服器。目前僅提供Linux版本。

建制完成後，使用者不需要安裝R，只需要打開瀏覽器，連上相關網址就可以使用。

參考資料

- Rstudio-Shiny 官方網站 http://www.rstudio.com/shiny/

作者

1. T.C. Hsieh (euler96@gmail.com)

- 清華統計所博士(2009-2013)，致力於發展沒人懂的統計方法與開發沒人用的統計軟體
- R 相關著作：
  - R套件，CARE1主要作者
  - Shiny應用，iNEXT-Online
- 研究領域：Statistics, ecology and genetics

2. Wush Wu (wush978@gmail.com)

- Taiwan R User Group Organizer
- R 相關著作：
  - RMessenger的作者
  - RSUS，這是On Shortest Unique Substring Query的實作
- 研究領域：Large Scale Learning，Text Mining和Uncertain Time Series

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)

想訂閱本雜誌的讀者，請按 雜誌訂閱 連結並填寫表單，我們會在每一期雜誌出刊時寄送通知與下載網址到您的信箱。

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 [程式人雜誌的出版方法] 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