

程式人

月刊
雜誌

Programmer

創刊號：2013年1月出刊

本期內容

- 看影片學電磁學
- Arduino 入門教學
- JavaScript 與 node.js
- Web 應用程式開發
 - Single Page Application
- 小談模型存取
- UART 程式設計

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



關於本雜誌

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，請加入我們的 facebook 社團：
<https://www.facebook.com/groups/programmerMagazine/> 一同共襄盛舉。

給雜誌閱讀者：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)

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。

授權聲明

本雜誌採用創作共用的「姓名標示、非商業性、相同方式分享」的授權，其中許多內容來自於維基百科與開放原始碼社群，以下是本文採用的授權網址。

<http://creativecommons.org/licenses/by-nc-sa/3.0/tw/>

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. 不可以作為商業用途。
3. 採用「姓名標示-非商業性-相同方式分享」的方式公開衍生著作。
4. 不得去除公益捐贈的相關描述。

程式人雜誌編輯 - 陳鍾誠 2012/11/13

內容目錄

程式人短訊.....	10
1.軟體短訊：Eclipse	10
2.硬體短訊：Arduino.....	11
3.法律短訊：創用 CC 授權.....	12
程式人介紹.....	14
1.Alan Turing：電腦科學之父.....	14
2.姚期智：2000 年圖零獎得主.....	15
3.陳鍾誠：金門大學資工系教師、程式人與部落客.....	16
4.趙琨堯：韌體工程師.....	17
程式人頻道.....	19
1.看影片學電磁學.....	19
程式人文集.....	27
1.Arduino 入門教學 (1) – 認識 Arduino (作者：Copper Maa).....	27
1.1.什麼是 Arduino.....	27

1.2.Arduino 的特色.....	28
1.3.Arduino 的應用.....	29
1.4.Arduino 硬體規格.....	33
1.4.1.數位 I/O Pins:.....	34
1.4.2.類比輸入 Pins:.....	34
1.5.Arduino 軟體開發環境.....	35
1.6.Arduino 硬體版本.....	36
1.7.Arduino 擴充板 (Shields).....	38
2.JavaScript (1) – 簡介與基本語法 (作者：陳鍾誠).....	41
2.1.程式人對 JavaScript 常見的誤解.....	41
2.2.JavaScript 的歷史.....	42
2.3.JavaScript 的基本語法.....	43
3.Web 應用程式開發(1) - Web 應用程式的演進與 Single Page Application (作者：李開文).....	48
3.1.Web 應用程式的演進.....	48
3.1.1.動態網頁.....	48
3.1.2.Rich Internet Application.....	49
3.2.Single Page Application.....	53

3.2.1.定義.....	53
3.2.2.所依賴的 Web 技術.....	54
3.2.3.SPA 的架構.....	57
3.3.結論.....	60
4.小談模型存取–以 Factory Pattern 實現 (作者： pojungwu).....	62
4.1.前言.....	62
4.2.相關背景.....	62
4.3.程式解析.....	63
4.4.後記.....	71
4.5.程式碼下載.....	72
5.UART 程式設計 (作者：趙琨堯).....	73
5.1.簡介.....	73
5.2.運作原理.....	73
5.2.1.平台硬體連接：.....	73
5.2.2.終端機設定的部份：.....	74
5.2.3.UART 的原理.....	76
5.2.4.程式導讀 – 主程式：.....	77

5.2.5. 程式導讀 – 處理指令函式(主程式).....	78
5.2.6. 程式導讀 – 處理測試程式函式部份.....	78
5.2.7. 程式導讀 – 處理 HELP 函式.....	83
5.2.8. 程式導讀 – 處理錯誤指令函式.....	84
5.2.9. 程式導讀 – 主程式說明.....	85
5.2.10. 副程式函式 – HELP 說明函式.....	85
5.2.11. 副程式函式 – ASCII 轉 BIN 碼函式.....	87
5.2.12. 副程式函式 – 清除顯示的函式.....	87
5.2.13. 副程式函式 – 顯示輸入資料函式(除錯資訊).....	88
5.2.14. 副程式函式 – 顯示 UART 傳送錯誤函式(除錯資訊).....	89
5.2.15. 副程式函式 – 解析接收指令函式.....	92
5.2.16. 副程式函式 – IIC 部份介紹.....	94
5.2.17. 副程式函式 – IIC 硬體設定.....	94
5.2.18. 副程式函式 – EEPROM 資料檢查碼函式.....	95
5.2.19. 副程式函式 – EEPROM 回覆預設值函式.....	96
5.2.20. 副程式函式 – EEPROM 讀取密碼(EEPROM 連續讀取資料範例).....	97
5.2.21. 副程式函式 – EEPROM 寫入密碼函式(EEPROM 連續寫入資料範例).....	98

5.2.22.副程式函式 – EEPROM.H 檔案部份	99
5.2.23.程式導讀 – 使用者規劃檔案	102
5.2.24.程式導讀 – 主程式預處理部份	105
5.2.25.程式導讀 – 中斷常式介紹	108
5.2.26.程式導讀 – 中斷向量位址	108
5.2.27.程式導讀 – 中斷服務常式	109
5.2.28.程式導讀 – 中斷服務常式說明	112
5.2.29.參考資料 – BAURD RATE 計算公式	112
5.2.30.參考資料 – PIC18F45J10 系列 – 波特率發生器 (BRG)	115
5.2.31.6.3 參考資料 – 常用的頻率與使用到的技術對應表 :	117
5.3.結語	117
邀稿與訂閱	118
1.讀者訂閱	118
2.作者徵求	118
3.參與編輯	119
4.公益資訊	119

程式人短訊

1. 軟體短訊：Eclipse

Eclipse 是著名的跨平台的自由整合式開發環境（IDE）。最初主要用來 Java 語言開發，目前亦有人透過外掛程式使其作為 C++、Python、PHP 等其他語言的開發工具。

Eclipse 的本身只是一個框架平台，但是眾多外掛程式的支援，使得 Eclipse 擁有較佳的靈活性。許多軟體開發商以 Eclipse 為框架開發自己的 IDE。

當您想寫 Java、Andriod 等程式時，就可以使用 Eclipse。Eclipse 可以說是開放原始碼領域的 Visual

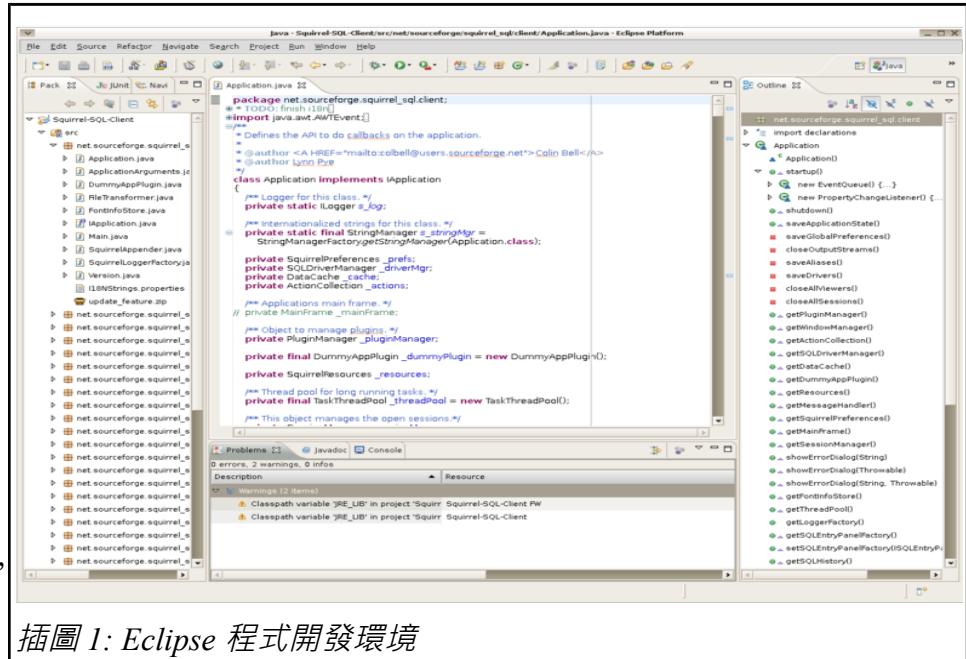


插圖 1: Eclipse 程式開發環境

Stdgio，功能很強大，也有類似 Intellisense 的視覺提示，寫起程式來會輕鬆愉快許多。【本文由陳鍾誠取材並修改自維基百科】

2. 硬體短訊：Arduino

Arduino，是一個開源的單板機控制器，採用了基於開放原始碼的軟硬體平台，構建於開放原始碼 simple I/O 介面版，並且具有使用類似 Java，C 語言的 Processing/Wiring 開發環境。

以下是 Arduino 的程式範例：LED 每秒閃爍一次。

```
int LED_PIN=13;

void setup () {
    // 啟用 Pin13
    pinMode (LED_PIN, OUTPUT);
}
```



Arduino.tw網站上的Arduino圖像

插圖 2: Arduino 單晶片開發板

```

void loop () {
    digitalWrite (LED_PIN, HIGH); // 打开 LED
    delay (1000); // 等待一秒
    digitalWrite (LED_PIN, LOW); // 关闭 LED
    delay (1000); // 等待一秒
}

```

【本文由陳鍾誠取材並修改自維基百科】

3. 法律短訊：創用 CC 授權

創作共用 Creative Commons 是哈佛大學法律教授 Lawrence Lessig 所創造的授權方法。Lessig 從開放原始碼領域得到靈感，制定了一系列可選擇的授權方式，目前通常簡稱為 CC 授權。

CC 授權共有四種屬性，其中的「姓名標示」為必選，可組合出六種基本授權，這些屬性與授權組合如下表所示，您可以透過 CC 授權釋出作品，也可以尋找網路上的 CC 授權文章、圖片、影片等進行衍生創作，修改後再度創作，本雜誌就是採用 CC 的「姓名標示、非商業性、相同方式分享」的授權釋出的。

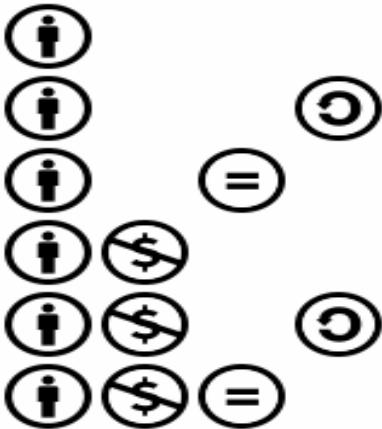
標誌	英文	縮寫	全稱
	Attribution	BY	姓名標示
	NonCommercial	NC	非商業性使用
	NoDerivs	ND	禁止改作
	ShareAlike	SA	相同方式分享

插圖 3: 創作共用的四種屬性

授權條款

姓名標示 (BY)

BY NC ND SA



姓名標示 (BY) - 相同方式分享 (SA)

姓名標示 (BY) - 禁止改作 (ND)

姓名標示 (BY) - 非商業性 (NC)

姓名標示 (BY) - 非商業性 (NC) - 相同方式分享 (SA)

姓名標示 (BY) - 非商業性 (NC) - 禁止改作 (ND)

插圖 4: 創作共用 (Creative Commons) 的六種基本授權

【本文由陳鍾誠取材並修改自維基百科】

程式人介紹

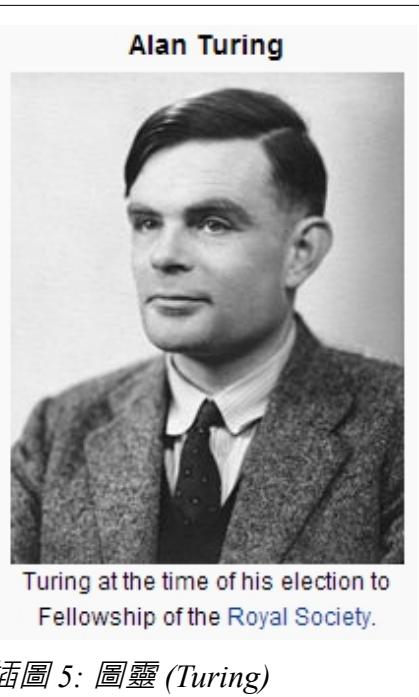
1. Alan Turing：電腦科學之父

圖靈 (Alan Mathison Turing) 是電腦領域的先驅，從 1930 年代電腦尚未發明的時候就開始研究電腦了，是公認的電腦科學之父。

念過計算理論的人一定會知道圖靈機 (Turing Machine) 這個機器 (小編也曾經在博士班的時後慘遭圖靈機的毒害)，透過這個機器圖靈探索了電腦能力的極限，並發現有些問題是電腦不可能解答的，像是著名的「停止問題」便是其中之一。

另外、著名的「圖靈測試」也是圖靈在一篇名為《機器會思考嗎？》(Can Machines Think?) 的論文中提出來的，他認為如果機器可以透過類似 MSN 或 facebook 的聊天過程成功的欺騙人類，讓人分不清楚對方到底是人還是機器，那就代表電腦是有智慧的。

圖靈於二戰時回到英國劍橋，曾協助軍方破解德國的著名密碼系統 Enigma，對盟軍取得了二戰的勝利有不少幫助。【本文由陳鍾誠取材並修改自維基百科】



Alan Turing

Turing at the time of his election to Fellowship of the Royal Society.

插圖 5: 圖靈 (Turing)

2. 姚期智：2000 年圖靈獎得主

姚期智（Andrew Chi-Chih Yao，1946 年 12 月 24 日—）是華人當中唯一得到計算機學術領域的最高榮譽「圖靈獎」的人。

姚期智主要的貢獻在計算理論領域，因為偽隨機數生成，密碼學與通信複雜性領域的貢獻而得到「圖靈獎」。

姚期智在台灣長大，1967 年 畢業於台灣大學，1972 年 獲哈佛大學物理學博士學位，1975 年 獲伊利諾大學香檳分校（UIUC）計算機科學博士學位。之後，他曾先後在麻省理工學院（1975—1976）、斯坦福大學（1976—1981，1983—1986）、加州大學伯克利分校（1981—1983）等美國高等學府從事教學和研究，1986 年至 2004 年任普林斯頓大學計算機科學系教授。目前他在中國大陸清華大學、香港中文大學都有任教。【本文由陳鍾誠取材並修改自維基百科】

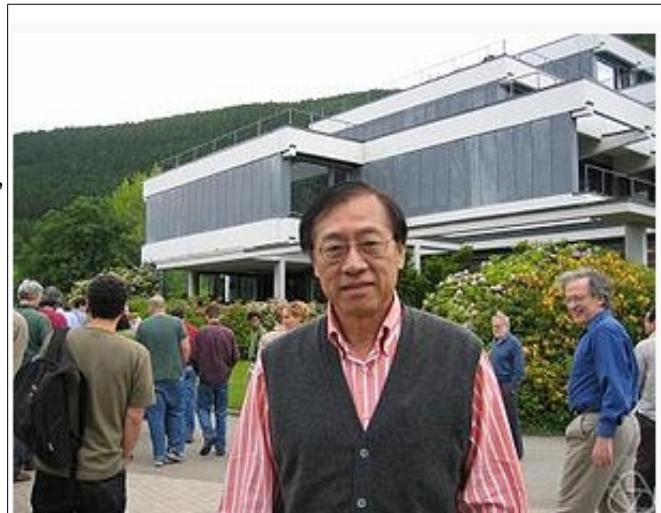


插圖 6: 姚期智

3. 陳鍾誠：金門大學資工系教師、程式人與部落客

真是不好意思，竟然將小編自己排在「圖靈」和「姚期智」後面，簡直就是「自抬身價，欺世盜名」。

不過還請各位讀者原諒，因為本刊第一期創刊，大家都很客氣，沒人敢跳出來讓我採訪，所以我只好「自己採訪自己」，也算是練習了周伯通的「左右互搏之術」吧！

小編服務於金門大學資工系，主要教授「視窗程式、網路程式、系統程式、微積分、機率統計、計算機結構、計算語言學、動畫設計」等課程。

小編熟悉的程式語言有 C, Java, C# 等，後來為了教動畫而學了 Blender 軟體，為了教機率統計而學了 R 軟體，發現這些開源軟體都超好用，在此極力推薦給大家。

2002 年開始接觸到開放原始碼、創作共用、



插圖 7: 陳鍾誠的網站

維基百科、開放教材之後，就開始對這個領域很感興趣，此次創辦「程式人」雜誌，希望能將開放原始碼的理念引入公益事業，還希望大家多多宣傳分享這樣的「開放公益出版」理念。(陳鍾誠的網站：<http://ccckmit.wikidot.com/> ,本文作者：陳鍾誠)

4. 趙琨堯：韌體工程師

趙琨堯在 facebook 上的網名是若竭，筆者第一次在臉書上注意到若竭是因為他與網友討論有關韓國與三星近來崛起的事情。若竭的很多想法讓我以為他是學社會或歷史的人，因為他對金觀濤的「興盛與危機」、以及古籍「商君論」等都有很獨到的見解。

結果後來我才知道，原來若竭竟然是個有八年經驗的韌體工程師，甚至也做過硬體工作，包含電路驗證與工程驗證測試(EVT)的工作都做過，這讓我非常的驚訝！

若竭所從事的韌體工作主要與微控制器 (MCU) 有關，像是第一份工作就是在製作掌上型 DVD 的部門，部門內部有 5 個人，分為處長負責產品規劃 HW 導入與客戶等，副處長負責料件與 BOM 等生產以及 QC 品管工作，一位 LAYOUT 與硬體協助，一位機構與 ID 工業設計工程師，以及由若竭擔任的韌體工程師，負責 PIC 組合語言以及主 IC MPEG



插圖 8: 趙琨堯

程式 DEBUG 和維護。

對於有志進入韌體與 MCU 領域的同學而言，需要學會甚麼樣的技術呢？若竭表示：「對於韌體工程師而言，在 MCU 的部份以 ASM 與 C 工具就夠，並且要具備 MCU 與電路設計的知識，和看得懂 SPEC，在現在的 ARM 趨勢多以 C 更高階的是以 C++ 但是用 C 也可以被 C++ 編輯器支援，編輯器像是 KEIL C , IAR , GCC , X86 架構的 Paradise , Eclipse 等 EDA 工具，而對於硬體工程師而言，要會用一些業界常用的 EDA 工具，如 ORCAD 、PROTEL 、POWERPCB 等等」。

或許接下來各為讀者就可以看到若竭為我們撰寫 MCU 領域的相關文章，我可是非常期待能看到他的文章呢！(若竭的 facebook : <https://www.facebook.com/kunyau.chao>)【本文由陳鍾誠透過 facebook 對若竭進行專訪】

程式人頻道

1. 看影片學電磁學

在很久很久以前，我還在新竹交通大學資訊科學系一年級上物理課的時候¹，老師在台上用一隻筆上課，對於資訊系的學生們，竟然只教到力學結束就完成了物理課。期末考最重要的一題是計算旋轉陀螺的平衡角度，然後、我們的物理課就上完了，甚麼是電磁學，雖然高中念過，但事實上我們幾乎是一無所知。

於是、二十五年來，不懂電磁學的我，也還是繼續寫著程式，甚至還當了大學資工系的老師，.....

但是，不懂電磁學畢竟是程式人心中的痛。於是，我找了一位老師來教我，那就是 Youtube，今天就讓我們跟著 Youtube 老師來學學電磁學吧！

首先登場的是加拿大 Carleton 大學的教授，我覺得他的實驗影片繼簡單又清楚，很適合筆者的程度。

主題	影片網址	說明
靜電 (1)	http://youtu.be/yU55lXbrV0U	Physics Lab Demo 1: Introduction to Electrostatics

¹ 那是 1988 年，電腦還處於 DOS 時代，還使用著磁碟片開機，我們都還不知道網路是甚麼東西的時代。

		摩擦生靜電，靜電極化後互相吸引。
靜電 (2)	http://youtu.be/cMM6hZiWnig	Physics Lab Demo 2: Electrostatic Induction 驗電器：靜電極化後互相排斥、兩片銅片的排斥實驗
靜電 (3)	http://youtu.be/AhLbYaIoxsE	Physics Lab Demo 3: Van der Graaf Experiment 范德格拉夫起電機：兩銅球之間靠近後放電 ²
示波器	http://youtu.be/QsvRHvRVuZk	Physics Lab Demo 4: Oscilloscope (示波器)
放電	http://youtu.be/SE0E9r9w1bo	Physics Lab Demo 5: Exploding Wire 大電容放電爆炸效果。
磁力線	http://youtu.be/GCHJmMdHNPo	Physics Lab Demo 6: Magnetic Field 磁場：鐵屑的磁力線實驗。
電磁感應 (1)	http://youtu.be/o1z2S3ME0cI	Physics Lab Demo 7: Thompson Experiment 電子槍：湯普生實驗，電子束受到磁鐵影響而彎曲 ³ 。
電磁感應 (2)	http://youtu.be/dNmMNC4qGS_A	Physics Lab Demo 8: Electric Motor ⁴ 電動馬達：電磁交互作用。

2 參考：http://en.wikipedia.org/wiki/Van_de_Graaff_generator

3 參考：<http://library.thinkquest.org/19662/high/eng/exp-thomson.html>, http://en.wikipedia.org/wiki/J._J._Thomson

4 參考：http://en.wikipedia.org/wiki/Electric_motor, Induction Motor How it works <https://www.youtube.com/watch?v=HWrNzUCjbkk> , Magnetism: Motors and Generators — https://www.youtube.com/watch?v=d_aTC0iKO68

電磁感應 (3)	http://youtu.be/7MTTyWzX1EY	Physics Lab Demo 9: Faraday's Experiment 法拉第實驗：線圈電流變化的時候，會引發另一個線圈產生電流 ⁵ 。
電磁感應 (4)	http://youtu.be/lCu3DrIY2w	Physics Lab Demo 10: Induced Electromagnetic Force (EMF) 感應電動勢：法拉第實驗 2，一迴路電流變化時，導至另一條迴路的燈亮了。
電磁感應 (5)	http://youtu.be/0_kKXJUargU	Physics Lab Demo 11: Jumping Ring 電磁線圈中心有個鐵塊，通電後造成上面的鐵圈跳起來。
電磁感應 (6)	http://youtu.be/OJvEOXsSuaQ	Physics Lab Demo 12: Eddy Currents 鐵版在磁鐵中心搖擺，造成反向作用力 (冷次定律)。

看完了教授的示範，覺得真的相當過癮，但是還是有些貪心，想再多看一些更棒的影片，於是我們找到了 IB Physics Help Video Podcast 這個頻道，裏面的實驗更是精彩。

主題	影片網址	說明
電磁互生	http://youtu.be/6fr4VR3fAuw	Applications of electromagnetic induction

5 參考：http://en.wikipedia.org/wiki/Michael_Faraday。

馬達	http://youtu.be/lw8MMq_FvOw	Motor effect
電流的磁場	http://youtu.be/3KkOqVEa1oI	Magnetic field of a current
感應電壓	http://youtu.be/_ud5yz-5CLQ	Voltage across the terminals of a battery
電磁感應	http://youtu.be/hajIIGHPeuU	Electromagnetic induction
直流馬達	http://youtu.be/5Usj4QbY2I4	Variation of the electric current in a DC motor
電磁感應	http://youtu.be/l-RjuauyuzM	IBPH Ep. 9 Electromagnetic Induction (讚！法拉第定律+冷次定律)
艾迪電流	http://youtu.be/SK0EdikjC24	Eddy Current Brake (艾迪電流煞車、冷次定律的一個明顯範例)

在以上的實驗中，最重要也最難理解的是電磁之間的交互作用，這點可以用法拉第定律來解釋，

$$\mathcal{E} = -\frac{d\Phi_B}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\Delta\Phi_B}{\Delta t}$$

其中的 \mathcal{E} 是感應電動勢，也就是磁場增減時所造成的電壓改變，而 $\Delta\Phi_B$ 是 Δt 時間內磁通量的改變量，這個公式是理解電磁交互作用的關鍵。



一個簡單的解釋是，當磁場改變的時候，會引發感應電動勢（電壓），因而造成電流，於是實驗中我們會發現當磁鐵從線圈中抽出的時候，造成了安培計上的電流指針晃動，而當我們將磁鐵放入線圈時，也造成了安培計上的電流指針晃動，而且兩者方向相反。

反過來看，當電流流動的時候，會造成磁場，一條長而且直的導線，其周遭的磁場是環狀的，磁場的方向可以根據右圖的安培右手定則來決定。而其電流大小與磁場則可由以下的安培環路定律決定。

$$\oint_C \mathbf{B} \cdot d\ell = \mu_0 I$$

(安培環路定律的解釋：電流 I 在一個曲面 S 上的通量，等於磁場 \mathbf{B} 沿著 S 的邊緣閉合迴路 C 的路徑積分)

如果我們將電線纏繞成一圈圈的環狀，通電候這些環狀電流會造成線圈內部形成單一方向的磁場，此時若再線圈內部加入一個鐵棒，則會增強磁場的強度，讓鐵棒變成強力磁鐵。

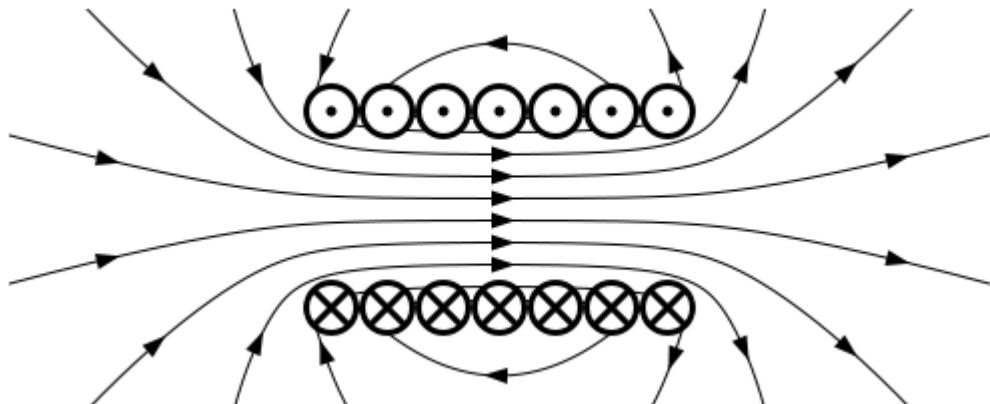
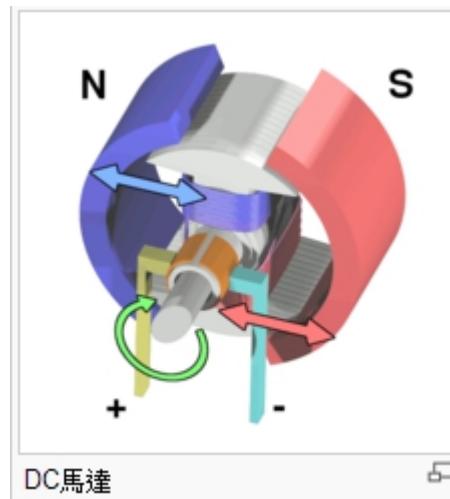


插圖 10: 環狀線圈中的磁場

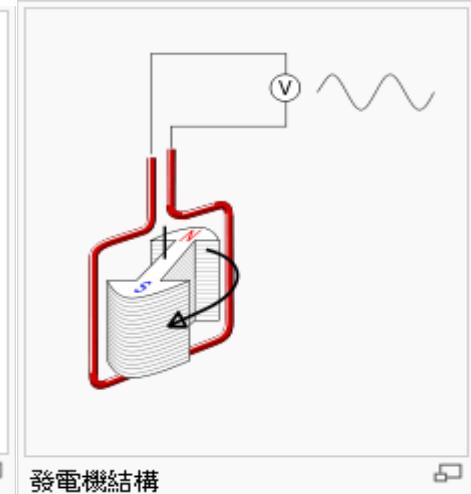
如果我們將纏有鐵棒的線圈放入永久磁鐵的兩極中，根據電流的流向就會產生吸引與排斥的效果，如果利用某種設計，讓電流的流向所產生的磁場每半圈反轉一次，那麼就能讓線圈磁鐵與永久磁鐵兩者間持續產生排斥力量，於是不停的旋轉，這就是馬達的原理。

相反的，如果我們用外力轉動線圈，那麼由於法拉第定律中的磁通量改變，也會產生電流，這種將馬達反轉過來使用的方式，就成了發電機。



DC馬達

插圖 11: 馬達



發電機結構

插圖 12: 發電機

以下影片解釋了馬達的運作原理：

主題	影片網址	說明
馬達 (1)	http://youtu.be/dNmMNC4qGSA	Physics Lab Demo 8: Electric Motor ⁶ 電動馬達：電磁交互作用。
馬達 (2)	http://youtu.be/HWrNzUCjbkk	Induction Motor How it works
馬達 (3)	http://youtu.be/d_aTC0iKO68	Magnetism: Motors and Generators
馬達 (4)	http://youtu.be/6fr4VR3fAuw	Applications of electromagnetic induction 包含馬達與發電機的示範。

在上數影片中曾經數次提到冷次定律 (Lenz's law)，冷次定律其實是這種電磁效應的一種自然結果，就像力學當中的作用力與反作用力一樣，任何的改變都會引發抵銷該改變的反向力量，透過這種想法會更容易的用直覺的概念去思考作用力、電流與磁場的方向。艾迪電流煞車 (Eddy Current Brake) 實際上就是冷次定律的一個很好的範例。

理解了電磁感應原理之後，我們就可以進一步理解電磁波的發射與接收原理了，當電流呈現震盪變動的情形時，由於其電場線與磁場線會隨著時間以光速向外傳播，於是會形成一圈一圈的封閉場線向外散開，而電場線與磁場線兩者是互相垂直的相互支撐結構，如下圖所示。

6 參考：http://en.wikipedia.org/wiki/Electric_motor, Induction Motor How it works <https://www.youtube.com/watch?v=HWrNzUCjbkk> , Magnetism: Motors and Generators — https://www.youtube.com/watch?v=d_aTC0iKO68

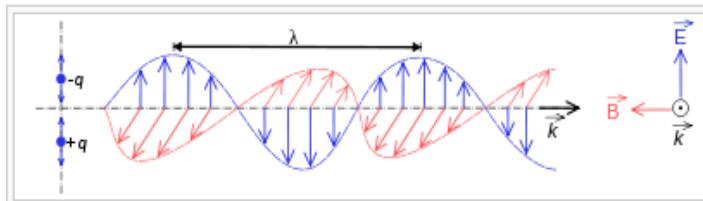


插圖 13: 電流震盪產生電磁波

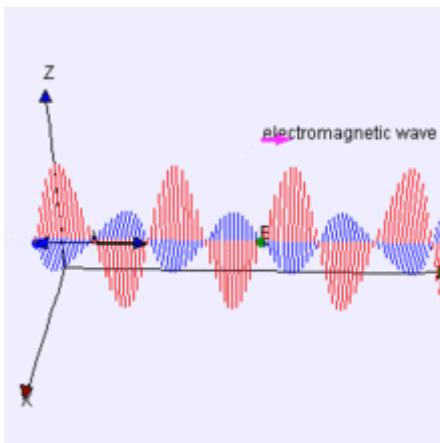


插圖 14: 電磁波 3D 動態圖(1)

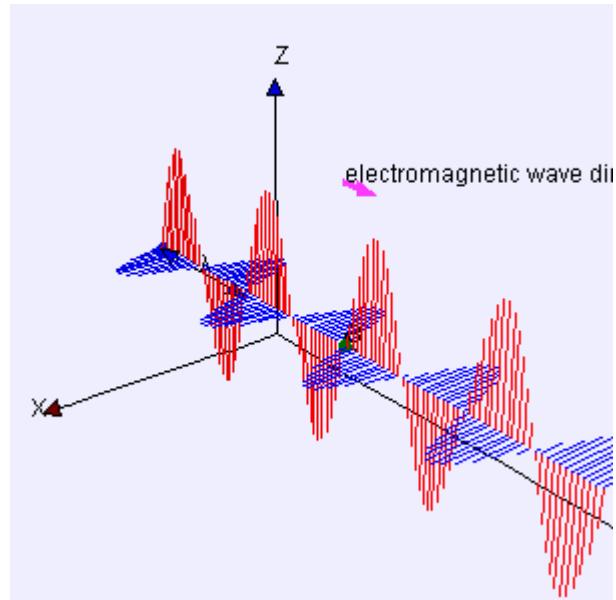


插圖 15: 電磁波 3D 動態圖(2)

【本文由陳鍾誠撰寫，圖片來自自維基百科】

1. Arduino 入門教學 (1) – 認識 Arduino (作者 : Copper Maa)

什麼是 Arduino？它的特色為何？可以拿來做什麼應用？本文將回答這些基本的問題，帶你認識 Arduino⁷。

1.1. 什麼是 Arduino

如下圖，Arduino 是一張微控制器板子(microcontroller board)，大約一個手掌大。

使用者可以在 Arduino 板子上接上各種電子裝置，例如 LED 燈、喇叭、馬達、開關、溫濕度感測器、紅外線發射與接收器、LCD 顯示裝置，以及 Ethernet,

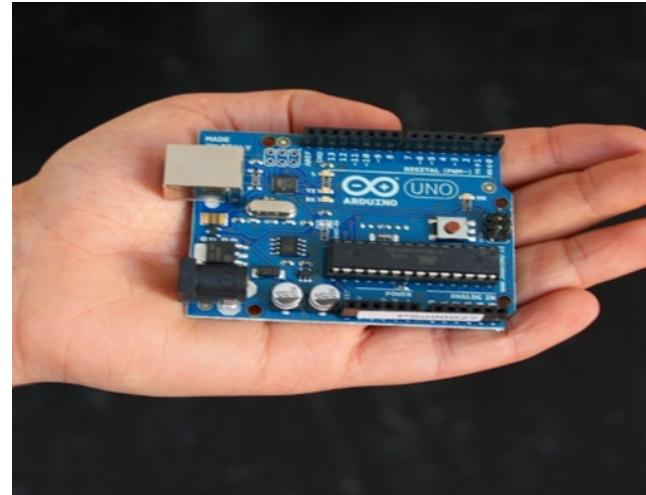


插圖 16: Arduino 微控制器開發板

7 分享一個 Arduino 教學心得：學習 Arduino 最好的方法，就是買張 Arduino 控制板和一些電子零件，捲起袖子實際動手做，唯有如此，才能夠真正體驗 Arduino。所以，本文不會花太多的篇幅在文字上作描述，將很快速的介紹 Arduino，讓你在最短的時間內認識 Arduino。

WiFi, XBee, Bluetooth, RFID, GPS 等各種通訊模組。

若再配合撰寫一些自動控制的程式，就能利用 Arduino 做出各式各樣的自動控制應用，例如利用溫度感測器控制風扇的運轉、使用可變電阻控制燈光的明暗、控制馬達的轉速、利用紅外線遙控家電／利用伺服機(Servo)控制機械手臂或機器人，以及製作自走車、飛行器等等。

1.2. Arduino 的特色

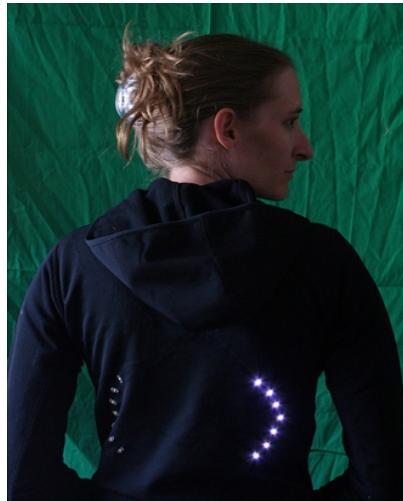
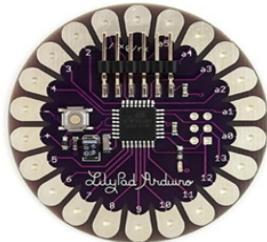
簡單來說，Arduino 有下列這些特色：

1. 開放源碼。不僅軟體是開放源碼，連硬體也是開放的。開發軟體用的 IDE 可免費下載，Arduino 的電路設計圖也可以從網路上下載。
2. 簡單好用資源多。傳統上，要開發微控制器的程式，開發者需要具備電子電機相關科系的背景，一般人不容易進入這個世界。Arduino 進入門檻低，即便沒有電子電機相關科系的背景，也很容易學會使用 Arduino。再者，由於 Arduino 開放的精神，很多人都樂於分享他們的作品，所以網路上有非常多的資源。很多時候，我們只要參考網友的作品，配合自己的需求調整一下設計，就可以在短時間內完成自己的作品。
3. 物美價廉。一張微控制器板子動輒 3000 元台幣，相較於這類微控制板子，Arduino 控制板只要 30 美元左右，可以說是俗擱大碗。

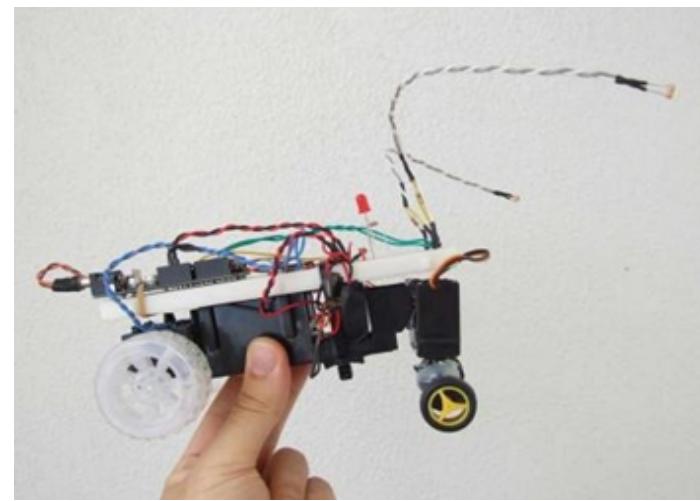
1.3. Arduino 的應用

Arduino 的應用非常多，不勝枚舉，底下只舉幾個我所知的應用：

LilyPad : 可穿戴在身上的 Arduino , 這東西應該可以拿來做鋼鐵人的發光手套。



Easy Robot : 簡易機器人 - 一個很簡單的機器人自走車，網路上有完整的製作教學⁸

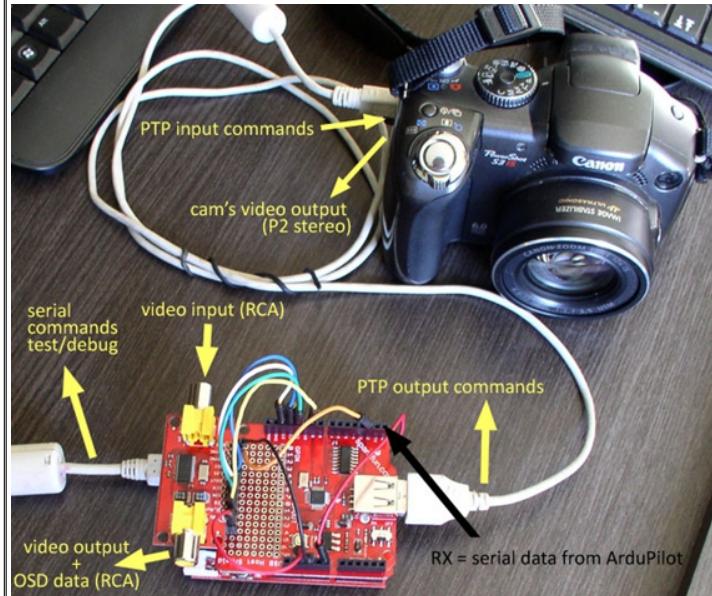


⁸ 影片網址：http://www.youtube.com/watch?feature=player_embedded&v=K93qAywU6H0

Robot ARM - 使用 Arduino 控制機械手臂⁹。



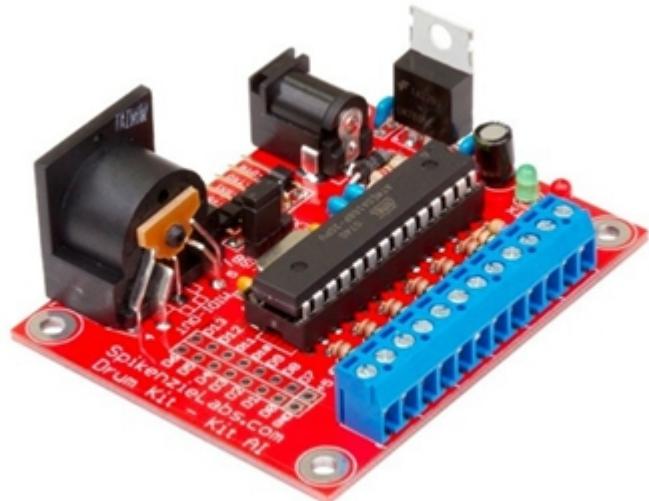
ArduCam - 利用 Arduino 遙控相機¹⁰



9 http://www.youtube.com/watch?feature=player_embedded&v=nz_tgDD8FNw

10 http://www.youtube.com/watch?feature=player_embedded&v=e_8vO9JBHrw

Drum Kit - 電子鼓，有了 Drum Kit，你也可以搖身一變成一個鼓手¹¹

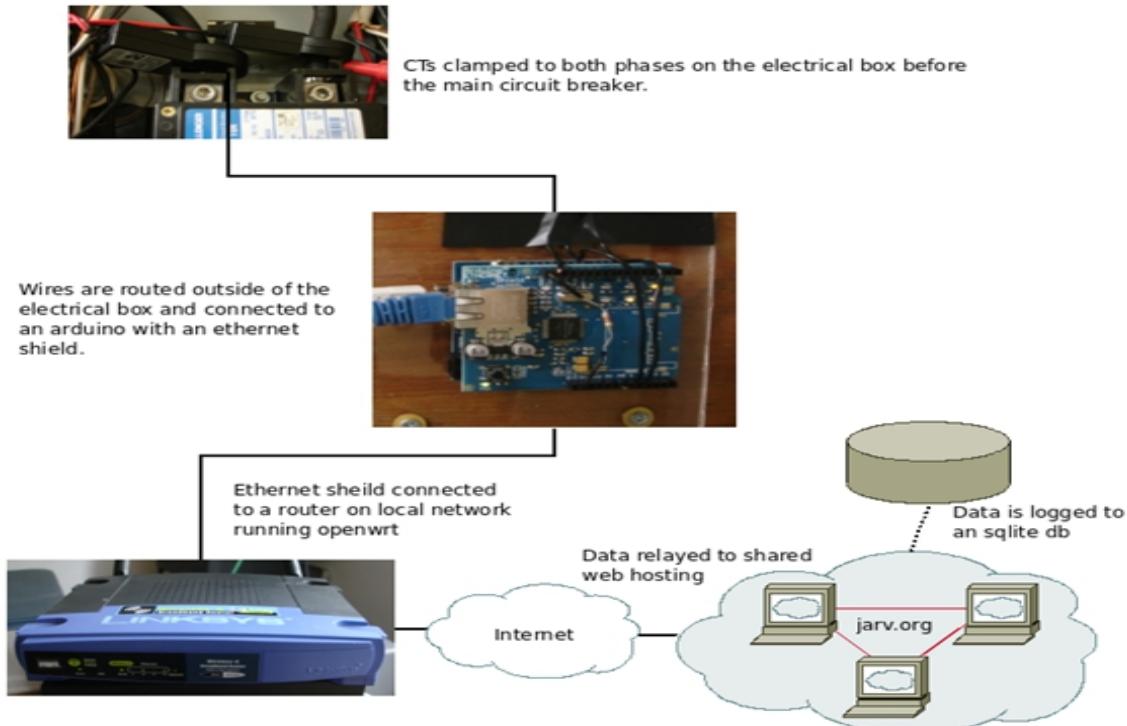


ArduCopter - 四軸直升機，哇嗚！這東西看了真是令人興奮，有錢應該買一台。



11 http://www.youtube.com/watch?feature=player_embedded&v=sQ6NIVnDXcU

Home Power Monitoring - 家庭用電監測系統，利用電流感測器(Current Transducer)監測家庭用電，並且透過 Ethernet 把用電數據傳到網路上，以 Flash 圖表呈現，讓使用者透過瀏覽器就可以看到家裏的用電情形。



1.4. Arduino 硬體規格

Arduino UNO 的外觀與硬體規格如下：

微控制器	ATmega328
工作電壓	5V
輸入電壓(建議)	7-12V
輸入電壓(限制)	6-20V
數位 I/O Pins	14 支 (其中有 6 支腳位可提供 PWM 輸出)
類入 Input Pins	6 支
I/O pin 直流電流	40mA
3.3V pin 直流電流	50mA
Flash 記憶體	32KB, 其中 0.5KB 拿去給 bootloader 使用
SRAM	2KB
EEPROM	1KB
時脈	16MHz

1.4.1. 數位 I/O Pins:

14 支數位 I/O Pins 可以當作 `input` 使用，也可以當作 `output` 使用，使用方法是透過 `pinMode()`, `digitalWrite()`, and `digitalRead()` 這幾個函式。這 14 支數位 I/O Pins，其中幾支腳有特殊的功能：

Serial 通訊	0(RX) 和 1 (TX) 這兩支腳。用來接收(RX)與傳輸(TX) TTL 訊號的序列資料。這兩支腳也連接到 USB Converter 晶片中。
外部中斷	2 和 3 這兩支腳。這兩支腳可以利用外部事件觸發中斷。詳細內容請參考 <code>attachInterrupt()</code> 函式。
PWM	3, 5, 6, 9, 10 和 11 共六支腳。透過 <code>analogWrite()</code> 函式可以提供 8-bit 的 PWM 輸出。
SPI	10 (SS), 11 (MOSI), 12 (MISO) 和 13 (SCK) 這四支腳。這四支腳搭配 SPI Library 可提供 SPI 序列通訊。
LED	13。內建一顆 LED，當 pin 腳為 HIGH 時，LED 打開，當 pin 腳為 LOW 時，LED 關閉。

1.4.2. 類比輸入 Pins:

Arduino Uno 有 6 支類比輸入腳，標記為 A0 到 A5，每支腳都可提供 10 位元的解析 (即 1024 種不同的數值)。這些腳位所用的參考電壓預設為 0 到 5V，不過參考電壓也是可以更改的，方法是透過 `AREF` 腳和 `analogReference()` 函式。

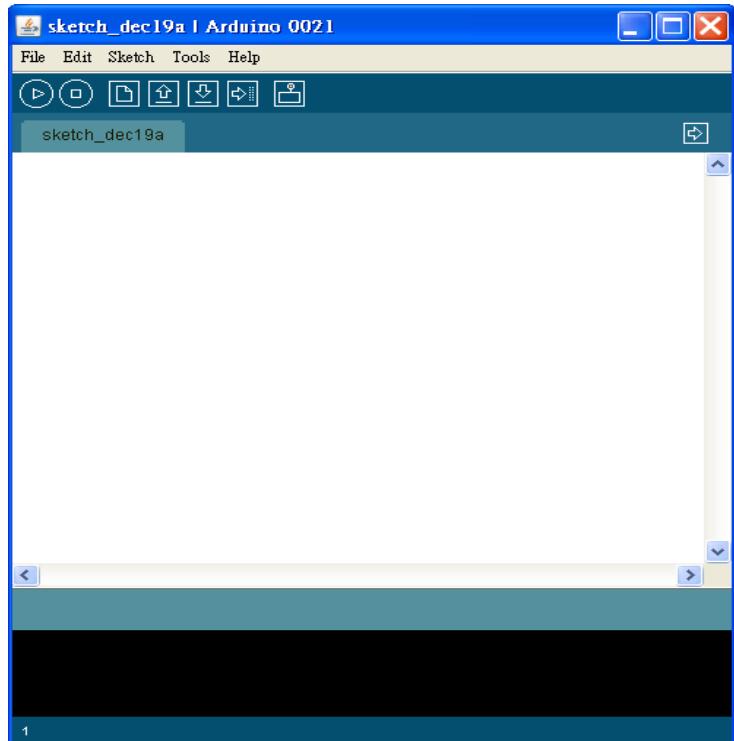
另外，有幾支腳也有特殊功能：

I2C	4 (SDA) 和 5 (SCL) 這兩支腳。透過 Wire library 可以提供 I2C 通訊。
AREF	類比輸入的參考電壓，搭配 analogReference() 函式一起使用。
Reset	當 Reset 腳為 LOW 時，微控制器會重置。

1.5. Arduino 軟體開發環境

Arduino 的軟體開發環境是開放源碼的 IDE (Open-source IDE)，可以在它的官網免費下載，它所用的程式語言語法類似於 C/C++，而且 Arduino IDE 是跨平台的，有 Windows, Macintosh OSX 和 Linux 的版本。Arduino IDE 的軟體介面如右圖。

Arduino 的程式叫作 Sketch，Sketch 意為腳本、素描、速寫或小品，因為 Arduino 程式都小小一個，不是很大，之所以取名為 Sketch，猜想大概有小品的意思。Arduino 程式主要由



`setup()` 和 `loop()` 這兩個函式組成：

因為 `setup()` 和 `loop()` 是每支 Arduino 程式都會用到的兩個函式，為了方便，Arduino IDE 已經幫大家準備好程式骨架了，我們在寫 Arduino 程式的時候，可以直接點選 `File > Examples > 1.Basics > BarMinimum` 這個範本檔，另存新檔後，然後再填寫 `setup()` 和 `loop()` 兩個函式的內容即可。

```
1 void setup() {  
2     // setup 函式只會跑一次  
3 }  
4  
5 void loop() {  
6     // loop 函式會不斷的執行  
7 }
```

1.6. Arduino 硬體版本

Arduino 控制板 (I/O Boards)有很多種版本，我們在使用的時候，可以依據自己的需求(例如用途、尺寸、容量、I/O 腳數量等)挑選適當的板子。底下條列幾款 Arduino 控制板，更多的資訊請上 Arduino 官網查詢。

Arduino Nano 3.0

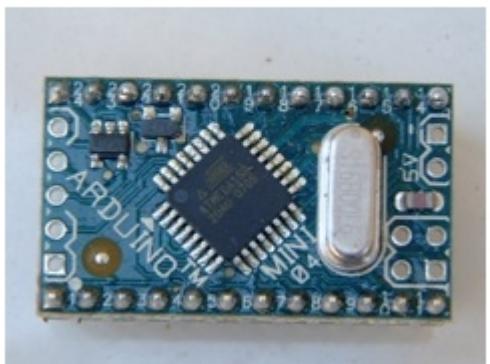
Arduino Uno_



Arduino Mega



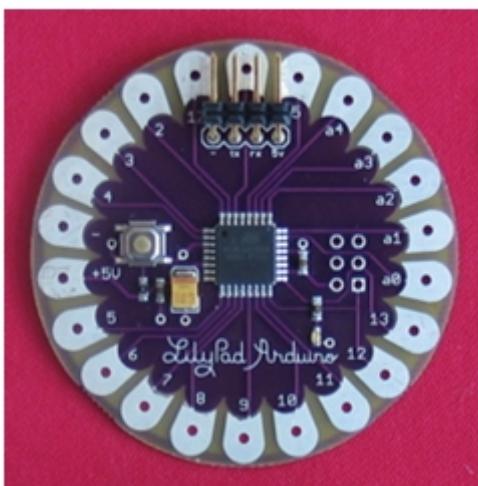
Arduino Mini



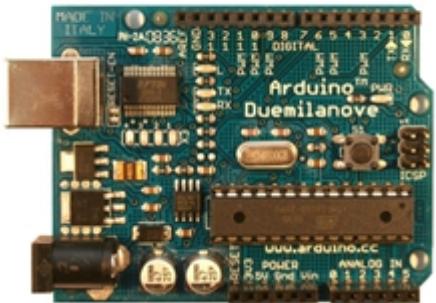
Arduino BT (BlueTooth)



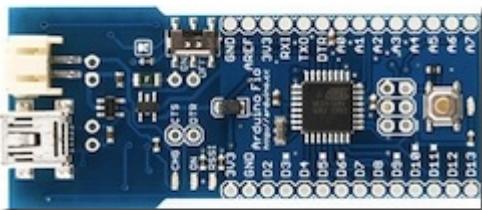
LilyPad Arduino 02



Arduino Duemilanove



Arduino Fio

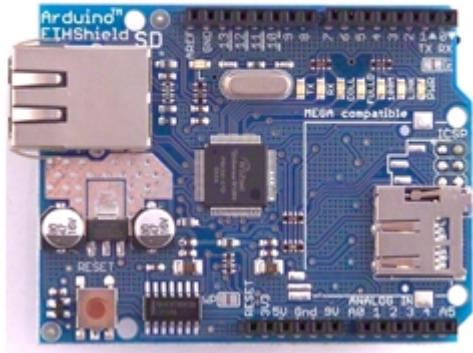


1.7. Arduino 擴充板 (Shields)

Shields 是擴充板，就像好像堆積木一樣，擴充板可以直接疊在 Arduino 控制板上，讓 Arduino 增加更的能力，例如控制搖桿、通訊、記憶卡、LCD 顯示、MP3 音樂播放等等。底下條列幾款擴充板，更多的資訊請上 [Arduino Shield List](#) 網站查詢。

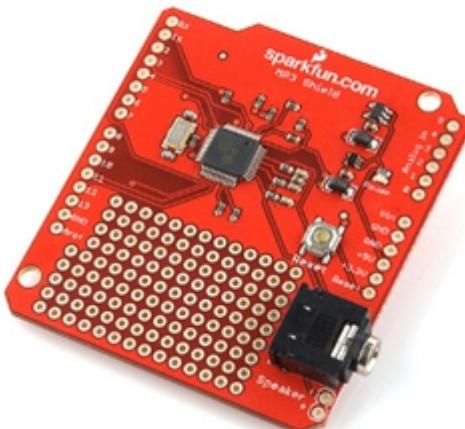
Ethernet Shield

提供 Ethernet 上網能力



MP3 Shield

提供 MP3 音樂解碼能力



Input Shield

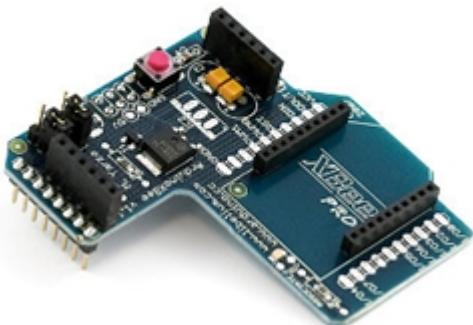
提供 Joystick 搖桿、按鍵及震動馬達能力



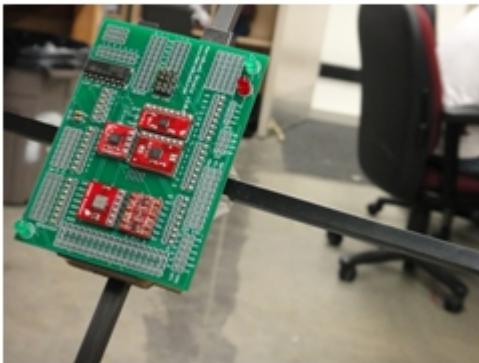
TouchShield
提供 OLED 觸控螢幕顯示能力



XBee Shield
提供 ZigBee 無線通訊能力



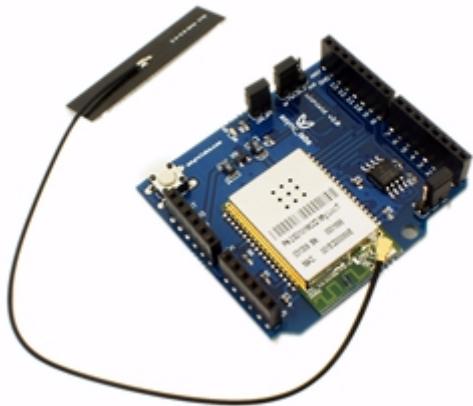
AeroQuad Shield
提供三軸陀螺儀與加速計的控制



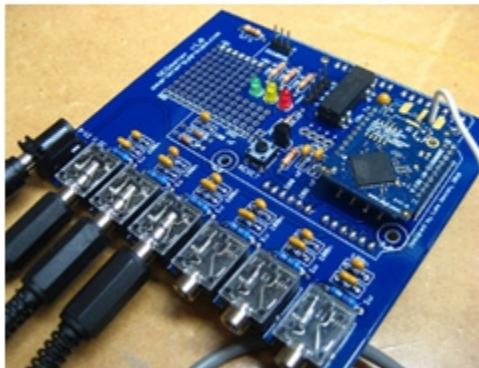
GPS Shield
提供 GPS 定位能力



WiShield
提供 Wi-Fi 上網能力



Smart Energy Groups SEGMeter
提供家庭與工業用電量測能力



(本文作者為馬萬圳，原為網誌上的文章，經授權後陳鍾誠編輯後成為此文。原文連結：
<http://coopermaa2nd.blogspot.tw/2010/12/arduino-arduino.html>)

2. JavaScript (1) – 簡介與基本語法 (作者：陳鍾誠)

本系列文章將採用淺入深的方式，逐步介紹 JavaScript 語言的特性，然後銜接 node.js 開發環境，讓讀者從 JavaScript 語言開始，逐步進入用 JavaScript + node.js 同時開發網站前後端的美麗世界。

2.1. 程式人對 JavaScript 常見的誤解

JavaScript 是目前在瀏覽器上唯一通用的程式語言，這種語言經常遭受到許多誤解，像是「JavaScript 就是 Java 語言的簡化版」、「JavaScript 語言很難用」、「JavaScript 語言設計很差勁」等等。

然而，這些誤解其實是我們不瞭解 JavaScript 所造成的。如果您用心去瞭解 JavaScript，您會發現這是一個「簡單、輕巧又優美」的語言，其原型導向的設計方式，用很簡單的概念達成了物件導向語言的功能，真的很適合做為瀏覽器上的共通語言。

JavaScript 雖然是一種物件導向語言，但是更精確的說，JavaScript 實際上是一種「原型導向」的語言，其中每個物件的 `_prototype` 欄位都可以指向他的原型，然後用來 `clone`(自體繁殖) 出新的物件，您可以使用 `function` 型態宣告一個物件，如此該物件就自動具有建構函數了，這種做法是非常簡單、奇特、但卻又彈性十足的方法。

雖然 JavaScript 語言有許多優點，但是也有不少缺點，例如有些語法的邏輯很奇怪，而且 JavaScript 的物件導向並非像 Java, C#, Ruby 等語言那樣傳統，而是採用原型導向的方式，這讓傳統物件導向的使用

者在學習 JavaScript 時會感到困難。不過一但您理解了原型導向的精神之後，就會發現這種方法真的有不少優點了。為了較深入的理解 JavaScript，讓我們先來看看 JavaScript 的歷史！

2.2. JavaScript 的歷史

在 1990 年 Web 開始成形，早期網際網路只有文字，沒有圖像、音效，且操作非常繁瑣。當時還是大學生的馬克·安德生（Marc Andreessen）被伊利諾伊大學的電腦應用中心聘為臨時工作人員，馬克·安德生提出設計一種簡單的瀏覽器程式想法，能方便的檢索網路資料，於是招聘了幾個程式設計師花了六週的時間開發。於是誕生了 Mosaic 瀏覽器。

Mosaic 的出現，算是點燃了後期網際網路熱潮的火種。後來在 1994 年 4 月，馬克·安德生和 Silicon Graphics 公司的創始人吉姆·克拉克（Jim Clark）在美國加州設立了一家公司，進一步改善瀏覽器技術，並且創造出了 Netscape 瀏覽器。

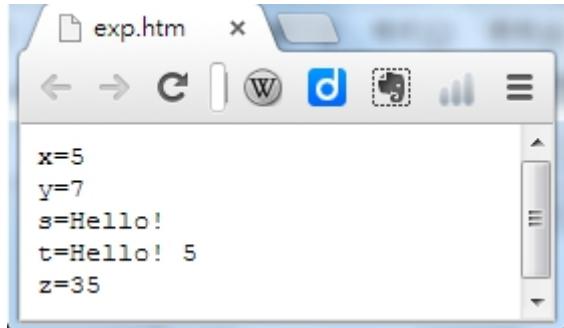
當時瀏覽器只能顯示文字與圖片，無法做出像功能表這樣需要即時互動的網頁，於是 Netscape 公司的 Brendan Eich 設計出了 LiveScript，這是一種動態、弱型別、基於原型的語言，用來寫一些讓網頁可以動起來的小程式。

由於 Netscape 在與昇陽（Sun）合作的關係，所以 Netscape 將 LiveScript 改名為 JavaScript，後來甲骨文（Oracle）公司買下了昇陽，所以 JavaScript 是甲骨文公司的註冊商標。為了避免侵犯商標權，於是 Ecma 國際（前身為歐洲電腦製造商協會）將其以 JavaScript 為基礎所制定的語言改稱為 ECMAScript。所以現在我們所說的 JavaScript，其真正意義其實應該是 ECMAScript。

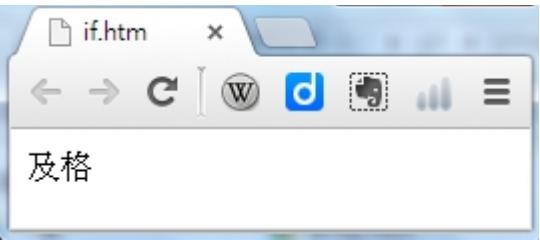
現在、JavaScript 已經成為一種非常重要的程式語言，JavaScript 除了用在瀏覽器之外，也被用在許多其他領域，像是您可以用 node.js 在 Server 端撰寫 JavaScript 的程式，也可以在 Unity3D 當中用 JavaScript 撰寫遊戲程式，更可以在 Tatinum 當中用 JavaScript 撰寫手機的 APP 然後放到 iPhone, iPad 或 Android 上面去執行。

2.3. JavaScript 的基本語法

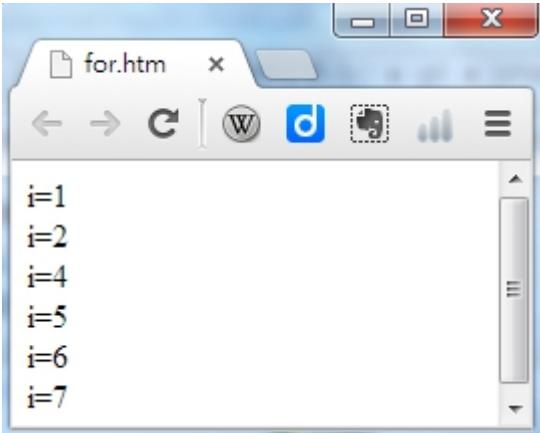
JavaScript 最常被嵌入到 HTML 網頁中，以便讓網頁顯示出一些特別的動態效果，因此含有 JavaScript 的網頁通常也被稱為動態網頁，以下是一個 HTML 內嵌 JavaScript 的簡單範例與顯示結果。

原始碼	執行結果
<pre><html> <body> <script type="text/javascript"> var x =5, y=7; var s = "Hello! " t = s + x; z = x * y; document.write("<pre>x="+x+"\ny="+y +"\\ns="+s+"\nt="+t+"\nzs="+z+"</pre>"); </script> </body> </html></pre>	 <p>The screenshot shows a web browser window titled 'exp.htm'. The page content displays the output of the JavaScript code, which includes variable assignments and their values: x=5, y=7, s=Hello!, t=Hello! 5, and z=35. The output is presented in a pre-formatted text block.</p>

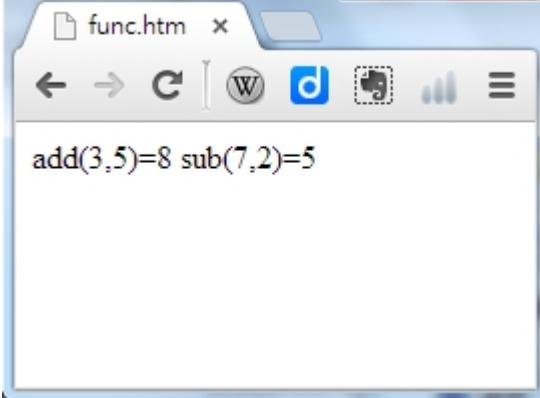
JavaScript 的 if 語法也很簡單，基本上與 C 、Java 、C# 等語言都相同，以下是一個簡單的範例。

原始碼	執行結果
<pre data-bbox="73 208 787 593"><html> <body> <script type="text/javascript"> var score = 61; if (score >= 60) document.write("及格"); else document.write("不及格"); </script> </body> </html></pre>	

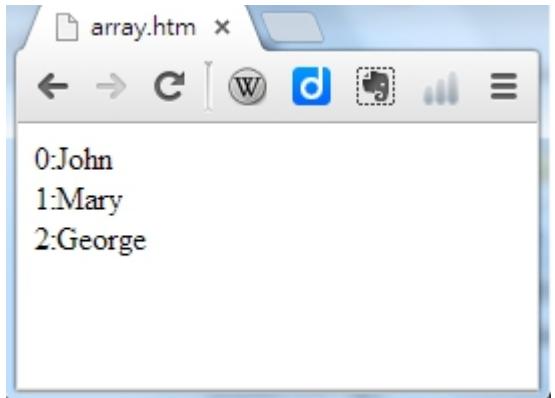
JavaScript 的迴圈語法也是繼承 C 語言的，以下是一個範例：

原始碼	執行結果
<pre><html> <body> <script type="text/javascript"> for (i=1;i<=10;i++) { if (i == 3) continue; if (i == 8) break; document.write("i="+i+"
"); } </script> </body> </html></pre>	 <p>i=1 i=2 i=4 i=5 i=6 i=7</p>

在 JavaScript 當中，函數的定義有兩種寫法，一種是正常的函數宣告，另一種是把函數當變數的指定方法。事實上，JavaScript 的函數也是一種基本型態，具有這種特點的語言通常稱為函數式語言。這種特性非常好用，因為在函數式語言當中，函數在也不是次等公民，而是一種基本型態，以下是一個 JavaScript 的函數宣告範例。

原始碼	執行結果
<pre><html> <body> <script type="text/javascript"> // 第一種寫法，將匿名函數指定給變數。 var add = function(a,b) { return a+b; } // 第二種寫法，直接宣告函數，該函式是一個函數物件 function sub(a,b) { return a-b; } document.write("add(3,5)="+add(3,5) +" sub(7,2)="+sub(7,2)); </script> </body> </html></pre>	

在 JavaScript 當中，您若要宣告一個陣列時，可以採用 `var a = new Array()` 的方法，建立一個陣列物件，然後用 `a[i]` 這樣的方法，存取陣列中的元素。

原始碼	執行結果
<pre><html> <body> <script type="text/javascript"> var x; var friends = new Array(); friends[0] = "John"; friends[1] = "Mary"; friends[2] = "George"; for (p in friends) { document.write(p + ":" + friends[p] + "
"); } </script> </body> </html></pre>	

以上我們已經介紹了 JavaScript 的基本語法，在下期當中，我們將介紹 JavaScript 語言中一個非常特別的地方，那就是「以原型為基礎的物件導向機制」，我們下期見！¹²

12 本文來自陳鍾誠網站電子書「JavaScript 程式設計」，原文網址為：<http://ccckmit.wikidot.com/js:syntax>

3. Web 應用程式開發(1) - Web 應用程式的演進與 Single Page Application (作者：李開文)

在 Web Service, Ajax, Web 2.0, REST 等 Web 應用與技術話題熱潮，帶動許多第二代的 Web 開發技術成長之後，這些話題也漸漸地消退。不過許多人可能不曾發現，其實這些技術名詞，是在慢慢地顯露一點：Web 應用程式逐漸從 Server Side 轉移到 Client Side，也就是瀏覽器身上。

本篇文章要從以往的 Server Side Web 應用程式，其開發方式與演進來介紹 Single Page Application(SPA) 與現今所有主流 Web 技術。

本篇文章也同步發佈於 <http://kiwi-grid.blogspot.tw/2007/09/single-page-application-web.html>

3.1. Web 應用程式的演進

3.1.1. 動態網頁

儘管 Web 並不是一個三言兩語能拿個版本號碼來解釋，但實際上 Web 技術確實有些明顯的分隔點。

最早期我們熟知的就是靜態網頁，這應該沒啥問題。儘管在 2000 前，php,asp 就開始流行，坊間的書上也都稱之為動態網頁。而我在此提及的動態網頁程式，實際上卻是從 php4 釋出的那一年開始(註釋*)。這邊要讓大家瞭解的分界點，其實是 php4 開始被許多商業公司所採用，而軟體的形式也更為套裝化，而不再像之前大家認定的「動態」網頁僅僅只是拿來完成一些簡單的區塊來與一般的靜態網頁整合。

在 2004 年的時候，Web Framework 的產生，創造了 Web 應用程式另一個新的高峰。而在這個時候也開始有一些 Rich Web Client 概念的雛形了。我將 Ruby On Rails 定為一個分界點是因為，他顛覆了傳統動態網頁還在使用設計方式，而改用 MVC。但要注意的是 Ruby On Rails 儘管整合了 Ajax 與進階 Javascript 函式庫，但還是沒有改變回傳完整或部分 HTML 的方式，意思便是 HTML 的產生始終在 Server Side。

3.1.2. Rich Internet Application

一直到現在，有相當多的 Web 應用程式，都還是維持使用 URL 來切換各種功能與畫面。而這些以「頁面」為主的程式，並不太需要控制 DOM，就不常遇到跨瀏覽器問題。然而自從 Firefox 逐漸也在市場佔有一席之地，Javascript 的應用普及之後，跨瀏覽器問題也接著發生。為了避開各種不同的瀏覽器所帶來的問題，各大企業都獨力發展自己可以嵌入在瀏覽器的應用程式。早期如 Java Applet 及微軟的 Active X，算是 Rich Internet Application(RIA)的開始，但效能方面還是差強人意。

直到 2004 年的時候，RIA 出現了兩種不同的實做方法：一種是承襲以往需要安裝額外 Runtime 或是在特定瀏覽器才能執行的方式，稱做 Sandbox；另一種是只採用 CSS,HTML，並以 Javascript 控制 HTML DOM 的 Dynamic HTML 方式，優點就是只需要瀏覽器就可以執行。而後者也延伸出利用 Offline

Database 或是 Ajax+Web Service 來傳送與儲存程式資料，並可以儲存成一個獨立頁面的 Web 應用程式，稱做 Single Page Application(SPA)。SPA 最典型的例子，就是 Gmail。Google 盡力克服了跨瀏覽器的問題，將 Javascript 發揮的淋漓盡致，讓大家驚嘆光靠純粹的 Web 技術竟能做到如此程度。

而我將 2005 定為 Sandbox RIA 真正開始的年代，也是因為 Adobe 併購 Macromedia，而有了較完整的開發環境與資源，並不是以往單純地嵌入 Flash。這個契機也促使微軟改變策略，比起效能較差的 Asp.Net，而拿 Silverlight 作為 Web 下一代主力軍。

RIA 或 SPA 都是學習歷程長，語言多又複雜的 Web 應用程式技術，也因此發展速度相當緩慢，但不可小看的是這些優點：

相較以往在 Server 上產生 HTML 並回傳至瀏覽器，任何畫面皆利用瀏覽器本身或附加的功能來產生。形同於借用了 Client Side CPU 的運算資源，減少 Server 成本。使用者感受到的互動性與回應速度皆有大幅的提升。

由於 Server 並不是每次都回傳複雜龐大的 HTML，而是利用 XML 或 JSON 傳輸資料的部分，使用的頻寬也相對變小。

Server Side 除了使用傳統 XML Web Service，更可以採用 REST，讓 Client 的應用程式可以更快速掌握資料的新增修改刪除(CRUD)並簡化呼叫的服務 URL。

能夠快速 Mashup 其他的 Web 應用程式資源，又能擁有高速的執行效能。

下表列出了 Web 技術的演進，要注意到後三種技術集合，其時間是並行的：

	靜態網頁	動態網頁程式	Web 應用程式	<i>Rich Internet Application with Sandbox</i>	<i>Single Page Application</i>
時期	2000 以前	2000(PHP4 釋出)~2004	2004(Ruby On Rails 釋出)以後	2005(macromedia 被 adobe 併購)以後	2004(Gmail 釋出 beta)以後
表現層	CSS	CSS,HTML,Javascript	CSS,HTML,Javascript	Flash, Sliverlight	CSS,HTML(DOM)
邏輯層	Javascript	Template 或自行撰寫	Web Framework	Action Script, C#	Javascript 或是撰寫 Web Service 的語言
資料層	HTML	Database(SQL)	Database(ORM)	Database(ORM)	Offline Database, Web Service
開發方式	網頁編輯程式	整合 HTML 及 Server Side 語言的編輯器	整合 Web Framework 的 IDE	整合 Sandbox 的 IDE	整合 Server Side 與 Client Side 語言的 IDE
運算資源	所有資料直接透過 Web Server 送出，除了硬碟讀取，幾	因為使用了 Server Side 語言來 Render 表現層，運算多半會消	因為使用了 Server Side 語言來 Render 表現層，	運算資源平均被分散在 Server 及 Client，但 Client 需要 Sandbox	運算資源平均被分散在 Server 及 Client

	乎不需要額外的運算	耗在 Server	運算多半會消耗在 Server	去執行，所以會消耗更多 CPU 資源	
資料格式	傳送完整的 HTML	傳送完整的 HTML	傳送部分或完整的 HTML	只需第一次傳送 HTML 及內嵌程式 (Flash 或 Silverlight)，其餘傳送 XML	只需第一次傳送 HTML 及 Javascript，其餘可傳送 XML 或 JSON
優點	簡單易學	學習同一種 Server Side 語言，搭配簡單的 HTML,CSS,JS 觀念便能夠有成果	整合 Ajax 或進階 Javascript 函式庫，REST 及 MVC。使得設計概念更為物件導向化	使用者介面反應快速，變化多且美觀。兼顧視窗程式的反應速度，且能部分相容傳統 HTML 應用。	完全相容傳統 HTML 應用，及任何可能的 Web 應用程式 Mashup。可以採用不同的傳輸方式，併和 REST 及瀏覽器快取來節省頻寬，使得整體反應相當快速。
缺點	無法讓使用者儲存任何應用程式資料，任何資料必須藉由人工設計	對於龐大的應用程式，便得花上大量的 Server 成本。程式反應速度受限於伺服器負載，需要叢集架構來彌補。	設計方式更為簡單快速，但相對於傳統的動態網頁程式付出更大的 Server 成本。	除了 CSS,HTML,JS 以外還需學習一兩種不同的語言才能進行開發。RIA 通常程式資料較大，在開始使用前必須等候一段下載時間。	除了 CSS,HTML,JS 以外還需學習一種撰寫 Web Service 的語言。需要相當熟悉 DOM 及 CSS，也需考慮瀏覽器的差異，開發起來相對地困難許多。

表現層的演進可以得知，不管是 RIA 利用 sandbox 或者是 SPA 利用 DHTML 作為表現層，相較起來傳統以文字 HTML 拼湊出畫面的作法已經無法符合使用者的需求。更動態，更彈性的作法才能讓使用者獲得操作感。

而在邏輯層上，演進到了 SPA 則是變得較為複雜。如果是使用 Web Service 作為 Server Side，除了必須撰寫該語言外，也還是得撰寫 Javascript 來控制畫面的呈現。而這也影響到開發方式，以往的編輯器多半著重一種主要的語言上，但現在的 Web IDE 多半都可以完整的處理所有瀏覽器的語言，及多種 Server Side 語言。例如 Aptana IDE 就是最好的例子。

資料層的演進就比較簡單，直到 RIA 的時期，還是相當依賴傳統的 Database。但 SPA 的時期，就可以採用 Offline Database，這裡指的就是 Google Gears。但如果要使用傳統的 Online Database，全部都尚未有 Javascript 的 Client，就必須透過 Web Service 來轉換資料。而在 SQL 到 ORM 的演進上，雖然使用物件導向的作法減緩執行速度，但相對降低開發難度，帶來更大的價值。

3.2. Single Page Application

3.2.1. 定義

請參考 <http://www.answers.com/topic/single-page-application?cat=technology>

Single Page Application 是一種 Web Application，完全地在瀏覽器上執行。也就是說，標準的 SPA 程式是不需要網路連線的。

像這些範例，你可以使用 firebug 來觀察連線的動作。

- TiddlyWiki <http://www.tiddlywiki.com/>：這個 wiki editor，除了整整一萬行的 javascript 以外，只有一個很簡單的起始頁。就算你拔了網路線，也只是不能夠儲存而已。但他要如何儲存編輯好的 wiki 內容呢？必須設定外部伺服器進行同步作業。
- Protopage <http://www.protopage.com/>：外型設計得很漂亮，操作感也很順暢。仔細觀察你也會發現在進行網路傳輸的時候都是傳遞 JSON。

而畢竟沒有網路連線的程式很難作為應用，所以我在 SPA 架構一節會提出兩種 SPA 程式的演進。

在此也必須提出一個概念，SPA 並非只是一個「不切換頁面」或是「URL 不變」的程式。而 SPA 所使用的觀念，就如同前言，已經是將瀏覽器當作 client 端了。

既然是 client 端，那勢必代表中間是必須傳輸資料，而並非 HTML；也因此 HTML 的產生也會發生在 client 端，而並非傳統由 server 產生 HTML 再由瀏覽器載入。

3.2.2. 所依賴的 Web 技術

Ajax 為 Asynchronous Javascript And XML，早期 Ajax 被用來傳輸單純的 HTML 或是 XML，並且利用 DOM 的 innerHTML 屬性來更新部分 HTML 內容。如今 Ajax 在 SPA 中被當作重要的傳輸媒介 (Transport)，無論範本資料到應用程式資料，都是利用 Ajax 在背景傳輸完後，再由 Javascript Template 來產生 HTML。

JSON 為 Javascript Simple Object Notation，在標準的 Javascript 語法中，以{}及[]這兩個語法，可以宣告物件與陣列，並可以使用 eval 函式將他轉換為 Javascript 物件。例如 object=eval("({a:'b'})")，此時 object 物件便有一個屬性 "a" 其值為 "b"。在 SPA 中，JSON 被運用來作為一種資料格式，藉此取代複雜的 XML，以節省頻寬。而傳輸的 JSON 資料又可以快速還原為 javascript 物件，又更節省程式執行的時間。

在 SPA 中，HTML DOM 是一個最重要的元素，尤其是 DIV 及 SPAN 等 Container 的操作更是。由於絕大多數的畫面都不進行任何換頁的動作，程式裡大部分都是在控制 DOM 及 Container。而對於 A (Anchor)而言，href 裡的 URL 也沒有太大意義，多半都是在 onclick 裡寫 javascript，或是用 Javascript 函式庫去 bind onclick 事件。由於直接呼叫瀏覽器提供的 DOM 函式庫功能，會遇到像 IE 一樣不符合 W3C 規格的問題。要選用一個合適的 Javascript 擴充函式庫，如 Prototype.js 或是 jQuery，如此才不會有太多跨瀏覽器問題。

CSS 在以往的 Web 應用程式中，多半都拿來當作畫面的修飾，布景主題或是顏色特校。但在 SPA 中，必須要熟悉 CSS 的 Dimentation(長寬控制)，Classification(顯示行為)，Positioning(定位)。在無法換頁的狀況下，只能靠著移動，隱藏，顯示這些方法來控制畫面的元素。如果要瞭解這些進階 CSS 的主題，都可以在 w3school 裡的教學找到。

Trimpah 是 Google 為了 SPA 而開發出來的一個函式庫集合，也可以說是 Rails 的 Javascript 版。如果要撰寫上述第一種 SPA，就必須利用到 Trimpah 的全部，而第二種只需要用到 Trimpah Javascript Template 即可。Javascript Template(JST)如同 PHP 的 Smarty 一樣，是標準的範本技術，只是採用的語言是 Javascript。為了撰寫 SPA，必須要好好地運用 JST。

快取的機制在 SPA 也相當的重要，為了達到讓使用者感受到程式的反應快速，就必須應用多方面的快取。

- View Cache：SPA 中的 View 就是指已經顯示出來的 HTML，很多常用，而不需要經常改變的 HTML，就可以將放在 Container(DIV 或 SPAN)裡。不用的時候就隱藏，需要的時候就顯示。而另一種方式是可以用 z-index 將選單或是清單的 Container 放在最下層，而要回到這個清單的時候就將蓋在其上的 container 隱藏。
- Template Cache：一般來說 JST 的範本資料只要讀取一次就可以，又因為這些只是字串，可以直接就存在 Hash 裡。
- Javascript Cache：在我提出的 SPA 實做中，有一個特性是將各個 JST 的「行為」程式碼分開，就如同 sap.net 將 aspx 與 cs 檔分開的作法一樣。而如果採用這個作法，不需要重複讀取的 javascript 就必須要快取。
- Data Cache：資料快取是最難的一部份，牽扯到了快取一致性的問題。而現在在 javascript 中並沒有對於 XML 或 JSON 的資料快取解決方案，未來如果能夠有這樣的函式庫，就能夠更提升整體的效率。

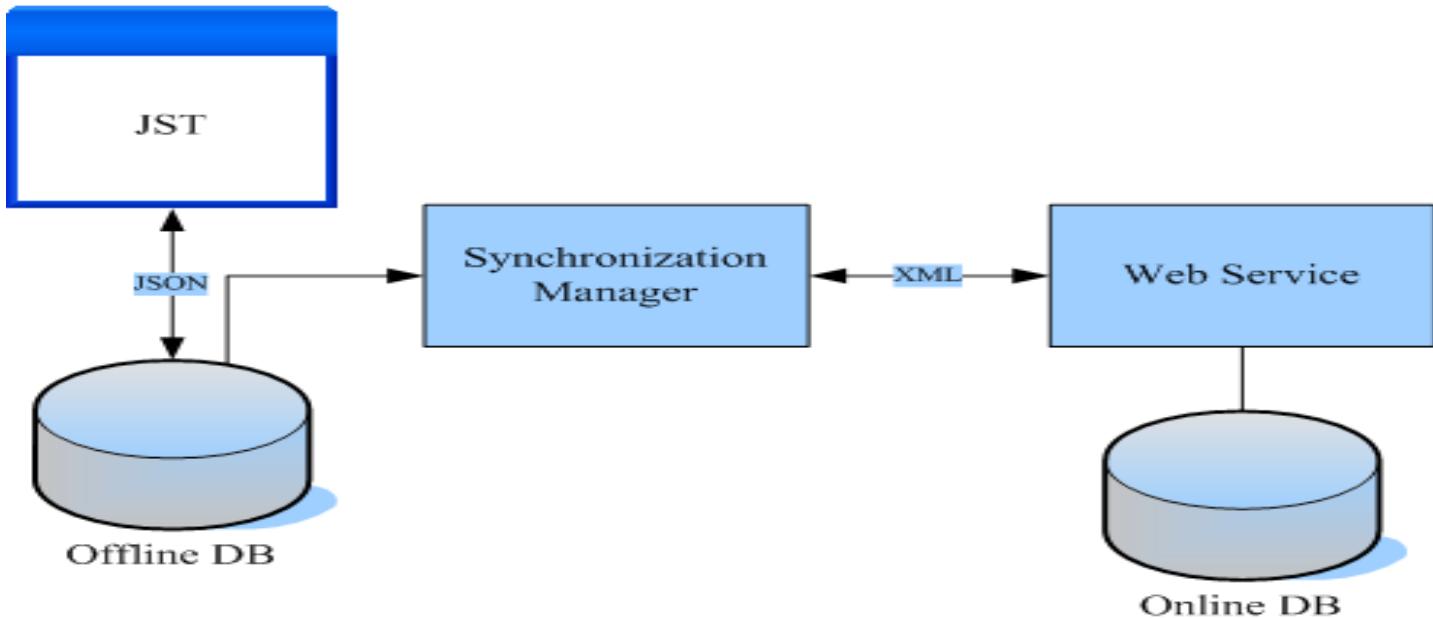
以上說明的都是 Client Side 所必須要使用到的技術，而 Server Side 的技術多半與 Service Design 息息相關。

REST 為 **Representational State Transfer**，他比較像是一種設計樣式(Design Pattern)，而不是 Web 技術。

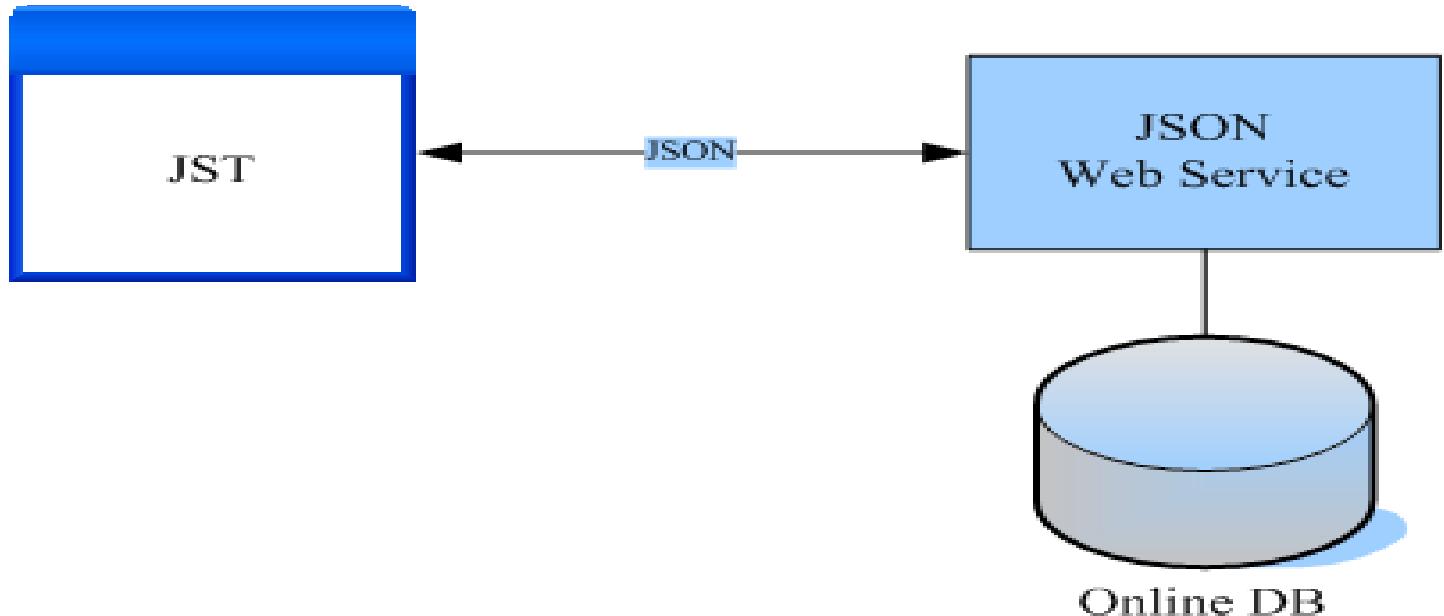
在以往 Web 應用程式的規劃中，URL 並不完全具有意義，傳輸的內容型態多半是 HTML，而 HTTP 的各種動作也並 未完全利用。在 SPA 中，由於需要在不同時間傳輸各種資料，如 HTML 範本，JS 範本，或是 XML 及 JSON 資料。此時 REST 的設計技巧就可以節省下很 多重複的命名，而讓程式碼整體更有意義。支援 REST 設計樣式的 Web Framework 如 Ruby On Rails，讓整體設計較為簡單。

3.2.3. SPA 的架構

SPA 就分類而言，算是 RIA 的一種，只是不採用任何的 sandbox 而已。典型的 SPA 是不需要任何的後端的 Web Service 或是 Offline Database，只需要一個 htm 檔或是一個網頁就能夠運作，如微軟的 HyperText Application(HTA)就是，但還是缺乏完善的資料儲存能力。



第一種 SPA 程式，如同著名的 Google Reader 離線版，具有一個離線資料庫與一個同步管理程式，在有網路連線的時候，會將資料同步回線上的資料庫。這個最大的優點就是完全利用了 Client 的 CPU 資源，使用者雖然看見的是網頁，但卻是在使用在本機執行的獨立應用程式，因此速度是相當流暢。比起一般的動態網頁，這樣子的使用體驗更能夠顛覆一般人對於「網頁」的看法，而逐漸瞭解何謂 Web 應用程式。另一個例子是使用 Trimpath 函式庫撰寫成的 NextAction，是一個多功能的 ToDo List。



另一種就是較簡單的 SPA，不具有離線瀏覽的能力，但是承襲了使用 javascript 的高效能。必須提及的是 Server Side 並不是採用 XML，而是可以快速轉換為 Javascript 物件的 JSON，來當作 Web Service。如此 Server Side 的語言只需要具備能夠快速將物件 serialize 成 JSON 的能力即可。

3.3. 結論

各位程式人讀者大家好，本篇文章是小弟在此雜誌上的連載第一篇。先感謝陳教授願意帶領大家來進行雜誌的編撰，沒有陳教授我想不可能會有這本數位雜誌。小弟不才，希望透過這樣的活動一方面讓大家間接幫助到需要被幫助的人，一方面多推廣小弟的文章及部落格，也歡迎大家寫信到 kiwi0530@gmail.com 來筆戰 XD，越多討戰文我越開心！我希望最終大家都能够透過討論來釐清事情的真相，這樣才能促進大家對軟體開發的瞭解與重視！

許多人都在說 Web 2.0 可能又是另一次的泡沫化，這個熱潮怎樣開始的，又怎樣消退的，也是相當明顯。網路上對於各種新技術名詞的炒作，將不同應用層次的技術，全部攬和在一起說明或稱做是最終解決方案，也模糊了使用者的眼睛。那麼，在這個時代，到底還有什麼可以相信，可以學習的？

唯一能夠做的就是重新審視這些技術，瞭解因果。就可以知道哪些作法是適合用在自己現在的專案，哪些是本質相同的，哪些是跨大其詞的。根本的觀念正確，就不需要擔心這些延伸的技術是否會有誤解或誤用。我相信下一個世代是 Web 及嵌入式系統的世代，換句話說 Web 及嵌入式系統工程師的需求量肯定是有增無減的。

在接下來的文章，我會為大家從兩個面像介紹 Web 開發，一個是使用純粹的 Server Side 語言，而這邊應該會介紹 Python 及 Javascript(node.js)的例子。另外一個就是使用 No-SQL 的資料儲存，搭配完全分離在不同主機上，並且利用 JSONP 來進行負載平衡通訊的 Javascript SPA 程式。在往後的日子，也會穿插介紹現在國外大多數人開發 Javascript Web 應用程式時所使用的設計技巧。

註釋*

在 2009 年 3 月的時候，對岸的一位作家周愛民老師有指點了我有關 php4 在這個時期的定位。周愛民老師當時的大作為「JavaScript 语言精髓与编程实践」，其中的第一章的各個第三小節，請各位參考。文章中也提及了那個年代的一些真實的狀況。

但我在這邊還是希望解釋清楚，確實用 php,asp 來定義「動態網頁」(甚至我還說 php4)並不是最正確的說法。但實際上我必須說明這是當時臺灣的書籍普遍描述的名詞與認知，我想台灣的早期的網頁程式設計師們應該對我的描述很有感受吧！

實際上真正的「動態網頁」應該指的是 Dynamic HTML (DHTML)，表示使用 javascript 去控制 HTML DOM 進而直接在網頁上產生動態效果的作法，而並非是 server side 的任何技術如 php,asp,jsp 等。但隨著時間演進，DHTML 這個字眼也逐漸被人淡忘，變成是 Ajax。當時的 DHTML 也已經有人嘗試著使用 javascript 與 server 通訊，那就是與現在的 Ajax 沒啥不同。至於說 php,asp 等的為啥會被說成動態網頁，也只是反映出那個時候台灣普遍對網頁設計的認知而已。

我很感謝周老師對我的指點，特此說明當時寫文章的想法。

4. 小談模型存取 – 以 Factory Pattern 實現 (作者： pojungwu)

4.1. 前言

在 3D 程式設計中，讀取 3D 模型是一個非常基本的工作。市面上常見的遊戲引擎都可讀取各種常見格式，如 obj、3ds、ply、md2 等。但架構上是如何實現的呢？如果我們採用 OpenGL 自己寫讀取器的時候又要如何寫呢？本文將以設計模式中的 Factory Pattern 工廠模式為大家做一個簡單的說明。

4.2. 相關背景

承如前文所述，本文將提及 Factory Pattern，那首先我們就來談談甚麼是設計模式吧！

它是對軟體設計中普遍存在（反覆出現）的各種問題，所提出的解決方案。

--節錄自維基百科

簡單的說，設計模式是一種解決問題的方式。牛頓曾說過他看的比別人遠是因為他站在巨人的肩膀上。人類使用電腦來解決問題（這裡通常指數學問題）已經非常多年，而成功解決這些問題的方法我們姑且先稱為演算法，可想而知同一個問題有非常多的演算法可以解決，使用那個方法是最快的，最有效的這是資料結構與演算法所探討的問題，而如何用物件導向的概念設計這些方法（或說怎麼寫比較靈

活) 則是設計模式所關注的重點。換言之，有許多既定的行為可以用既定的模式實現。所以研究各式設計模式則可以使得程式人站在巨人(前人)的肩膀上更快更有效的設計出符合物件導向原則的程式。

常見的設計模式如 Gof 所著 **DESIGN PATTERNS: ELEMENTS OF REUSABLE OBJECT-ORIENTED SOFTWARE** 筆者就不再闡述，當然，本文所用的工廠模式就是其中一種。工廠模式的焦點在於操作行為，其將資料的存取視為一個元件，而我們可以藉由工廠來“生產”各式元件，至於對這些元件所做的操作則可視為流水線，是一致的。舉例而言，假設我們在設計一個編輯器，我們先不考慮文件格式，而先將基本的操作如讀檔，存檔，關閉定義完畢，在討論各種格式是如何被操作的，而這裡所謂的各種格式則由工廠生產之，如此一來，我們就可以任意添加或移除各種各樣的格式只要我們通知工廠即可，而存檔關閉等操作完全不用重新寫，這就是物件導向所提及的代碼重新利用(**Code Reused**)。

4.3. 程式解析

以下是一個簡單的工廠模式範例，如果您閱讀上覺得吃力，那煩請閱讀物件導向相關教學，尤其是繼承等問題。

```
#include <cstdlib>
#include <iostream>
#include <string>
using namespace std;

class Operator{
public:
    virtual int getResult()=0;
```

```
    void setNum(int A,int B){Operand1=A;Operand2=B; }

protected:
    int Operand1;
    int Operand2;
};

class Operator_Add: public Operator{
public:
    int getResult(){
        return Operand1+Operand2;
    }
};

class Operator_Sub: public Operator{
public:
    int getResult(){
        return Operand1-Operand2;
    }
};

class OperatorFactory {
public:
    Operator* creatOperator(string type){
        if(type=="+") return new Operator_Add();
        if(type=="-") return new Operator_Sub();
        return nullptr;
    }
}
```

```
};

int main() {
    Operator* _operator=nullptr;
    OperatorFactory *factory= new OperatorFactory();
    _operator=factory->creatOperator("+");
    _operator->setNum(5,6);
    cout<<_operator->getResult();
    system("PAUSE");
}
```

上述程式為 C/C++ 版，如果您需要參考 JAVA 或 C#可以參考網路文章或書籍，在此筆者推薦可閱讀程杰所著《大話設計模式》。

本程式為一個 ALU（算數邏輯單元）的雛形，我們都知道一個簡易的運算包括一個運算子（operator），以及其所作用的對象即運算元（operand），而運算子所作用的結果則是我們所熟悉的運算結果（result）。以下就各類別做說明：

1. `Operator` 類別為基礎類別（base）作為所有運算子的範本（或說是各種命令的範本），其中包括兩個運算元 Operand1，Operand2 作為其成員變數，並有一個名為 setNum() 的函式用以設定這兩個運算元，而整個運算則由 getResult() 函式負責，我們藉由呼叫該函式即可得到其結果，值得說明的是，這個函式被宣告為虛擬函式而使得 Operator 類別成為抽象類別，這是因為 Operator 類別的衍生類別必須以不同的方式（加法、減法等）來計算結果。

2. `Operator_Add` 類別是一個簡易的加法運算子類別，當然，這個類別繼承自 `Operator`，並依抽象類別的設計原則必須對 `getResult()` 進行實作，將兩個運算元以加法運算子的方式運算並且傳出。
3. `Operator_Sub` 類別為一個減法運算子類別。與 `Operator_Add` 類別設計方式一樣，就不多做說明了。
4. `OperatorFactory` 類別故名思意為所有運算子的工廠。工廠裡面將負責「**生產**」各種元件（運算子類別），至於要生產何種元件則以字串辨識要用哪個運算子，並回傳對應的運算類別。函式 `creatOperator()` 就是負責生產對應的運算類別，並回傳一個 `Operation` 指標形態，而這正是基礎類別。所以我們只需要接收回傳指標，並且呼叫其中的 `getResult()` 函式就可以得到對應的運算結果。具體使用方式則可以參考第 33-40 行的主程式宣告，筆者就不再闡述。

在上述的這個範例中，我們可以發現如果我們想要添加新的運算子（如乘法，除法甚至開平方根）則只需要寫一個新的運算子類別繼承 `Operator`，並將 `getResult()` 函式重新實作出來，並告知工廠 (`OperatorFactory` 類別) 即可。如此一來，程式就變得非常靈活。

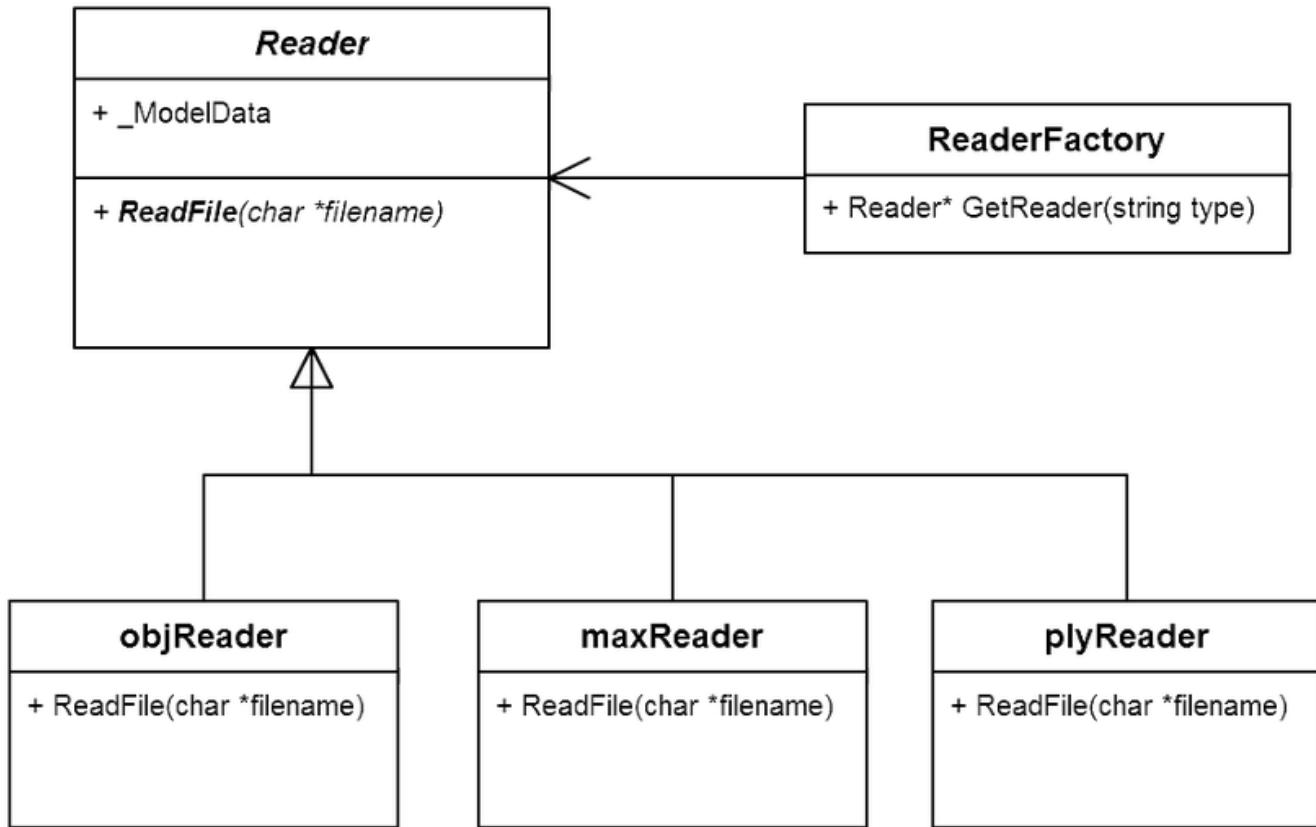


插圖 17: 模型讀取器 (Reader) 設計示意圖

現在我們來討論一下如何將這種概念運用到模型讀取中，如果您對於點跟面的概念不是很了解，則可能要請您參考一下電腦圖學。就筆者定義的一個模型而言，最基本的資訊包括 屬性（擁有多少個點，擁有多少個面等）、資料（實際點的資訊及面的索引）、讀取器（即 Reader）、渲染方式等。本篇所關注的重點為讀取器，具體如何使用 OpenGL 渲染模型，如何開檔讀檔等筆者就不再多做說明，如果有興趣讀者可以下載對應的程式碼閱讀。如圖一所示，我們設計了一個 Reader 的抽象類別，其中要求所衍生的類別必須對 ReadFile() 函式實作。而工廠（ReaderFactory）則負責辨識且產生對應的讀取器。整體的程式如後所示：

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

class Reader {
public:
    virtual void ReadFile(char *filename) = 0;
    int modelData;
};

class objReader: public Reader{
public:
    void ReadFile(char *filename) {
```

```

        cout<<"using objReader" << endl;
        cout<<"Trying to read "<<filename << endl;
    }
};

class plyReader: public Reader{
public:
    void ReadFile(char *filename) {
        cout<<"using plyReader" << endl;
        cout<<"Trying to read "<<filename << endl;
    }
};

class ReaderFactory {
public:
    Reader* GetReader(string type){
        if ( type == "obj" ) return new objReader();
        if ( type == "ply" ) return new plyReader();
        return NULL;
    }
};

class myModel{
public:
    myModel() {
        _readerFactory= new ReaderFactory();

```

```

        _reader=_readerFactory->GetReader("obj");
    }
    void changeReader(string type){
        _reader=_readerFactory->GetReader(type);
    }
    void ReadModel(char* filename){
        _reader->ReadFile(filename);
    }
private:
    ReaderFactory *_readerFactory;
    Reader* _reader;
};

int main(){
    myModel test;
    //test.changeReader("ply");
    test.ReadModel("test.obj");
    system("PAUSE");
}

```

以下就上述程式各類別做討論：

1. `Reader`類別，這就是讀取器（Reader）的雛型（基礎類別），在這個基礎類別裡面我們定義了虛擬

的函數 `ReadFile()` 用以模擬讀取各種模型格式（注意，每種格式讀取實際操作都不一樣，但是對系統來說，都是 `ReadFile` 這個動作。）。但礙於版面問題，具體如何讀取 `obj` 以及 `ply` 或 `3dmax` 檔案筆者在此就略過了。有興趣的朋友可以參考本文的，內以 `obj` 作為讀取範例。

2. `objReader` 與 `plyReader` 類別，這兩個類別都繼承自 `Reader` 類別，在此我們並無對格式做詳細的操作，而是做一個簡單的 列印說明。主要是用於我們要驗證這個方法的可行性。

3. `myModel` 類別，在這裡類別裡，擁有讀取器(Reader)以及工廠(ReaderFactory)，並且他們都是私有的變數，避免別人直接存取變數，我們可藉由 `changeReader()` 這個函數來重新設定讀取器，而範例中 `ReadModel()` 則為讀取模型的方式，當然，這裡僅是範例，真正的讀取模型當還包括對檢查檔案，讀取標頭，掃描各種資訊等繁複的工作，筆者就不在這裡詳談了。筆者將這個類別的讀取器預設為 `obj` 格式（預設在建構子的時候就將讀取器切換到 `obj` 模式），當然，我們也可以用 `test.changeReader("ply");` 這樣的指令將讀取器切換成 `plyReader`。而未來如果有新的模型格式，只需要設計新的衍生類別即可。這使得未來的相容性變高。

4.4. 後記

藉由這樣的範例，我們可以比較具體的了解如何使用工廠模式來設計模型的讀取器，同時也發現設計模式的魅力所在，這也是筆者想要表達的重點。所以才有人說 「**物件導向是一種概念，設計模式則是一種思考的藝術**」。再此也要特別感謝 陳鍾誠 老師的成立這份雜誌所做的貢獻。本篇為筆者第一次嘗試投稿，如有闡述不明或錯誤之處還請來信指教。

有關本文所提及的內容，如果對該內容感興趣想知道更詳細的程式（如 obj 讀取過程、OpenGL 渲染）筆者所亦由創立的 glModel，您可以擇一方式下載。

Enjoy !!! Open Source , Yeah! ← 筆者 pojungwu 的心聲

本文作者為 pojungwu，email : wupojung@gmail.com

4.5. 程式碼下載

1. 本文程式碼的 GitHub 下載網址 - <https://github.com/wupojung/glModel>
2. 本文程式碼的 Google Code 下載網址 - <http://code.google.com/p/glmodel/>

5. UART 程式設計 (作者：趙琨堯)

5.1. 簡介

在這裡介紹UART的程式，對於接觸MCU工作的工程師來說UART相當的實用，在DEBUG上或是與其他裝置溝通以及一些搭配IIC進行的程式工具都可以透過UART，並且這協定在與PC溝通的程式又很成熟，可以透過VISUAL C++、BASIC或是其他PC程式設計來做工具來與主機連接使用，本程式是單介紹UART控制程式部份，以及搭配IIC操作控制EEPROM讀寫的部份，這兩種協定因為是業界成熟並且資源多，而範例中的程式提供一個使用者輸入控制的平台出來控制主機。

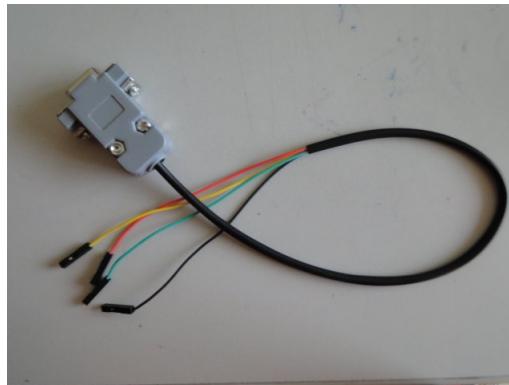
5.2. 運作原理

5.2.1. 平台硬體連接：

透過MCU的UART連接PORT與PC端的COM透過RS232 TO TTL轉換線連接，PC端的TX與RX到主機上的RX與TX，就只有這兩條線而已，不要亂接到CTL或是RTL等其他線，GND的地線也是一定要連接的，VCC不用接沒關係。

首先準備一條RS232連接線，如果你電腦沒有COM PORT的話也可以使用USB TO RS232來轉換，另外要再準備一個能將RS232訊號轉成TTL的線或是轉換器如下，如果利用Y拍或是露天的話，以下這

兩組線材都不貴：都 60 元不包含運費¹³，燦坤要買一條\$600...



5.2.2. 終端機設定的部份：

開啟一新終端機給自訂檔名，以及使用連線選擇到 COM?，看你使用的 COM 端口選擇：

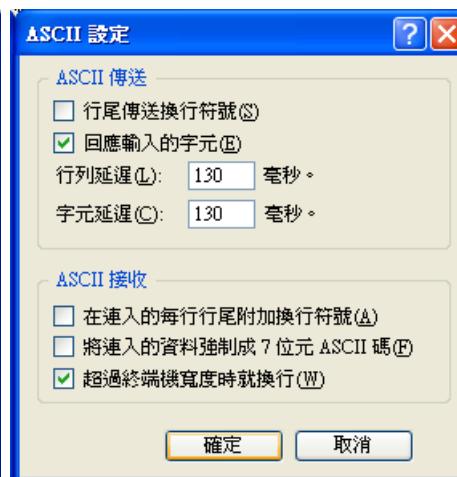
13 參考線材：<http://goods.ruten.com.tw/item/show?21109149093683> , <http://goods.ruten.com.tw/item/show?21210243632703>



然後資料傳輸位元選設定好的 BAUD RATE 速率，這與程式中要搭配，選完後點確定。

之後再把設定打開，點選到設定那一頁，見下圖，再點選 ASCII 設定，右下的按鈕，開啟後將回應輸入的字元勾選，並且輸入延遲時間，這裡建議先從 300 毫秒開始，實際執行時再看畫面會不會卡字或破損再調低，最低可以調到 0





5.2.3. UART 的原理

UART 傳輸時序：



以 38400 的 BAUDRATE 的速率收發，那一個資料就是 1/38400 的時序，所以 8BIT 加頭尾段的時序，差不多會花到 11 個資料時序約是 0.286mSEC，因此 RX_INT 將會在每 0.286mSEC 觸發一次，在接收之間的時序就是處理 RS_buf 的接受資料部份，解析指令與放置命令都要在這時間內處理完成，在 ASM 的處理上 0.286mSEC 是 286 行 1uSEC 的指令，但是 PIC 的 C18 可不是這樣的... LITE 版本是免費的，轉換間會損失時序，一行程式變成 2-4 個機械周期，甚至 WHILE() 或是 FOR(;;) 更會一下吃掉 39 個左右的機械周期，而實際轉成的 ASM 檔呢！！一看裡面竟然被塞了一堆的 NOP();

天阿！！真想給 MICRO CHIP 翻桌了... 級錢版 CODE 就變小，時序效能也變高...
所以這部份還是要注意一下，才不會出了怪怪問題！

5.2.4. 程式導讀 — 主程式：

接收後到處理指令的程式部份：

```
if((BM_STATUS_IND == 0) && (rs_ready == 1))
{
    RX_IE_ISR = 0;           // Close UART Rx interrupt
    Monitor();
    RX_IE_ISR = 1;           // Open UART Rx interrupt
    rs_ready = 0;
}
```

在 monitor(); 中所處理的就是把 RS_buf 中的指令與我們定的命令透過比對，然後去執行動作。

5.2.5. 程式導讀 — 處理指令函式(主程式)

```
void Monitor(void)
{
    char verfity, MACHINE_STD;
    char new_pwd_num;
    char show_num1, show_num2;
    char NUM_BIN, SHOW_BIN;

    new_pwd_num = parse_parse_token();

#ifdef DBG_ShowRs
    show_num1 = PWD_NUM+0x30;
    show_num2 = new_pwd_num+0x30;

    func_DBG_ShowRs(show_num1, show_num2);
#endif

#ifdef DBG_UART_ERR
    func_DBG_UART_ERR();
#endif
```

5.2.6. 程式導讀 — 處理測試程式函式部份

```
// ===== Test Function Start =====
```

```

#define DBG_TEST_FUNC
else if((strncpypgm2ram(str1, (const far rom char *) "EEP", 3)==0) ||
(strncpypgm2ram(str1, (const far rom char *) "eep", 3)==0)) {
    if((strncpypgm2ram(str2, (const far rom char *) "WR", 2)==0) ||
(strncpypgm2ram(str2, (const far rom char *) "wr", 2)==0)) {
        EEByteWrite(DEF_EEPROM_DEV, EEPROM_VERFINE, str3[0]);
        BM_RX_IND = 0;
        Delay1mSec(T_uart_rs_ind);
        putrsUSART((const far rom char *) "EEPROM WRITE = ");
        Delay1mSec(T_uart_rs_end);
        WriteUSART((char)str3[0]);
        Delay1mSec(T_uart_rs_end);
        BM_RX_IND = 1;
    }

    else if((strncpypgm2ram(str2, (const far rom char *) "RD", 2)==0) ||
(strncpypgm2ram(str2, (const far rom char *) "rd", 2)==0)) {

        NUM_BIN = (char)Asc2Bin(str3);
        SHOW_BIN = (char)EERandomRead(DEF_EEPROM_DEV, NUM_BIN);

        BM_RX_IND = 0;
        Delay1mSec(T_uart_rs_ind);
        putrsUSART((const far rom char *) "EEPROM READ = ");
    }
}

```

```

Delay1mSec(T_uart_rs_end);
WriteUSART((char)SHOW_BIN);
Delay1mSec(T_uart_rs_end);
BM_RX_IND = 1;

}

else if((strncpypgm2ram(str2, (const far rom char *) "SAVE", 4)==0) ||
(strncpypgm2ram(str2, (const far rom char *) "save", 4)==0)) {
    Save_Password(str3,new_pwd_num);
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const far rom char *) "SAVE OK!");
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
}

else if((strncpypgm2ram(str2, (const far rom char *) "LOAD", 4)==0) ||
(strncpypgm2ram(str2, (const far rom char *) "load", 4)==0)) {
    Load_Password();
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const far rom char *) "LOAD OK!");
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
}

```

```

    else if((strncpypgm2ram(str2, (const far rom char *) "LDF", 3)==0) ||
(strncpypgm2ram(str2, (const far rom char *) "ldf", 3)==0)) {
    Load_Default();
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const far rom char *) "Load_Default OK!");
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
}
else if((strncpypgm2ram(str2, (const far rom char *) "SHOW", 4)==0) ||
(strncpypgm2ram(str2, (const far rom char *) "show", 4)==0)) {
    strncpypgm2ram(EEP_PWD, (const far rom char *) " ", STR_THREE);
    Delay1mSec(T_CLR_STR1);
    strncpypgm2ram(EEP_IMM_PWD, (const far rom char *) " ",
", STR_THREE);
    Delay1mSec(T_CLR_STR1);

    Load_Password();

    show_num1 = PWD_NUM + 0x30;
    show_num2 = PWD_IMM_NUM + 0x30;

    verfity = (char)EERandomRead(DEF_EEPROM_DEV, EEP_VERFINE);
    Delay1mSec(T_I2C_TWC);
}

```

```

    MACHINE_STD =
(char) EERandomRead(DEF_EEPROM_DEV,EEP_SAT_MACHINE_STD);
    Delay1mSec(T_I2C_TWC);

BM_RX_IND = 0;
Delay1mSec(T_uart_rs_ind);
WriteUSART((char)verfity);
Delay1mSec(T_uart_rs_end);
putsUSART(RS_buf);
Delay1mSec(T_uart_rs_end);
putsUSART(EEP_PWD);
Delay1mSec(T_uart_rs_end);
putsUSART(EEP_IMM_PWD);
Delay1mSec(T_uart_rs_end);
WriteUSART((char)show_num1);
Delay1mSec(T_uart_rs_end);
WriteUSART((char)show_num2);
Delay1mSec(T_uart_rs_end);
WriteUSART((char)MACHINE_STD);
Delay1mSec(T_uart_rs_end);
BM_RX_IND = 1;
}
#endif

```

```
// ===== Test Function End =====
```

5.2.7. 程式導讀 — 處理 HELP 函式

```
// ===== HELP START =====
else if((strncpypgm2ram(str1, (const far rom char *)"help", 4)==0) ||
(strncpypgm2ram(str1, (const far rom char *)"?", 1)==0)) {
    MonHelp();
}
else if((strncpypgm2ram(str1, (const far rom char *)"FWVER", 5)==0) ||
(strncpypgm2ram(str1, (const far rom char *)"fwver", 5)==0)) {
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const MEM_MODEL rom char *)FW_VER_STR);
    Delay1mSec(T_uart_rs_end);
    putrsUSART((const MEM_MODEL rom char *)FW_VER_ID);
    Delay1mSec(T_uart_rs_end);
    putrsUSART((const MEM_MODEL rom char *)FW_Version);
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
}
else if((strncpypgm2ram(str1, (const far rom char *)"RESET", 5)==0) ||
(strncpypgm2ram(str1, (const far rom char *)"reset", 5)==0)) {
    BM_RX_IND = 0;
```

```

Delay1mSec(T_uart_rs_ind);
putrsUSART((const MEM_MODEL rom char *)"Reset");
Delay1mSec(T_uart_rs_end);
BM_RX_IND = 1;
Reset();
}
// ===== HELP END =====

```

5.2.8. 程式導讀 — 處理錯誤指令函式

```

// ===== Command Err =====
else {
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const MEM_MODEL rom char *)"COMMAND ERROR");
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
    Reset();
}
// ===== End =====

func_clr_str();
}

```

程式會把 str1、str2、str3 與我們定的指令與參數去做比對，然後對應到程式動作執行。

5.2.9. 程式導讀 — 主程式說明

這程式的完整動作就是透過連接 UART PIN 接到電腦的 COM 上，TX 接 RX，RX 接 TX，然後在終端機，輸入你所設定的指令，

如：#HELP# + [ENTER]

就會出現對應的動作，如果你把你想要的功能寫進去，在寫 FUNCTION 進去，就可以像是 DOS 那樣輸入指令執行動作滿好玩的，我後來都這樣寫一段來做 MCU 的 DEBUG 動作。

5.2.10. 副程式函式 — HELP 說明函式

```
void MonHelp()
{
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const far rom char *) "=====Monitor Help =====");
    Delay1mSec(T_uart_rs_end);
    putrsUSART((const far rom char *) "===== Monitor Help =====");
    Delay1mSec(T_uart_rs_end);
    putrsUSART((const far rom char *) "=====Monitor Help =====");
    Delay1mSec(T_uart_rs_end);
    putrsUSART((const far rom char *) "Type (#help# or #?#)+end show HELP");
    Delay1mSec(T_uart_rs_end);
    putrsUSART((const far rom char *) "#EEP*WR*Data# Chang EEPROM
data");
    Delay1mSec(T_uart_rs_end);
```

```

putrsUSART((const far rom char *) "#eep*rd*addr# Show EEPROM Addr");
Delay1mSec(T_uart_rs_end);
putrsUSART((const far rom char *) "#eep*save*PWD# Save EERPOM PWD");
Delay1mSec(T_uart_rs_end);
putrsUSART((const far rom char *) "#eep*load# Load EERPOM PWD");
Delay1mSec(T_uart_rs_end);
putrsUSART((const far rom char *) "#eep*ldf# Default EERPOM Value");
Delay1mSec(T_uart_rs_end);
putrsUSART((const far rom char *) "#eep*show# Show EERPOM Value");
Delay1mSec(T_uart_rs_end);
putrsUSART((const far rom char *) "#fwver# Show F/W Version");
Delay1mSec(T_uart_rs_end);
putrsUSART((const far rom char *) "#reset# Reset Machine");
Delay1mSec(T_uart_rs_end);
putrsUSART((const far rom char *) "Prompt:\r>");
Delay1mSec(T_uart_rs_end);
BM_RX_IND = 1;
}

```

這部份可以放一些說明指令與動作的文字，我放這東西是因為會怕忘記，按一個"?"就可以看到說明。

為什麼我在傳送 TX 之間要放 Delay1mSec(T_uart_rs_end); 和 Delay1mSec(T_uart_rs_ind);

這是因為原本設計是透過一個藍芽模組傳到裝置上，所以我要等待一些時間讓模組調整好，而我打程式喜歡先把全部功能都打完，之後的時間在慢慢去調整 T_uart_rs_ind 、T_uart_rs_end 的時間差距，並且在給雜訊狀況下找到正確的值！！

5.2.11. 副程式函式 — ASCII 轉 BIN 碼函式

```
BYTE AsclBin(char asc)
{
    if(asc>='0' && asc <='9') return (asc - '0');
    if(asc>='a' && asc <='f') return (asc - 'a' + 0x0a);
    if(asc>='A' && asc <='F') return (asc - 'A' + 0x0a);
}

BYTE Asc2Bin(char *s)
{
    WORD bin;

    bin = 0;
    while(*s != '\0' && *s != ' ' && *s != '#') {
        bin = bin<<4;
        bin = bin + AsclBin(*s);
        s++;
    }
    return bin;
}
```

因為 UART 在傳送時是透過字串的方式 STR，所以放個將字串轉成 BIN 的程式對之後處理也滿不錯。

5.2.12. 副程式函式 — 清除顯示的函式

這是針對這調整 UART 顯示時會出現的錯誤做的，這個部份是將輸入的字串在終端機重覆顯示一次出來看的。

```

void func_clr_str(void)
{
    strncpypgm2ram(str1, (const far rom char *)"           ", STR_THREE);
    Delay1mSec(T_CLR_STR1);
    strncpypgm2ram(str2, (const far rom char *)"           ", STR_THREE);
    Delay1mSec(T_CLR_STR2);
    strncpypgm2ram(str3, (const far rom char *)"           ", STR_THREE);
    Delay1mSec(T_CLR_STR3);

    strncpypgm2ram(RS_buf, (const far rom char *)"           ", STR_MAX);
    Delay1mSec(T_CLR_RS_BUF);
}

```

5.2.13. 副程式函式 — 顯示輸入資料函式(除錯資訊)

```

#ifndef DBG_ShowRs
void func_DBG_ShowRs(char num1, char num2)
{
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putsUSART(RS_buf);
    Delay1mSec(T_uart_rs_end);
    putsUSART(str1);
    Delay1mSec(T_uart_rs_end);
}

```

```

    putsUSART(str2);
    Delay1mSec(T_uart_rs_end);
    putsUSART(str3);
    Delay1mSec(T_uart_rs_end);
    WriteUSART((char)num1);
    Delay1mSec(T_uart_rs_end);
    WriteUSART((char)num2);
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;

}
#endif

```

5.2.14. 副程式函式 — 顯示 UART 傳送錯誤函式(除錯資訊)

這部份是模組的接收如果出現一些錯誤時的反應訊息，我會把這些訊息開出來看，有時候是 TX、RX 太長了造成錯誤，有時候是溢位元和覆傳動作，因為模組有時候會偷偷睡覺，所以你熊熊拿起來按個指令就會送兩次，造成傳太快而 ERROR，我透過這些錯誤去調整裝置與模組的相容度。

```

#ifndef DBG_UART_ERR
void func_DBG_UART_ERR(void)
{
    if(UART_STATUS&USART_OERR)
    {
        BM_RX_IND = 0;
    }
}

```

```

Delay1mSec(T_uart_rs_ind);
putrsUSART((const far rom char *)"USART_OERR");
Delay1mSec(T_uart_rs_end);
BM_RX_IND = 1;
UART_STATUS&= (~USART_OERR);
}

if(UART_STATUS&USART_FERR)
{
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const far rom char *)"USART_FERR");
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
    UART_STATUS&= (~USART_FERR);
}

if(UART_STATUS&USART_EOF)
{
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const far rom char *)"USART_EOF");
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
    UART_STATUS&= (~USART_EOF);
}

```

```

    }

if(UART_STATUS&USART_RETURN)
{
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const far rom char *)"USART_RETURN");
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
    UART_STATUS&= (~USART_RETURN);
}

if(UART_STATUS&USART_NUL)
{
    BM_RX_IND = 0;
    Delay1mSec(T_uart_rs_ind);
    putrsUSART((const far rom char *)"USART_NUL");
    Delay1mSec(T_uart_rs_end);
    BM_RX_IND = 1;
    UART_STATUS&= (~USART_NUL);
}

#endif

```

5.2.15. 副程式函式 — 解析接收指令函式

這部份就是將 ## 和分隔的 * 做解析的函式了，所以之後 COMMAND 會被放在 str1 中，參數會被放到 *str2 中，和值放到 *str3 中。

```
Byte parse_parse_token(void)
{
    Byte i,str_num=0,token_num=0;
    char s1=0,s2=0,s3=0;

    for(i=0;i<STR_MAX;i++)
    {
        if(RS_buf[i]=='#') {
            i++;
            token_num++;
            if(token_num>1)
                str_num++;
        }
        else if(RS_buf[i]=='*') {
            i++;
            str_num++;
        }
        else if(((RS_buf[i]>='0') && (RS_buf[i]<='9')) || ((RS_buf[i]>='A')
&& (RS_buf[i]<='Z')) || ((RS_buf[i]>='a') && (RS_buf[i]<='z'))) {
            }
    }
}
```

```
else
    i++;

switch(str_num)
{
case 0:
    str1[s1++] = RS_buf[i];
    break;
case 1:
    str2[s2++] = RS_buf[i];
    break;
case 2:
    str3[s3++] = RS_buf[i];
    break;
case 3:
    break;
}
// PWD_NUM = s1;           // for SAVE Password NUM 20110907
return s3;
}
```

5.2.16. 副程式函式 — IIC 部份介紹

另外增加 IIC 控制 EEPROM 的部份：

參數部份：

```
#define DEF_EEP_SSPADD           27 // 39

// =====
// Read / write to EEPROM          - OK 8/16
// When writing to the EEPROM, first disable the interrupts.
// The address and data bytes are set followed by writing DEF_EEP_VERIFY
// and then 0xaa to EECON2.
// The details are in section 7.0 of the PIC18F4550 datasheet.
// =====

// EEByteWrite
// Return Value: 0 if there were no errors
// -1 if there was a bus collision error
// -2 if there was a NOT ACK error
// -3 if there was a write collision error
```

5.2.17. 副程式函式 — IIC 硬體設定

```
void Open_IIC(void)
{
    OpenI2C(MASTER, SLEW_OFF);           // Initialize I2C module
```

```

SSPADD = DEF_EEP_SSPADD;           // 400kHz Baud clock(9) @16MHz
                                    // 100kHz Baud clock(39) @16MHz
}

DEF_EEP_SSPADD 這數值也是實際用視波器調整出來的，這一段要放在 MAIN()前面部份打開 IIC 功能。

```

5.2.18. 副程式函式 — EEPROM 資料檢查碼函式

```

// =====
// Func Name : Check_Verify
// Check EEprom Vreify postion
// Return Vreify Result
// 0 - Vreify Fail
// 1 - Vreify TRUE
// =====
Byte Check_Verify(void)
{
    char chk;

    chk = (char)EERandomRead(DEF_EEPROM_DEV,EEP_VERFINE);
    Delay1mSec(T_I2C_TWC);

    if(chk == DEF_EEP_VERIFY)
        return (Byte)1;
    else
        return (Byte)0;
}

```

5.2.19. 副程式函式 — EEPROM 回覆預設值函式

這是從 EEPROM 把預設值給寫到 EEPROM 的功能，Verify Code 的功能就是在每一次改版後，更改 FW 版本後一起變更 Verify Code 值，因為這是透過一筆資料直接去比對，如果升級了程式但是 EEPROM 裡面值還是舊的.... 那會是一場災難... 所以這個部份可以放到一開機的 MAIN(); 的 init(); 部份去做，可以這樣：

```
if(!Check_Verify())
    Load_Default();
=====
void Load_Default(void)
{
    BYTE i;

    EEByteWrite(DEF_EEPROM_DEV, EEP_VERFINE, DEF_EEP_VERIFY);           // Verify
Code
    Delay1mSec(T_I2C_TWC);

    EEByteWrite(DEF_EEPROM_DEV, EEP_PWD_NUM, DEF_EEP_PWD_NUM);          // Default
Password Num
    Delay1mSec(T_I2C_TWC);

    for(i=0;i<6;i++) {
        EEByteWrite(DEF_EEPROM_DEV, (EEP_PWD_BYTE_1+i), DEF_EEP_PWD);   // Default Password
        Delay1mSec(T_I2C_TWC);
```

```

    }

EEByteWrite(DEF_EEPROM_DEV, EEP_SAT_MACHINE_STD, DEF_SAT_MACHINE_STD);
Delay1mSec(T_I2C_TWC);
}

```

5.2.20. 副程式函式 — EEPROM 讀取密碼 (EEPROM 連續讀取資料範例)

這個取密碼的函式是抓取 EEPROM 中連續位置的範例，透過 EEPROM.H 中的 ENUM 列舉 MAPPING 到實際的 EEPROM 中，如果你要取的值超過 EEPROM 一頁 2K 範圍，也可以繼續列舉在函式中可以用列舉值除 256 得到頁值，然後列舉值%256 得到位址值。

```

=====

// Function Name: Load_Password
// Action: Load EEPROM PWD
=====

void Load_Password(void)
{
    BYTE i;

    PWD_NUM = (char)EERandomRead(DEF_EEPROM_DEV, EEP_PWD_NUM);
    Delay1mSec(T_I2C_TWC);
    PWD_IMM_NUM = (char)EERandomRead(DEF_EEPROM_DEV, EEP_IMM_PWD_NUM);
    Delay1mSec(T_I2C_TWC);
}

```

```

for(i=0;i<6;i++) {
    EEP_PWD[i] = (char)EERandomRead(DEF_EEPROM_DEV, (EEP_PWD_BYTE_1+i));
    // Delay1mSec(T_I2C_TWC);
}
for(i=0;i<6;i++) {
    EEP_IMM_PWD[i] = (char)EERandomRead(DEF_EEPROM_DEV,
(EEP_IMM_PWD_BYTE_1+i));
    // Delay1mSec(T_I2C_TWC);
}

```

5.2.21. 副程式函式 — EEPROM 寫入密碼函式 (EEPROM 連續寫入資料範例)

這是連續存值 EEPROM 範例.

```

//=====
// Function Name: Save_Password
// Param: PWD - Password
// Param: len - Passwoed Length
// Action: Save PWD to EEPROM
//=====
void Save_Password(char *str,char num)
{
    BYTE i;

```

```

    EEByteWrite(DEF_EEPROM_DEV, EEP_PWD_NUM, num); // 
    Password Num
    Delay1mSec(T_I2C_TWC);

    for(i=0;i<6;i++) {
        EEByteWrite(DEF_EEPROM_DEV, (EEP_PWD_BYTE_1+i), str[i]);
        Delay1mSec(T_I2C_TWC);
    }
}

```

5.2.22. 副程式函式 — EEPROM.H 檔案部份

預處理定義：

```

// =====
// EEPROM.H
// =====
#ifndef __EEPROM_H
#define __EEPROM_H

#define DEF_EEP_READ          1
#define DEF_EEP_WRITE          0

#define DEF_EEPROM_DEV         0xA0

```

```
// ===== Default =====
#define DEF_EEP_PWD_NUM          0x04
#define DEF_EEP_PWD              '0'
#define DEF_SAT_MACHINE_STD      0x00

EEPROM 位址 MAPPING :
// === EEPROM MAP =====
enum {
    EEP_VERFINE = 0,
    EEP_FW_VER_01,
    EEP_FW_VER_02,
    EEP_FW_VER_03,
    EEP_FW_VER_04,
    EEP_FW_VER_05,
    EEP_FW_VER_06,
    EEP_FW_VER_07,
    EEP_FW_VER_08,
    EEP_FW_VER_09,
    EEP_FW_VER_10,
    EEP_PWD_NUM,
    EEP_PWD_BYTE_1,
    EEP_PWD_BYTE_2,
    EEP_PWD_BYTE_3,
```

```
EEP_PWD_BYTE_4,  
EEP_PWD_BYTE_5,  
EEP_PWD_BYTE_6,  
  
EEP_SAT_MACHINE_STD,  
  
EEP_MAC_PWD_NUM,  
  
EEP_MAC_PWDBYTE_1,  
EEP_MAC_PWDBYTE_2,  
EEP_MAC_PWDBYTE_3,  
EEP_MAC_PWDBYTE_4,  
EEP_MAC_PWDBYTE_5,  
EEP_MAC_PWDBYTE_6,  
  
EEP_MAC_STATUS,  
  
EEP_MAX_ADDR = 0xFF  
};  
  
extern void Open_IIC(void);  
extern void Load_Default(void);  
  
#endif
```

5.2.23. 程式導讀 — 使用者規劃檔案

進入程式的規劃後，一開始當然是從 main.c 開始寫，但是這次要多加一個 config.h 的檔案，這樣我們在做後續程式時就會簡單容易的多，為避免一些隱藏問題和出貨版與工程師 DEBUG 版簡單切換，就增加這個檔案來設計吧!!

首先介紹 Config.h 檔案，在寫程式時一些細節我們要考慮好。

```
// =====
// Config.h
// Factory Mode Use ICSP Tools To Lock Mcu,
// Then BT Moudle RS232 Tools Link, Download BT FW or Config BT
// =====
#ifndef __Config_H
#define __Config_H

#include "TypeDefs.h"
#include "pconfig.h"

#include "p18f25j10.h"
#include "HW_ProfilePIC18.h"
#include "portb.h"

#include "delay.h"
```

一開始我們在檔案前要先加上檔案的註解，這是一個寫程式者的好習慣，這樣我們會很容易了解檔案的用法與功能為

何，然後是 .h 檔的定義避免重複的宣告問題，所以我們加上了這兩行：

```
#ifndef __Config_H  
#define __Config_H
```

如果未定義那麼我們就定義，如果已經定義呢?! 代表這程式的 [視野] scope 已經存在了，那就不會再重複的定義到湖裡面，英文是湖也可以說定字池，之後就是我們需要的檔案，在 config.h 的檔案中它的視野是最高的，也就是說幾乎所有的 .c 檔都會呼叫到這檔案，因此我們把通用的標準 c 函式庫都定義到這檔案中，我們在做定義時或是很多的 c 指令時，都要遵守一個觀念，那就是先去閱讀你使用的編譯器的手冊與程式範例的文件，這非常重要，多數的微處理器 MCU 都有特殊的編譯器，寫程式不熟悉編譯器是一個很危險的事，因此要先去了解這編譯器的架構與使用原則，以 C18 為例這是一個透過 ASM 組合語言所組成的編譯器，這套並不是標準 C 編譯器，所以一些標準 C 的用法不一定都可以支援，這是一個隱藏殺手，在安裝完 MICROCHIP 提供的 C18 編譯器後，在安裝檔案夾中會找到函式庫的檔案和說明，與編譯器的說明檔案，不要因為怕麻煩而不去閱讀這東西，像是以下這幾個檔案：

[MPLAB C18 C Compiler User's Guide and Release Notes.pdf](#)

[MPLAB C18 Libraries Documentation.pdf](#)

[MPLAB-C18-Getting-Started.pdf](#)

[MPLAB-C18-Libraries.pdf](#)

以上文件請熟讀，邊進行程式進行中邊讀也可以，在這個編譯器中有一個方法可以不要一個檔案一個檔案的 include 進來，但是我不想要這樣，我要控管好每一個程式視野，這樣才會有效率和降低 code size，甚至編譯好的 asm 我也

會看一下.

這部份就是要定義其他的設定與 DEBUG 使用的部份檔案，我把所有的 DEBUG 都放到一個定義裡面，這技巧是跟許多原廠的 TOTAL SOLUTION 學的，這樣雖然寫的時候麻煩點，用的時候就很方便!!

```
// ===== Define Debug =====  
  
#define debug  
#ifdef debug  
// #define DBG_ShowRs           // Show RX Recevice Data + end  
#define DBG_UART_ERR          // Show RX Recevice Error Status  
#define DBG_TEST_FUNC          // For Test Function  
#endif  
  
#endif /* _CONFIG_H_ */
```

包覆在 debug 定義中的定義是一定要在 debug 打開才能使用，而 debug 打開後內部的子定義也不一定要全開，可以一部份 debug 完一部份就關掉，到最後所剩下的資訊就會變成單純的，比如我要進行接收某個模組的 RS 訊號，要顯示我就把這部份打開，關掉的話就不會顯示這模組的 RS 訊息出來，你也可以做成分層寫法，就是將 debug 定義為一個 0~FF 的數值，然後在 FF 層就是最高所有訊息都會打開，0 層就是關閉所有訊息，這樣也可以。

PS. 為什麼對微處理機的程式要特別小心 Scope 問題?! 因為這一顆的 RAM 只有 1K 能夠用 C 寫已經是天大的恩惠

了!!

中斷也只有兩個，分為高和低的中斷，之後程式會介紹到。

5.2.24. 程式導讀 — 主程式預處理部份

程式在 main 的部份開始先放上一些初始化設定部份：

```
void baudUSART ( unsigned char baudconfig);
void OpenUSART ( unsigned char config,  unsigned spbrg);

//-----
// General Hardware Setup
//-----

#define F_OSC          8000000UL           // 8MHz
#define USART_SYNC      0
#define USART_BRG16     1
#define USART_BRGH      1
#define USART_BAUDRATE  38400

#if (USART_SYNC == 1)
    #define USART_BRG_DIV        4
#elif ((USART_BRG16 == 0) && (USART_BRGH == 0))
    #define USART_BRG_DIV        64
#elif ((USART_BRG16 == 1) && (USART_BRGH == 1))
    #define USART_BRG_DIV        4
```

```

#else
    #define USART_BRG_DIV          16
#endif

#define USART_BRG      (((F_OSC/USART_BAUDRATE) +
(USART_BRG_DIV/2))/USART_BRG_DIV)-1 // + (USART_BRG_DIV/2)
#define CONFIG_USART_BRG16()        BAUDCONbits.BRG16=USART_BRG16,
TXSTAbits.BRGH=USART_BRGH

Open_IIC();

IPR1bits.RCIP = 1;           // Make [1/0] RX interrupt [HI/LOW] priority
IPR1bits.TXIP = 1;           // Make [1/0] TX interrupt [HI/LOW] priority

baudUSART (    BAUD_IDLE_RX_PIN_STATE_HIGH &
                BAUD_IDLE_TX_PIN_STATE_HIGH &
                BAUD_16_BIT_RATE &
                BAUD_WAKEUP_OFF &
                BAUD_AUTO_OFF );

OpenUSART (    USART_TX_INT_OFF &
                USART_RX_INT_ON &
                USART_SYNCH_MODE &
                USART_EIGHT_BIT &
                USART_CONT_RX &

```

```

USART_BRGH_HIGH &
USART_ADDEN_OFF,
USART_BRG);

RCONbits.IPEN = 1;           //Enable interrupt priority
INTCONbits.GIEL = 1;         //Enable low priority/peripheral
interrupts
INTCONbits.GIEH = 1;         //Enable high priority/global interrupts

// USART_RX_INT_ON      //這設定決定 UART 使用 INT 的方式接受.
// USART_BRG            //上一頁的公式巨集就將 FOSC、BAUDRATE 計算寫在巨集中.

```

最後在打開 UART 的 ENABLE 和 INT 的高與低觸發，這樣 UART 就可以接收與發送資料了，建議是用視波器在設定完後，將實際資料抓出來看時序是否正確，我也是因為實際去看訊號才摸出了原本計算公式的缺點，並且在最佳化調整出 38400 是我所需要最合用的 BAUDRATE，技術不是憑想像冒出的東西，而是基礎與經驗的驗證，很多的平臺別人做不來，但是我們可以做的很好，差距就是在這 3%的誤差這也是 $+(USART_BRG_DIV/2)$ 的補公式差值由來，SPEC 中可能不會有，我忘記在那裡找到的了...

5.2.25. 程式導讀 — 中斷常式介紹

定義部份預處理：

```
#define STR_MAX      24
#define STR_THREE    8
#define STR_MAC      14

// ===== F/W Version ===== //
#define FW_VER_STR    "F/W Version :"
#define FW_VER_ID     "Monitor"
#define FW_Version     "v1.0"           // 長度 10 位碼
#define DEF_EEP_VERIFY 'A'           // VERIFY CODE
```

5.2.26. 程式導讀 — 中斷向量位址

這裡是定義 INT 中斷向量的位址，並且讓 vector 位置指到我們的中斷服務程式中。

```
// interrupt *****
#pragma code high_vector = 0x08
void interrupt_at_high_vector (void)
{
    _asm goto high_isr _endasm
}

#pragma code low_vector = 0x18
```

```
void interrupt_at_low_vector(void)
{
    _asm GOTO low_isr_endasm
}
```

5.2.27. 程式導讀 — 中斷服務常式

```
// === C O D E === //
#pragma code
#pragma interrupt low low_isr

void low_isr(void)
{
    // NO low ISR
    return;
}

#pragma interrupt high_isr
void high_isr (void)
{
    char c;

    if(PIE1bits.RCIE && PIR1bits.RCIF)          // Did USART cause interrupt
    {
        if(!rs_ready)
```

```

{
#endif DBG_UART_ERR
if (RCSTAbits.OERR)
{
    RS_buf[0] = ReadUSART(); // clear the receiver fifo
    RS_buf[0] = ReadUSART(); // ReadUSART();
    RCSTAbits.CREN = 0;      // clear the OVR flag
    RCSTAbits.CREN = 1;
    UART_STATUS |= USART_OERR; // send (prepare) error message
    return;
}
if (RCSTAbits.FERR)
{
    RS_buf[0] = ReadUSART(); // remove the broken byte
    UART_STATUS |= USART_FERR; // send (prepare) error message
    return;
}

/* Get the character received from the USART */
c = ReadUSART();
if (c == EOF)
    UART_STATUS |= USART_EOF;
if (c == '\n')
    UART_STATUS |= USART_RETURN;

```

```

if(c == ' ')
    UART_STATUS |= USART_NUL;
#endif

RS_buf[RS_in++] = c;
if(RS_in>=STR_MAX) {
    Rs_hotkey = 0;
    RS_in     = 0;
}

if(c == '#') {
    Rs_hotkey++;
    if(Rs_hotkey>= 2) {
        rs_ready = 1;
        Rs_hotkey = 0;
        RS_in     = 0;
    }
    else
        rs_ready = 0;
}
}

PIR1bits.RCIF = 0;           // Clear the interrupt flag
}

return;

```

}

5.2.28. 程式導讀 — 中斷服務常式說明

中斷分為低與高的中斷函式：

```
void low_isr(void) // 這裡面沒有放程式，如果你高興也可以把 HI ISR 的程式 COPY 進去，但設定要改為 LOW 觸發。  
void high_isr (void)  
  
c = ReadUSART();  
RS_buf[RS_in++] = c; // 這是接收資料移到 RS_BUF 裡面，看你想收多大的資料可以自己設一個最大值  
STR_MAX
```

我寫成接收指令為 #COMMAND# 在兩個 ## 中間的指令會被 parse 出來，看你怎麼規劃命令的結構都可以，但要對應程式解法!!

解出的指令會被放到 RS_buf 裡面，在變數中 ['#', 'C', 'O', 'M', 'M', 'A', 'N', 'D', '#']，接收完成後 rs_ready 會被設定為 '1'，main 程式中的部份會放入判斷式，檢查 rs_ready 是否為 '1' 進入 Monitor()；函式中，那接收的部份就解決了。

5.2.29. 參考資料 — BAURD RATE 計算公式

我想要先介紹 UART 一下但是不曉得這會不會太複雜，因為要寫 MCU 的話 DEBUG 工具是很重要的，一般整合性

的 IDE 像是 KEIL C 會有提供 DEBUG 的功能，但是在程式設計使用不同 IC 可能會沒有很完整的 DEBUG 工具，那麼你就會需要一個好辦法幫忙，這個程式的基本架構約 4K 你可以自己斟酌來加函式，而一關掉 DEBUG 呢，就一點都不影響程式了，以下是程式部份與解說：

我當初在寫這程式用的 MCU 是 PIC25J10 聽代理商說核心是 MIPS 的，使用的是 C18 的編譯器，MICRO CHIP 是 PIC 的公司，我覺得他們的工具做的有點糟，而一般車用領域大部份也不太用 PIC，比較多會用瑞薩的或是 TI 的 MSP 系列，工具部份我不介紹，因為我不用那東東來 DEBUG 我在編輯器上改完後就透過 C18 編譯上傳到機器，然後就可以控制了，所以我只用兩顆按鈕 1. 編譯 2. 燒錄

因為這一顆是 RISC 架構的所以指令都是除 4 週期計算，先來算一下鮑率吧!!

FOSC		BaudRate	SYNC	BRG16	BRGH	誤差率
4	MHz	38400	0	1	1	1.0000%
4000000	104.17	26.04				
N	BRG 值					
4	25					

單一計算 Baud Rate			
BRG 值	覆算	誤差率	USE
25	38461.54	0.16026%	可用

青色可更改
紅色字絕對勿動
灰色字勿變

PIC18F25J10
目標波特率 = FOSC/(64 ([SPBRGH:SPBRG] + 1))
X = ((FOSC/ 目標波特率) - 1 // 1200 以下
X = (((FOSC/ 目標波特率)+(N/2))/N) - 1 // 2400 以上
= (((8000000/9600)+2)/4) - 1
= [207.833] = 207
波特率計算結果 = 8000000/(4 (207 + 1))
(波特率計算結果 - 目標波特率) / 目標波特率
= (9615 - 9600)/9600 = 0.16%

BaudRate	USE	BRG	BRG(實)	覆算	誤差率(絕)	誤差率
110	可用	9089	9089	110.01	0.01000%	-0.01000%
300	可用	3332	3332	300.03	0.01000%	-0.01000%
1200	可用	832	832	1200.48	0.04002%	-0.04002%
2400	可用	416	416	2398.08	0.07994%	0.07994%
4800	可用	207	207	4807.69	0.16026%	-0.16026%
9600	可用	103	103	9615.38	0.16026%	-0.16026%
19200	可用	51	51	19230.77	0.16026%	-0.16026%
38400	可用	25	25	38461.54	0.16026%	-0.16026%
57600	不可用	16	16	58823.53	2.12418%	-2.12418%
115200	不可用	8	8	111111.11	3.54938%	3.54938%

以上就是 PIC18 系列所使用的波特率的計算表格與公式，這的試算表的公式要比官方提供的強多了，我當初找官網的資料，不是很清楚就決定自己寫了，你們可以新增試算表檔案，再把上表轉貼過去，其中可以更改的是青色字的部份，比如 FOSC 4 MHZ，可以更改為 11.0592 或是其他的值，也可以在 4 那一格輸入 =11.0592/4，這樣的計算式那就會得到 2.76 MHZ，頻率 FOSC 降低了為何反而鮑率可以支援到更高？！原因就是自然界所使用的振蕩器都有一定的頻率，而透過 IC 內部的 PLL 一般動作是將 FOSC 來降頻或是升頻，所以我們得到的是一個整除於公式的數，因此不能很準確的達到標準鮑率的使用頻，也就是說 11.0592 這頻率去除 115200 或是其他的鮑率是可以整除的，你們可以試試，答案永遠都很簡單，設計上的問題很多都源於設計複雜度與金錢，而工程師的立場不單純是屬於科學家

要去探究未知，而是透過可行的方式將問題找到並且解決，所以我們輸入了 FOSC 代表使用的晶振是這個頻率，以及 SYNC、BRG16 和 BRGH 設定好，這三個旗標(FLAG)所代表的就是：
FOSC /[(64 OR 16 OR 4) * (N+1)]

設完後還有誤差值的部份也設定一下，一般都是 5% 以內就可以用，我設是 1%，也可以設為 0.5%或是更低的誤差值，這沒有差因為每一次收完 UART 就重計，誤差不會累計的，越準的鮑率當然是容錯率會較好。(這是奇樣子問題)
PS. 8051 是 CISC 架構所以指令週期是除 12，過去常用的是 11.0592 MHZ 的晶振，或是 12MHZ 的晶振，想當然 11.0592MHZ 要比 12MHZ 來的支援比特率要更高，因為可以被整除。

5.2.30. 參考資料 — PIC18F45J10 系列— 波特率發生器 (BRG)

BRG 是一個專用的 8 位或 16 位發生器，支持 EUSART 的異步和同步模式。默認情況下，BRG 工作在 8 位模式下，將 BRG16 位 (BAUDCON<3>) 置 1 可選擇 16 位模式。

SPBRGH:SPBRG 寄存器對控制一個獨立運行定時器的周期。在異步模式下，BRGH (TXSTA<2>) 位和 BRG16 (BAUDCON<3>) 位也控制波特率。在同步模式下，BRGH 位會被忽略。

表 16-1 所示為不同 EUSART 模式的波特率計算公式，但僅適用於主控模式 (由內部產生時鐘信號)。

給定目標波特率和 FOSC 的情況下，可以使用表 16-1 中的公式計算 SPBRGH:SPBRG 寄存器中的近似整數值，從而確定波特率誤差。

例 16-1 紹出了計算示例。

表 16-2 中給出了不同異步模式下典型的波特率和誤差值。

使用高波特率 (BRGH = 1) 或 16 位 BRG 有利於減少波特率誤差，或者在快速振盪頻率條件下實現低波特率。向 SPBRGH:SPBRG 寄存器寫入新值會引起 BRG 定時器復位 (或清零) 。

這可以確保 BRG 無需等待定時器溢出就可以輸出新的波特率。

16.1.1 在功耗管理模式下的操作器件時鐘用於產生所需的波特率。

當進入一種功耗管理模式時，新時鐘源可能會工作在一個不同的頻率下。

這可能需要調整 SPBRG 寄存器對中的值。

16.1.2 採樣

擇多檢測電路對 RX 引腳採樣三次，以判定 RX 引腳上出現的是高電平還是低電平。

表 16-1： 波特率公式

配置位			BRG/EUSART 模式	波特率计算公式
SYNC	BRG16	BRGH		
0	0	0	8 位 / 异步	Fosc/[64 (n + 1)]
0	0	1	8 位 / 异步	Fosc/[16 (n + 1)]
0	1	0	16 位 / 异步	
0	1	1	16 位 / 异步	Fosc/[4 (n + 1)]
1	0	x	8 位 / 同步	
1	1	x	16 位 / 同步	

图注： x = 任意值， n = SPBRGH:SPBRG 寄存器对的值

5.2.31. 6.3 參考資料 — 常用的頻率與使用到的技術對應表：

像是 3.58 是 US 泛美系統使用在 TV 上的，
4.43 是歐洲區域用在 TV 上的，27M 是
DTV 和 MPEG 使用的色載波用頻率與 DOT
CLOCK 的基頻使用頻率。

5.3. 結語

本 UART 的設計設計是透過 H/W 觸發接收訊號後，將接收資料暫存器移到 BUFF 裡面，之後在主程式中去處理接收到的指令，使用 UART 對於 MCU 使用上來說是很方便的，IIC 也是很方便使用，一般來說 SPI 要比 IIC 速度快，但是佔用的 PIN 要較多而且隨裝置越多 PIN 使用會更多，一般我都比較喜歡用 IIC 和 UART 來設計，SPI 的協定比較類似 UART，一個 DO 一個 DI 輸出入可以同時，還有一 PIN 是做 CLOCK 使用，很像是 IIC+UART 的合體，有一些

Frequency (MHz)	Typical Application	Frequency (MHz)	Typical Application
1.54	DS1	18.43	Microprocessor Clock
1.84	Microprocessor Clock	19.2	Cellular
2.05	CEPT1	19.44	SONET
2.3	SONET Tributaries	19.68	Cellular
3.09	DS1 (2X)	20	DSP, Microprocessor Clock, Hub
3.15	DS1 C	20.28	Microprocessor Clock
3.46	SONET Tributaries	20.48	ISDN, CEPT1 (10X)
3.58	US Color TV Subcarrier	22.12	Microprocessor Clock
3.69	Microprocessor Clock	23.16	DS1 (15X)
4	TOKEN RING, Clock	24	DSP, Microprocessor Clock
4.03	Microprocessor Clock	24.58	CEPT1 (12X), MPEG AUDIO
4.1	CEPT1 (2X)	24.7	DS1 (16X)
4.43	European TV	25	DSP, Microprocessor Clock, Hub
5	Microprocessor Clock	26.66	DSP, Microprocessor Clock, Hub
5.07	Microprocessor Clock	27	DIGITAL TV, MPEG Video
6.14	CEPT1 (3X)	27	DSP, Microprocessor Clock, Hub
6.18	DS1 (4X)	28	DSP, Microprocessor Clock, Hub
6.31	DS2	28.22	56K Modem
6.4	ISDN	29	DSP, Microprocessor Clock, Hub
6.91	SONET Tributaries	29.49	V.34 Modem
7.37	Microprocessor Clock	30	DSP, Microprocessor Clock, Hub
8	Microprocessor Clock	30.88	DS1 (20X)
8.06	Microprocessor Clock	32	DSP, Microprocessor Clock, Hub
8.19	CEPT1 (4X)	32.06	DS3-Japan
8.45	CEPT2	32.77	CEPT1 (16X)
9.22	Microprocessor Clock	33.33	Ethernet, LAN
10	Microprocessor Clock	34.37	CEPT3
10.7	TV	35.33	CEPT1 (17.25X)
11.06	Microprocessor Clock	36	DSP, Microprocessor Clock, Hub
12	Microprocessor Clock	36.86	CEPT1 (18X)
12.29	CEPT1 (6X), MPEG	37.06	DS1 (24X)
12.35	DS1 (8X)	38.88	SONET
12.5	Microprocessor Clock	40	DSP, Microprocessor Clock, Hub
12.62	DS2 (2X)	44.24	V.34 Modem
12.8	CEPT1 (6.25X)	44.74	DS3
13	Cellular	48	DSP, Microprocessor Clock, Hub
13.5	DIGITAL TV	49.15	CEPT1 (24X)
14	Microprocessor Clock	50	DSP, Microprocessor Clock, Hub
14.32	Microprocessor Clock	51.84	SONET OC-1
14.34	CEPT1 (7X)	56.45	56K Modem
14.4	Cellular	60	SCSI
14.75	Microprocessor Clock	66	DSP, Microprocessor Clock, Hub
15	Microprocessor Clock	66.67	DSP, Microprocessor Clock, Hub
15.36	CEPT1 (7.5X)	66.67	DSP, Microprocessor Clock, Hub
15.44	DS1 (10X)	77.76	SONET
16	TOKEN RING, DSP Clock	80	DSP, Microprocessor Clock, Hub
16.13	Microprocessor Clock	100	DSP, Microprocessor Clock, Hub
16.38	CEPT1 (8X)	106.25	FDDI (Fiber Channel)
16.93	VIDEO	125	FDDI (Fiber Channel)
17.73	European TV	155.52	SONET OC-3, SDH STM1

FLASH 是用 SPI 來設計，這樣很省錢速度也不錯，我不太了解使用者會碰到什麼問題... 因為我沒有碰到問題所以沒辦法理解是否會出現問題，本程式是在接 TECH WELL 的平台時看到原廠設計，我覺得他們這種用法很方便，就把接收程式改寫了，透過中斷來使用，實際使用上也很方便... 大概是這樣

邀稿與訂閱

1. 讀者訂閱

想訂閱本雜誌的讀者，請按以下的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。

[我想訂閱程式人雜誌](#)

2. 作者徵求

程式人雜誌非常歡迎您加入專欄作者的行列，如果您想撰寫任何文章或投稿，請加入我們的 facebook 社團並與我們一同討論，社團網址如下。

<https://www.facebook.com/groups/programmerMagazine/>

3. 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 [程式人雜誌的 facebook 社團](#) 來與我們討論相關事宜。

4. 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