

# 程式人

月刊  
雜誌



## Programmer



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



# 前言

## 編輯小語

由於小編放暑假暫回火星休息，因此「Verilog, R, 開放電腦計畫」等系列文章暫停一期。

不過這期的程式人雜誌更加精采，因為網友們投稿踴躍，有包含「Arduino、PROLOG、x86 組合語言、R 與 Java 的連接」等主題，希望讀者會喜歡。

----（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！

# 程式人短訊

## 課程短訊 – 線上教育 **MOOC** 的新發展

還記得我們在 2013 年 3 月號刊登訊息中介紹了線上教育 MOOC 的情況，其網址如下：

- [網路短訊-MOOC 大量線上教育](#)

經過了幾個月之後，MOOC 領域似乎有了一些新進展，像是 MOOC 的三大平台 (Coursera, Udacity 與 edX) 當中，由麻省理工 (MIT) 主導的 edX 這個平台，以 AGPL 授權的方式釋出了原始碼，您可以參考下列新聞：

- [edX 學習平台全面開放源碼](#)

其原始碼的下載網址如下：

- <http://code.edx.org/>

因此若您也想成立一個自己的 MOOC 網站，應該可以不用自己重寫程式了。

另外、台灣也開始有自己本土的 MOOC 網站了，網址如下：

- <http://www.sharecourse.net/>

該網站是由「捷鑿科技」製作的，內容目前大部分來自「清大、交大、資策會」等單位，課程目前還是集中在「資訊程式類」的領域較多，也有少數「媒體類」的課程。

雖然如此，筆者自己的課程還是習慣放在 YouTube 上，然後開 facebook 社團讓學生加入，作業也出在 facebook 上，然後用 markdown+pandoc 寫電子書，上傳到 dropbox 與 github 上公開，以下是筆者最近逐漸在整理的一些課程與教材，或許這也算是一種 MOOC 的實施方法吧！

- 陳鍾誠的教科書 -- <http://ccckmit.github.io/home/>

## 參考文獻

- [edX 學習平台全面開放源碼](#), OpenFoundary 電子報: 2013-06-25, 謝良奇.
- [edX learning platform now all open source](#), 05 June 2013, 16:36.
- <http://code.edx.org/>
- <http://www.sharecourse.net/>
- [清大+捷鑿 大學優質課「學聯網」傳給大眾](#), 2013/01/19, 【聯合報／記者李青霖／新竹報導】

【本文由陳鍾誠取材並修改自維基百科】

# 程式人介紹

## 約耳趣談軟體的作者 - Avram Joel Spolsky

很多人可能會問，Avram Joel Spolsky 是誰，但如果我說他是「約耳趣談軟體」的作者，或許很多人就知道了。在中國大陸，很多人都稱他為「周思博」，我想這應該是他的中文名字吧！



Joel 生於 1965 年，是一位猶太人，生於新墨西哥州的阿爾伯克基，並在那裡生活至15歲，此後便和家人搬到以色列耶路撒冷。在耶路撒冷，他就讀高中，並於其後以傘兵的身份服兵役。

1987年間，他返回美國讀大學。他在賓夕法尼亞大學就讀一年後，轉學到耶魯大學，就讀其下屬的皮爾遜學院，並於1991年畢業，獲得計算機科學最優等理學學士學位。

在1991年~1995年間，他擔任了 Microsoft Excel 小組的專案經理。並設計出 Excel Basic，主導了微軟的 Visual Basic for Application 之開發策略。

在1995年，他遷至紐約市，並先後就職於 Viacom 傳媒公司與 Juno 在線服務公司。

2000年，他創立了 Fog Creek Software 公司，並開設了一個稱為 [Joel on Software](#) 的 Blog，這些文章後來集合在「約耳趣談軟體」這本書中，該書流傳甚廣，而這些文章，很多在他的下列網站都可以免費閱讀。

- [Joel on Software](#)
- [周思博趣談軟體](#)
- [《約耳談軟體\(Joel on Software\)》翻譯計畫](#)

2005年間，Joel 與他人合作攝製了紀錄片《Aardvark'd: 12 Weeks with Geeks》，記錄了 Fog Creek Software在開發遠程協助工具 Project Aardvark 時的一些片段，該片網址如下：

- Aardvark'd: 12 Weeks with Geeks
  - <http://youtu.be/0NRL7YsXjSg>

2008 年，他與 Jeff Atwood 合作創建了 [StackOverflow](http://stackoverflow.com/) ( <http://stackoverflow.com/> )這個著名的程式 Q&A 網站。

後來 [StackOverflow](#) 持續擴展到不同領域，納入了數百個 Q&A 網站，成為 Stack Exchange Network，Joel 擔任

了 Stack Exchange Network 的 CEO。

2011 年，基於「看板管理」(Kanban) 理論，Joel 開發了一個簡潔的在線專案管理應用 [Trello](#)。

到目前為止，他已經出版了五本書籍，列表如下：

- [User Interface Design for Programmers](#), Apress, 2001. ISBN 1-893115-94-1
- [Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity](#), Apress, 2004. ISBN 1-59059-389-8
- [The Best Software Writing I: Selected and Introduced by Joel Spolsky](#), Apress, 2005. ISBN 1-59059-500-9
- [Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent](#), Apress, 2007. ISBN 1-59059-838-5
- [More Joel on Software: Further Thoughts on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity](#), Apress, 2008. ISBN 1-4302-0987-9

其中有兩本被翻譯為中文版，列表如下：

- [約耳趣談軟體：來自專案管理的現場實錄](#)
- [約耳續談軟體：探究軟體經營的根本實學](#)

## 參考文獻

- [Joel on Software](#)
- [周思博趣談軟體](#)
- [《約耳談軟體\(Joel on Software\)》翻譯計畫](#)
- [Joel's Personal website](#)
- [維基百科:周思博](#)
- <http://stackoverflow.com/>
- <http://stackexchange.com/>
- [Wikipedia:Stack Exchange Network](#)
- [Wikipedia:Trello](#)

【本文由陳鍾誠取材並修改自維基百科】

# 程式人頻道

## 看影片學 **Angular JS** 互動式網頁框架



最近在網友分享的訊息中，注意到有越來越多人分享了 Angular JS 的相關內容，讓我開始好奇 這到底是什麼樣的技術呢？

Angular JS 是一個網頁前端的 JavaScript 函式庫框架，與 ExtJS、BackboneJS、EmberJS、KnockoutJS 等框架的定位類似，比較適合用來開發資料管理 CURD (新增：Create, 修改：Update, 查詢：Read, 刪除：Delete) 類型的程式。

AngularJS 是由 Google 維護，通常用在單一頁面應用 (single-page applications) 上，使用 MVC (Model-View-Controllers) 架構。

其運作原理是採用雙向資料綁定 (two-way data binding) 的方式，將前端與後端結合起來，在前端利用「自定義標籤屬性」的 HTML 進行定義，而後端則透過樣板的方式進行溝通。

爲了解 Angular JS 到底是甚麼，我們從 Angular JS 的官網上找了一些影片，以便瞭解這個技術：

影片名稱	影片網址
AngularJS Hello World	<a href="http://youtu.be/uFTFsKmkQnQ">http://youtu.be/uFTFsKmkQnQ</a>
AngularJS Fundamentals In 60-ish Minutes	<a href="http://youtu.be/i9MHigUZKEM">http://youtu.be/i9MHigUZKEM</a>
Introduction to AngularJS	<a href="http://youtu.be/k4qVkJWh1EAo">http://youtu.be/k4qVkJWh1EAo</a>

以上的影片都是比較基礎級的，最簡單的是 AngularJS Hello World 這個只有 3 分 57 秒的影片中，您可以從中很快的看到 Angular JS 的功能示範，然後再看後面兩個影片。

如果還不過癮、您也可以從 Angular JS 的官方頻道當中，看到許多 Angular JS 的相關教學影片，網址如下：

- <http://www.youtube.com/user/angularjs>

另外、下列網站中也有中文的 Angular JS 文章與教學 (不過也有些是英文的)，閱讀中文的朋友可以參考看看：

- <http://www.good2u.com.tw/95>

筆者對 AngularJS 也還是一頭霧水，但是希望透過這篇文章的資源導引，讓對 AngularJS 有興趣的讀者可以找到學習的起點。

## 參考文獻

- 前端工程的極致精品：[AngularJS 開發框架介紹](#)
- AngularJS 官網 -- <http://angularjs.org/>
- Wikipedia:AngularJS -- <http://en.wikipedia.org/wiki/AngularJS>
- 維基百科:AngularJS -- <http://zh.wikipedia.org/wiki/AngularJS>

【本文由陳鍾誠修改自維基百科】



# 程式人討論區

## 新科技討論 - 南加州大學：Google量子計算機是真的

2013年7月2日，我看到以下的新聞，但是感到很疑惑，因為在我的印象中，量子電腦應該還沒辦法產品化阿！該怎麼看這樣的新聞呢？

- 南加州大學：Google量子計算機是真的
  - <http://www.hksilicon.com/kb/articles/180950/>

所以我就分享了這篇文章，發了下列訊息，看看有沒有高人可以提供訊息，

- <https://www.facebook.com/groups/programmerMagazine/permalink/668603293156390/>

以下是網友們的回應：

- Y○○：為什麼原創公司沒有說明呢？
- I○○：天網～
- 陳鍾誠 雖然我知道退火，也知道量子電腦，但是兩者之間的關係我還真的搞不清楚啊！
- Y○○：我樸素地理解，模擬退火是用亂數來模擬自然界亂的狀態，並且在狀態上藉著某些參數的調整來模擬降低溫度而讓狀態自然地達到穩定點而了解穩點解，然而因為狀態很多，所以就需要很大的計算量。量子上因為一個狀態是多重狀態的線性疊加，因此以一個狀態去作操作可能可以達到數倍（或更多？）狀態計算的結果，因此有可能達到近似於「直覺看到解答」的最終效果，而快速收斂到穩定解。我大概是這樣理解的。
- D○○：不好意思，我一直不太懂多重狀態如何增加運算能量？可以請先進幫助解釋嗎？感謝！
- U○○：個人比較好奇為什麼需要第三方的機構（南加州大學）用推測的方式「嘗試證明」是量子電腦，官方公司卻不提供詳細細節？（雖然原因大概能猜出來... 唉）
- 邱○○：量子電腦 還有一段長的路要走 例如 外部量子訊息如何讀取（磁碟片）如何進行運算（CPU）量子訊息如何Output（外部儲存裝置）而且這個系統必須要可以不斷的擴充 而且還有計算的結果必須要有高的可靠度（因為溫度會改變 量子的狀態）不過我對於他計算質數的能力 非常有興趣 可以是傳統電腦的一萬倍以上 這表示 以後要破解加密的密碼 不用花到一秒鐘
- 梁○○：這個小故事或可以說明量子電腦運算模型跟傳統之不同。愛迪生～沒讀太多書，所以找來一個數學很強的科學家來幫他設計燈泡，有一次～愛迪生丟一個燈泡給科學家：請告訴我它的容積。結果這個科學家～用了很複雜的微積分計算半天～還沒結果，愛迪生覺得很奇怪，為什麼搞這麼久～還沒答案？過去一看～這位老兄還低頭忙著計算。愛迪生說～這有什麼好算的？裝水進去～再倒出來量～不就有答案了！？傳統電腦模型就像計算微積分，按照步驟運算。目前期望可以帶來驚奇的量子運算～則是用測量得到答案！
- S○○：「量子退火」應該只是類比的名詞跟傳統資訊的操作不太一樣～不過說來好玩的是這原本也是模擬物理現象
- T○○：應該沒有意義，同樣的計算，換另一個方向去想，一般二進制計算的電腦也辦得到，例如以一個DWORD代表1個變數的 $2^{32}$ 個狀態，用現有的計算方式足夠了，不論是模糊邏輯，還是類神經元運算，至於運算能力，用分散式運算，遠比打造一台超貴的量子電腦會划算得多了
- S○○：在量子物理的詭異特性下，作某些運算的確會比傳統電腦快...
- T○○：目前，超級電腦的效能是以PFLOPS來評估的，以每秒 $10^{15}$ 次的浮點運算為1 PFLOPS，2011年以後，世界上有進排名的超級電腦，運算能力都10 PFLOPS以上了。如果Google不公布那台運算能力的話，那可能宣傳目的，大於實質意義，且基於電子學，線性運算，仍有不少的限制，一但超過工作區，也就等同二進

制運算的溢位了

- C○○：我第一個想到帽子世界=. =...
- T○○：換個角度去想，為何Intel當年要推 IA64，可是最後還是得跟著AMD的腳步去抄襲AMD64 改名叫作 Intel 64 的 x86-64 技術呢？因為x86的遺產龐大，讓User沒辦法拋棄它，就算量子運算的電腦能夠量產，如果還是得用模擬的，才能跑現有的軟體的話，我想它也沒辦法取代現有的數位運算的市場，只能用於特定用途，如此一來，還不如做成介面卡，直接擴充現有的數位運算的電腦
- U○○：這種東西會先是科學研究、大量運算使用這些領域就可以不用考慮什麼x86的問題，而且這種電腦架構是和普遍現在的電腦非常不同的，很多寫好的一般（假設1 bit絕對是0或是1）軟體也不一定能在上面跑 量子電腦其中一個重要特性就是1 bit可能不只是0或1，可以表達更多的狀態
- Y○○：以前在學校時聽到的量子力學演講，是例如我們常用的非對稱加密金鑰系統，是基於兩個大質數很難作因式分解。但是用量子電腦確有機會快速分解。目前一堆 SSL 憑證等等都靠這些東西。因此如果量子電腦成真，會是一些有高度價值的特殊應用！

然後我去查了維基百科當中「量子電腦」的內容，以下是摘要：

量子計算機是一種使用量子邏輯實現通用計算的設備。不同於電子計算機，量子計算用來存儲數據的對象是量子位元，它使用量子演算法來進行數據操作。一般認為量子計算機仍處於研究階段。然而2011年5月11日，加拿大的D-Wave System Inc. 發布了一款號稱「全球第一款商用型量子電腦」的計算設備「D-Wave One」。該量子設備是否真的實現了量子計算目前還沒有得到學術界廣泛認同。2013年5月D-Wave System Inc宣稱NASA和Google共同預定了一台採用512量子位的D-Wave Two量子電腦。.....

量子計算最本質的特徵為量子疊加性和量子相干性。....

在量子力學裏，態疊加原理（superposition principle）表明，假若一個量子系統的量子態可以是幾種不同量子態中的任意一種，則它們的歸一化線性組合也可以是其量子態。稱這線性組合為「疊加態」。假設組成疊加態的幾種量子態相互正交，則這量子系統處於其中任意量子態的機率是對應權值的絕對值平方 ...

但是這樣還是不夠清楚，所以我找到英文維基百科內的說明，發現寫得好很多，有興趣的讀者建議看以下這篇：

- [https://en.wikipedia.org/wiki/Quantum\\_computer](https://en.wikipedia.org/wiki/Quantum_computer)

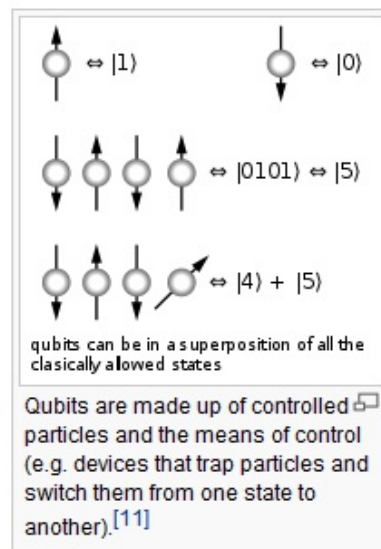
以下是其中一個關鍵段落的截圖：

## Bits vs. qubits [edit]

A quantum computer with a given number of qubits is fundamentally different from a classical computer composed of the same number of classical bits. For example, to represent the state of an  $n$ -qubit system on a classical computer would require the storage of  $2^n$  complex coefficients. Although this fact may seem to indicate that qubits can hold exponentially more information than their classical counterparts, care must be taken not to overlook the fact that the qubits are only in a probabilistic superposition of all of their states. This means that when the final state of the qubits is measured, they will only be found in one of the possible configurations they were in before measurement. Moreover, it is incorrect to think of the qubits as only being in one particular state before measurement since the fact that they were in a superposition of states before the measurement was made directly affects the possible outcomes of the computation.

For example: Consider first a classical computer that operates on a three-bit register. The state of the computer at any time is a probability distribution over the  $2^3 = 8$  different three-bit strings 000, 001, 010, 011, 100, 101, 110, 111. If it is a deterministic computer, then it is in exactly one of these states with probability 1. However, if it is a **probabilistic computer**, then there is a possibility of it being in any *one* of a number of different states. We can describe this probabilistic state by eight nonnegative numbers  $A, B, C, D, E, F, G, H$  (where  $A$  = probability computer is in state 000,  $B$  = probability computer is in state 001, etc.). There is a restriction that these probabilities sum to 1.

The state of a three-qubit quantum computer is similarly described by an eight-dimensional vector  $(a, b, c, d, e, f, g, h)$ , called a **ket**. However, instead of adding to one, the sum of the *squares* of the coefficient magnitudes,  $|a|^2 + |b|^2 + \dots + |h|^2$ , must equal one. Moreover, the coefficients can have **complex values**. Since the absolute square of these complex-valued coefficients denote probability amplitudes of given states, the phase between any two coefficients (states) represents a meaningful parameter, which presents a fundamental difference between quantum computing and probabilistic classical computing.<sup>[12]</sup>



筆者的理解是，量子電腦神奇的地方在於：「量子位元」是種機率狀態的穩態，而這些「機率狀態」在「量子運算」的過程中可以進行機率性的疊加，這讓一些原本在傳統電腦架構下難以解決的問題，像是質數判定與分解等問題，有了被快速解決的機會。

不知這樣的理解是否正確，就得請教真正的量子電腦專家了，筆者以上講的這段話，可是不能算數的阿 ...

## 參考文獻

- 維基百科:[量子電腦](#)
- 維基百科:[態疊加原理](#)
- 維基百科:[量子相干性](#)
- [Wikipedia:Quantum computer](#) -- [https://en.wikipedia.org/wiki/Quantum\\_computer](https://en.wikipedia.org/wiki/Quantum_computer)
- 南加州大學：[Google量子計算機是真的](#) -- <http://www.hksilicon.com/kb/articles/180950/>

【本文由陳鍾誠修改自維基百科】

# 程式與科學

## 資訊科學到底算不算是科學呢？（作者：陳鍾誠）

### 前言

Computer Science 通常被翻成中文的「資訊科學」，不過更精準的翻譯應該是「電腦科學」或「計算機科學」。

但是、Computer Science 真的能算是一門「科學」嗎？

或許有些人會覺得納悶，這是甚麼怪問題阿！

既然是 Computer 「Science」，當然是科學啦！

但是、Computer Science 究竟有多科學呢？

另外、Computer Science 的研究有甚麼障礙等待這些「科學家」去克服呢？

這是本文想探討的問題。

### 哪些學問算是科學呢？

如果我說「物理」是一門科學，或者說「生物」是一門科學，那我想應該很少人會有意見，因為「物理、化學、生物」這些領域可以說是典型的科學研究領域，如果這些不能被稱為科學的話，那其他領域就完全無法被稱為「科學」了。

那麼、「心理學」、「社會學」、「經濟學」或「歷史學」，也算是科學嗎？

關於這點，我想就有很多人有意見了！

以上問題見仁見智，我們就不企圖在此進行爭論了。

接著、我們再來看看一個比較有趣的問題，那就是所有科學都需要用到的 -- 「數學」，可以算是一門科學嗎？

要談論這個問題，得讓我們先回到「工業革命」的時代！

眾所周知的是，西洋的科學文化通常會追溯到希臘三哲人的時代，然後經過了兩千年的漫長旅程，到了十五世紀文藝復興之後，開始又復甦起來，然後更連接到「威尼斯、荷蘭、西班牙、葡萄牙」的大航海時代，接著英法等國逐漸掌握了海權，並且在英國興起了「工業革命」之後，科學的重要性才逐漸的凸顯了出來。

所以科學和工業革命事實上是歷史上難以分開的兩個兄弟，但是、工業與科學到底有甚麼關係呢？

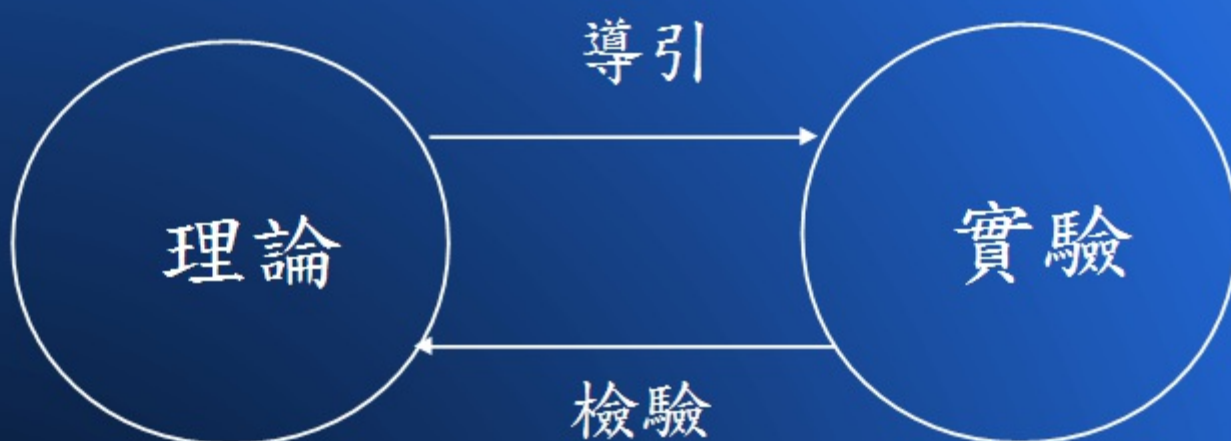
在我大學的時代，一直對這個問題很好奇，直到有一天，我看了金觀濤「[創造與反思](#)」一書中的幾篇文章之後，概念逐漸清晰了起來，這些文章列表如下：

- 科學技術的整體觀
- 近代科學技術結構的成長
- 中國近代科學落後的原因

以下是我從這些文章中整理出來的幾個圖，讓我們用這些圖來說明「科學、實驗與工業」之間的關係。

首先讓我們聚焦在「科學與實驗之間的關係」這張圖上，我們可以看到實驗對科學的重要性，實驗可以用來檢驗科學理論是否有誤，而科學理論則對實驗該如何進行提供了指導方向。

# 理論與實驗相互促進論



來源：金觀濤，西方社會結構的演變

圖、科學與實驗之間的關係

這種想法在 Popper (常譯為波柏或波普爾) 進化認識論當中表現得特別明顯，以下是從「[波柏的進化認識論](#)」這篇文章中摘錄出來的一段話：

針對某一特定現象作出精確預言，並且承認：符合預言的事實不能證實自己的理論，但不符合預言的事實卻能否證這一理論，這才是真正的科學，否則即是前科學或是偽科學。

換句話說，實驗可以用來否證一個理論，但是卻不能「證明」某個理論 (只能說該理論沒有被推翻)。

於是 Popper 發展出了他著名的「進化認識論」，論述那些「可以被外在事實或實驗檢驗」的問題，才算是科學問題，而那些無法被「實驗檢驗」的問題，就不屬於科學性的問題。

因此、像是宗教上面論述神是否存在、或者說「只有某些特定的人才能見證到神的存在」之類的問題，都屬於無法被「可重複的實驗結果」所檢驗的，因此無法被稱為科學問題。

如果從這個觀點來看，「物理、化學、生物」等領域，都依賴實驗來檢驗理論，因此都屬於典型的科學領域，但是「心理、歷史、經濟與社會」等領域，由於都與人有密切的關聯，而且很難進行「可重複的實驗」，因此就不屬於典型科學領域的範疇。

而上面所說的數學呢？由於數學並不具有「可用外在世界實驗檢驗理論」的特性，因此在 Popper 的這種想法中，並不能算是科學性的領域。

雖然數學並不算是科學的領域，但這並不代表數學是不重要的，相反的，數學在科學上的價值是有目共睹的，因為大部分的理論，只有在能夠表達成某種數學之後，才能夠被檢驗。舉例而言，牛頓第二運動定律  $F=M*A$  這條數學式，一旦被寫出來之後，物理學家們就可以去做實驗，想辦法否證這個定律，而經過千百次的檢驗之後，力學運動大致都符合這個定律，沒有實驗能明顯的否證此一定律時，我們才能說這是一個「定律」，否則就只能稱為「假說」而已。

透過「實驗」來驗證理論，正是「科學」與「非科學」領域之間的最大差異。

但是、科學或不科學到底有甚麼關係呢？難道科學的興起與工業革命之間有關連嗎？且讓我們再來看看以下圖形。



圖、科學與工業之間的關係

在上圖中，除了原本「理論與實驗」間的良好循環之外，又加上了「科學與工業」間的循環，這個循環解釋了為何「工業革命與科學革命」同時發生，而且兩者個關係如此緊密的原因。

「金觀濤」在上述文章中用了很清楚的邏輯，說明了「理論與實驗」、「科學與工業」間的增強循環，是如何在 15 世紀之後發生，然後不斷增強與進步的，非常建議有興趣的讀者可以閱讀金觀濤的一系列作品。

## Computer Science 究竟有多科學呢？

再度回到我們的問題上，究竟 Computer Science 到底算不算是一門科學呢？首先讓我們看看 Computer Science 到底在研究些甚麼？

根據筆者的研究經驗，我大致將 Computer Science 的研究分為三種類型，第一類著重於「執行速度」、第二類著重於「使用空間」，第三類則著重於「正確率」的研究，第四類則是著重於「優化某種數字」的研究。

像是「演算法、計算機結構、網路通訊」等領域的研究，通常是為了讓「軟體、硬體、網路」更有效率，速度更快而進行的研究，這類的研究是以「執行速度」為衡量標準的研究。

而像「資料結構、影像壓縮、檔案結構」等領域的研究，則是為了用更少的空間，達成相同或更好的效能，這類的研究是屬於第二類的「使用空間」為衡量標準的研究。

然後、像是「影像辨識、語音辨識、手寫辨識、機器翻譯、自然語言」等領域，則是著眼於提升「翻譯或辨識」的正確率。

最後、有些研究是在尋找某種更好的解答，像是「某個量化數字更好」等等，這類的研究通常稱為「最佳化」或「優化」類的研究。

對於一、二類的研究而言，我們可以採用某種衡量方法，或者實際的去執行程式，以便檢驗究竟哪種方法較好。而對於第四類的研究而言，那些數字是某個固定的函數，所以也可以很容易的計算出來，以檢驗方法的好壞。

但是對於第三類的研究而言，正確率往往會「與人有關」，這時候最後的檢驗標準必須用人來判斷，這類的研究以「人工智慧」領域最多，其中有些問題是人類通常有共同答案的，像是「影像辨識、語音辨識、手寫辨識」等，這些就比較容易有一致的檢證標準。

在第三類的研究當中，有些問題連人類也常有不同答案的，像是「機器翻譯」的問題，同一句英文會被翻譯成什麼樣的中文，是每個人都有不同想法的，甚至對於同一個翻譯而言，有些人覺得這樣翻很好，也有人可能會覺得這樣翻很糟，沒有固定的標準，這種研究在客觀上就有衡量的困難。

對於「自然語言」的研究更是如此，舉例而言，在 1950 年時資訊科學領域的開山始祖「艾倫·圖靈」(Alan Turing) 就曾經提出一個稱為「圖靈測試」(Turing Test) 的測驗，用來檢驗「一台電腦是否具有智慧」這件事情，其方法很容易理解，現代的讀者可以想像當有個陌生人連進來與你做 MSN 或 facebook 訊息的交談時，您必須判斷對方到底是一個真人，或者只是一支「聊天程式」，假如有個程式厲害到讓人判斷不出來，那麼這個程式就通過了「圖靈測試」。

在 Alan Turing 想出「圖靈測試」這個檢驗方法之後，很多人就開始想如何才能做出這樣一台有智慧的電腦，一支有智慧的程式呢？這個問題在 1964 被 MIT 的 Joseph Weizenbaum 用一個非常投機取巧的方式，設計出了一支稱為 Eliza 的聊天程式，不算完整的解決了「圖靈測試」這個問題。

事實上、Eliza 可以說是一支很擅長呼攏欺騙的程式，如果翻譯成中文來看，他其實幾乎都用那些寒暄招呼語，以及模擬兩可的用語在與人對談，例如「你好嗎？」、「請繼續」...，另外、如果你談到「父母兄弟姊妹...」，他就說「可以多和我聊聊有關你家人的事情嗎？」之類的泛用語，Eliza 甚至會利用樣式比對取出你的一些話後來回答你，例如你如果說：「我心情不好！」，Eliza 就會回答說：「為何你心情不好？」，其中的(心情不好)這幾個字是從你的問句中用「我 \*」的方式取出來的。

以下是筆者與自己設計的「中文 Java 版 Eliza」聊天的一個過程。

您好，我的朋友！有心事嗎？

>> 我不開心

為何不開心？

>> 和同學吵架了

這樣喔！知道！

>> 心情很惡劣

可否多告訴一些呢？

>> 我最好的朋友罵我

你認識他多久了呢？

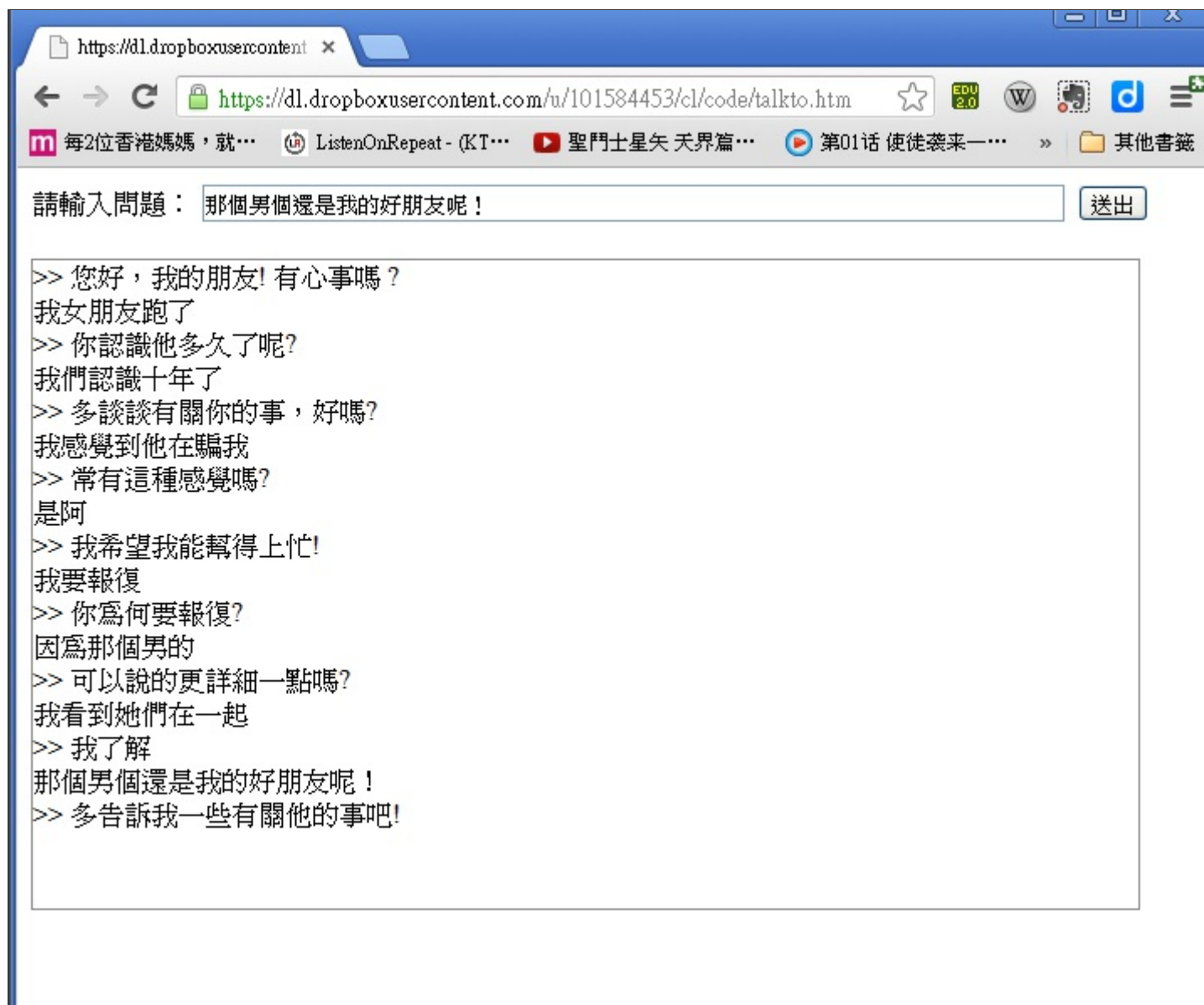
>> 很久了，超過十年了

...

如果您也想試試看與 [Eliza](#) 聊天的感覺，可以連線到筆者的下列網頁，試著與「中文網頁版的 [Eliza](#)」聊聊天：

- <https://dl.dropboxusercontent.com/u/101584453/cl/code/talkto.htm>

以下是筆者與自己寫的網頁版程式聊天的一個畫面（不過我女朋友沒有跑掉就是了 ....）。



圖、筆者與網頁版 [Eliza](#) 聊天的一個畫面



由於這類的問題牽涉到人類的判斷，而且研究者可以利用某些人類特性去發展出一些取巧的方法，因此往往是最難評量的一類問題。

對於第三類問題而言，假如問題的判斷最後取決於人的，那麼其檢證標準就不太容易客觀的存在，因為與人們的判斷有關，有時候可能會見仁見智，這種類型的研究，反倒有點像是「心理學」的研究了。

## 後記

經過了 30 年的程式訓練之後，筆者深深感覺「資訊科學」其實不太像「物理、化學、生物」這樣有一個外在的世界可以用來檢驗某個程式的。相反的，「資訊科學」反倒是比較像「數學」一樣，是從某種「公理系統」出發，這個最初公理系統就是電腦的「機器指令」，程式設計者透過「寫程式」的方式，告訴電腦一個「推演的方法」，然後讓那個「程式」去進行某種「自動推演」，以便找出問題的解答。

當然、由於程式的寫法無窮無盡，因此每個人寫出的程式也就大不相同，這些程式背後所根據的方法也各有差異，因此在同一個問題上，程式的表現也就有優劣之分，但是要到底哪個程式好，哪個程式不好，則不一定有絕對的標準，因為對於兩組不同的輸入 A, B 而言，可能「程式 1」在 A 上表現比「程式 2」好，但是「程式 1」在 B 上表現又比「程式 2」糟。因此最後就得有個「綜合指標」將這些程式的表現量化，但是這種「綜合指標」必然會導致某種的不客觀或不公平，所以在很多「資訊科學」的問題上，評量是 很難客觀的。

另外、對於那些與人有關的問題，其評量必須耗費大量的人力、時間與金錢，然而即使作完評量，這樣的評量卻又與人有關，因此很容易產生不客觀的爭議，因此這些領域也就不容易有快速的研究進展，像是「機器翻譯」與「自然語言」等都在評量上都存有很大的障礙。

甚至、有些障礙不只存在問題本身，而是存在「資訊科學」的研究文化上。在早期、電腦架構各不相同，執行環境難以統一的年代，資訊科學的研究最後都是化成數學符號，然後撰寫成論文的，這個文化一直持續到網路發達的今日，資訊科學的研究者往往在發表論文的時候並沒有附上「程式與測試資料」，這讓 想要「重複進行該研究實驗」的研究者難以檢驗論文與方法的好壞，這種文化顯然阻礙了「資訊科學」的進步速度。

我想，資訊科學領域的研究者有必要向「開放原始碼」領域的程式設計者學習，盡可能的將「程式與測試資料」公開，讓後續的研究者得以「精準的重複該實驗」並「檢驗論文所提出的方法」，然後從「程式原始碼當中學習該方法」，這樣才能讓「資訊科學」成爲一門「可重複的實驗結果」的領域，而這也正是 [波柏的進化認識論](#) 所認爲「科學應該有的必要條件」阿！

## 參考文獻

- 可重複的實驗結果是科學進步的基礎？文 / 陳紹慶（慈濟大學人類發展學系專任助理教授）
  - <http://pansci.tw/archives/43584>
- 金觀濤的理論 -- 工業革命爲何沒發生在中國？
  - <https://dl.dropboxusercontent.com/u/101584453/slide/JinGuanTao2.odp>
- 維基百科：[亞里斯多德](#)
- [孔恩的科學革命](#) -- [http://www.nhu.edu.tw/~sts/class/class\\_01\\_2.htm](http://www.nhu.edu.tw/~sts/class/class_01_2.htm)
- [波柏的進化認識論](#) -- [http://www.nhu.edu.tw/~sts/class/class\\_01\\_1.htm](http://www.nhu.edu.tw/~sts/class/class_01_1.htm)
- [創造與反思](#) -- <http://www.books.com.tw/exep/prod/booksfile.php?item=0010024931>
- [Wikipedia:Turing Test](#) -- [http://en.wikipedia.org/wiki/Turing\\_test](http://en.wikipedia.org/wiki/Turing_test)
- 維基百科：[圖靈測試](#) -- <http://zh.wikipedia.org/wiki/%E5%9B%BE%E7%81%B5%E6%B5%8B%E8%AF%95>
- [Wikipedia:Eliza](#) -- <http://en.wikipedia.org/wiki/ELIZA>

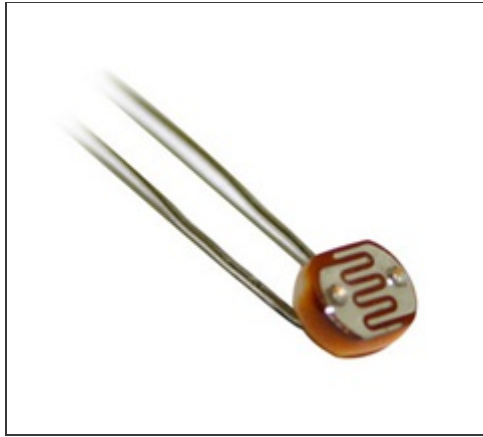
- 陳鍾誠的網站:自然語言處理：Eliza -- <http://ccckmit.wikidot.com/nlp:eliza>
- 自己動手設計交談機器人 (Eliza 中文版) - 使用 Java -- <http://ccckmit.wikidot.com/code:eliza>

# 程式人文集

## Arduino 入門教學(8) – 使用光敏電阻控制 LED 的開關 (作者：Cooper Maa)

### 實驗目的

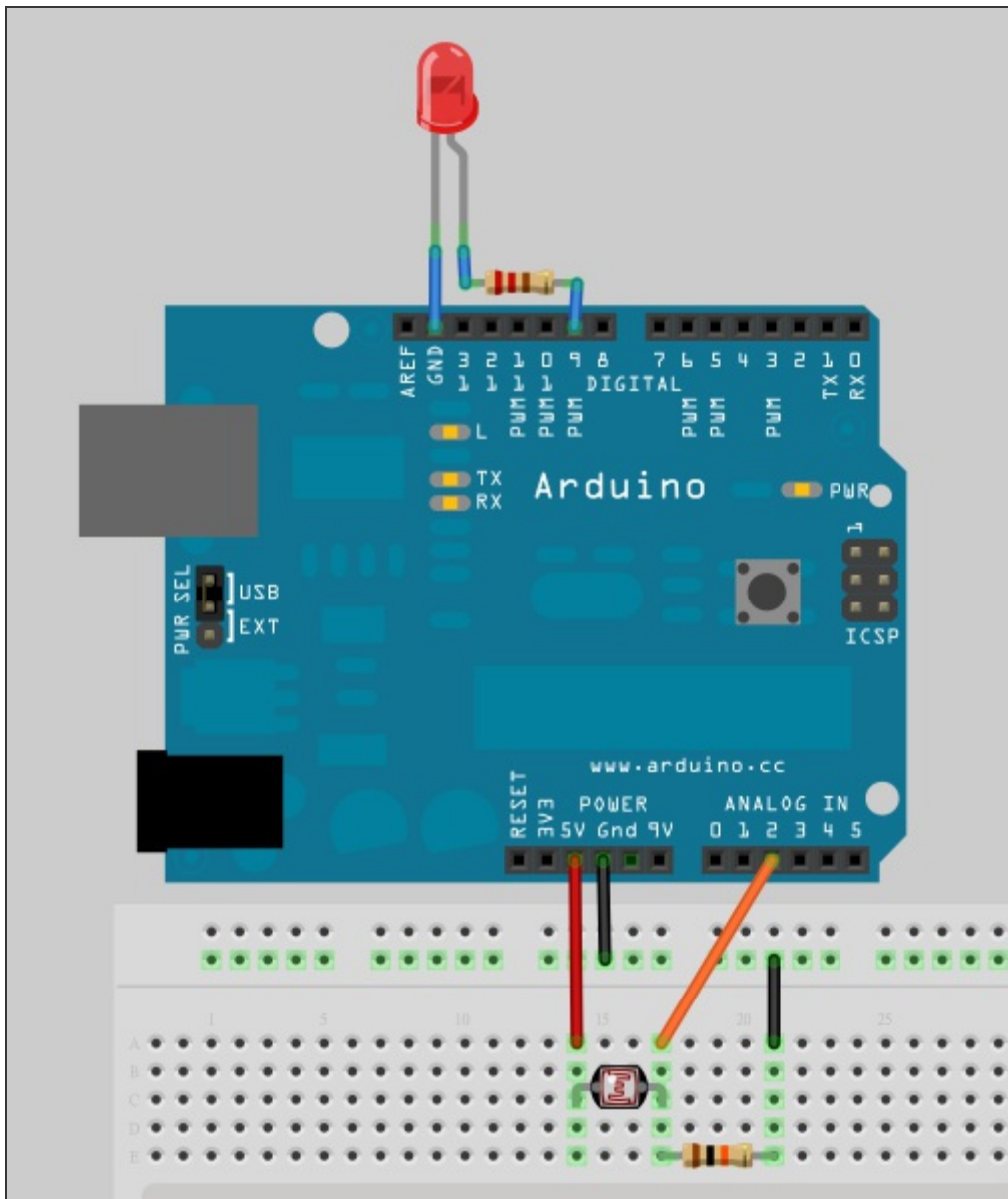
利用光敏電阻做一個 LED 的自動開關，在光線不足時，自動打開 LED 燈，反之，光線充足時便關掉 LED 燈。



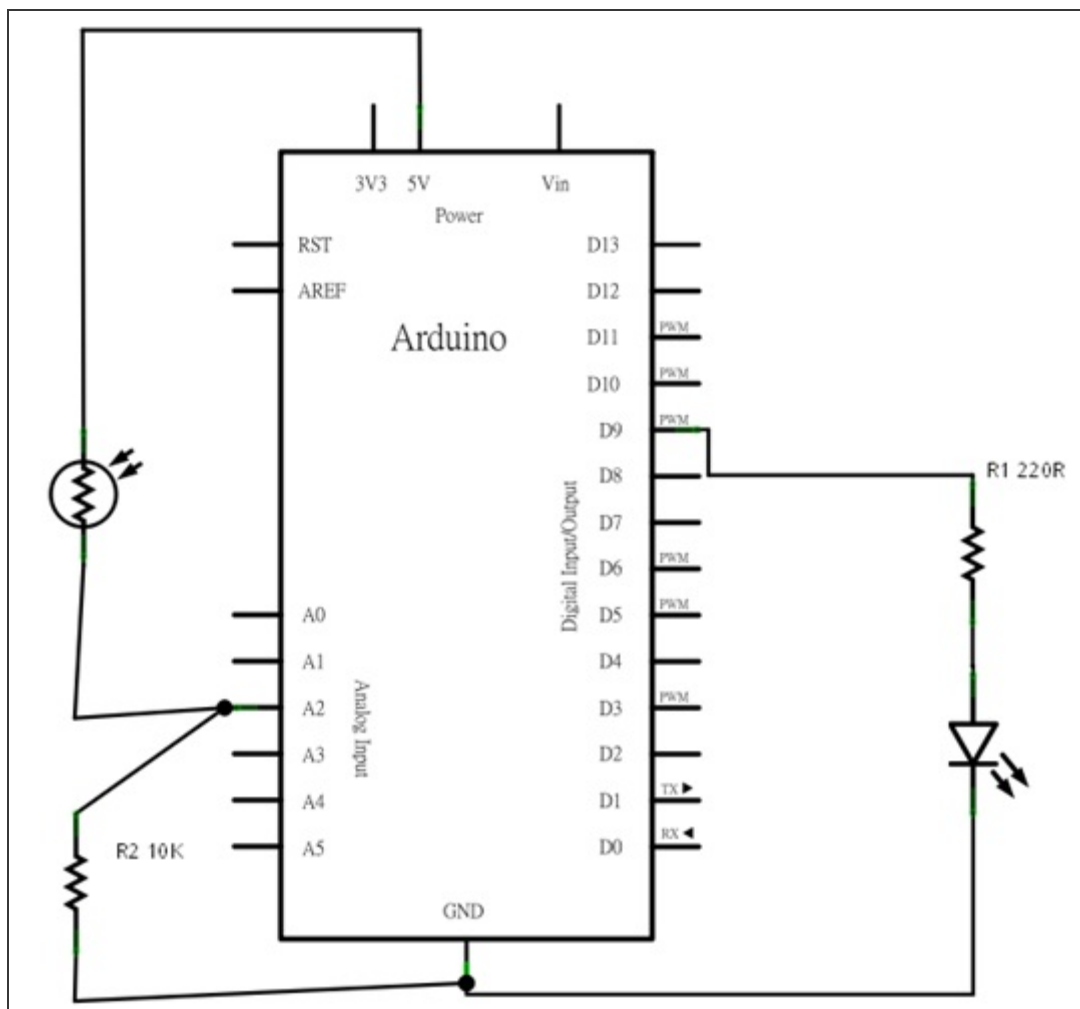
### 材料

- 麵包板 x 1
- Arduino 主板 x 1
- 光敏電阻(photocell) x 1
- 220 ohm 電阻 x 1
- 10k ohm 電阻 x 1
- 單心線 x N
- 接線

光敏電阻一支腳接到 5V，另一支腳串接一顆 10k 電阻接到 analog pin 2 LED 接到 pin9 和 GND，長腳(陽極)串接一顆 220 ohm 電阻到 pin9，短腳(陰極)直接接到 GND



電路圖



## 程式碼

實驗一(Photocell.pde)：讀取光敏電阻並輸出到 Serial Port，先確定光敏電阻可以運作

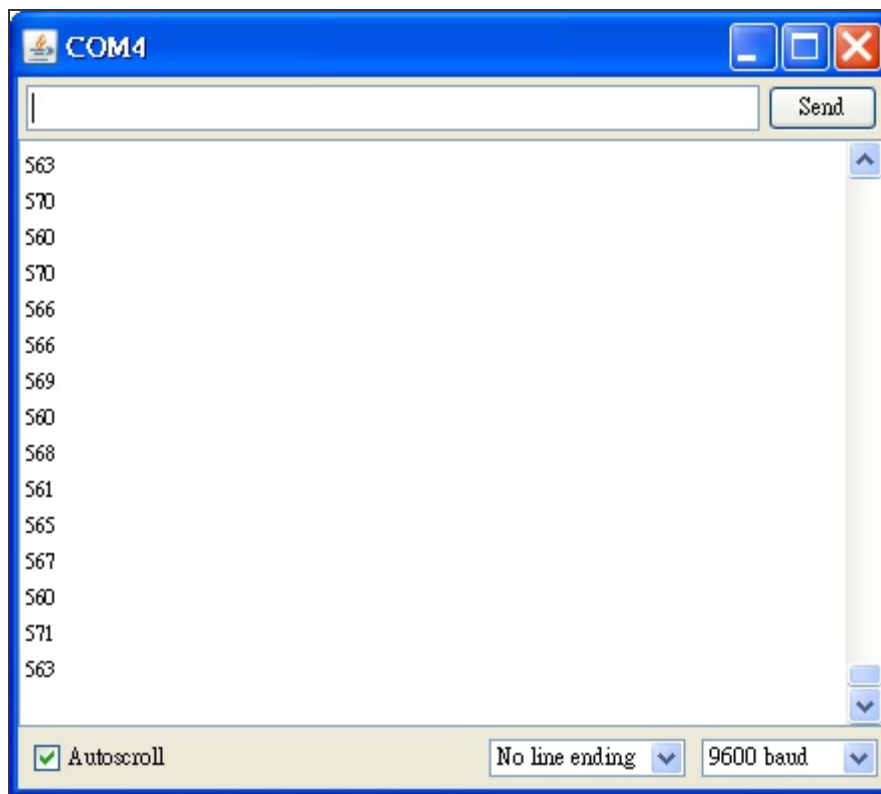
```
// LAB8 - 讀取光敏電阻 (v1)

int photocellPin = 2; // 光敏電阻 (photocell) 接在 analog pin 2
int photocellVal = 0; // photocell variable

void setup() {
  Serial.begin(9600);
}

void loop() {
  // 讀取光敏電阻並輸出到 Serial Port
  photocellVal = analogRead(photocellPin);
  Serial.println(photocellVal);
  delay(100);
}
```

假如一切順利的話，如下圖，打開 Serial Monitor 會看到 COM Port 不斷收到一串數據，那便是光敏電阻的讀值，試著用手去遮住光線，應該會看到電阻值跟著變小，代表光敏電阻的運作是正常的。



實驗二(Photocellv2.pde)：加上 LED 的開關控制，在光線不足時，自動打開 LED 燈，反之，光線充足時便關掉 LED 燈

```
// LAB8 - 讀取光敏電阻 (v2)
```

```
int photocellPin = 2; // 光敏電阻 (photocell) 接在 analog pin 2
```

```
int photocellVal = 0; // photocell variable
```

```
int minLight = 200; // 最小光線門檻值
```

```
int ledPin = 9;
```

```
int ledState = 0;
```

```
void setup() {  
  pinMode(ledPin, OUTPUT);  
  Serial.begin(9600);  
}
```

```
void loop() {  
  // 讀取光敏電阻並輸出到 Serial Port  
  photocellVal = analogRead(photocellPin);  
  Serial.println(photocellVal);  
  
  // 光線不足時打開 LED  
  if (photocellVal < minLight && ledState == 0) {  
    digitalWrite(ledPin, HIGH); // turn on LED  
    ledState = 1;  
  }  
}
```

```
// 光線充足時關掉 LED
if (photocellVal > minLight && ledState == 1) {
    digitalWrite(ledPin, LOW); // turn off LED
    ledState = 0;
}

delay(100);
}
```

## 範例照片／影片

- Arduino 筆記 -- Lab8 使用光敏電阻控制 LED 的開關
  - <http://youtu.be/3qLCi0Ughk0>

## 動動腦

現在的筆記型電腦，通常都會自動調整螢幕背光亮度，在光線比較充足時，跟著調高背光亮度，在環境比較暗時，就降低背光的亮度，這樣使用者在操作筆電時就不會有不適的感覺，在暗的地方螢幕就不會太刺眼，在亮的地方也不怕不看到螢幕上的東西。試解釋筆電自動調整背光的原理。

【本文作者為馬萬圳，原文網址為：<http://coopermaa2nd.blogspot.tw/2010/12/arduino-lab8-led.html>，由陳鍾誠編輯後納入本雜誌】

## JavaScript (8) – 中文版 Eliza 聊天程式(作者：陳鍾誠)

在這期程式人雜誌期的「程式與科學」主題中，我們已經介紹過下列主題。

- [資訊科學到底算不算是科學呢？](#)

其中我們提到了「圖靈測試」( [Turing Test](#) ) 這個評量「程式是否有智慧」的方法，並且談到 1964 年 Joseph Weizenbaum 創造了一個稱為 [Eliza](#) 的聊天程式，這個程式讓許多人誤以為是在和真人聊天，因此與它聊得很起勁，所以 [Eliza](#) 在某種程度上通過了「圖靈測試」。

在本文中，我們將模仿 Joseph Weizenbaum 在 [Eliza](#) 中採用的方法，製作出一個中文版的 [Eliza](#) 聊天程式。我們將這個程式寫成一個 HTML + JavaScript 的網頁，您只要擁有瀏覽器就可以執行了 (筆者測試這個程式使用的瀏覽器是 Google Chrome)。

## 使用畫面

以下是筆者自己與聊天程式對談的一個聊天過程的畫面，當您輸入完每一句後請按下「enter 鍵」或「送出鈕」，等待程式回答。



圖、與本聊天程式對談的畫面

如果想試用一下這個程式，可以連線到筆者的下列網頁，試著與「中文網頁版的 Eliza」聊聊天。

- <https://dl.dropboxusercontent.com/u/101584453/cl/code/talkto.htm>

## 原始程式碼：**talkto.htm**

這個程式的運作原裏很簡單，程式裏的 `qaList` 變數是一個 Q&A 陣列，當某個 Q 欄位中的詞彙 (或樣式) 出現在使用者輸入的文句中時，就會被觸發，然後程式會從 A 欄位隨機選出一個答案，來回答使用者。舉例而言，當您輸入：

我不開心

這句話時，程式就會比對 `qaList` 中的樣式，發現以下 QA 物件中的「不」出現在「我不開心」這個句子裏，於是就觸發了這個規則

```
{ Q:"不", A:"為何不?|所以你不?"},
```

接著程式會用比對到問句的那個項目，也就是「不」字，去比對「我不開心」，於是比對結果為「我(不)開心」，然後將句尾的「開心」取出，並設定 `tail = "開心"`。



接著程式會從 A 欄位的「為何不?」、「所以你不?」這兩個可能的回答中，隨機選出一個進行回答，如果選到的是「為何不\*?」，那就會將其中的 \* 取代成「開心」兩個字，因此作出「為何不開心?」這樣的回答。

但是，有時使用者輸入的句子當中有「你」或「我」的對話角色用語，在回答時就必須將兩者倒過來，例如：假如使用者輸入

```
我不喜歡你
```

這時候比對結果是「我(不)喜歡你」，如果直接取出句尾替換，應該回答

```
為何不喜歡你?
```

但是這樣的答案顯然很不恰當，因此若將「你」與「我」的角色對換，就會變成：

```
為何不喜歡我?
```

這樣的答案在角色上就比較對，所以程式中的下列段落，就是在處理這種情況。

```
tail = tail.replace("我", "#").replace("你", "我").replace("#", "你");
```

以下是整個「中文版 Eliza」，也就是 talkto.htm 這個程式的完整原始碼，請讀者參考：

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
</head>
<body onload="onRuleLoaded()">
  <div>
    請輸入問題:
    <input id="say" name="say" type="text" value="" size="80" onkeydown="keyin(event)"/> <!-- 按 enter
    <input type="submit" value="送出" onclick="say()"/> <!-- 當送出按鈕按下，就呼叫 say() 函數回答 -->
  </div>
  <BR/>
  <div id="dialogBox" style="width:95%; height:80%; overflow:auto; border:ridge 1px #888888; ">
  >> 您好，我的朋友！有心事嗎？<BR/>
  </div>
  <script type="text/javascript">
/* 以下為本程式回答問題時使用的 Q&A 規則，例如對於以下 Q&A 規則物件
{ Q:"想 | 希望", A:"為何想*呢?|真的想*?|那就去做阿?為何不呢?"},
```

代表的是，當您輸入的字串中有「想」或「希望」這樣的詞彙時，程式就會從 A: 欄位中的回答裏隨機選出一個來回答。

回答語句中的 \* 代表比對詞彙之後的字串，舉例而言、假如您說：

我想去巴黎

那麼我們的程式從這四個可能的規則中隨機挑出一個來產生答案，產生的答案可能是：

為何想去巴黎呢？

真的想去巴黎？

那就去做阿？

為何不呢？

Eliza 就是一個這麼簡單的程式而已。\*/

```
// Q&A 陣列宣告
```

```
var qaList = [
```

```
{ Q:"謝謝", A:"不客氣!"},
```

```
{ Q:"對不起 | 抱歉 | 不好意思", A:"別說抱歉 !|別客氣，儘管說 !"}，
```

```
{ Q:"可否 | 可不可以", A:"你確定想*?"},
```

```
{ Q:"我想", A:"你為何想*?"},
```

```
{ Q:"我要", A:"你為何要*?"},
```

```
{ Q:"你是", A:"你認為我是*?"},
```

```
{ Q:"認為 | 以為", A:"為何說*?"},
```

```
{ Q:"感覺", A:"常有這種感覺嗎?"},
```

```
{ Q:"為何不", A:"你希望我*!"},
```

```
{ Q:"是否", A:"為何想知道是否*?"},
```

```
{ Q:"不能", A:"為何不能*?|你試過了嗎?|或許你現在能*了呢?"},
```

```
{ Q:"我是", A:"你好，久仰久仰!"},
```

```
{ Q:"甚麼 | 什麼 | 何時 | 誰 | 哪裡 | 如何 | 為何 | 因何", A:"為何這樣問?|為何你對這問題有興趣?|你認為答
```

```
{ Q:"原因", A:"這是真正的原因嗎?|還有其他原因嗎?"},
```

```
{ Q:"理由", A:"這說明了甚麼呢?|還有其他理由嗎?"},
```

```
{ Q:"你好 | 嗨 | 您好", A:"你好，有甚麼問題嗎?"},
```

```
{ Q:"或許", A:"你好像不太確定?"},
```

```
{ Q:"不曉得 | 不知道", A:"為何不知道?|在想想看，有沒有甚麼可能性?"},
```

```
{ Q:"不想 | 不希望", A:"有沒有甚麼辦法呢?|為何不想*呢?|那你希望怎樣呢?"},
```

```
{ Q:"想 | 希望", A:"為何想*呢?|真的想*?|那就去做阿?為何不呢?"},
```

```
{ Q:"不", A:"為何不*?|所以你不*?"},
```

```
{ Q:"請", A:"我該如何*呢?|你想要我*嗎?"},
```

```
{ Q:"你", A:"你真的是在說我嗎?|別說我了，談談你吧!|為何這麼關心我*?|不要再說我了，談談你吧!|你自己
```

```
{ Q:"總是 | 常常", A:"能不能具體說明呢?|何時?"},
```

```
{ Q:"像", A:"有多像?|哪裡像?"},
```

```
{ Q:"對", A:"你確定嗎?|我了解!"},
```

```
{ Q:"朋友", A:"多告訴我一些有關他的事吧!|你認識他多久了呢?"},
```

```
{ Q:"電腦", A:"你說的電腦是指我嗎?"},
```

```
{ Q:"難過", A:"別想它了|別難過|別想那麼多了|事情總是會解決的"},
```

```
{ Q:"高興", A:"不錯Y|太棒了|這樣很好Y"},
{ Q:"是阿|是的", A:"甚麼事呢?|我可以幫助你嗎?|我希望我能幫得上忙!"},
{ Q:"", A:"我了解|我能理解|還有問題嗎 ?|請繼續說下去|可以說的更詳細一點嗎?|這樣喔! 我知道!|然後呢? 承
];
```

```
function random(n) { // 從 0 到 n-1 中選一個亂數
return Math.floor(Math.random()*n);
}
```

```
function say() { // 當送出鍵按下時，會呼叫這個函數進行回答動作
  append(document.getElementById("say").value); // 先將使用者輸入的問句放到「對話區」顯示。
answer(); // 然後回答使用者的問題。
}
```

```
function keyin(event) { // 當按下 enter 鍵時，會呼叫此函數進行回答
  var keyCode = event.which; // 取出按下的鍵
if (keyCode == 13) say(); // 如果是換行字元 \n，就進行回答動作。
}
```

```
function append(line) { // 將 line 放到「對話區」顯示。
  var dialogBox = document.getElementById("dialogBox"); // 取出對話框
  dialogBox.innerHTML += line+"<BR/>\n"; // 加入 line 這行文字，並加入換行 <BR/>\n
  dialogBox.scrollTop = dialogBox.scrollHeight; // 捲動到最下方。
}
```

```
function answer() { // 回答問題
  setTimeout(function () { // 停頓 1 到 3 秒再回答問題 (因為若回答太快就不像人了，人打字需要時間)
    append(">> "+getAnswer());
  }, 1000+random(2000));
}
```

```
function getAnswer() {
  var say = document.getElementById("say").value; // 取得使用者輸入的問句。
  for (var i in qaList) { // 對於每一個 QA
    try {
      var qa = qaList[i];
      var qList = qa.Q.split("|"); // 取出 Q 部分，分割成一個一個的問題字串 q
      var aList = qa.A.split("|"); // 取出回答 A 部分，分割成一個一個的回答字串 q
      for (var qi in qList) { // 對於每個問題字串 q
        var q = qList[qi];
        if (q=="") // 如果是最後一個「空字串」的話，那就不用比對，直接任選一個回答。
          return aList[random(aList.length)]; // 那就從答案中任選一個回答
        var r = new RegExp("(.*")+q+"([^?;]*)", "gi"); // 建立正規表達式 (.*?) q ([^?;]*)
        if (say.match(r)) { // 比對成功的話
```

```
tail = RegExp. $2; // 就取出句尾
// 將問句句尾的「我」改成「你」，「你」改成「我」。
tail = tail. replace("我", "#"). replace("你", "我"). replace("#", "你");
return aList[random(aList.length)]. replace(/\*/, tail); // 然後將 * 改為句尾進行回答
}
}
} catch (err) {}
}
return "然後呢?"; // 如果發生任何錯誤，就回答「然後呢？」來混過去。
}
</script>
</body>
</html>
```

## 結語

雖然 **Eliza** 的運作原理非常簡單，但是卻成功的欺騙了不少人，讓人們可以與程式聊天，這是人工智慧史上一個重要的進展，雖然這個程式並不是真的很有「智慧」，或者說根本就是在「欺騙」人，但是卻能達成與人聊天這麼困難的任務，實在是一個非常令人驚訝的小程式。

如果讀者曾經用過 **Apple iPhone** 上的 **Siri**，那麼應該可以體會到這種程式的用途。當然、**Siri** 比 **Eliza** 還要聰明一些，但事實上也沒有聰明太多，只要我們加入一些特殊的規則，**Eliza** 也可以輕易的被擴充成類似 **Siri** 的功能，例如看到「上市公司名稱」加上「價錢描述」，我們就猜測使用者是想查該公司的股票價格，於是就顯示「該公司的股價走勢」。

這種方法其實在某種程度上抓到了使用者的意圖，因此在自然語言處理領域，透過規則比對其實是最容易撰寫與使用的一種方法，很多所謂的智慧型交談系統，其實都是這樣做出來的喔！

## 參考

- **Wikipedia:Turing Test** -- [http://en.wikipedia.org/wiki/Turing\\_test](http://en.wikipedia.org/wiki/Turing_test)
- 維基百科：圖靈測試 -- <http://zh.wikipedia.org/wiki/%E5%9B%BE%E7%81%B5%E6%B5%8B%E8%AF%95>
- **Wikipedia:Eliza** -- <http://en.wikipedia.org/wiki/ELIZA>
- 陳鍾誠的網站:自然語言處理：Eliza -- <http://ccckmit.wikidot.com/nlp:eliza>
- 自己動手設計交談機器人 (**Eliza** 中文版) - 使用 **Java** -- <http://ccckmit.wikidot.com/code:eliza>



# Prolog 入門 (作者：KuoE0)

這學期第二個教的程式語言是 Prolog，這個語言更奇妙了！之所以稱作 Prolog 的理由是，他是「Programming in logic」的縮寫。被廣泛使用於人工智慧的領域，更常被作為設計專家系統的語言。

老實說，Prolog 個人覺得真的不好學習，對於習慣 imperative programming 的來說，思維模式比 functional programming 更難令人接受！Prolog 是由我們預先設計好一些規則作為其知識庫，接下來我們就可以給予問題詢問 Prolog，Prolog 就會搜尋其知識庫並回答出符合其知識庫中的正確答案。Prolog 也是有許多不同的實現，這邊我選擇了 SWI Prolog。

## 詞法結構 Syntax

先看看 Prolog 的基本語法，在 Prolog 中有三種主要語法結構，分別是：fact、clause 以及 query，而這三種語法結構都是由 predicate 為基礎，中文稱之為「謂詞」。predicate 有點像是其他語言的函式一般，由一個名稱與一些參數組成，且其回傳值僅為 boolean 值。見下表的詞法結構：

TYPE	BNF
predicate	<code>&lt;predicate&gt; ::= &lt;P&gt;(&lt;ARGS&gt;)</code>
fact	<code>&lt;fact&gt; ::= &lt;predicate&gt;.</code>
clause	<code>&lt;clause&gt; ::= &lt;predicate&gt; :- &lt;predicate&gt; {(, ;) &lt;predicate&gt; }.</code>
query	<code>&lt;query&gt; ::= ?- &lt;predicate&gt; {(, ;) &lt;predicate&gt; }.</code>

先看到 fact，其語法是 predicate 後，加上一個句點(.)，表示該 predicate 為真！而 clause 其語法可以是多個 predicate，表示當所有 clause 所包含的 predicate 都為真時，該 clause 為真！另外，在 clause 中，逗號(,)表示「且 (and)」，分號 (;) 表示「或 (or)」，所以，clause 就像是一個條件式。用分號分隔的 clause 也可能分開成多條 clause 來寫。例如：

```
clause :-  
    goal1 ;  
    goal2.
```

可以改寫成，

```
clause :- goal1.  
clause :- goal2.
```

從上面的例子中也可以發現，Prolog 的語法都可以跨行，並不一定要寫在單一行！在 clause 中，還可以利用括號來做群組。另外，還有一個符號 -> 用來作條件判斷，當置於 -> 之前的敘述結果為 true 時，才執行 -> 之後的敘述，一樣直接看例子：

```
clause :-
```

```
( cond1 ->
  goal1
;  cond2 ->
  goal2
;  goal3
).
```

利用括號與 `->`，可以很容易的實現 **if-else** 的條件敘述。

最後，是 **query**，長個很像 **fact**，唯一的差別是前面加上了 `?-` 的符號，表示一個查詢，Prolog 會幫我們找出令該 **predicate** 為真的結果。

另外，其實 **fact** 也可以看作是一個 **clause**，以下是其 BNF 語法：

```
<fact> ::= <predicate> :- true.
```

## 範例

我們先假設幾條 **predicate**：

```
isMom(A, B)      % A 是 B 的媽媽
isDad(A, B)      % A 是 B 的爸爸
isParent(A, B)   % A 是 B 的父母
isGrandParent(A, B) % A 是 B 的祖父或祖母
```

有了這幾個 **predicate** 後，我們可以建立這些 **predicate** 的關係，也就是 **clause**：

```
isParent(A, B) :- isMom(A, B).  % A 是 B 的媽媽的話，表示 A 是 B 的父母
isParent(A, B) :- isDad(A, B).  % A 是 B 的爸爸的話，表示 A 是 B 的父母
isGrandParent(A, C) :- isParent(A, B), isParent(B, C).  % A 是 B 的父母且 B 是 C 的父母的話，
```

再來我們需要針對這些 **predicate** 建立一些 **fact**：

```
isMom(毛福梅, 蔣經國).
isMom(蔣方良, 蔣孝文).
isMom(蔣方良, 蔣孝武).
isDad(蔣介石, 蔣經國).
isDad(蔣經國, 蔣孝文).
isDad(蔣經國, 蔣孝武).
```

那個該怎麼在 SWI Prolog 中宣告這些規則呢？首先先開啓 SWI Prolog 的 **interactive shell**，在終端機中輸入 `swipl` 開啓。執行後會輸出以下訊息：

```
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.2.6)
Copyright (c) 1990-2012 University of Amsterdam, VU Amsterdam
```

```
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,  
and you are welcome to redistribute it under certain conditions.  
Please visit http://www.swi-prolog.org for details.  
For help, use ?- help(Topic). or ?- apropos(Word).  
?-
```

會發現 SWI Prolog 的 prompt 是 ?-，還記得前面提過的 Prolog 的詞法結構嗎？可以發現這就是 query 的開頭，所以在 SWI Prolog 的 interactive shell 中，我們能做的就是進行 query ！

只能 query 的話，該怎麼使用上述的規則呢？我們建立一個檔案名為 `chiangfamily.pl`（注意檔名不可以使用大寫）內容如下：

```
isParent(A, B) :- isMom(A, B). % A 是 B 的媽媽的話，表示 A 是 B 的父母  
isParent(A, B) :- isDad(A, B). % A 是 B 的爸爸的話，表示 A 是 B 的父母  
isGrandParent(A, C) :- isParent(A, B), isParent(B, C). % A 是 B 的父母且 B 是 C 的父母的話，  
isMom(毛福梅, 蔣經國).  
isMom(蔣方良, 蔣孝文).  
isMom(蔣方良, 蔣孝武).  
isDad(蔣介石, 蔣經國).  
isDad(蔣經國, 蔣孝文).  
isDad(蔣經國, 蔣孝武).
```

並在這個檔案存在的資料夾中開啓 SWI Prolog，接著輸入 `consult(chiangfamily).` 的 query，這時候就可以將這個檔案中的規則都讀取進來到 Prolog 的知識庫中。

讀取後我們就可以開始查詢蔣家的親屬關係了！例如我們想知道蔣介石是不是蔣經國的爸爸？只要輸入 `isDad(蔣介石, 蔣經國).`，這時候 Prolog 就會回傳一個 `true` 回來，按下 `enter` 表示確定。不過這個例子只是查詢我們預先定義的事實罷了，好像沒什麼！那我們再來試試，蔣介石是不是蔣孝文的祖父呢？輸入 `isGrandParent(蔣介石, 蔣孝文).`，這時候 Prolog 一樣回傳了 `true`，這次查詢就不是我們給定的事實了，而是 Prolog 自己根據 `fact` 與 `clause` 所推論出來的！因為我們存在 `isDad(蔣介石, 蔣經國).` 與 `isDad(蔣經國, 蔣孝文).` 這兩個事實，再根據推論方法 `isParent(A, B) :- isDad(A, B).` 與 `isGrandParent(A, C) :- isParent(A, B), isParent(B, C).` 即可得知，這是利用 Prolog 的\*比對能力！

再來，我們在看個 Prolog 更厲害的地方—列舉。如果，我們想知道蔣介石的孫子有誰呢？我們只要輸入 `isGrandParent(蔣介石, X).`，Prolog 會回傳 `X = 蔣孝文.`，這時候先不要按下 `enter` 確定，可以按下 `space` 取得更多答案，就會再回傳一個 `X = 蔣孝武.` 的結果，前面的 `X = 蔣孝文` 後面也會由句號變成分號，這表示蔣孝文跟蔣孝武都是蔣介石的孫子！一樣的我們可以查詢誰是蔣孝文的祖父母？輸入 `isGrandParent(X, 蔣孝文).`，這時候 Prolog 一樣先回傳了 `X = 毛福梅.`，如果按下 `space` 可以得到更多答案，也就是 `X = 蔣介石.`。如果我們再進一步的按下 `space` 想知道更多的答案的話，會得到一個 `false`，這表示已經沒有符合的值了。

欲離開 SWI Prolog 的話，只要輸入 `halt.` 即可。以下是完整的執行流程：

```
$ swipl  
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.2.6)  
Copyright (c) 1990-2012 University of Amsterdam, VU Amsterdam  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
```



```
and you are welcome to redistribute it under certain conditions.
```

```
Please visit http://www.swi-prolog.org for details.
```

```
For help, use ?- help(Topic). or ?- apropos(Word).
```

```
?- consult(chiangfamily).
```

```
% chiangfamily compiled 0.00 sec, 10 clauses
```

```
true.
```

```
?- isDad(蔣介石, 蔣經國).
```

```
true.
```

```
?- isGrandParent(蔣介石, 蔣孝文).
```

```
true .
```

```
?- isGrandParent(蔣介石, X).
```

```
X = 蔣孝文 ;
```

```
X = 蔣孝武.
```

```
?- isGrandParent(X, 蔣孝文).
```

```
X = 毛福梅 ;
```

```
X = 蔣介石 ;
```

```
false.
```

```
?- halt
```

```
$
```

## 資料結構 **Data Type**

首先，我們稱任何資料單位都叫做項 (**term**)。term 可以是原子 (**atom**)、數字 (**number**)、變數 (**variable**) 或是複合項 (**compound term**)，其中 **atom** 與數字屬於常數 (**constant**)。我個人是不太喜歡把 **term** 跟 **atom** 還有 **compound term** 給翻成中文，總覺得怪怪的，接下來這幾個詞還是以英文為主。

### 原子 **atom**

小寫開頭的文字序列，可以包含空白等特殊符號，但若是包含空白或特殊符號就需要用單引號 (') 包圍。若是要使用大寫開頭的 **atom**，也必須要使用單引號 (') 包圍，以區別跟變數的不同。另外，空列表 [] 也算是一個 **atom**。

### 數字 **number**

任何整數或浮點數，在 **Prolog** 中並沒有限制整數的大小，也就是支援整數的大數運算。

### 變數 **variable**

任何字母或是數字組成且由大寫或是底線開頭的文字串列。變數可以經過 **unification** 的機制而實例化，實例化表示將該變數用其他 **term** 替代，其他 **term** 可以是 **atom**、數字、變數或是 **compound term**。另外，在 **Prolog** 中有個特別的變數為 **\_**，就單一一個底線，代表為匿名變數，表示我們對他的內容並不關心，它可以是任何值。

舉個匿名變數的例子：

定義一個 **fact** 為 **isMom**(\_, 蔣經國)，一樣把它放進 **chiangfamily.pl** 的檔案中，並開啓 **SWI Prolog** 載入。然後我們來查詢誰是蔣經國的媽媽？輸入 **isMom**(**X**, 蔣經國)。會得到以下兩個結果：

```
true ;  
X = 毛福梅.
```

第一個 **true** 表示，我們根本不關心 **X** 到底是多少，至少第二個參數是蔣經國，就可以回傳 **true** 了。但我們也有定義另一個 **fact** 是 **isMom(毛福梅, 蔣經國)**，所以也會有第二個答案出現 **X = 毛福梅**。既然我們不關心 **X** 的值是多少，那麼是不是隨便填都應該回傳 **true** 呢？那我們來試試 **isMom(宋美齡, 蔣經國)**，得到的結果也是 **true**，但我們並沒有把 **isMom(宋美齡, 蔣經國)** 寫入 **fact** 中。爲了尊重蔣前總統，所以我就不再拿其他例子試啦！

另外，變數是俱有狀態的，前面提過透過 **unification** 的機制，變數可以被實例化，而被實例化的變數它就已經被綁定到某個值了，所以會說該變數已被綁定了。

## 複合項 **compound term**

一個由 **functor** 與參數表所組成的結構，可以把它想像成是 **C/C++** 中的 **structure**，應該有點 **ADT** 的概念。**functor** 就是一個類似函式的名稱的東西，那後面接著參數表，其實就大概可以預見長什麼樣子了。直接舉幾個例子：**car('Ferrari', red, 'XY-1234')**、**PersonInfo('KuoE0', '1990-05-06', 'National Cheng Kung University')**。其實，**atom** 可以被視爲一個沒有參數列表的 **compound term**。通常我們會用 **f/n** 來表示一個 **compound term**，其中 **f** 爲 **functor**，**n** 爲其 **arity**（**arity** 爲參數數量）。

前面提過，**Prolog** 的語法基礎是 **predicate**，再來看一次 **predicate** 的 **BNF** 語法：

```
<predicate> ::= <P>(<ARGS>)
```

把 **<P>** 當作 **functor**，可以發現原來 **predicate** 本身就是一個 **compound term** 了，所以其實 **Prolog** 的所有基礎都建立在 **compound term** 這個資料結構！

以下是幾個用於 **compound term** 的操作：

**arg(N, Term, Arg)**：對 **compound term** 的第 **N** 個參數進行操作。

```
?- arg(1, car('Ferrari', red, 'XY-1234'), X).  
X = 'Ferrari'.  
?- arg(2, car('Ferrari', X, 'XY-1234'), blue).  
X = blue.
```

**functor(Term, Functor, NumberOfArgs)**：獲取 **compound term** 的名稱與參數數量，或是建構一個俱有特定名稱且擁有特定數目個自由變數的 **compound term**。

```
?- functor(car('Ferrari', X, 'XY-1234'), F, A).  
F = car,  
A = 3.  
?- functor(C, car, 3).  
C = car(_G891, _G892, _G893).
```

**=..**：建構或解構一個 **compound term**。

```
?- car('Ferrari', X, 'XY-1234') =.. X.  
X = [car, 'Ferrari', X, 'XY-1234'].  
?- X = [car, 'Ferrari', X, 'XY-1234'].  
X = car('Ferrari', X, 'XY-1234').
```

## list 列表

其實 **list** 在 **Prolog** 也是一個 **compound term**，只是 **Prolog** 賦予了她一個語法糖—利用中括號來使用。前面提過，一個空的 **list** 是一個 **atom**，寫作 `[]`，而一個包含 `a`、`b`、`c`、`d` 四個物件的 **list** 可以寫作 `[a, b, c, d]`。那為什麼說 **list** 是 **compound** 呢？其實中括號是一個 **functor** 為 `.`，且俱有兩個 **arty** 的 **compound term**，用 **compound term** 地表示方式為 `./2`。`.` 俱有裡個參數，其中第一個可以是任何物件，但是第二個必須是一個 **list**。一個包含一個物件的 **list**，我們寫作 `.(a, [])`。前面包含 `a~d` 的例子用 `.` 的話就要寫作 `.(a, .(b, .(c, .(d, []))))`。這兩種寫法都是可以混用的，所以包含 `a~d` 的 **list** 也可以寫作 `.(a, [b, c, d])`。利用這個 **functor**，我們很容易可以得到一個 **list** 的頂端元素與剩餘元素，就像 **Lisp** 中的 `car` 與 `cdr`，以下是其 **clause**：

```
car(X, L) :- L = .(X, _).  
cdr(X, L) :- L = .(_, X).
```

來看看 `car`，由於 `X` 必須要是 `L` 的開頭，所以他應該也要是 `.` 這個 **compound term** 的第一個參數。另外，我們不需要知道該 **list** 的剩餘元素是什麼，所以就直接利用匿名變數 `_` 來表示即可。`cdr` 的原理也是一樣的。

假如說，現在要實現一個功能，但該功能僅需要 **list** 中的第一個元素，那麼就可以利用 `.` 這個 **compound term**。直接看例子，現在我要把第一個元素給加上 `1`：

```
addOneToHead(. (H, T), X) :- Y is H + 1, X = .(Y, T).
```

執行 `addOneToHead([1, 2, 3], X)` 後，會得到 `X = [2, 2, 3]` 的結果。

但是，`.` 這個 **compound term** 其實還滿不易讀得我覺得，在 **Prolog** 中可以利用中括號 `[]` 與豎線 `|` 完全取代 `.` 的功能，用法就像是 `.`，傳入兩個參數，第一個是要放置在 **list** 前面的元素，第二個則是一個 **list**，然後用豎線隔開。前面包含 `a~d` 的例子就可以改寫為 `[a|[b|[c|[d|[]]]]]`，或是 `[a|[b, c, d]]`。利用這樣的方式改寫的 `car` 與 `cdr` 如下：

```
car(X, L) :- L = [X|_].  
cdr(X, L) :- L = [_|X].
```

前面的 `addOneToHead` 的例子也可以改寫為：

```
addOneToHead([H|T], X) :- Y is H + 1, X = [Y|T].
```

如此一來，程式碼的閱讀上也較為容易！

## string 字串

字串在 **Prolog** 中，字串是由雙引號 (`"`) 所包圍住的文字串列，例如：`"KuoE0"`。其實字串在 **Prolog** 終究相當於一

個整數的 `list`，這些整數就是這些字元所對應的 ASCII/UTF-8 code。以 "KuoE0" 這個字串來說，就相當於 [75, 117, 111, 69, 48] 這個 `list`。

另外有個用於 `list` 與字串的操作 `name/2`。其定義為 `name(Text, List)`，可以將 `Text` 轉換為 ASCII/UTF-8 code 的 `list` 後存給 `List`，反之亦然。

## 一致化 Unification

`Prolog` 如此強大的能力是基於其 `unification` 的機制，我們接下來就來看看什麼是 `unification`。

當兩個 `predicate` 的名稱與參數相同時，就可以促成 `unification` 的機制進行。而 `unification` 要成功的話，兩個 `predicate` 需要能夠化做一模一樣的形式。規則如下：

當 `term1` 與 `term2` 皆為常數時，可以 `unify` 的條件為當兩者為相同的 `atom` 或是數字。當 `term1` 是變數而 `term2` 為任意型別，則 `term1` 可以被 `term2` 取代進行 `unify`，反之亦然。當兩者皆為變數時，兩者都可以互相取代進行 `unify`，而兩者的值也將會相同。當 `term1` 跟 `term2` 都是 `compound term` 時，必須要符合下列條件才能進行 `unify`：俱有相同的 `functor` 與 `arity` 所有參數都可以 `unify` 相同變數只能 `unify` 到相同的值 以下是利用 `Prolog` 實作 `unify` 的程式碼，如果看得懂的話，應該更能體會 `unify` 發生的條件與過程：

```
unify(A, B) :-
    atomic(A), atomic(B), A = B.
unify(A, B) :-
    var(A), A = B.
unify(A, B) :-
    nonvar(A), var(B), A = B.
unify(A, B) :-
    compound(A), compound(B),
    A =.. [F|ArgsA], B =.. [F|ArgsB],
    unify_args(ArgsA, ArgsB).
unify_args([A|TA], [B|TB]) :-
    unify(A, B),
    unify_args(TA, TB).
unify_args([], []).
```

一樣用前面的例子來看，假設我們要問蔣經國的爸爸是誰？我們會輸入 `isDad(X, 蔣經國)`，前面一直忘記提到，在 `Prolog` 中，大寫開頭的 `token` 是變數的意思！看看我們給定的 `fact` 中，可以找到三個 `isDad` 的 `predicate`：

```
isDad(蔣介石, 蔣經國).
isDad(蔣經國, 蔣孝文).
isDad(蔣經國, 蔣孝武).
```

由於常數不可以被取代，所以 `query` 中的「蔣經國」不可以被替換掉，那麼這三個 `isDad` 的 `fact` 就又被過濾掉了一些，並只剩下一個 `isDad(蔣介石, 蔣經國)`。而變數是可以被替換掉的，所以如果我們將變數 `X` 替換成「蔣介石」的話就可以符合我們給定的 `fact` 了，因此有了 `true` 的情況發生，`Prolog` 就會回傳一個 `X = 蔣介石` 的結果給我們！

再來看另一個例子，我們想查詢所有的父子關係呢？輸入 `isDad(X, Y)`. 進行查詢，一樣可以發現有三個 `isDad` 的 `fact`，由於我們查詢中並沒有常數，所以這三個 `fact` 都可以帶入得到 `true` 的結果，因此該次查詢可以得到三種答案！

那麼如果找不到可以符合的結果的話，就會回傳 `false`！例如我們想知道蔣介石的爸爸是誰？輸入 `isDad(X, 蔣介石)`. 就會得到 `false` 的結果。

透過 `clause` 推論規則，加上 `unification` 機制，`Prolog` 強大的推論便油然而生。來看個簡單的例子，`isParent(A, B) :- isMom(A, B)`. 與 `isParent(A, B) :- isDad(A, B)`. 這兩條 `clause` 都是 `isParent` 的推論規則。所以當我們想知道蔣經國的父母有誰？就可以透過 `isParent(X, 蔣經國)`. 來做查詢，然後得到 `X = 毛福梅`. 與 `X = 蔣介石`. 這兩個結果。其推論的流程是先將 `clause` 替換成我們定義的推論規則，在這個例子中，`isParent(X, 蔣經國)`. 會變成 `isMom(X, 蔣經國)`. 或是 `isDad(X, 蔣經國)`.。再透過 `isMom(X, 蔣經國)`. 經由 `unification` 的機制將 `X` 帶入毛福梅後可以得到相對應的 `fact`，也就是得到 `true` 的結果，所以會回傳 `X = 毛福梅`. 的結果。然而，`isParent(X, 蔣經國)`. 也可以推論為 `isDad(X, 蔣經國)`.，將 `X` 帶入蔣介石後也可以找到相對應的 `fact`，所以也會回傳 `X = 蔣經國`.。

再看一個比較複雜一點點的範例，查詢所有祖孫關係。要查詢所有祖孫關係的話，需要輸入 `isGrandParent(X, Y)`.。 `isGrandParent(X, Y)`. 可以推論為 `isParent(X, Z), isParent(Z, Y)`. 也就是要找到 `isParent(X, Z)`. 的結果是 `true` 且 `isParent(Z, Y)`. 的結果是 `true` 的情況才會回傳 `true`。回傳結果如下：

```
X = 毛福梅,  
Y = 蔣孝文 ;  
X = 毛福梅,  
Y = 蔣孝武 ;  
X = 蔣介石,  
Y = 蔣孝文 ;  
X = 蔣介石,  
Y = 蔣孝武 ;  
false
```

最後的 `false` 表示已經找不到這樣的關係了，所以可以發現四組祖孫關係。要瞭解 `Prolog` 的詳細推論流程的話，可以在 `SWI Prolog` 中輸入 `trace`. 打開 `trace mode`。這樣一來，`Prolog` 將會將所以推論流程通通顯示在螢幕上！以下是 `isParent(蔣經國, X)`. 的推論流程：

```
?- trace.  
true.  
[trace] ?- isParent(蔣經國, X).  
Call: (6) isParent(蔣經國, _G1558) ? creep  
Call: (7) isMom(蔣經國, _G1558) ? creep  
Fail: (7) isMom(蔣經國, _G1558) ? creep  
Redo: (6) isParent(蔣經國, _G1558) ? creep  
Call: (7) isDad(蔣經國, _G1558) ? creep  
Exit: (7) isDad(蔣經國, 蔣孝文) ? creep  
Exit: (6) isParent(蔣經國, 蔣孝文) ? creep  
X = 蔣孝文 ;
```

```
Redo: (7) isDad(蔣經國, _G1558) ? creep
Exit: (7) isDad(蔣經國, 蔣孝武) ? creep
Exit: (6) isParent(蔣經國, 蔣孝武) ? creep
```

```
X = 蔣孝武.
[trace] ?-
occurs check
```

在 **unification** 中有個 **issue** 是 **occur check**，直接看個例子：

```
?- X = f(X)
X = f(X).
```

可以看到變數 **X** 被 **unify** 成一個 **compound term**，而且這個 **compound term** 還包含他自己。因此他成了一個可無限遞回的結構，這麼一來我們除了可以把 **X** unify 為 **f(X)** 外，還可以 unify 為 **f(f(X))**、**f(f(f(X)))**、**f(f(f(f(X))))** 等等無窮循環下去。

但在 **Prolog** 預設的 **unify** 方式就會像是上面的例子，如果不希望有這樣的 **unify** 情況發生，可以使用 **unify\_with\_occurs\_check** 來進行 **unification**。看以下例子：

```
?- unify_with_occurs_check(X, f(X)).
false.
```

因為有 **occurs check**，可以檢測到 **X** 會被 **unify** 為一個包含自己的 **compound term**，所以不會發生 **unify**。

### 編寫獨立的 Prolog 程式

就我個人來說，看了許多教學後，其實我還是不知道該怎麼寫出一個 **Prolog** 程式，應該說可以直接執行而不是進入 **interactive environment** 後在自己執行。看了 **Wikipedia** 後，發現其 **quick sort** 範例中有個 **:- initialization(q).** 的 **clause**，發現這個敘述前面竟然沒有東西，直接就接下 **:-** 的符號，我在想應該就是 **interactive environment** 開始的位置，而 **q** 就是自己編寫的 **predicate**，並從該處開始。其實有點像 **C/C++** 裡的 **main** 一樣，在 **Prolog** 的程式進入點就是 **initialization**。不過，其實並不是 **initialization** 是 **Prolog**，只要開頭是 **:-** 這樣的符號，**Prolog** 都會直接執行該敘述。但為了方便程式有個進入點，就把 **initialization** 當作進入點吧！

以下程式碼引用自 **Wikipedia**，另外我加上了 **halt** 來離開程式，並儲存為 **quicksort.pl**：

```
q:- L = [33, 18, 2, 77, 66, 18, 9, 25], last(P, _), (quicksort(L, P, _), write(P), nl), halt.
partition([], _, [], []).
partition([X|Xs], Pivot, Smalls, Bigs) :-
    ( X @< Pivot ->
      Smalls = [X|Rest],
      partition(Xs, Pivot, Rest, Bigs)
    ; Bigs = [X|Rest],
      partition(Xs, Pivot, Smalls, Rest)
    ).
quicksort([]) --> [].
quicksort([X|Xs]) -->
```

```
{ partition(Xs, X, Smaller, Bigger) },
quicksort(Smaller), [X], quicksort(Bigger).
:- initialization(q).
```

在終端機輸入下列指令即可直接執行該程式碼：

```
$ swipl -q -s quicksort.pl
[2, 9, 18, 18, 25, 33, 66, 77]
```

其中 `-s` 是指將接下來的檔案以 `script` 的方式執行，而 `-q` 則是叫 SWI Prolog 安靜一點，不要輸出一些有的沒有。

【本文原文網址為：<http://kuoe0.ch/2288/prolog-tutorial/>，由陳鍾誠編輯後納入本雜誌】

## x86 machine code 初探 (0) - base register address mode (作者：Descent Sung)

我從來沒想過我會學習 machine code，而且 x86 machine code 實在太複雜，花了很多心力才有點進展。本來直接拿起 intel 那令人害怕的手冊硬 K，果然不行，有看沒有懂，再看 ref 3，還是看不懂。而在閱讀了 ref 1, 2 後，我如獲至寶，看懂之後再翻閱 ref 3，終於有了進展。當能看懂時，心中的興奮之情，恐怕只有和我一樣在 machine code 裡頭掙扎的同好才能理解。

不過 x86 machine code 實在太複雜，我並沒能搞懂所有格式。

為什麼要研究 machine code 呢？因為要改寫執行檔的位址，做類似 dynamic loader/linker 的事情，會看這篇文章的朋友，難道你對於 dynamic loader/linker 的原理沒有興趣嗎？不會想個方法來實驗這件事情嗎？Binary Hacks--駭客秘傳技巧一百招 #72 就在談這個。

讓我們從 intel memory address 開始。intel 著名的 segmentation memory address，將記憶體位址分為 segment part, offset part，本文章重點擺在 offset part。而台灣翻譯的基底/索引定址法（中國的翻譯則是 ...）則為 base/index。

offset part 有三個欄位：

1. base register
2. index register multiplied 1, 2, 4, 8 (1, 2, 4, 8 被稱為 scale factor)
3. displacement

$base + index * scale + displacement = offset\ part$

ex:

at&t syntax add 0x12345678(%eax, %esi, 4), %esi intel syntax add esi, [eax+esi\*4+0x12345678]

本篇文章大量參考 Programming THE 80386 (instruction encoding - p59):



這是 1987 年的書 (我寫這篇文章的時間是 20130612), 在電腦界這麼快速更新的時代, 照理說應該是過期的書籍, 不過由於相容性的緣故, 裡頭提到的東西到現在都還是可以用的, 而且簡單不少 (也沒有 64 bit mode), intel 手冊已經變得又厚又硬難以閱讀, 而這本書講的比較好理解。有些術語和最新的 intel 手冊有些不同, 不過不影響理解。

之前在閱讀 [IBM 80X86 組合語言實務](#) 被其複雜的定址模式搞得我好亂, 現在一次把它搞懂。這些複雜的 address mode 是針對 offset part 而來, 大概就是 base register 和 index register mode 這兩種比較複雜。而 16 bit 和 32 bit 又有點不同, 64 bit 沒研究, 就不提了。

16 bit 和 32 bit 可用來當 base/index register 的暫存器有些不同。為什麼呢? 讀懂 machine code 就可以回答這問題。

x86 machine code 順序, 後面的數字是 byte 數目:

1. prefix 0-4
2. opcode 1-2
3. modrm 0-2
4. displacement 0-4
5. immediate 0-4

這是這本書的解釋, 和目前的 intel 手冊有些不同, 目前的 intel 手冊把 modrm 分為 modrm + sib, 不過沒關係, 原則都是一樣的。我建議先看過 [ref 1](#), [ref 2](#) 再看這篇, 如果真的沒時間至少也要看過 [ref 2](#), 因為他們提到的東西, 我不會特別說明, 請花點時間看, 若你真想理解這玩意, 應該有覺悟要佔去你不少休閒時間。

檔案: address\_mode.S

```
1 # practice x86 machine code
2 .code16
3 #.code32
4 .text
5 .global begin
```



```
6 begin:
7   add (%bx), %ax
```

看看 L7 的組語, 這使用了 **base register address mode**。

```
objdump -d -m i8086 address_mode.elf

1 descent@w-linux:x86_machine_code$ objdump -d -m i8086 address_mode.elf
2
3 address_mode.elf:      file format elf32-i386
4
5
6 Disassembly of section .text:
7
8 00000100 <_text>:
9 100:   03 07                add    (%bx), %ax
```

由於 intel cpu 有 16/32/64 bit mode, 所以我們得選擇要讓組譯器用那種 bit mode 來翻譯出 machine code, 這個例子是 16 bit (ref address\_mode.S L2, L3)。

03 07 是 machine code, 對應到 opcode modrm 這兩欄, 沒有 prefix, displacement, immediate 這些部份。

來看看 add 的 opcode:

<http://css.csail.mit.edu/6.858/2011/readings/B86/ADD.htm>

有好幾個, 看以下這個:

```
03 /r      ADD r16,r/m16      2/6      Add r/m word to word register
clock 那欄我看不懂, 有請大大解惑。
所以 03 是 opcode, 03 搞定, 那 07 是什麼?

07 是 modrm 這欄, 再細分解為 2:3:3 欄位,
0000 0111 -> 00 000 111
mod: 00
reg: 000
r/m: 111
```

把 mod, r/m 拿來查表。

mod = 00 時的對照表 (16 bit)

Effective Address	r/m
[BX+SI]	000
[BX+DI]	001

[BP+SI]	010
[BP+DI]	011
[SI]	100
[DI]	101
disp16	110
[BX]	111

得到 [BX], 這是 base register address mode

reg: 000 代表 ax register

register table

REG	Register
000	EAX/AX
001	ECX/CX
010	EDX/DX
011	EBX/BX
100	ESP/SP
101	EBP/BP
110	ESI/SI
111	EDI/DI

得到 `add ax, [bx]` (intel syntax), 恭喜, 終於看懂 machine code 了。但是要從 `add ax, [bx]` 得到 `03 07` 就比較難了。

這在 intel 術語稱為: one-byte address mode encoding。

這篇好像有點長了, 不過打鐵趁熱, 來看看 prefix 的例子。

檔案: address\_mode1.S

```
1 # practice x86 machine code
2 .code16
```

```
3 #.code32
4 .text
5 .global begin
6 begin:
7     add (%bx), %eax
```

和 `address_mode.S` 的差別僅在 `%eax`, `%ax` 改成 `%eax`。我一直被 16 bit 程式碼可以使用 32 bit register 所疑惑, 這樣的程式碼到底是 16 bit 還是 32 bit? 當然是 16 bit, 那你和我覺得疑惑可以使用 32 bit register 嗎? 讓我們透過 `prefix` 解除這疑惑。

```
objdump -d -m i8086 address_mode.elf

1 descent@w-linux:x86_machine_code$ objdump -d -m i8086 address_mode.elf
2
3 address_mode.elf:      file format elf32-i386
4
5
6 Disassembly of section .text:
7
8 00000100 <_text>:
9 100:    66 03 07                add    (%bx),%eax
```

`machine code` 則多了一個 `66`, 這就是 `prefix`, 用來切換 `operand size`, 這裡的例子是從 16 bit 切換成 32 bit, 本程式執行在 16 bit 下, 而 `%eax` 是 32 bit operand, 所以要加上 `prefix 0x66`, 就這麼簡單。

## 想想看:

什麼是 32 bit 執行環境? A: ...

若在 32 bit 執行環境執行這個 `machine code` 又是什麼意思?

A:

```
.code32
.text
.global begin
begin:
    add (%edi), %ax
```

很有意思吧, `x86` 就是這麼討厭又讓人喜愛。

好用的組譯器: <http://radare.org/y/?p=examples&f=rasm>, 玩 `machine code` 的朋友一定要試試。

## 參考文獻

1. `x86/x64` 指令編碼內幕 (适用于 AMD/Intel): <http://www.mouseos.com/x64/index.html>
2. 学习 OpCode: <http://www.luocong.com/learningopcode.htm>

### 3. Programming THE 80386

4. <http://ref.x86asm.net/coder32.html>

【本文原文網址為：<http://descent-incoming.blogspot.tw/2013/06/x86-machine-code-0-base-register.html>，由陳鍾誠編輯後納入本雜誌】

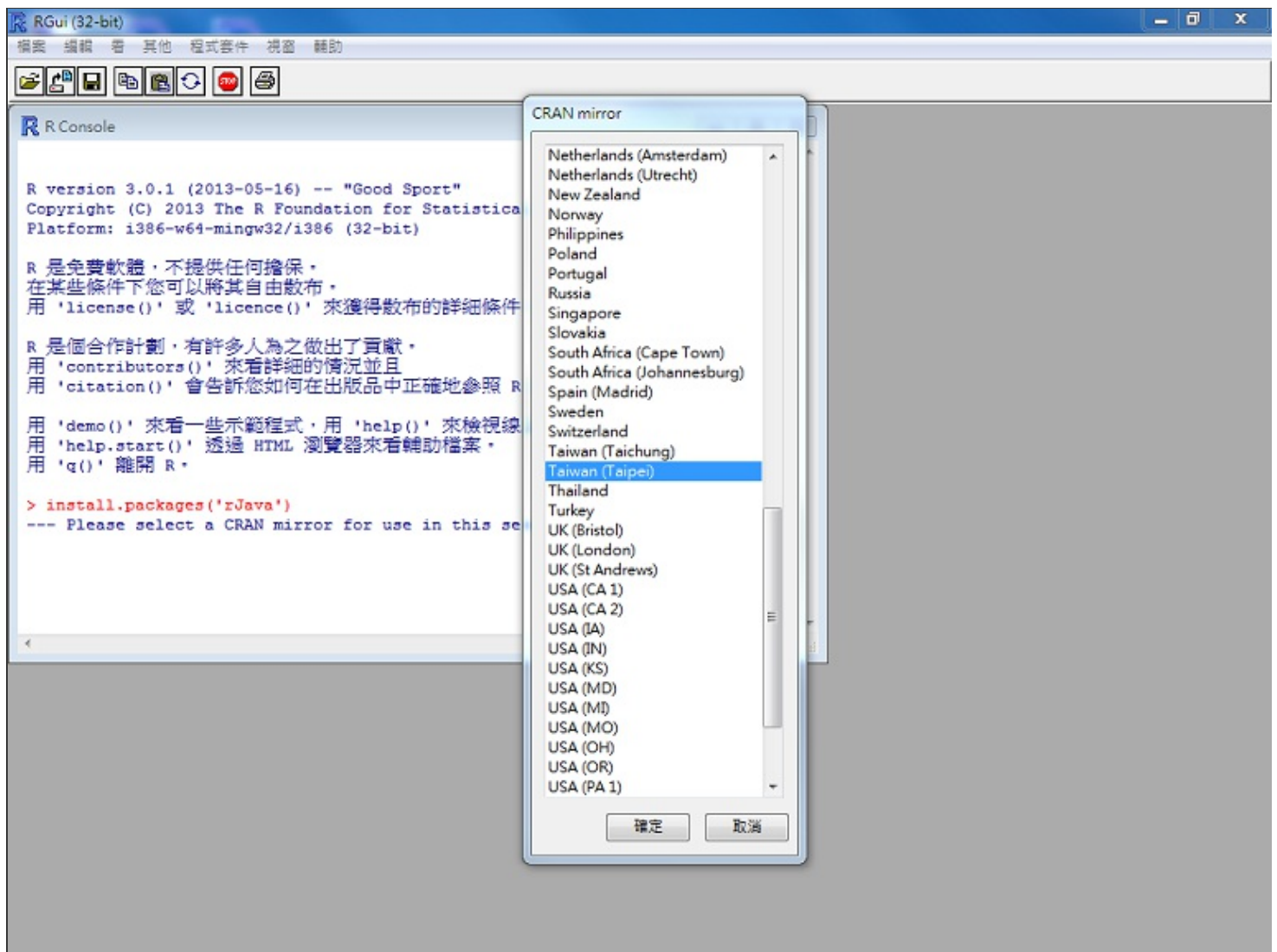
## 如何讓 R 與 Java 溝通 - rJava (作者：Taiwan R User Group)

R 是目前最熱門的 Open Source 統計語言。除了傳統的統計檢定之外，R也有套件支援許多 Machine Learning 和 Data Mining 的技術。因此使用者可以很方便的在R上實作各種分析方法。

rJava 在 [Top 100 R packages for 2013](#) 名單中排名第17名，是讓 R 呼叫 Java 撰寫的物件(Class)、實例(Instance)和方法(Method)的套件。這個套件降低R調用Java既有的資源的難度，例如Google APIs、Hadoop等等。

### 安裝步驟

輸入`install.packages('rJava')`，選取CRAN位置之後，便完成安裝。



install

## 初始化rJava

每次在使用rJava之前，都要先啟動JVM：

```
library(rJava)
.jinit()
```

## Hello World

我們先依照[rJava官網文件](#)的介紹，來Demo一個Hello World吧！首先，使用 `.jnew` 來產生一個 `java.lang.String` 的 instance "s"，

```
s <- .jnew("java/lang/String", "Hello World!")
s
```

```
## [1] "Java-Object {Hello World!}"
```

從輸入s之後就可看出，s屬於Java-Object，且其內容為Hello World!。

## 建立Java物件的Reference

透過 `J`，使用者可以取出Java物件的Reference，進而存入一個R的變數：

```
pi <- J("java.lang.Math")
pi
```

```
## [1] "Java-Class-Name: java.lang.Math"
```

## 建立Java實例

透過 `.jnew`，使用者可以建立某個Java物件的實例：

```
s <- .jnew("java/lang/String", "Hello World!")
s
```

```
## [1] "Java-Object {Hello World!}"
```

## 取得Java物件的屬性值

取得Java物件的屬性有兩種方法：

- `.jfield`
- `+$Field名`

```
.jfield("java.lang.Math", , "PI")
```

```
## [1] 3.142
```

```
pi$PI
```

```
## [1] 3.142
```

## 使用物件的方法

呼叫Java物件的方法可以用

- `.jcall` .jcall的第一個參數是java物件變數，第二個是Return Type(定義請見下圖，[出處](#))，第三是Java物件的方法名稱。
- `$.Method`名

```
.jcall(s, "I", "length")
```

```
## [1] 12
```

```
s$length()
```

```
## [1] 12
```

若是要查詢物件的Method，則可以利用：

- `.jmethods`
- `names(obj)`
- `obj$+TAB鍵`(自動完成)

```
.jmethods(pi)
```

```
## [1] "public static int java.lang.Math.abs(int)"
## [2] "public static double java.lang.Math.abs(double)"
## [3] "public static long java.lang.Math.abs(long)"
## [4] "public static float java.lang.Math.abs(float)"
## [5] "public static double java.lang.Math.sin(double)"
## [6] "public static double java.lang.Math.cos(double)"
## [7] "public static double java.lang.Math.tan(double)"
## [8] "public static double java.lang.Math.atan2(double, double)"
## [9] "public static double java.lang.Math.sqrt(double)"
## [10] "public static double java.lang.Math.log(double)"
## [11] "public static double java.lang.Math.log10(double)"
## [12] "public static double java.lang.Math.pow(double, double)"
## [13] "public static double java.lang.Math.exp(double)"
```

```
## [14] "public static long java.lang.Math.min(long, long)"
## [15] "public static double java.lang.Math.min(double, double)"
## [16] "public static int java.lang.Math.min(int, int)"
## [17] "public static float java.lang.Math.min(float, float)"
## [18] "public static float java.lang.Math.max(float, float)"
## [19] "public static double java.lang.Math.max(double, double)"
## [20] "public static long java.lang.Math.max(long, long)"
## [21] "public static int java.lang.Math.max(int, int)"
## [22] "public static double java.lang.Math.scalb(double, int)"
## [23] "public static float java.lang.Math.scalb(float, int)"
## [24] "public static int java.lang.Math.getExponent(double)"
## [25] "public static int java.lang.Math.getExponent(float)"
## [26] "public static float java.lang.Math.signum(float)"
## [27] "public static double java.lang.Math.signum(double)"
## [28] "public static double java.lang.Math.asin(double)"
## [29] "public static double java.lang.Math.acos(double)"
## [30] "public static double java.lang.Math.atan(double)"
## [31] "public static double java.lang.Math.toRadians(double)"
## [32] "public static double java.lang.Math.toDegrees(double)"
## [33] "public static double java.lang.Math.cbrt(double)"
## [34] "public static double java.lang.Math.IEEEremainder(double, double)"
## [35] "public static double java.lang.Math.ceil(double)"
## [36] "public static double java.lang.Math.floor(double)"
## [37] "public static double java.lang.Math rint(double)"
## [38] "public static long java.lang.Math.round(double)"
## [39] "public static int java.lang.Math.round(float)"
## [40] "public static double java.lang.Math.random()"
## [41] "public static float java.lang.Math.ulp(float)"
## [42] "public static double java.lang.Math.ulp(double)"
## [43] "public static double java.lang.Math.sinh(double)"
## [44] "public static double java.lang.Math.cosh(double)"
## [45] "public static double java.lang.Math.tanh(double)"
## [46] "public static double java.lang.Math.hypot(double, double)"
## [47] "public static double java.lang.Math.expm1(double)"
## [48] "public static double java.lang.Math.log1p(double)"
## [49] "public static double java.lang.Math.copySign(double, double)"
## [50] "public static float java.lang.Math.copySign(float, float)"
## [51] "public static double java.lang.Math.nextAfter(double, double)"
## [52] "public static float java.lang.Math.nextAfter(float, double)"
## [53] "public static double java.lang.Math.nextUp(double)"
## [54] "public static float java.lang.Math.nextUp(float)"
## [55] "public final native void java.lang.Object.wait(long) throws java.lang.InterruptedException"
## [56] "public final void java.lang.Object.wait() throws java.lang.InterruptedException"
## [57] "public final void java.lang.Object.wait(long, int) throws java.lang.InterruptedException"
## [58] "public native int java.lang.Object.hashCode()"
```

```
## [59] "public final native java.lang.Class java.lang.Object.getClass()"
## [60] "public boolean java.lang.Object.equals(java.lang.Object)"
## [61] "public java.lang.String java.lang.Object.toString()"
## [62] "public final native void java.lang.Object.notify()"
## [63] "public final native void java.lang.Object.notifyAll()"
```

**names**(pi)

```
## [1] "class"          "E"              "PI"             "abs("
## [5] "abs("          "abs("          "abs("          "sin("
## [9] "cos("          "tan("          "atan2("        "sqrt("
## [13] "log("          "log10("        "pow("          "exp("
## [17] "min("          "min("          "min("          "min("
## [21] "max("          "max("          "max("          "max("
## [25] "scalb("        "scalb("        "getExponent("  "getExponent("
## [29] "signum("       "signum("       "asin("         "acos("
## [33] "atan("         "toRadians("    "toDegrees("    "cbrt("
## [37] "IEEEremainder(" "ceil("         "floor("        "rint("
## [41] "round("        "round("        "random()"       "ulp("
## [45] "ulp("          "sinh("         "cosh("         "tanh("
## [49] "hypot("        "expm1("        "log1p("        "copySign("
## [53] "copySign("    "nextAfter("    "nextAfter("    "nextUp("
## [57] "nextUp("
```

## pi\$+TAB

```
R Console

[17] "min("          "min("          "min("          "min("
[21] "max("          "max("          "max("          "max("
[25] "scalb("        "scalb("        "getExponent("  "getExponent("
[29] "signum("       "signum("       "random()"       "IEEEremainder("
[33] "acos("         "asin("         "atan("         "cbrt("
[37] "ceil("         "copySign("    "copySign("    "cosh("
[41] "expm1("        "floor("        "hypot("        "log1p("
[45] "nextAfter("    "nextAfter("    "nextUp("       "nextUp("
[49] "rint("         "round("        "round("        "sinh("
[53] "tanh("         "toDegrees("    "toRadians("    "ulp("
[57] "ulp("

> pi$
pi$class      pi$E          pi$PI         pi$abs(
pi$sin(       pi$cos(      pi$tan(       pi$atan2(
pi$sqrt(      pi$log(      pi$log10(     pi$pow(
pi$exp(       pi$min(      pi$max(       pi$scalb(
pi$getExponent( pi$signum(   pi$random()   pi$IEEEremainder(
pi$acos(      pi$asin(     pi$atan(      pi$cbrt(
pi$ceil(      pi$copySign( pi$cosh(      pi$expm1(
pi$floor(     pi$hypot(    pi$log1p(     pi$nextAfter(
pi$nextUp(    pi$rint(     pi$round(     pi$sinh(
pi$tanh(      pi$toDegrees( pi$toRadians( pi$ulp(
> pi$|
```



## 使用Java Library Jar

如果使用者要呼叫非JDK內建的物件(如：[SWT](#))，則必須先匯入定義該物件的jar檔。具體操作如下：

- 用 `.jaddClassPath` 設定R 搜尋 jar 的路徑。

```
.jaddClassPath(dir("C:/rJava", full.names = TRUE))
```

完成後，可用 `.jclassPath()` 來確認。

```
.jclassPath()
```

```
## [1] "/home/wush/R/x86_64-pc-linux-gnu-library/3.0/rJava/java"
```

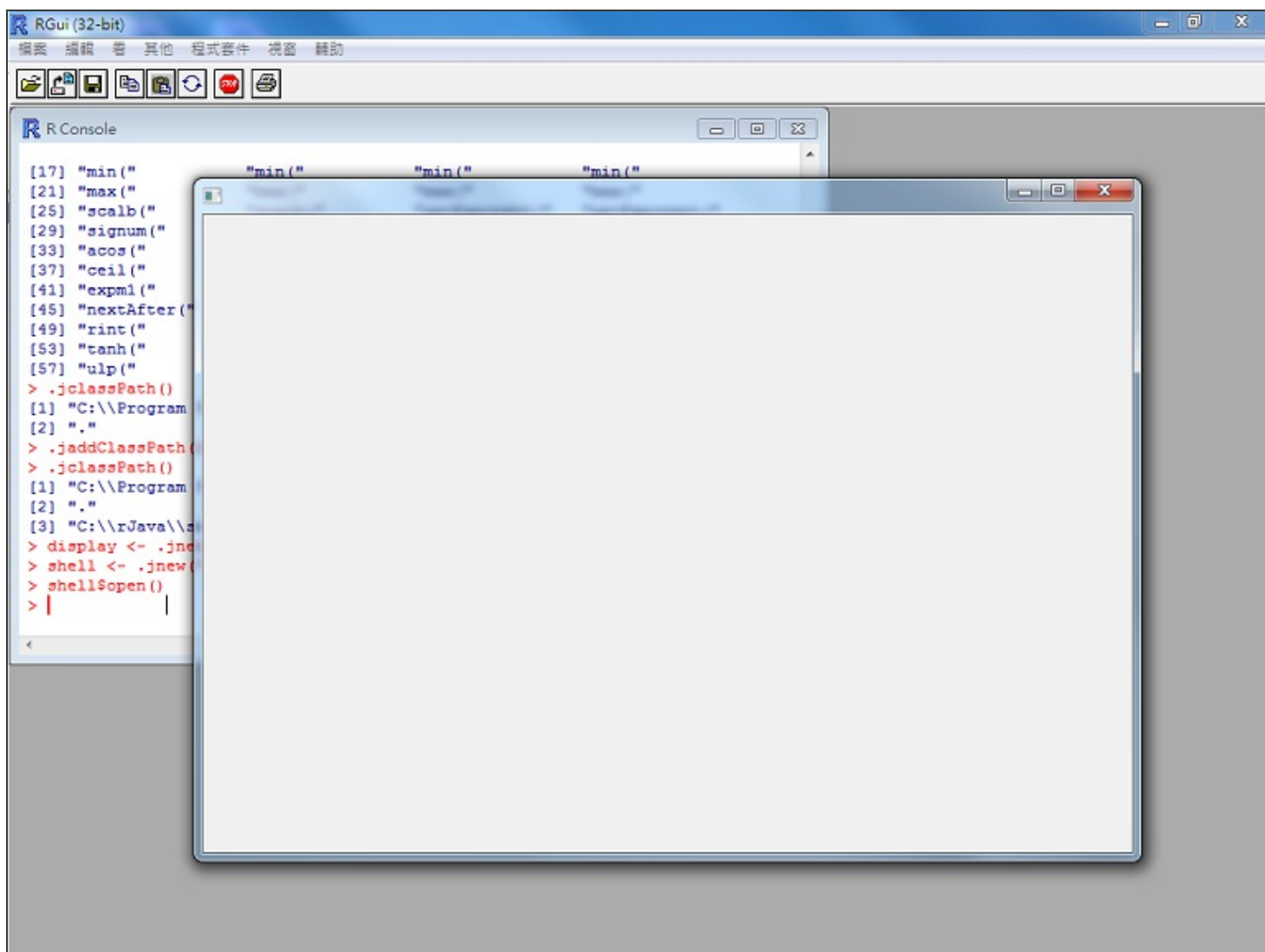
接下來，新增一個SWT的Display物件。

```
display <- .jnew("org/eclipse/swt/widgets/Display")
```

再新增一個Shell物件來裝Display物件。

```
shell <- .jnew("org/eclipse/swt/widgets/Shell", display)
shell$open()
```

便可產生出SWT視窗。



SWT Demo

## Java 程式碼包裝成 R Package

接下來依照那，[Hello Java World! A Tutorial for Interfacing to Java Archives inside R Packages](#)，撰寫一個使用 rJava 的 R Package。這樣所有之前繁瑣的設定動作，全部簡化成 `library(xxx)`，而且需要的 jar 檔也可以隨著套件散佈，並安裝到適當的位置。

## 利用 Google SMTP 傳送 Email

首先，撰寫好 Java 傳送 Email 的程式打包成 jar 檔，放置套件根目錄(範例中是 `C:\helloJavaWorld`)內的 `inst\java` 目錄下，並於套件根目錄的 R 目錄下，新增 `email.R`：

```
email <- function(s, o) {
  email <- .jnew("addEvent")
  email$GamilSender(s, o)
}
```

接下來要設定套件，讓 `email` 函數可以供其他使用者呼叫：

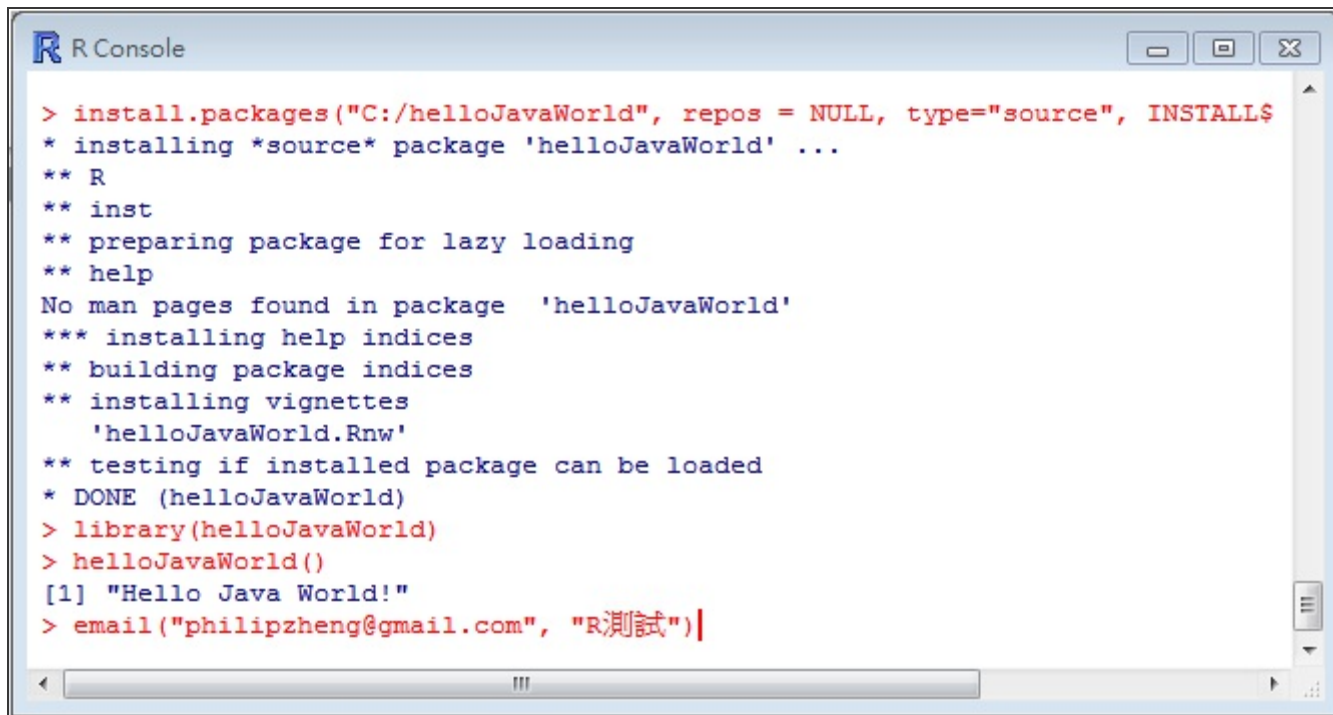
- 到套件根目錄下的 DESCRIPTION 檔案之內，在 Collate: 下加上 'email.R'。

- 修改套件根目錄下的NAMESPACE檔案，在檔案中加上export("email")。

完成後，透過

```
install.packages("C:/helloJavaWorld", repos = NULL, type = "source", INSTALL_opts = "--no-multiarch")
```

安裝剛剛建立的套件後，就可以用email函數來寄信了！



```
R Console
> install.packages("C:/helloJavaWorld", repos = NULL, type="source", INSTALL$
* installing *source* package 'helloJavaWorld' ...
** R
** inst
** preparing package for lazy loading
** help
No man pages found in package 'helloJavaWorld'
*** installing help indices
** building package indices
** installing vignettes
'helloJavaWorld.Rnw'
** testing if installed package can be loaded
* DONE (helloJavaWorld)
> library(helloJavaWorld)
> helloJavaWorld()
[1] "Hello Java World!"
> email("philipzheng@gmail.com", "R測試")
```

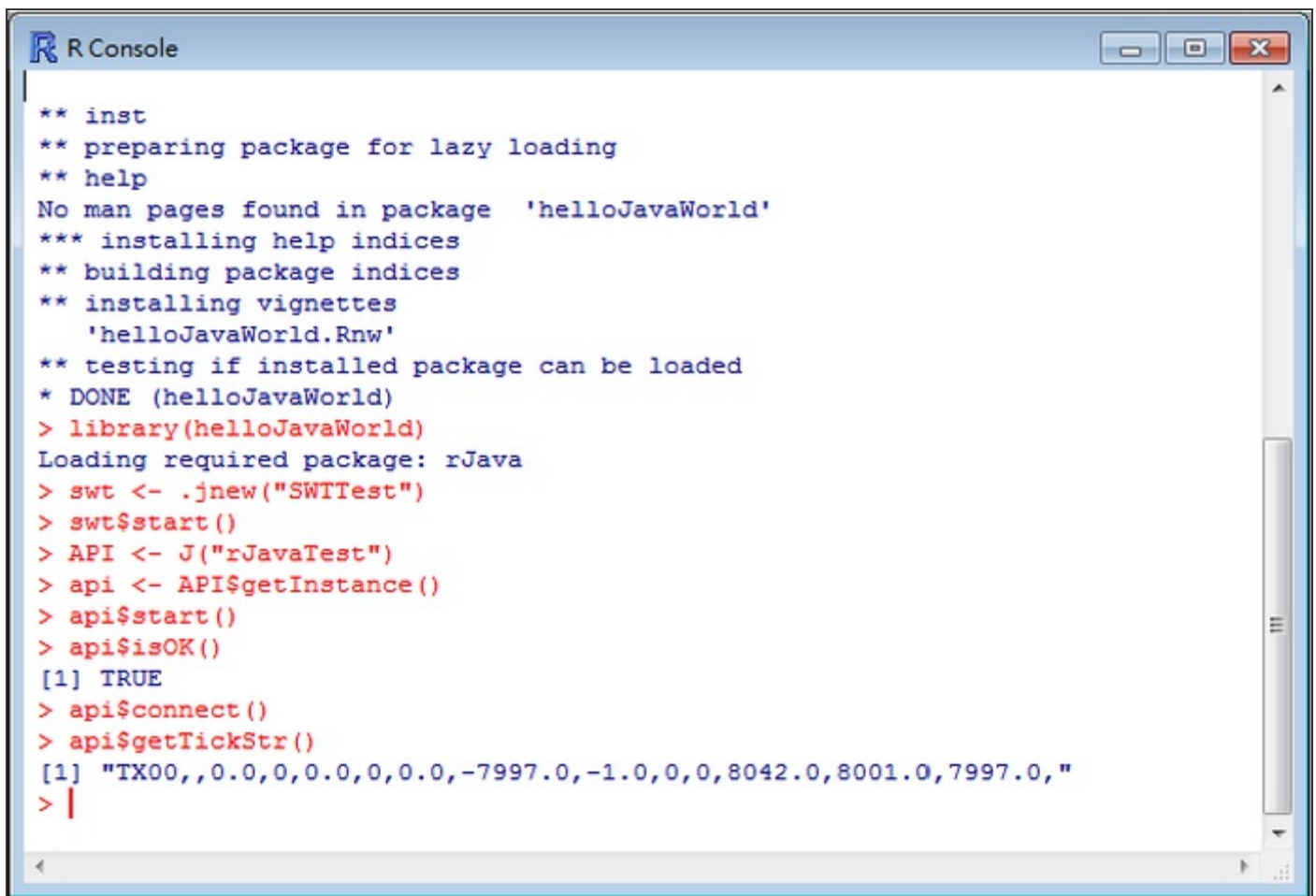
Email Demo

## SWT GUI While Loop 問題

以下是另一個使用SWT的範例，[rJavaTest.java](#)。

因R 是單緒程(Single Thread)，所以直接使用SWT語法的while loop，會發生R一直停在那Java程式(blocking)。爲了避免blocking，我們將SWT的使用方法改寫成使用Java Thread物件，並利用Design Pattern – Singleton來取值，這是用rJava呼叫Java程式時可能會遇到的狀況。

```
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep()
}
display.dispose()
```



```
R Console
** inst
** preparing package for lazy loading
** help
No man pages found in package 'helloJavaWorld'
*** installing help indices
** building package indices
** installing vignettes
'helloJavaWorld.Rnw'
** testing if installed package can be loaded
* DONE (helloJavaWorld)
> library(helloJavaWorld)
Loading required package: rJava
> swt <- .jnew("SWTTest")
> swt$start()
> API <- J("rJavaTest")
> api <- API$getInstance()
> api$start()
> api$isOK()
[1] TRUE
> api$connect()
> api$getTickStr()
[1] "TX00,,0.0,0,0.0,0,0.0,-7997.0,-1.0,0,0,8042.0,8001.0,7997.0,"
> |
```

Singleton Demo

## 作者

### Philipz (philipzheng@gmail.com)

- [Taiwan R User Group](#) Officer
- 研究領域：Image Processing, Software Engineering, Algorithmic Trading
- 開放原始碼專案：[TradingBot](#) 程式交易機器人
- Blog: [Philipz學習日誌](#)

### 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「羅慧夫顏顏基金會 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

財團法人羅慧夫顱顏基金會	<a href="http://www.nncf.org/">http://www.nncf.org/</a> <a href="mailto:lynn@nncf.org">lynn@nncf.org</a> 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	<a href="http://www.cyga.org/">http://www.cyga.org/</a> <a href="mailto:cyga99@gmail.com">cyga99@gmail.com</a> 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