**初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )**

*網頁最後修改時間：2017/01/07*

|  |
| --- |
| [https://2.bp.blogspot.com/-1P1JsxTe9Rs/VHF3CWkw-rI/AAAAAAAAGGs/btzUKd94RSc/s1600/ThingSpeak_IoT_Demo-blog.png](http://2.bp.blogspot.com/-1P1JsxTe9Rs/VHF3CWkw-rI/AAAAAAAAGGs/btzUKd94RSc/s1600/ThingSpeak_IoT_Demo-blog.png) |
| ThingSpeak 環境溫濕度記錄圖 |

網路是構成物聯網 ( IoT, Internet of Things ) 其中一項不可或缺的技術，而 IoT Server 就是用來蒐集並記錄由世界各地傳過來的數據資料，這些資料可以自己使用也可以分享給其他人做使用，這對於需要在世界各地蒐集資料的科學家或是程式設計師，善用這些資料可以加速其研究分析的工作，節省很多的時間。例如，對於科技農夫來說，可以蒐集整片土地上面各處土壤的溫溼度、光照程度在特定時間間隔的分佈狀態，以長時間的數據來分析影響土地上各植物的生長狀態的因素，幫助農夫可以做製程改善達到最大的產量輸出。當然，還有很多的應用，可以利用物聯網的方式來實現，而本篇網頁提出一個實際的例子來說明，如何使用賣場的 "[ESP8266 IoT 入門學習套件](http://goods.ruten.com.tw/item/show?21447787125749)" 每隔 60 秒傳送環境溫濕度到 IoT Server ( ThingSpeak ) 做紀錄。  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
[*物聯網，維基百科的解釋*](http://zh.wikipedia.org/wiki/%E7%89%A9%E8%81%94%E7%BD%91)  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
本篇網頁會使用到的材料，可自行準備或是到露天賣場購買

* [萬物皆聯網-ESP8266 IoT(Internet of Things)入門學習套件(樹莓派、Arduino、單晶片)](http://goods.ruten.com.tw/item/show?21447787125749)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
我所使用的 IoT Server 是由 ThingSpeak 所提供，支援很多微處理器和控制器，也提供相對應的使用範例做為使用上的參考，上手很快也很方便。重點是：ThingSpeak 是開源的，可以下載並安裝在自己的電腦上做 IoT Server。  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Internet of Things -*[*ThingSpeak*](https://thingspeak.com/)

*ThingSpeak 開放原始碼 -*[*GitHub*](https://github.com/iobridge/thingspeak)

ThingSpeak is an open source “Internet of Things” application and API to store and retrieve data from things using HTTP over the Internet or via a Local Area Network. With ThingSpeak, you can create sensor logging applications, location tracking applications, and a social network of things with status updates.

*ThingSpeak 教學 -*[*Tutorials*](http://community.thingspeak.com/tutorials/)  
  
*參考網頁：*

* *ThingSpeak -*[*Using an Arduino + Ethernet Shield to Update a ThingSpeak Channel*](http://community.thingspeak.com/tutorials/arduino/using-an-arduino-ethernet-shield-to-update-a-thingspeak-channel/)
* *Instructable -*[*ESP8266 Wifi Temperature Logger*](http://www.instructables.com/id/ESP8266-Wifi-Temperature-Logger/) *網頁主要是參考這篇文章裡的內容，作者給了很大的幫助！但是實際上，照著做並不能成功上傳資料，原因在於：(1) 作者的 ESP8266 韌體有可能是舊版的，或是已事先輸入幾個設定的 AT 指令進入模組，因為沒有出現在程式碼中，直接使用程式碼是無法正常上傳資料的；(2) 作者使用 3V3 的 Arduino 板子和 DS18B20 溫度感測器，線路是不一樣的，程式碼必須重新修改；(3) 作者使用單筆資料而非多筆資料同時上傳，這部分需要上 ThingSpeak 網站上面找一下資料。*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

下面分為幾個步驟來實現溫溼度上傳到 IoT Server - ThingSpeak 的目的：

* 申請一個 ThingSpeak 帳號  
  .
* 建立一個給 DHT11 使用的 Channel  
  .
* 取得剛剛建立的 ThingSpeak Channel 的( Write ) API KEY，並手動測試  
  .
* HTTP 方法：POST 和 GET  
  .
* 佈線電路圖說明  
  .
* 程式測試說明

*Note：所有這網頁需要的資料全部都已經上傳到賣場雲端硬碟中，請至目錄中下載或是直接移至自己的雲端硬碟。*  
*電路圖：****ESP8266 IoT 入門學習套件 / schematics / Arduino\_IoT\_Demo***  
*程式碼：****ESP8266 IoT 入門學習套件 / codes / Arduino / IoT\_demo***  
  
  
**申請一個 ThingSpeak 帳號：**  
  
ThingSpeak 帳號申請很簡單，直接進入 [**ThingSpeak.com**](http://thingspeak.com/) 主網頁點擊右上角的 [**Sign Up**](https://thingspeak.com/users/sign_up)連結進入註冊頁面。進入頁面後，依照欄位輸入相對應的資料，如下圖所示。

|  |
| --- |
| [https://2.bp.blogspot.com/-A9R4CLpRC4U/VHHYhlT7UlI/AAAAAAAAGHk/S7BOyt72jvE/s1600/ThingSpeak%2B-%2BSign%2BUp_desc.png](http://2.bp.blogspot.com/-A9R4CLpRC4U/VHHYhlT7UlI/AAAAAAAAGHk/S7BOyt72jvE/s1600/ThingSpeak%2B-%2BSign%2BUp_desc.png) |
| ThingSpeak 新帳號申請頁面 |

**建立一個給 DHT11 使用的 Channel：**  
  
完成帳號申請之後，按下 **Create Account** 按鈕就會跳轉到下面的頁面，這時已經確認產生新的帳號，可以開始設定 Channel

|  |
| --- |
| [https://4.bp.blogspot.com/-EDNaMrt0SZA/VHHaxqpmwmI/AAAAAAAAGH8/3N2mATZDpTg/s1600/ThingSpeak%2B-%2Bafter%2Bsign%2Bup-blog.png](http://4.bp.blogspot.com/-EDNaMrt0SZA/VHHaxqpmwmI/AAAAAAAAGH8/3N2mATZDpTg/s1600/ThingSpeak%2B-%2Bafter%2Bsign%2Bup-blog.png) |
| ThingSpeak - My Channels 頁面 |

**Channel Settings** 的頁面中，要設定的只有三個地方

* **Name**：Channel 的名稱，這裡請設定為 **DHT11**
* **Field 1**：資料欄位的顯示名稱，請設定為 **Temperature ( degC )**
* **Filed 2**：資料欄位的顯示名稱，請設定為 **Humidity ( % )**

|  |
| --- |
| [https://3.bp.blogspot.com/-ILwOzSwwNkI/VHHeDRwETWI/AAAAAAAAGII/9OoK3FnlZ14/s1600/ThingSpeak%2B-%2BChannel%2BSetting-blog.png](http://3.bp.blogspot.com/-ILwOzSwwNkI/VHHeDRwETWI/AAAAAAAAGII/9OoK3FnlZ14/s1600/ThingSpeak%2B-%2BChannel%2BSetting-blog.png) |
| ThingSpeak Channel Settings - 欄位名稱設定 |

這些設定的文字只是為了辨識輸入的資料欄位是代表什麼，實際從遠端輸入資料到 ThingSpeak IoT Server 時，資料欄位的輸入都是有固定的格式，程式設計師要知道的最重要的東西是這個 Channel 的 **Write API Key** 和 **Read API Key，**從遠端輸入資料時，就是傳送這個 API Key 給 ThingSpeak IoT Server，Server 才會知道這資料是屬於哪一個使用者的哪一個 Channel 的欄位資料；每一個 ThingSpeak 的 Channel 最多有 8 個 資料欄位。  
  
輸入完畢後按下最下方的 **Save Channel** 按鈕儲存資料；另外，Clear Channel 按鈕是用來清除所有以記錄的資料，這等一下會用到，要記得清除資料是在這裡清除！  
  
  
**取得剛剛建立的 ThingSpeak Channel 的( Write ) API KEY，並手動測試：**  
  
在同樣的頁面，上方有一顆 **API Keys** 按鈕，按進去就會出現剛剛這個 Channel 的寫入的 API Key，將 Write API Key 先複製或是記下來，等一下測試會用到！

|  |
| --- |
| [https://1.bp.blogspot.com/-zTlPzM_IKzo/VHHzthzkt3I/AAAAAAAAGIY/hho18Lvmt-o/s1600/ThingSpeak%2B-%2BAPI%2BKeys-M-blog.png](http://1.bp.blogspot.com/-zTlPzM_IKzo/VHHzthzkt3I/AAAAAAAAGIY/hho18Lvmt-o/s1600/ThingSpeak%2B-%2BAPI%2BKeys-M-blog.png) |
| ThingSpeak - Channel's API Key |

假設我們取得剛剛產生的 Channel 的 **Write API Key** 是 *ALSKUJG6DR5PLODE*，要設定的溫濕度分別是 **25** 度 C 與 **75** %，那輸入的標準格式是  
  
***http://api.thingspeak.com/update?key=[THINGSPEAK\_KEY]&field1=[溫度]&field2=[濕度]***  
  
根據我們所設定的溫度，則完整的輸入就會變成  
  
***http://api.thingspeak.com/update?key=ALSKUJG6DR5PLODE&field1=25&field2=75***  
  
打開瀏覽器，複製上面的資料 ( 記得換上自己的 **Write API Key** )，按下 **Enter**，若成功傳入資料到剛剛在 ThingSpeak 新增的 Channel，那麼就會在瀏覽器視窗收到 **1**

|  |
| --- |
| [https://2.bp.blogspot.com/-Q0mbYFw5gCc/VHH5MJeUjUI/AAAAAAAAGI0/gQHqiflcCXg/s1600/Send%2Bdata%2Bto%2BThingSpeak-blog.png](http://2.bp.blogspot.com/-Q0mbYFw5gCc/VHH5MJeUjUI/AAAAAAAAGI0/gQHqiflcCXg/s1600/Send%2Bdata%2Bto%2BThingSpeak-blog.png) |
| 傳送測試的溫溼度資料到 ThingSpeak |

到底資料傳進去了沒有，我們來確認一下！  
  
同樣在 Channels 頁面下，按下 **Private View** 就會看到兩個趨勢圖 ( 方便顯示起見，我將下方的圖移到右邊 )，而顯示的結果與我們剛剛輸入的資料相同，這也就表示上面的輸入格式與資料都是正確的。

|  |
| --- |
| [https://3.bp.blogspot.com/-F4aGS25pku4/VHH7kYcQFDI/AAAAAAAAGJA/6sYlGFQX-kA/s1600/ThingSpeak%2B-%2BChannel%2BPrivate%2BView-M-blog.png](http://3.bp.blogspot.com/-F4aGS25pku4/VHH7kYcQFDI/AAAAAAAAGJA/6sYlGFQX-kA/s1600/ThingSpeak%2B-%2BChannel%2BPrivate%2BView-M-blog.png) |
| 檢視測試輸入資料的結果 |

使用上面剛剛建立的資料輸入方式，只要更改 **field1=** 與 **field2=** 後面的數值，就可以將資料上傳到 ThingSpeak 所建立的 Channel 相對應的資料欄位中，容易吧！  
  
這段建立的資料格式將會使用到 Arduino 程式碼中，負責將整段格式移交給 ESP8266 無線模組來處理。但是瀏覽器傳送時是使用 HTTP 的方法傳送，類似於我們在網頁輸入表單資料後，需要按下表單按鈕以傳送資料到伺服器去，伺服器再根據資料回傳給我們，所以並不是直接輸入上面的格式就可以。這就是我們下一段要討論的 HTTP Method：GET 和 POST 方法。  
  
  
**HTTP 方法：POST 和 GET**  
  
ThingSpeak 提供了使用 Arduino 配合有線網路模組的[例子](http://community.thingspeak.com/tutorials/arduino/using-an-arduino-ethernet-shield-to-update-a-thingspeak-channel/)，傳送的 HTTP 方法是 POST，與上面所做測試的方法是不一樣的。因為使用 POST 是看不到傳送的欄位資料出現在瀏覽器的網址列中，所以在上面傳送資料的 HTTP 方法是使用 GET。  
  
這兩種 HTTP 方法主要的不同在於，  
  
**GET：查詢字串 ( name / value pairs，名稱  / 值對 ) 是在 GET 請求的 URL 中發送的**

* 請求數據長度有限制

**POST：查詢字串 ( name / value pairs，名稱 / 值對 ) 是在 POST 請求的消息主體中發送的**

* 請求數據長度沒有要求

*上面只簡單列出跟網頁相關最重要的，其他的可參考 "*[*淺談 HTTP Method：表單中的 GET 與 POST 有什麼差別？*](http://blog.toright.com/posts/1203/%E6%B7%BA%E8%AB%87-http-method%EF%BC%9A%E8%A1%A8%E5%96%AE%E4%B8%AD%E7%9A%84-get-%E8%88%87-post-%E6%9C%89%E4%BB%80%E9%BA%BC%E5%B7%AE%E5%88%A5%EF%BC%9F.html)*" 這篇網頁中的說明。*  
  
ThingSpeak 裡面所使用的例子是使用 POST 發送數據，會換成使用 GET 的方式是因為他的傳送格式簡單，只要一行就可以搞定；若是利用 POST 要傳送一堆標頭才能傳送本體 ( 資料放在此 )，有點浪費時間！尤其是在微處理機使用上，盡量簡單就好。等到那一天發現到這方法不適用時，再換成 POST 方式就好。  
  
  
**佈線電路圖說明：**

|  |
| --- |
| [https://1.bp.blogspot.com/-MiG5s6RLQ88/WGp2v0G9xfI/AAAAAAAAKf8/WFA0cWc4x48SOlOGeDsUwnxLMtE77wEyACPcB/s640/IoT_Demo_Schematics_crop_v0.2.png](https://1.bp.blogspot.com/-MiG5s6RLQ88/WGp2v0G9xfI/AAAAAAAAKf8/WFA0cWc4x48SOlOGeDsUwnxLMtE77wEyACPcB/s1600/IoT_Demo_Schematics_crop_v0.2.png) |
| IoT\_Demo 電路圖 V0.2 - 包括 UART 除錯 |

測試時，Arduino 是直接插在筆電，與筆電通訊和供電，所有外部模組和零件所用到的，不管是 3V3 或是 5V電源都是來自 Arduino。雖然我這樣測試是正常的，但不代表其他人就會一樣！若出現問題，請先確認 ESP8266 無線模組所供應的電源是否在範圍內；簡單一點的方法就是換成套件中的電源模組做 ESP8266 的 3V3 電源，這會解決大部分因為電源不穩定所引起的問題。  
  
上面的電路圖中，USB 轉 TTL 模組是用來做除錯用的。程式執行的過程中會送出除錯的訊息，依但程式出現問題時才抓的到；硬體 UART 用來與 ESP8266 直接通訊用，程式寫起來會輕鬆許多。  
  
**所需材料：**

* 1  x  pcs  [ Arduino 板子 ]
* 1  x  pcs  [ ESP8266, ESP-01 無線模組 ]
* 1  x  pcs  [ [USB 轉 TTL 模組](http://goods.ruten.com.tw/item/show?21447787125749) ]
* 1  x  pcs  [ [DHT11 ( 溫溼度感測器 )](http://goods.ruten.com.tw/item/show?21211203887342) ]
* 1  x  pcs  [ [Level Shifter ( 3V3 / 5V 電壓準位轉換模組 )](http://goods.ruten.com.tw/item/show?21527956141979) ]
* 1  x  pcs  [ 4.7 K 電阻 ]
* 1  x  pcs  [ 麵包板 ]，哪一種形式都可以，能用就好！
* 杜邦線 ( 母對母，公對公 )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
*如果需要中間那個電壓轉換模組，可以依照 "*[*使用 N-Channel MOSFET 做 I2C 電壓準位轉換電路*](http://ruten-proteus.blogspot.tw/2014/06/N-MOSFET-I2C-Level-Shifter.html)*" 網頁中的說明自己佈線，或是到下面連結購買*

* [*四通道雙向邏輯電壓準位(適用1.8V,2.8V,3.3V,5.0V )轉換模組(支援 I2C, SPI, 一般邏輯準位)*](http://goods.ruten.com.tw/item/show?21527956141979)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  |
| --- |
| [https://4.bp.blogspot.com/-ji_aFCNrb68/VHF3BoYgdvI/AAAAAAAAGGY/mMbReznMyrg/s1600/DSC03572-01-blog.JPG](http://4.bp.blogspot.com/-ji_aFCNrb68/VHF3BoYgdvI/AAAAAAAAGGY/mMbReznMyrg/s1600/DSC03572-01-blog.JPG) |
| 實際接線圖 |

**程式測試說明：**  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
***2016/12/20 更新：***  
*由於使用的感測器並不一定會與網頁上使用的 ( DHT11 ) 相同，因此並未將所使用的 DHT11 函式庫連結附上，主要是考量到這只是 ESP8266 的應用，但因為常有私信問我要，所以檔案連結在下面，不需要再寫信跟我要了！*  
  
[*DHT11 Arduino Library (最新版)*](http://bit.ly/1TjMosn)  
  
*若上面最新版本使用有問題，可改用下面舊版本函式庫使用*  
[*DHT11 Arduino Library (網頁程式使用的舊版本 DHT11 函式庫)*](http://bit.ly/2ibGvSA)  
  
*直接複製到 "****文件/{Arduino IDE}/libraries****" 目錄下，重新開啟 Arduino IDE 即可！*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

接好線路之後 ( USB 轉 TTL 模組的線路可以不接，除非想要看到實際的輸出情形，或是需要除錯 )，完整的程式碼列在下面 ( 雲端硬碟裡面的程式碼裡面有特別做註解 )，將其複製到 Arduino IDE 中。

Note：請注意使用的 ESP8266 板子的通訊速率是使用多少 ? 若是購買賣場藍色板子，就不需要修改下面的程式碼可以直接使用；但若是使用黑色板子，下面的 ***#define \_baudrate***要改成 ***115200***

#include <SoftwareSerial.h>

#include "DHT.h"

#define DEBUG

#define \_ledpin 13

*//\*-- Hardware Serial*

#define \_baudrate 9600

*//\*-- Software Serial*

*//*

#define \_rxpin 2

#define \_txpin 3

SoftwareSerial debug( \_rxpin, \_txpin ); *// RX, TX*

*//\*-- DHT11*

#define \_dhtpin 8

#define \_dhttype DHT11

DHT dht11( \_dhtpin, \_dhttype );

uint8\_t dhtbuf[2];

*//\*-- IoT Information*

#define SSID "[無線網路熱點名稱]"

#define PASS "[無線網路連線密碼]"

#define IP "184.106.153.149" *// ThingSpeak IP Address: 184.106.153.149*

*// 使用 GET 傳送資料的格式*

*// GET /update?key=[THINGSPEAK\_KEY]&field1=[data 1]&filed2=[data 2]...;*

String GET = "GET /update?key=[ThingSpeak\_(Write)API\_KEY]";

void setup() {

Serial.begin( \_baudrate );

debug.begin( \_baudrate );

sendDebug("AT");

delay(5000);

if(Serial.find("OK"))

{

debug.println("RECEIVED: OK\nData ready to sent!");

connectWiFi();

}

// DHT11

dht11.begin();

pinMode( \_ledpin, OUTPUT );

digitalWrite( \_ledpin, LOW );

}

void loop() {

dhtbuf[0] = dht11.readHumidity();

dhtbuf[1] = dht11.readTemperature();

// 確認取回的溫溼度數據可用

if( isnan(dhtbuf[0]) || isnan(dhtbuf[1]) )

{

debug.println( "Failed to read form DHT11" );

}

else

{

digitalWrite( \_ledpin, HIGH );

char buf[3];

String HH, TT;

buf[0] = 0x30 + dhtbuf[1] / 10;

buf[1] = 0x30 + dhtbuf[1] % 10;

TT = (String(buf)).substring( 0, 2 );

buf[0] = 0x30 + dhtbuf[0] / 10;

buf[1] = 0x30 + dhtbuf[0] % 10;

HH = (String(buf)).substring( 0, 2 );

updateDHT11( TT, HH );

#ifdef DEBUG

debug.print("Humidity: ");

debug.print( HH );

debug.print(" %\t");

debug.print("Temperature: ");

debug.print( TT );

debug.println(" \*C\t");

#endif

digitalWrite( \_ledpin, LOW );

}

delay(60000); // 60 second

}

void updateDHT11( String T, String H )

{

// 設定 ESP8266 作為 Client 端

String cmd = "AT+CIPSTART=\"TCP\",\"";

cmd += IP;

cmd += "\",80";

sendDebug(cmd);

delay(2000);

if( Serial.find( "Error" ) )

{

debug.print( "RECEIVED: Error\nExit1" );

return;

}

cmd = GET + "&field1=" + T + "&field2=" + H +"\r\n";

Serial.print( "AT+CIPSEND=" );

Serial.println( cmd.length() );

if(Serial.find( ">" ) )

{

debug.print(">");

debug.print(cmd);

Serial.print(cmd);

}

else

{

sendDebug( "AT+CIPCLOSE" );

}

if( Serial.find("OK") )

{

debug.println( "RECEIVED: OK" );

}

else

{

debug.println( "RECEIVED: Error\nExit2" );

}

}

void sendDebug(String cmd)

{

debug.print("SEND: ");

debug.println(cmd);

Serial.println(cmd);

}

boolean connectWiFi()

{

Serial.println("AT+CWMODE=1");

delay(2000);

String cmd="AT+CWJAP=\"";

cmd+=SSID;

cmd+="\",\"";

cmd+=PASS;

cmd+="\"";

sendDebug(cmd);

delay(5000);

if(Serial.find("OK"))

{

debug.println("RECEIVED: OK");

return true;

}

else

{

debug.println("RECEIVED: Error");

return false;

}

cmd = "AT+CIPMUX=0";

sendDebug( cmd );

if( Serial.find( "Error") )

{

debug.print( "RECEIVED: Error" );

return false;

}

}

整段程式中最重要的就是剛剛在 ThingSpeak 所得到的結果，還有就是無線網路名稱  (SSID) 與連線密碼 ( PASS )，這幾個重要的設定都在程式開頭 IoT Information 的地方。  
  
不過除了SSID 和 PASS 修改為你的無線網路 AP 之外，其他的不需要做修改！

*//\*-- IoT Information*

#define SSID "[無線網路熱點名稱]"

#define PASS "[無線網路連線密碼]"

#define IP "184.106.153.149" *// ThingSpeak IP Address: 184.106.153.149*

*// 使用 GET 傳送資料的格式*

*// GET /update?key=[THINGSPEAK\_KEY]&field1=[data 1]&filed2=[data 2]...;*

String GET = "GET /update?key=[ThingSpeak\_(Write)API\_KEY]";

插上 Arduino 板子的線到電腦的 USB 插槽，設定好 Arduino IDE 板子類型 ( 和晶片類型 ) 和通訊埠號碼後，按下 **upload**按鈕上傳此 **Sketch**到 Arduino 板子上。  
  
如果有裝上 USB 轉 TTL 模組，可以先將其插到 PC，再打開串列埠軟體等待 Arduino 通電之後輸出的數據。  
  
開機之後得到的數據就如下面所示，遮蔽的部分是私有的資料，執行你自己的程式碼之後就會看到完整的輸出

|  |
| --- |
| [https://3.bp.blogspot.com/-j0KeHGukcqE/VHF3C0aEUsI/AAAAAAAAGGw/JjIoRiDT22M/s1600/ThingSpeak_esp826_dht11_log_M-blog.png](http://3.bp.blogspot.com/-j0KeHGukcqE/VHF3C0aEUsI/AAAAAAAAGGw/JjIoRiDT22M/s1600/ThingSpeak_esp826_dht11_log_M-blog.png) |
| 訊息輸出 |

因為每隔 60 秒才會上傳一次，因此可以過一段時間再上去 ThingSpeak 看。下面是過了差不多20幾分鐘後的數據，我放在電腦的出風口並在旁邊開電風扇循環，不然溫度會變化不大！

|  |
| --- |
| [https://2.bp.blogspot.com/-1P1JsxTe9Rs/VHF3CWkw-rI/AAAAAAAAGGs/btzUKd94RSc/s1600/ThingSpeak_IoT_Demo-blog.png](http://2.bp.blogspot.com/-1P1JsxTe9Rs/VHF3CWkw-rI/AAAAAAAAGGs/btzUKd94RSc/s1600/ThingSpeak_IoT_Demo-blog.png) |
| ThingSpeak 環境溫濕度記錄圖 |

結論：  
  
完成這個例子之後，應該對於程式碼裡面的ESP8266 AT 指令有點陌生！但是這不要緊，因為這就是一個實際例子，先知道 ESP8266 無線模組可以做些什麼，再來了解 AT 指令的操作會更容易了解！  
  
ESP8266 無線模組，可當作一般串口轉無線模組使用，也可以利用它開放的 SDK 修改或是自己製作它的韌體，開發成一個具有無線網路功能的產品或是項目，相關的資料會持續地發佈在部落格上與大家做分享！  
  
相信使用一段時間的 AT 指令之後，對於使用 Arduino IDE 與 AT 指令的配合或許不是那麼的滿意！那麼可以試試使用 ESP8266 Arduino AT 函式庫，相關的資料可以參考下面這三個網頁中的說明，與此篇網頁程式的加強版

* [自行更新 ESP8266  無線模組韌體至 - AT：1.2.0.0, SDK：1.5.4.1](http://ruten-proteus.blogspot.com/2016/12/esp8266-at1200-sdk1541.html)
* [給我一個可操作 AT 指令的 Arduino 函式庫 - 解決 AT 指令處理的蛋疼問題 (親測 AT v1.2.0.0 韌體)](http://ruten-proteus.blogspot.com/2017/01/arduino-at1200-lib-recommend.html)
* [初遇 IoT ( Internet of Thing, 物聯網 ) 加強版 - 使用 ESP8266 Arduino AT 函式庫上傳溫溼度資料到 Thingspeak ( AT v1.2.0.0 based on SDK 1.5.4.1)](http://ruten-proteus.blogspot.com/2017/01/esp8266-at-lib-iot-demo-thingspeak.html)

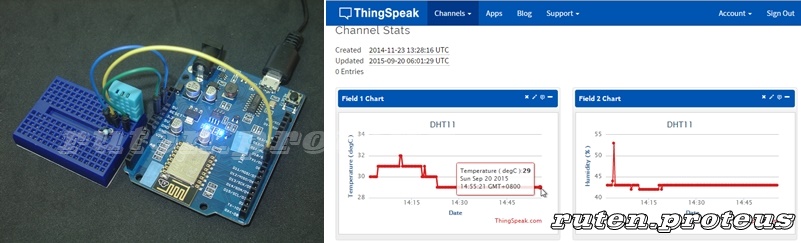
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
***2017/01/03 更新：***  
*因為很多問與答都是關於無法上傳資料到 Thingspeak 的問題，所以在這邊做個總結回覆，大抵上這個網頁中的問題就是這樣解決。*  
 *這個問題除了接線的問題之外，就是在程式執行之前必須確定在網址列手動輸入資料是可以正確成功的，再來就是修改程式碼需要的欄位，基本上就不會出現問題 ! 因為手動可以，程式碼就可以正確地將你的資料網上傳，所以只要出問題，就是檢查剛剛修改的欄位是否出現錯誤，因為這並不是程式出現問題，而是輸入參數設定錯誤造成上傳失敗。在相同的'網路環境之下，只是將手動變成自動而已，出現問題要先檢查這些部分 !*  
 *上述的問題都解決之後，另外會出現的問題可能就是上傳一段時間之後會失敗會是回傳 error 的問題。最近一位網友提問了這個問題，並且問到了重點，所以進行了幾封郵件的來回 (原文如*[*附件*](http://bit.ly/2hLkaQd)*(若連結失效，請回報)，文中遮蔽私人資料)，若有相同問題可看裡面的討論 !*  
 *"*[*給我一個可操作 AT 指令的 Arduino 函式庫 - 解決 AT 指令處理的蛋疼問題 (親測 AT v1.2.0.0 韌體)*](http://ruten-proteus.blogspot.tw/2017/01/arduino-at1200-lib-recommend.html)*" 網頁中，對於程式範例也是採用這個方式解決亂碼或是通訊失敗的問題。*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<< 部落格相關文章 >>

* [初遇 IoT ( Internet of Thing, 物聯網 ) 加強版 - 使用 ESP8266 Arduino AT 函式庫上傳溫溼度資料到 Thingspeak ( AT v1.2.0.0 based on SDK 1.5.4.1)](http://ruten-proteus.blogspot.com/2017/01/esp8266-at-lib-iot-demo-thingspeak.html)
* [給我一個可操作 AT 指令的 Arduino 函式庫 - 解決 AT 指令處理的蛋疼問題 (親測 AT v1.2.0.0 韌體)](http://ruten-proteus.blogspot.com/2017/01/arduino-at1200-lib-recommend.html)
* [自行更新 ESP8266  無線模組韌體至 - AT：1.2.0.0, SDK：1.5.4.1](http://ruten-proteus.blogspot.com/2016/12/esp8266-at1200-sdk1541.html)
* [ESP8266 ( ESP-01 ) 無線模組燒錄 NodeMCU 韌體 ( Lua 語法支援 )](http://ruten-proteus.blogspot.com/2015/05/esp8266-lua-nodemcu-firmware-update.html)
* [自行更新 ESP8266 , ESP-01 無線模組韌體至 - AT：0.23, SDK：1.0.1](http://ruten-proteus.blogspot.com/2015/05/esp8266-esp-01-at023-sdk101.html)
* 初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )
* [操控 ESP8266 無線模組 - 經由 AP、STA 和 AP+STA 三種模式，學習 ESP8266 AT 指令](http://ruten-proteus.blogspot.com/2014/12/esp8266-at-command.html)
* [如何燒錄 ESP8266 無線模組的韌體 ( 使用一鍵燒錄軟體，燒錄版本：0018000902 )](http://ruten-proteus.blogspot.com/2014/12/esp8266-firmware-burning.html)
* [初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )](http://ruten-proteus.blogspot.com/2014/11/internet-of-thing-arduino-esp8266.html)
* [使用 Arduino IDE 開發 ESP8266 物聯網應用 - ThingSpeak, HTTP GET / POST 資料上傳方法](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-send.html)
* [使用 Arduino IDE 開發 ESP8266 物聯網應用 - 取回 ThingSpeak 特定 Channel 和 Field 最後一筆資料](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-retrieve.html)
* [ESP8266 入門學習套件支援 Arduino IDE 開發環境之安裝、使用說明與範例](http://ruten-proteus.blogspot.com/2015/09/esp8266-kits-support-arduino-ide.html)

### 使用 Arduino IDE 開發 ESP8266 物聯網應用 - ThingSpeak, HTTP GET / POST 資料上傳方法

*網頁最後修改時間：2015/09/22*

[](http://1.bp.blogspot.com/-OrfuwII3yHQ/Vf9QXrDPJMI/AAAAAAAAHRA/yYu1ghtxhlo/s1600/ESP-ArduIDE-Thingspeak-blog.jpg)

在部落格的另一篇關於物聯網應用的[網頁](http://ruten-proteus.blogspot.com/2014/11/internet-of-thing-arduino-esp8266.html)中，已經說明了使用 HTTP GET 上傳感測器資料到到 IoT ( Internet-of-Thing, 物聯網 ) Server ( 這裡以 [ThingSpeak](http://thingspeak.com/) 為例 ) 的方法 ( HTTP POST 的方法請查閱該網頁所提供的參考連結 )。而在此篇網頁，將延續在 Arduino IDE 撰寫程式碼的方式，但會捨棄使用 Arduino 板，改用 ESP8266 做為微控制器與週邊裝置通訊，撰寫與說明如何在 Arduino IDE 撰寫 ESP8266 使用 [HTTP GET 和 POST](http://blog.toright.com/posts/1203/) 上傳資料到 ThingSpeak.com 的 Sketch！  
  
看過此篇之後就會發現到，使用 [**ESP8266 Arduino IDE 開發板**](http://goods.ruten.com.tw/item/show?21537656243813) ( 下面簡稱：**開發板** )

* ESP8266 無線網路模組就是微控制器
* 不需要額外處理與設置 AT 指令
* 利用熟悉的 Arduino 語法開發 ESP8266 就像開發其他 Arduino 板子一樣簡單
* ...

更多、更多好處等著使用者自己去發掘 !

*/\*-/--\*-\*/\*/\*/\*/\*\*\*//-\*-\*-\*\*-\*/\*-\*-/\*/\*/\*-\*-/-////--/\*\*/\*\*--\*\*/--///--//\*\*----\*\*//--\*\*//\*\*----\*\*\*//\*-\*\*//\**  
有購買商品的使用者，網頁中所需相關資料已放置於雲端硬碟，請自行下載使用！  
其餘的使用者，請自行依照提供之連結下載相關資料，程式碼複製貼上使用！  
*/\*-/--\*-\*/\*/\*/\*/\*\*\*//-\*-\*-\*\*-\*/\*-\*-/\*/\*/\*-\*-/-////--/\*\*/\*\*--\*\*/--///--//\*\*----\*\*//--\*\*//\*\*----\*\*\*//\*-\*\*//\**  
  
*NOTE：本篇網頁所使用的程式經過*[***ESP8266 UNO Arduino IDE 開發板 (Type B1)***](http://goods.ruten.com.tw/item/show?21537656243813)*測試通過！不過，同樣適用於其他使用 Arduino IDE 開發的* ***ESP8266, ESP-##*** *型號的板子，但請自行測試！*  
*除了與網頁內容相關的討論之外，像是 Arduino IDE 與其他 ESP8266 型號連線的問題等，請使用者自行搞定！！！*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
本網頁所使用到的零件可至下面連結訂購：

* [ESP8266 UNO Arduino IDE 開發板 (Type B1) - ESP-12F, 32MBits](http://goods.ruten.com.tw/item/show?21537656243813)
* [ESP8266 Nano Arduino IDE 開發板 (Type A1) - ESP8266EX、CP2102、32MBits](http://goods.ruten.com.tw/item/show?21741203493792)
* DHT11 - 溫濕度感測器 (可用於 Raspberry Pi, Arduino, 單晶片, 8051, AVR)

ESP8266 相關商品可至下面連結訂購：

* [萬物皆聯網-ESP8266 IoT(Internet of Things)入門學習套件](http://goods.ruten.com.tw/item/show?21447787125749)
* [ESP8266, UART 轉 WiFi 透傳模組 ( 樹莓派、Arduino、單晶片、微控制器可用 )](http://goods.ruten.com.tw/item/show?21538692035419)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**開發板在 Arduino IDE 中的設置：**

*進行下面設置之前，使用必須已經在 Arduino IDE 安裝好 ESP8266 的硬體套件 ( Hardware Package )；有購買開發板的使用者，請參閱賣場所提供的套件安裝說明手冊進行安裝！*

在 Arduino IDE 的選單 "Tools" 設置下面參數，使用者可以根據需要進行修改 ( 若是沒自行修改過，應該跟下面參數設置一樣 )，一般維持與下面相同的預設值就好

* Upload Using: "Serial"
* CPU Frequency: 80 MHz
* Upload Speed: 460800
* Port: COM#  
  選擇開發板連接的哪一個 COM PORT 號碼

**ThingSpeak 的設定：**  
  
參考 "[初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )](http://ruten-proteus.blogspot.com/2014/11/internet-of-thing-arduino-esp8266.html)" 網頁，完成下面這三個部分：

* 申請一個 ThingSpeak 帳號
* 建立一個給 DHT11 使用的 Channel
* 取得剛剛建立的 ThingSpeak Channel 的( Write ) API KEY，並手動測試

因為這篇網頁是上篇網頁中的接續，所以使用相同的設定，但資料在測試之前已全部清除！  
  
**線路：**  
  
比較一下就可以發現！相對於使用 Arduino 板作為微控制器，開發板的接線反而簡化許多。**DHT11** 只需要三條跳線與一支上拉用的  ( **4.7K ~ 10 K** ) 電阻連接到開發板就完成整個佈線，然後將開發板使用手機的 MicroUSB 線與電腦連接就可以開始撰寫與上傳程式。  
  
如下圖。藍線為 **DHT11** 的第**一**支接腳 ( **VCC** ) 接至開發板的 **3.3V** 接腳；黃線為 **DHT11** 的第**二**支接腳 ( **OUT** ) 接至開發板的 **D8** 接腳；綠線為 **DHT11** 的第**四**支接腳 ( **GND** ) 接至開發板的 **GND** 接腳。最後將電阻分別接到 **DHT11** 的第**一**支與第**二**支接腳就完成整個接線。

|  |
| --- |
| [https://4.bp.blogspot.com/-40w1mdSJc2M/Vf9qlClQUVI/AAAAAAAAHRY/V1s6_SXXSLE/s640/DSC03661blog.JPG](http://4.bp.blogspot.com/-40w1mdSJc2M/Vf9qlClQUVI/AAAAAAAAHRY/V1s6_SXXSLE/s1600/DSC03661blog.JPG) |
| 實際完成之接線圖 |

此時要不要先接上 MicroUSB 都可以！  
  
**程式碼說明：**  
  
下面分別以 HTTP GET 和 HTTP POST 兩個方式，說明如何使用 ESP8266 Arduino IDE 開發板撰寫上傳資料到 ThingSpeak 的程式碼；由於兩個程式碼只差別在上傳的格式，所以只給出其中一份完整的 ( HTTP GET ) 程式，另外一份 ( HTTP POST ) 則只給出差異的部分，請使用者自行修改其中不同之處。  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.  
*在 "*[*初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )*](http://ruten-proteus.blogspot.com/2014/11/internet-of-thing-arduino-esp8266.html)*" 網頁中所使用的 DHT11 的函式庫只適用於 Arduino 板子，因為沒有根據微控制器調整速度的功能，所以不適用於 ESP8266 為主的微控制器，因此若系統中已經安裝賣場所提供的 DHT11 函式庫，則請先移除或是移動出 {Arduino}/libraries 目錄，或安裝下面提供的另一版本的 DHT11 函式庫*

[*DHT Sensor Library V1.2.0*](http://bit.ly/1Kvu2zp)

*函式庫的使用可參考下面的範例。*  
 *NOTE: 我是移除掉現有的 DHT 函式庫，使用開發板的硬體套件中所包含的 DHT 函式庫，這兩者經過比較都是相同的，安不安裝取決於使用者的選擇，不影響程式碼的編譯！*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
下面是以 HTTP GET 方法與硬體套件中所帶的 DHT 函式庫所撰寫的 Arduino Sketch，修改至  
"[初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )](http://ruten-proteus.blogspot.com/2014/11/internet-of-thing-arduino-esp8266.html)" 網頁中的例子。比較兩者之間的差異，就會發現其中不同之處！好不好使，見"人"見智！

#include "DHT.h" *// 使用 ESP8266 自帶的 DHT11 函式庫*

*// 如果 libraries 已有其他的 DHT 函式庫，請先移除*

#include <ESP8266WiFi.h>

*//\*-- Hardware Serial*

#define \_baudrate 9600

*//\*-- DHT11*

#define \_dhtpin D8

#define \_dhttype DHT11

*// Initialize DHT sensor*

DHT dht11( \_dhtpin, \_dhttype );

float temperature, humidity;

*//\*-- IoT Information*

#define SSID "路由器名稱"

#define PASS "路由器連線密碼"

#define HOST "api.thingspeak.com" *// ThingSpeak IP Address: 184.106.153.149*

#define PORT 80

*// 使用 GET 傳送資料的格式*

*// GET /update?key=[THINGSPEAK\_KEY]&field1=[data 1]&filed2=[data 2]...;*

String GET = "GET /update?key=0123456789ABCDEF";

void setup() {

Serial.begin( \_baudrate );

Serial.println( "ESP8266 Ready!" );

// Connecting to a WiFi network

Serial.print("Connect to ");

Serial.println( SSID );

WiFi.begin( SSID, PASS );

// 持續等待並連接到指定的 WiFi SSID

while( WiFi.status() != WL\_CONNECTED )

{

delay(500);

Serial.print( "." );

}

Serial.println( "" );

Serial.println( "WiFi connected" );

Serial.println( "IP address: " );

Serial.println( WiFi.localIP() );

Serial.println( "" );

// DHT11

dht11.begin();

Serial.println( "DHT11 Ready!" );

delay(2000);

}

void loop() {

humidity = dht11.readHumidity();

temperature = dht11.readTemperature();

// 確認取回的溫溼度數據可用

if( isnan( humidity ) || isnan( temperature ) )

{

Serial.println( "Failed to read form DHT11" );

return;

}

else

{

// DHT11 溫度與濕度傳送

Serial.print( "Humidity: " );

Serial.print( humidity );

Serial.print( ", Temperature: " );

Serial.println( temperature );

updateDHT11();

}

// 每隔多久傳送一次資料

delay( 20000 ); // 20 second

}

void updateDHT11()

{

// 設定 ESP8266 作為 Client 端

WiFiClient client;

if( !client.connect( HOST, PORT ) )

{

Serial.println( "connection failed" );

return;

}

else

{

// 準備上傳到 ThingSpeak IoT Server 的資料

// 已經預先設定好 ThingSpeak IoT Channel 的欄位

// field1：溫度；field2：濕度

String getStr = GET + "&field1=" + String((int)temperature) +

"&field2=" + String((int)humidity) +

" HTTP/1.1\r\n";;

client.print( getStr );

client.print( "Host: api.thingspeak.com\n" );

client.print( "Connection: close\r\n\r\n" );

delay(10);

//

// 處理遠端伺服器回傳的訊息，程式碼可以寫在這裡！

//

client.stop();

}

}

程式裡面，拿掉了原先 debug 的串列通訊，取而代之的！是使用硬體串列埠作為訊息(或除錯)用，節省了一個 USB 轉 TTL 模組。  
  
程式中需要注意的及修改的地方，其中  
  
**\_baudrate** 代表的是電腦與開發板的通訊鮑率，可以設定不同的值！

*//\*-- Hardware Serial*

#define \_baudrate 9600

**SSID** 與 **PASS** 是路由器的名稱與密碼；**0123456789ABCDEF** 是要傳送到 ThingSpeak 的 Channel 中的 Write API KEY，記得修改成自己的

//\*-- IoT Information

*#define SSID "路由器名稱"*

*#define PASS "路由器連線密碼"*

*#define HOST "api.thingspeak.com" // ThingSpeak IP Address: 184.106.153.149*

*#define PORT 80*

// 使用 GET 傳送資料的格式

// GET /update?key=[THINGSPEAK\_KEY]&field1=[data 1]&filed2=[data 2]...;

String GET = "GET /update?key=0123456789ABCDEF";

/\*--\*//\*\*---/\*///\*\*---\*-\*////\*\*\*--\*/\*///\*\*\*----\*///--\*/\*///\*\*--\*/\*//\*\*--\*\*/\*//

**\* HTTP POST 修改部分：**

*//\*-- IoT Information*

#define SSID "路由器名稱"

#define PASS "路由器連線密碼"

#define HOST "api.thingspeak.com" *// ThingSpeak IP Address: 184.106.153.149*

#define PORT 80

String APIKEY = "NAQE1STDCH6ZPKFG";

void updateDHT11()

{

*// 設定 ESP8266 作為 Client 端*

WiFiClient client;

if( !client.connect( HOST, PORT ) )

{

Serial.println( "connection failed" );

return;

}

else

{

*// 準備上傳到 ThingSpeak IoT Server 的資料*

*// 已經預先設定好 ThingSpeak IoT Channel 的欄位*

*// field1：溫度；field2：濕度*

String postStr = APIKEY + "&field1=" + String((int)temperature) + \

"&field2=" + String((int)humidity) + "\r\n\r\n";

client.print( "POST /update HTTP/1.1\n" );

client.print( "Host: api.thingspeak.com\n" );

client.print( "Connection: close\n" );

client.print( "X-THINGSPEAKAPIKEY: "+APIKEY+"\n" );

client.print( "Content-Type: application/x-www-form-urlencoded\n" );

client.print( "Content-Length: " );

client.print( postStr.length() );

client.print( "\n\n" );

client.print( postStr );

client.stop();

}

}

/\*--\*//\*\*---/\*///\*\*---\*-\*////\*\*\*--\*/\*///\*\*\*----\*///--\*/\*///\*\*--\*/\*//\*\*--\*\*/\*//

上面參數修改好了之後，將開發板接上電腦，並按下 ***"Upload"*** 按鈕編譯與上傳程式碼。成功連接和開始傳輸時，就會出現如下輸出畫面直到 **100%** 並出現 **Leaving...** 完成上傳

|  |
| --- |
| [https://4.bp.blogspot.com/-6dO7Qgo2oy4/Vf-MheD1wpI/AAAAAAAAHRo/u3bykhPoUcM/s640/upload-01blog.png](http://4.bp.blogspot.com/-6dO7Qgo2oy4/Vf-MheD1wpI/AAAAAAAAHRo/u3bykhPoUcM/s1600/upload-01blog.png) |
| 開發板編譯、上傳輸出畫面 |

打開 ***Serial Monitor*** ( 記得選擇程式碼所設定的 **\_baudrate** ) ，再按下板子上面的 ***RESET1*** 按鈕，就會看到如下的畫面輸出 ( 開始會有一小段亂碼是正常的，下圖前頭沒有亂碼是嘗試幾次試出來的！)：

|  |
| --- |
| [https://4.bp.blogspot.com/-HAOcWcIxBe4/Vf-Q65ae3CI/AAAAAAAAHR8/yXjQlIYrnVk/s1600/Serial-monitor-01blog.png](http://4.bp.blogspot.com/-HAOcWcIxBe4/Vf-Q65ae3CI/AAAAAAAAHR8/yXjQlIYrnVk/s1600/Serial-monitor-01blog.png) |
| Serial Monitor 輸出畫面 |

上傳資料到 ThingSpeak，必須注意，上傳的間隔要在 **15** 秒以上！(程式每 20 秒上傳一次)  
  
經過一段時間的資料上傳之後，前往 ThingSpeak 所創建的 Channel 中，就會看到類似下面的圖形

|  |
| --- |
| [https://2.bp.blogspot.com/-rSCeOr_F4Ek/Vf-MqCXwzlI/AAAAAAAAHRw/NjBgxhlziCE/s1600/Channel_chartsblog.png](http://2.bp.blogspot.com/-rSCeOr_F4Ek/Vf-MqCXwzlI/AAAAAAAAHRw/NjBgxhlziCE/s1600/Channel_chartsblog.png) |
| ThingSpeak 環境溫濕度記錄圖 |

**結論：**  
  
使用 ESP8266 Arduino IDE 開發板來開發物聯網相關應用非常方便！但是相比於 Arduino 板來說，函式庫支援數量相對較低，但持續開發與增加中，長期來說對於開發物聯網原型應用來說，還是具有相當大的吸引力與前景！  
  
另外關於 HTTP POST 與 HTTP GET 上傳的方法，差異在於： GET 速度比較快，但上傳的資料可以被看到；反之使用 POST 用在傳輸隱密性較高且資料量大的情況之下。要用哪一個 ? 使用者自己決定 ！  
  
建議使用者能夠比較相關網頁間的程式碼，了解達到相同目的的程式碼之間不同之處，這就是寫這篇網頁的目的！

<< 部落格相關網頁連結 >>

* [初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )](http://ruten-proteus.blogspot.com/2014/11/internet-of-thing-arduino-esp8266.html)
* [使用 Arduino IDE 開發 ESP8266 物聯網應用 - ThingSpeak, HTTP GET / POST 資料上傳方法](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-send.html)
* [使用 Arduino IDE 開發 ESP8266 物聯網應用 - 取回 ThingSpeak 特定 Channel 和 Field 最後一筆資料](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-retrieve.html)

**使用 Arduino IDE 開發 ESP8266 物聯網應用 - 取回 ThingSpeak 特定 Channel 和 Field 最後一筆資料**

*網頁最後修改時間：2015/09/22*

|  |
| --- |
| [https://1.bp.blogspot.com/-IG906BwzibI/Vf-vqfjvOBI/AAAAAAAAHSM/SSj7_LstQjE/s640/retrieve-field1-serial-monitor-blog.jpg](http://1.bp.blogspot.com/-IG906BwzibI/Vf-vqfjvOBI/AAAAAAAAHSM/SSj7_LstQjE/s1600/retrieve-field1-serial-monitor-blog.jpg) |
|  |

上一篇 "[使用 Arduino IDE 開發 ESP8266 物聯網應用 - ThingSpeak, HTTP GET / POST 資料上傳方法](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-send.html)"網頁中提到了使用 HTTP GET 與 HTTP POST 上傳資料到 ThingSpeak IoT Server 的方法；這一篇反過來，要說明如何取回 ThingSpeak IoT Server 特定 Channel 和 Field 上**最後一筆**資料。  
  
*ps. 其實寫到最後發現！只有一個情況下，不管如何只能取回最後一筆資料；但另外一個情況則是，相同的程式碼處理下，取回多筆資料會比取回最後一筆資料來的容易解釋！*  
  
為什麼是最後一筆，而不是全部或是其中一點的資料 ? 原因是：除非在資料集中的每一個數據都很重要，不然最後一筆資料就是最重要的數據；另外一點，就是語法都差不多，差異在於後面所接的參數不同而已，只要看過下面所舉的例子之後，舉一反三絕不會是問題！  
  
*/\*-/--\*-\*/\*/\*/\*/\*\*\*//-\*-\*-\*\*-\*/\*-\*-/\*/\*/\*-\*-/-////--/\*\*/\*\*--\*\*/--///--//\*\*----\*\*//--\*\*//\*\*----\*\*\*//\*-\*\*//\**  
有購買商品的使用者，網頁中所需相關資料已放置於雲端硬碟，請自行下載使用！  
其餘的使用者，請自行依照提供之連結下載相關資料，程式碼複製貼上使用！  
*/\*-/--\*-\*/\*/\*/\*/\*\*\*//-\*-\*-\*\*-\*/\*-\*-/\*/\*/\*-\*-/-////--/\*\*/\*\*--\*\*/--///--//\*\*----\*\*//--\*\*//\*\*----\*\*\*//\*-\*\*//\**

*NOTE：本篇網頁所使用的程式經過* [***ESP8266 UNO Arduino IDE 開發板 (Type B1)***](http://goods.ruten.com.tw/item/show?21537656243813)***(  本文簡稱開發板  )****測試通過！不過，同樣適用於其他使用 Arduino IDE 開發的****ESP8266, ESP-##****型號的板子，但請自行測試！*  
*除了與網頁內容相關的討論之外，像是 Arduino IDE 與其他 ESP8266 型號連線的問題等，請使用者自行搞定！！！*  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
本網頁所使用到的零件可至下面連結訂購：

* [ESP8266 UNO Arduino IDE 開發板 (Type B1) - ESP-12F, 32MBits](http://goods.ruten.com.tw/item/show?21537656243813)
* [ESP8266 Nano Arduino IDE 開發板 (Type A1) - ESP8266EX、CP2102、32MBits](http://goods.ruten.com.tw/item/show?21741203493792)

ESP8266 相關商品可至下面連結訂購：

* [萬物皆聯網-ESP8266 IoT(Internet of Things)入門學習套件](http://goods.ruten.com.tw/item/show?21447787125749)
* [ESP8266 UART 轉 WiFi 透傳模組 ( 樹莓派、Arduino、單晶片、微控制器可用 )](http://goods.ruten.com.tw/item/show?21538692035419)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*/\*-/--\*-\*/\*/\*/\*/\*\*\*//-\*-\*-\*\*-\*/\*-\*-/\*/\*/\*-\*-/-////--/\*\*/\*\*--\*\*/--///--//\*\*----\*\*//--\*\*//\*\*----\*\*\*//\*-\*\*//\**

在開始此篇網頁之前，建議使用者依序看過下面這兩篇部落格關於 ESP8266 說明網頁後

，再繼續往下看！

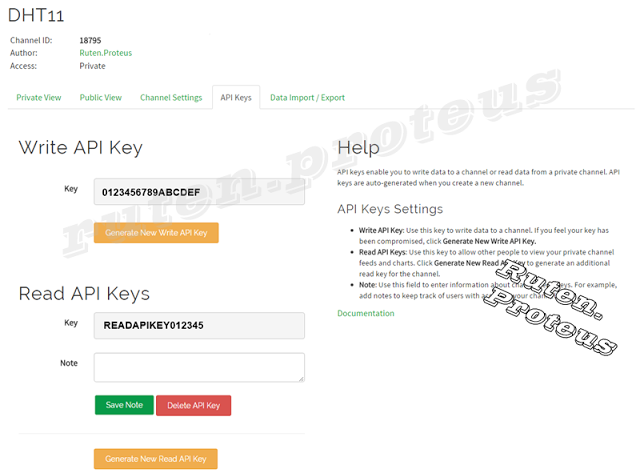
* [初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )](http://ruten-proteus.blogspot.com/2014/11/internet-of-thing-arduino-esp8266.html)
* [使用 Arduino IDE 開發 ESP8266 物聯網應用 - ThingSpeak, HTTP GET / POST 資料上傳方法](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-send.html)

*/\*-/--\*-\*/\*/\*/\*/\*\*\*//-\*-\*-\*\*-\*/\*-\*-/\*/\*/\*-\*-/-////--/\*\*/\*\*--\*\*/--///--//\*\*----\*\*//--\*\*//\*\*----\*\*\*//\*-\*\*//\**

打開之前所建立的 ThingSpeak 中的一個 Channel，也就是之後要用來取出數值的 Channel

|  |
| --- |
| [https://2.bp.blogspot.com/-mgunbAwglXQ/Vf--zJDoR-I/AAAAAAAAHSc/KiP26SdHZPo/s640/Thingspeak_Channel_ID-blog.png](http://2.bp.blogspot.com/-mgunbAwglXQ/Vf--zJDoR-I/AAAAAAAAHSc/KiP26SdHZPo/s1600/Thingspeak_Channel_ID-blog.png) |
| Example: ThingSpeak DHT11 Channel |

首先記住上面的 **Channel ID** ( 我的是 **18795** )。接著，若是這個 Channel 是設定為 Public ( 看 **Access** 欄位 )，則不需要取得 **READ API KEY**；反之若是 Private，按下 "**API Keys**" 進入畫面，在畫面左下方按下按鈕 "**General New Read API Key**" 產生一個新的  **READ API KEY**，記下它 ( **READAPIKEY012345** )，後面要使用這個 Key 才能讀取 Channel 裡面的資料。

[](http://1.bp.blogspot.com/-vlI0KFuKdDQ/Vf_BqCTr6wI/AAAAAAAAHSo/nvDLHZl1QPU/s1600/ThingSpeak-readapikey-create-02-blog.png)

**HTTP GET / POST 與 ThingSpeak Channels 的關係：**  
  
要取回 ThingSpeak IoT Server 特定 Channel 中指定的 Field# ( # = 1 ... 6 ) 中的資料，依照官網上的說明，只能選擇使用 HTTP GET 的方法，相關的格式說明，請點擊下方連結並開啟一個新視窗做為等一下對照之用  
  
[ThingSpeak, Doc, Channels](https://thingspeak.com/docs/channels)  
  
開啟之後，在網頁中找到文字 "**Response in:**" ( ***Ctrl+F*** )，旁邊有不同的響應方式可以選擇 ( 有 TEXT、JSON 和 XML 三種 )。選擇並點擊 **JSON** ，JSON 會變為綠色，且相對應的 **HTTP GET** 格式也會跟著改變。使用這種方式所傳送的 HTTP GET 格式，ThingSpeak 所回應的資料就是 JSON 格式，也是最後我們要處理的東西。  
  
ThingSpeak Doc, Channels 網頁中，關於資料的取回的段落有下面幾個：

* [Get a Channel Feed](https://thingspeak.com/docs/channels#get_feed)  
  使用這個 HTTP GET 格式，可以取回指定的 Channel 中所有 Fileld1 ~ Field6 的指定時間區隔、數量的數據，並且進行這些數據中有效值的加總、平均和中間值的運算。  
  實際範例請參考該網頁。  
  .
* Get Last Entry in a Channel Feed  
  使用這個 HTTP GET 格式，可以取回指定的 Channel 中所有 Fileld1 ~ Field6 最後一筆輸入的數據。  
  實際範例請參考該網頁。  
  .
* Get Specific Entry in a Channel  
  使用這個 HTTP GET 格式，可以取回指定的 Channel 中所有 Fileld1 ~ Field6 第幾個輸入數據的資料。  
  實際範例請參考該網頁。  
  .
* [**Get Channel Field Feed**](https://thingspeak.com/docs/channels#get_field)使用這個 HTTP GET 格式，可以取回指定的 Channel 中，特定 Fileld1 ~ Field6 其中一個 Filed 的指定時間區隔、數量的數據，並且進行這些數據中有效值的加總、平均和中間值的運算。實際範例請參考該網頁，或看下面的額外說明。 **.**
* **Get Last Entry in a Field Feed**使用這個 HTTP GET 格式，可以取回指定的 Channel 中特定 Fileld1 ~ Field6 其中一個] Field 最後一筆輸入的數據。實際範例請參考該網頁，或看下面的額外說明。  
  .

上面的各個 HTTP GET 都配置了一些 **Key / Value** 的參數讓使用者可以選擇性的加入或是移除。下面將會針對最後兩種取回資料的 HTTP GET 格式深入說明，若對於其他 HTTP GET 格式有疑問的話，請自行參考官方網頁的說明。只要能夠了解 ThingSpeak, Doc, Channel 上面關於使用 HTTP GET 取回資料的格式，使用[上一篇](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-send.html)網頁的程式碼對送出的 HTTP GET 字串做修改，就能開始針對 JSON 做分析，取出相對應的欄位名稱與值作為他用。  
  
/\*--\*//\*\*---/\*///\*\*---\*-\*////\*\*\*--\*/\*///\*\*\*----\*///--\*/\*///\*\*--\*/\*//\*\*--\*\*/\*//  
**\* Get Channel Field Feed ( Response in:** JSON **)：**  
  
先看一下此種 HTTP GET 的格式  
https://api.thingspeak.com/channels/CHANNEL\_ID/fields/FIELD\_ID.json   
  
其中，

* CHANNEL\_ID：就是使用者想要取出的 Channel ID，也就是 **18795**
* FIELD\_ID       ：就是 Channel 中要取出的那一個 Field 的數字，也就是 **1** ( 對於 DHT11 來說，代表溫度值 )

網址後面可接的東西就是直接在 HTTP GET 格式之後加上 **?** 再加上下面列出的參數與值，例如  
  
***{HTTP GET}?[參數1=值]&[參數2=值]...***  
  
這個格式的參數有好幾個，每個參數後面都有可輸入的值格式的說明：

* **api\_key** (string) Read API Key for this specific Channel (optional--no key required for public channels)
* **results** (integer) Number of entries to retrieve, 8000 max, default of 100 (optional)
* **days** (integer) Number of 24-hour periods before now to include in feed (optional)
* **start** (datetime) Start date in format YYYY-MM-DD%20HH:NN:SS (optional)
* **end** (datetime) End date in format YYYY-MM-DD%20HH:NN:SS (optional)
* **timezone** (string) [Timezone identifier](https://thingspeak.com/docs#timezones) for this request (optional)
* **offset** (integer) Timezone offset that results should be displayed in. Please use the [timezone](https://thingspeak.com/docs#timezones)parameter for greater accuracy. (optional)
* **status** (true/false) Include status updates in feed by setting "status=true" (optional)
* **metadata** (true/false) Include Channel's metadata by setting "metadata=true" (optional)
* **location** (true/false) Include latitude, longitude, and elevation in feed by setting "location=true" (optional)
* **min** (decimal) Minimum value to include in response (optional)
* **max** (decimal) Maximum value to include in response (optional)
* **round** (integer) Round to this many decimal places (optional)
* **timescale** (integer or string) Get first value in this many minutes, valid values: 10, 15, 20, 30, 60, 240, 720, 1440, "daily" (optional)
* **sum** (integer or string) Get sum of this many minutes, valid values: 10, 15, 20, 30, 60, 240, 720, 1440, "daily" (optional)
* **average** (integer or string) Get average of this many minutes, valid values: 10, 15, 20, 30, 60, 240, 720, 1440, "daily" (optional)
* **median** (integer or string) Get median of this many minutes, valid values: 10, 15, 20, 30, 60, 240, 720, 1440, "daily" (optional)
* **callback** (string) Function name to be used for JSONP cross-domain requests (optional)

Please note that the results parameter is not compatible with timescale, sum, average, or median.

可以指定數據的起始 ( **start** ) 和結束 ( **end** ) 時間間隔；指定時間顯示的時區 ( timezone )；或是要取回的數據數量 ( **results** ) ... 等多種的格式組合。

以取回 Channel ID = 18795，Filed1 的最後一筆數據 ( 溫度值 ) 為例，將下面組合之後的 HTTP GET 格式輸入到瀏覽器的網址中

https://api.thingspeak.com/channels/18795/fields/1.json?api\_key=READAPIKEY012345&results=1

或是下面將 **api\_key** 變成 **key**，ThingSpeak 都可以接受

https://api.thingspeak.com/channels/18795/fields/1.json?key=READAPIKEY012345&results=1

成功傳輸的話，ThingSpeak 就會回覆如下類似的 JSON 字串在瀏覽器視窗中

{"channel":{"id":18795,"name":"DHT11","field1":"Temperature ( degC )","field2":"Humidity ( % )","created\_at":"2014-11-19T13:53:29Z","updated\_at":"2015-08-28T14:08:14Z","last\_entry\_id":25},"feeds":[{"created\_at":"2015-08-28T14:08:14Z","entry\_id":25,"field1":"29.5"}]}

將上面 JSON 字串複製並貼到 [Json Parser Online](http://json.parser.online.fr/) 網站上，就可以得到整理之後的 JSON 以條列式的方式表示

* {
  + "channel":{
    - "id":18795,
    - "name":"DHT11",
    - "field1":"Temperature ( degC )",
    - "field2":"Humidity ( % )",
    - "created\_at":"2014-11-19T13:53:29Z",
    - "updated\_at":"2015-08-28T14:08:14Z",
    - "last\_entry\_id":25

},

* + "feeds":[
    - {
      * "created\_at":"2015-08-28T14:08:14Z",
      * "entry\_id":25,
      * "field1":"29.5"

}

* + ]
* }

對照一下下面圖形上最後一個數據點，將上面 **feeds:** 裡的 **created\_at:2015-08-28T14:08:14Z** 時間加上 **+8** 小時，就會與圖形所顯示的時間一樣。這是因為伺服器回傳是 UTC 時間而不是本地時間，使用者要根據自己所處區域自己設定，不過可以再深入討論一下

|  |
| --- |
| [https://1.bp.blogspot.com/-BOORssbcB0o/Vf_ft9qL4sI/AAAAAAAAHS4/lH64iormArc/s1600/Thingspeak_Channel_ID-Field1-blog.png](http://1.bp.blogspot.com/-BOORssbcB0o/Vf_ft9qL4sI/AAAAAAAAHS4/lH64iormArc/s1600/Thingspeak_Channel_ID-Field1-blog.png) |
| ThingSpeak, Channel ID 18795, Filed1 圖形 |

若是直接使用 **timezone** 的功能，則 HTTP GET 格式如下：

https://api.thingspeak.com/channels/18795/fields/1.json?api\_key=READAPIKEY012345&timezone=Asia/Taipei&results=1

輸出就會如下面所示

{"channel":{"id":18795,"name":"DHT11","field1":"Temperature ( degC )","field2":"Humidity ( % )","created\_at":"2014-11-19T21:53:29+08:00","updated\_at":"2015-08-28T22:08:14+08:00","last\_entry\_id":25},"feeds":[{"created\_at":"2015-08-28T22:08:14+08:00","entry\_id":25,"field1":"29.5"}]}

將上面 JSON 字串複製並貼到 [Json Parser Online](http://json.parser.online.fr/) 網站上，就可以得到整理之後的 JSON 以條列式的方式表，可看出其中時間部分已經更正為台灣時間

* {
  + "channel":{
    - "id":18795,
    - "name":"DHT11",
    - "field1":"Temperature ( degC )",
    - "field2":"Humidity ( % )",
    - "created\_at":"2014-11-19T21:53:29+08:00",
    - "updated\_at":"2015-08-28T22:08:14+08:00",
    - "last\_entry\_id":25

},

* + "feeds":[
    - {
      * "created\_at":"2015-08-28T22:08:14+08:00",
      * "entry\_id":25,
      * "field1":"29.5"

}

* + ]
* }

至於其他的參數，請自行嘗試！  
  
/\*--\*//\*\*---/\*///\*\*---\*-\*////\*\*\*--\*/\*///\*\*\*----\*///--\*/\*///\*\*--\*/\*//\*\*--\*\*/\*//  
\* **Get Last Entry in a Field Feed ( Response in:** JSON **)：**  
  
至於這個部分的 HTTP GET 格式就相對的簡單  
https://api.thingspeak.com/channels/CHANNEL\_ID/fields/FIELD\_ID/last.json  
CHANNEL\_ID 與 FIELD\_ID 前面已經有說過，至於此部分可使用的參數相對比較少，大致上都跟上一節一樣，所以我們直接使用例子來說明。  
以取回 Channel ID = 18795，Filed1 的最後一筆數據 ( 溫度值 ) 為例，將下面組合之後的 HTTP GET 格式輸入到瀏覽器的網址中

https://api.thingspeak.com/channels/18795/fields/1/last.json?api\_key=READAPIKEY012345&timezone=Asia/Taipei

成功傳輸的話，ThingSpeak 就會回覆如下類似的 JSON 字串在瀏覽器視窗中

{"created\_at":"2015-08-28T22:08:14+08:00","entry\_id":25,"field1":"29.5"}

將上面 JSON 字串複製並貼到 [Json Parser Online](http://json.parser.online.fr/) 網站上，就可以得到整理之後的 JSON 以條列式的方式表示

* {
  + "created\_at":"2015-08-28T22:08:14+08:00",
  + "entry\_id":25,
  + "field1":"29.5"

}

//\*\*---\*\*///\*-/\*---\*/\*---/\*-/--\*-/\*-\*--\*\*\*////\*--\*/\*/\*/-\*-/\*/---/\*/-\*/\*/\*/-  
  
上面兩個特別說明的 ThingSpeak HTTP GET 格式，如果還不了解箇中原由的話，請重新回頭看！動手做一下測試，直到清楚為止！  
  
因為，現在才剛要進入主題！  
  
  
**ThingSpeak 的 HTTP GET 回應：**  
  
上一段介紹了 HTTP GET 與 ThingSpeak IoT Server 通訊並取回資料的方法，當中看到了由瀏覽器取回的回應的 JSON 字串。既然如此，為什麼還要在這段落再說一次 ?  
  
原因是：眼見不一定為真！  
  
ESP8266 不是瀏覽器，它不會預先幫使用者過濾一些 HTTP GET 回應的封包內容，但是瀏覽器會！就如同 HTML 檔案丟到瀏覽器，顯示出來的是經過瀏覽器處理之後要讓使用者看到的東西，還是顯示出 HTML 原始檔案格式 ?  
  
所以說，上一段落的確是我們最終要得到的 JSON 格式的字串！但是使用 ESP8266 所得到的卻不是只有 JSON 格式的字串，而是 ThingSpeak 完完整整的 HTTP GET 回應的原始檔，裡面包含了 HTTP 傳輸上的一些東西 ( 簡單解釋看[這篇](http://blog.toright.com/posts/1203/) )，一般不會直接看到！但是直接接收來自於 ThingSpeak 的回應，卻是會收到這些額外我們不需要的東西，而這也是我們必須要去處理並且過濾掉的，才能取出我們所需要的 JSON 字串。  
  
在 Arduino IDE 開啟一個新的 Sketch ，複製下面程式碼貼上並且儲存它，命名為 ***ThingSpealReply***

#include <ESP8266WiFi.h>

*//\*-- Hardware Serial*

#define \_baudrate 9600

*//\*-- IoT Information*

#define SSID "路由器名稱"

#define PASS "路由器連線密碼"

#define HOST "api.thingspeak.com" *// ThingSpeak IP Address: 184.106.153.149*

#define PORT 80

#define READAPIKEY "READAPIKEY012345" *// READ APIKEY for the CHANNEL\_ID*

void setup() {

Serial.begin( \_baudrate );

Serial.println( "ESP8266 Ready!" );

// Connecting to a WiFi network

Serial.print("Connect to ");

Serial.println( SSID );

WiFi.begin( SSID, PASS );

// 持續等待並連接到指定的 WiFi SSID

while( WiFi.status() != WL\_CONNECTED )

{

delay(500);

Serial.print( "." );

}

Serial.println( "" );

Serial.println( "WiFi connected" );

Serial.println( "IP address: " );

Serial.println( WiFi.localIP() );

Serial.println( "" );

}

void loop() {

// 每隔多久取一次資料

retrieveField( 18795, 1 ); // filed\_id=1 是 DHT11 的溫度值

delay( 20000 ); // ms

}

void retrieveField( uint32\_t channel\_id, uint8\_t field\_id )

{

// 設定 ESP8266 作為 Client 端

WiFiClient client;

if( !client.connect( HOST, PORT ) )

{

Serial.println( "connection failed" );

return;

}

else

{

////// 使用 GET 取回最後一筆 FIELD\_ID 資料 //////

/\*\*\* Method 1: \*\*\*/

//

//-- Get a Channel Field Feed --//

// To view a Channel's field feed, send an HTTP GET to

//

// https://api.thingspeak.com/channels/CHANNEL\_ID/fields/FIELD\_ID.json

//----

String GET = "GET /channels/" + String(channel\_id) + "/fields/" + String(field\_id) + ".json?key=" +

READAPIKEY + "&results=1";

//----

Serial.println( "\*\*-- Get a Channel Fiels Feed --\*\*" );

String getStr = GET + " HTTP/1.1\r\n";;

client.print( getStr );

client.print( "Host: api.thingspeak.com\n" );

client.print( "Connection: keep-alive\r\n\r\n" );

delay(10);

// 讀取所有從 ThingSpeak IoT Server 的回應並輸出到串列埠

while(client.available())

{

String line = client.readStringUntil('\r');

Serial.print(line);

}

/\*\*\* Method 2: \*\*\*/

//

//-- Get Last Entry in a Fiels Feed --//

// To get the last entry in a Channel's field feed, send an HTTP GET to

//

// https://api.thingspeak.com/channels/CHANNEL\_ID/fields/FIELD\_ID/last.json

//----

GET = "GET /channels/" + String(channel\_id) + "/fields/" + String(field\_id) +

"/last.json?key=" + READAPIKEY;

//

//----

Serial.println( "\*\*-- Get Last Entry in a Fiels Feed --\*\*" );

getStr = GET + " HTTP/1.1\r\n";;

client.print( getStr );

client.print( "Host: api.thingspeak.com\n" );

client.print( "Connection: close\r\n\r\n" );

delay(10);

// 讀取所有從 ThingSpeak IoT Server 的回應並輸出到串列埠

while(client.available())

{

String line = client.readStringUntil('\r');

Serial.print(line);

}

client.stop();

}

}

使用之前，請記得修改 **SSID**、**PASS** 和 **READAPIKEY** 這三個地方的參數設定；另外，一開始就忘了將 Channel ID 定義在前面，所以使用者必須將自己的 Channel ID 輸入到 **loop(**) 裡的 **retrieveField()** 函式

void loop() {

// 每隔多久取一次資料

retrieveField( **18795**, 1 ); // filed\_id=1 是 DHT11 的溫度值

delay( 20000 ); // ms

}

都完成後，按下 "***Upload***" 按鈕編譯與上傳程式到開發板。  
  
打開 "***Serial Monitor***" 再按下開發板的 ***RESET1*** 按鈕，兩種 HTTP GET 方法取得的 ThingSpeak 回應的原始資料，就會完完整整的顯示在輸出視窗中 ( 資料太多，只節錄其中一部分 )

|  |
| --- |
| [https://4.bp.blogspot.com/-y03L1BS3rB0/VgC-sciO66I/AAAAAAAAHTI/h9kuS3jceko/s320/retrieve-thingspeak-response-01-blog.png](http://4.bp.blogspot.com/-y03L1BS3rB0/VgC-sciO66I/AAAAAAAAHTI/h9kuS3jceko/s1600/retrieve-thingspeak-response-01-blog.png) |
| ThingSpealReply.ino "***Serial Monitor***" 輸出畫面 |

發送 HTTP GET 需求給 ThingSpeak 後，所得到的回應就是像上面一樣的資料，而 JSON 格式的字串就藏在最後面倒數第二行。  
  
*補充說明：*

* *倒數第一行：數字 0，表示輸出完成；應該不會有另外不同的數字出現。*
* *倒數第三行：十六進制的數字表示，指出下一行 ( 倒數第二行 )  有多少個字*

詳細的輸出資料，都在下面兩個小節之中。  
  
/\*--\*//\*\*---/\*///\*\*---\*-\*////\*\*\*--\*/\*///\*\*\*----\*///--\*/\*///\*\*--\*/\*//\*\*--\*\*/\*//  
**\* Get Channel Field Feed ( Response in:**JSON **) ThingSpeak 回應之原始資料：**

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Transfer-Encoding: chunked

X-Cnection: close

Status: 200 OK

X-Frame-Options: ALLOWALL

Access-Control-Allow-Origin: \*

Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH

Access-Control-Allow-Headers: origin, content-type, X-Requested-With

Access-Control-Max-Age: 1800

ETag: "e5c482641fa6b8d74d546aee0ca3e952"

Cache-Control: max-age=0, private, must-revalidate

X-Request-Id: 88d1901e-790b-4b8a-834c-12816a957507

X-Runtime: 0.007428

X-Powered-By: Phusion Passenger 4.0.57

Date: Tue, 22 Sep 2015 01:51:56 GMT

Server: nginx/1.9.3 + Phusion Passenger 4.0.57

10a

{"channel":{"id":18795,"name":"DHT11","field1":"Temperature ( degC )","field2":"Humidity ( % )","created\_at":"2014-11-19T13:53:29Z","updated\_at":"2015-08-28T14:08:14Z","last\_entry\_id":25},"feeds":[{"created\_at":"2015-08-28T14:08:14Z","entry\_id":25,"field1":"29.5"}]}

0

/\*--\*//\*\*---/\*///\*\*---\*-\*////\*\*\*--\*/\*///\*\*\*----\*///--\*/\*///\*\*--\*/\*//\*\*--\*\*/\*//  
\* **Get Last Entry in a Field Feed ( Response in:** JSON **) ThingSpeak 回應之原始資料：**

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Transfer-Encoding: chunked

Connection: close

Status: 200 OK

X-Frame-Options: ALLOWALL

Access-Control-Allow-Origin: \*

Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH

Access-Control-Allow-Headers: origin, content-type, X-Requested-With

Access-Control-Max-Age: 1800

ETag: "ba589660c39a80fd60ecad73df80c219"

Cache-Control: max-age=0, private, must-revalidate

X-Request-Id: 9f6f3443-3760-454c-9dc8-c36b57c07a20

X-Runtime: 0.006541

X-Powered-By: Phusion Passenger 4.0.57

Date: Tue, 22 Sep 2015 01:51:59 GMT

Server: nginx/1.9.3 + Phusion Passenger 4.0.57

43

{"created\_at":"2015-08-28T14:08:14Z","entry\_id":25,"field1":"29.5"}

0

/\*-/\*--\*/\*/\*-\*---\*////\*\*---\*\*/-\*-\*-/-\*-\*-/-\*-\*-\*---\*-\*-\*-\*-/-\*\*--\*-/-/\*\*-/-/-/  
  
  
**解析 JSON 字串：**  
  
當知道了回傳的資料格式之後，接下來的動作就是如何處理出 JSON 字串。首先，可以知道這些資料以"**行**"的方式輸入，每一行最後面以 **\r** 結束直到倒數第四行是以 **\n** 代表為一個空白行。接著，出現代表數字的字串輸入，可以處理也可以不處理，視使用者的需求決定，程式預設是保留但不處理。然後就是 JSON 字串，直接取出並保存到字串變數中即可。最後一行就直接跳過，不于理會。  
  
/\*--\*//\*\*---/\*///\*\*---\*-\*////\*\*\*--\*/\*///\*\*\*----\*///--\*/\*///\*\*--\*/\*//\*\*--\*\*/\*//  
**\* Get Last Entry in a Field Feed**  
  
延續剛剛的 Sketch 程式碼 ***ThingSpealReply***，只需增加一個處理 JSON 的函式庫標頭檔於最前面的地方

#include <ESP8266WiFi.h>

#include <ArduinoJson.h>

然後複製下面的程式碼取代 **retrieveField()** 裡面 **else{ }** 所有的東西，另存新檔為 ***RetrieveHttpGetM2***

else

{

////// 使用 GET 取回最後一筆 FIELD\_ID 資料 //////

/\*\*\* Method 2: \*\*\*/

//

//-- Get Last Entry in a Field Feed --//

// To get the last entry in a Channel's field feed, send an HTTP GET to

//

// https://api.thingspeak.com/channels/CHANNEL\_ID/fields/FIELD\_ID/last.json

//

// replacing CHANNEL\_ID with the ID of your Channel and FIELD\_ID with the ID of your field.

//

// Example:

//

// https://api.thingspeak.com/channels/18795/fields/1/last.json?key=READAPIKEY012345

//----

String GET = "GET /channels/" + String(channel\_id) + "/fields/" + String(field\_id) +

"/last.json?key=" + READAPIKEY;

//

//----

String getStr = GET + " HTTP/1.1\r\n";;

client.print( getStr );

client.print( "Host: api.thingspeak.com\n" );

client.print( "Connection: close\r\n\r\n" );

delay(10);

// 讀取所有從 ThingSpeak IoT Server 的回應並輸出到串列埠

String section="HEAD";

while(client.available())

{

String line = client.readStringUntil('\r');

//\*\* parsing the JSON response here ! \*\*//

// parse the HTML body

if(section == "HEAD" ) // HEAD

{

Serial.print( "." );

if( line == "\n" ) // 空白行

{

section = "LENGTH";

}

}

else if( section == "LENGTH" )

{

// 這裡可以取回 JSON　字串的長度

// String content\_length = line.substring(1);

/\* 有需要處理的寫程式在這裡 \*/

section = "JSON";

}

else if( section == "JSON" ) // print the good stuff

{

Serial.println( "" );

section = "END";

String jsonStr = line.substring(1); // 給定一個從索引到尾的字串

// 開始解析 JSON

int size = jsonStr.length() + 1;

char json[size];

jsonStr.toCharArray(json, size);

Serial.println( json );

StaticJsonBuffer<200> jsonBuffer;

JsonObject& jsonParsed = jsonBuffer.parseObject(json);

if (!jsonParsed.success())

{

Serial.println("parseObject() failed");

return;

}

const char \*createdat = jsonParsed["created\_at"];

int entryid = jsonParsed["entry\_id"];

float field1 = jsonParsed["field1"];

Serial.println("-- Decoding / Parsing --");

Serial.print( "Created at: " ); Serial.println( createdat );

Serial.print( "Entry id: " ); Serial.println( entryid );

Serial.print( "Field 1: " ); Serial.println( field1, 1 );

}

}

client.stop();

}

按下 "***Upload***" 按鈕編譯與上傳程式到開發板。  
  
打開 "***Serial Monitor***" 再按下開發板的 ***RESET1***按鈕，得到的結果就如下面所示

|  |
| --- |
| [https://4.bp.blogspot.com/-DhnKFEoVQEE/VgDlNFFypxI/AAAAAAAAHTY/mUleN2g44nQ/s640/serial-monitor-01-blog.png](http://4.bp.blogspot.com/-DhnKFEoVQEE/VgDlNFFypxI/AAAAAAAAHTY/mUleN2g44nQ/s1600/serial-monitor-01-blog.png) |
| Get Last Entry in a Field Feed 範例程式輸出 |

程式會一直取回 ThingSpeak 上面 Channel 18795, Field 1 最後一筆資料進行解析，並且輸出解析之後的結果。  
 *-- Decoding / Parsing --* 上面一行是程式取出的 JSON 字串，下面一行開始，就是解析之後的欄位值。  
  
解析之後的欄位名稱與值存放在 **jsonparsed** 物件中，要取出欄位值必須先指定欄位名稱。例如，要取出 Field 1 的資料，就輸入像下面的程式碼  
  
**float field1 = jsonParsed["field1"];**  
  
對照一下 JSON 字串裡面的關係，就不難了解到底要輸入什麼在 **jsonParsed** 物件陣列中取出相關欄位名稱的值了。另外一點，要特別注意取值時，要指定正確的資料格式以取出 **jsonParsed** 物件陣列的值。  
  
/\*--\*//\*\*---/\*///\*\*---\*-\*////\*\*\*--\*/\*///\*\*\*----\*///--\*/\*///\*\*--\*/\*//\*\*--\*\*/\*//  
**\* Get a Channel Field Feed**  
  
相對於前一小節 **Get Last Entry in a Field Feed**，這一小節就比較複雜，因為不管取回的數據有多少，都會被分成 **channel:{},feeds"[{},{},...]** 的格式，例如取回 3 筆資料

https://api.thingspeak.com/channels/18795/fields/1.json?api\_key=READAPIKEY012345&results=3

去除掉前面不需要的部分，只留下 JSON 字串

{"channel":{"id":18795,"name":"DHT11","field1":"Temperature ( degC )","field2":"Humidity ( % )","created\_at":"2014-11-19T13:53:29Z","updated\_at":"2015-08-28T14:08:14Z","last\_entry\_id":25},"feeds":[{"created\_at":"2014-11-20T03:30:12Z","entry\_id":23,"field1":"24"},{"created\_at":"2014-11-20T04:03:57Z","entry\_id":24,"field1":"66"},{"created\_at":"2015-08-28T14:08:14Z","entry\_id":25,"field1":"29.5"}]}

將上面 JSON 字串複製並貼到 [Json Parser Online](http://json.parser.online.fr/) 網站上，就可以得到整理之後的 JSON 以條列式的方式表示

* {
  + "channel":{
    - "id":18795,
    - "name":"DHT11",
    - "field1":"Temperature ( degC )",
    - "field2":"Humidity ( % )",
    - "created\_at":"2014-11-19T13:53:29Z",
    - "updated\_at":"2015-08-28T14:08:14Z",
    - "last\_entry\_id":25

},

* + "feeds":[
    - {
      * "created\_at":"2014-11-20T03:30:12Z",
      * "entry\_id":23,
      * "field1":"24"

},

* + - {
      * "created\_at":"2014-11-20T04:03:57Z",
      * "entry\_id":24,
      * "field1":"66"

},

* + - {
      * "created\_at":"2015-08-28T14:08:14Z",
      * "entry\_id":25,
      * "field1":"29.5"

}

* + ]
* }

這樣的 JSON 字串沒有辦法被原本使用的函式庫處理並解析出每一個欄位值，必須再經過處理。  
  
處理的方式見"人"見智，個人使用的方式：

1. 刪除字串最後面的 **]}**
2. 尋找字串 **},{** 再將其中的逗號更改為空白
3. 以空白字元作為字串切割的依據，循環切割並處理

切割之後的字串，就是函式庫可以處理的 JSON 格式字串。  
  
同樣的，然後複製下面的程式碼取代 ***RetrieveHttpGetM2*** 裡 **retrieveField()** 裡面 **else{ }** 所有的東西，另存新檔為 ***RetrieveHttpGetM1***

else

{

////// 使用 GET 取回最後一筆 FIELD\_ID 資料 //////

// 有兩種方式可以取回 Field\_id 中的最後一筆資料

//

// NOTE:

// If your channel is public, you don't need a key.

// If your channel is private, you need to generate a read key.

//

/\*\*\* Method 1: \*\*\*/

//

//-- Get a Channel Field Feed --//

// To view a Channel's field feed, send an HTTP GET to

//

// https://api.thingspeak.com/channels/CHANNEL\_ID/fields/FIELD\_ID.json

//

// replacing CHANNEL\_ID with the ID of your Channel and FIELD\_ID with the ID of your field.

//

// Example:

// https://api.thingspeak.com/channels/18795/fields/1.json?key=READAPIKEY012345&results=1

//

// DHT11, field 1: temperature, field 2: humidity

//----

String GET = "GET /channels/" + String(channel\_id) + "/fields/" + String(field\_id) + ".json?key=" +

READAPIKEY + "&results=3";

//----

String getStr = GET + " HTTP/1.1\r\n";;

client.print( getStr );

client.print( "Host: api.thingspeak.com\n" );

client.print( "Connection: close\r\n\r\n" );

delay(10);

// 讀取所有從 ThingSpeak IoT Server 的回應並輸出到串列埠

String section="HEAD";

while(client.available())

{

String line = client.readStringUntil('\r');

//\*\* parsing the JSON response here ! \*\*//

// parse the HTML body

if(section == "HEAD" ) // HEAD

{

Serial.print( "." );

if( line == "\n" ) // 空白行

{

section = "LENGTH";

}

}

else if( section == "LENGTH" )

{

// 這裡可以取回 JSON　字串的長度

// String content\_length = line.substring(1);

/\* 有需要處理的寫程式在這裡 \*/

section = "JSON";

}

else if( section == "JSON" ) // print the good stuff

{

Serial.println( "" );

section = "END";

int size = line.length();

String jsonStr = line.substring(1, size - 2);

size = jsonStr.length() + 1;

char json[size];

jsonStr.toCharArray(json, size);

Serial.println("--Json String --");

Serial.println( json );

for(int i = 0; i < size; i++)

{

if(json[i] == '}')

{

if(json[i+1] == ',')

{

if(json[i+2]=='{')

{

json[i+1] = ' ';

}

}

}

}

char\* feeds = (strstr( json, "\"feeds\"") + 9);

Serial.println("-- Removed comma inside feeds string --");

Serial.println(feeds); // 輸出 JSON 字串

char\* field;

field = strtok( feeds, " " );

while(field)

{

Serial.println("-- Field --");

Serial.println(field);

StaticJsonBuffer<200> jsonBuffer;

JsonObject& jsonParsed = jsonBuffer.parseObject(field);

if (!jsonParsed.success())

{

Serial.println("parseObject() failed");

return;

}

const char \*createdat = jsonParsed["created\_at"];

int entryid = jsonParsed["entry\_id"];

float field1 = jsonParsed["field1"];

Serial.println("-- Decoding / Parsing --");

Serial.print( "Created at: " ); Serial.println( createdat );

Serial.print( "Entry id: " ); Serial.println( entryid );

Serial.print( "Field 1: " ); Serial.println( field1, 1 );

field = strtok( NULL, " " );

}

}

}

client.stop();

}

按下 "***Upload***" 按鈕編譯與上傳程式到開發板。  
  
打開 "***Serial Monitor***" 再按下開發板的 ***RESET1***按鈕，得到的結果就如下面所示

|  |
| --- |
| [https://3.bp.blogspot.com/-G9SR-_fffmw/VgF6kaE2H5I/AAAAAAAAHTo/H6RjcHAf-Xg/s640/serial-monitor-retrieve-M1-blog.png](http://3.bp.blogspot.com/-G9SR-_fffmw/VgF6kaE2H5I/AAAAAAAAHTo/H6RjcHAf-Xg/s1600/serial-monitor-retrieve-M1-blog.png) |
| Get a Channel Field Feed 範例程式輸出 |

原本只要取回最後一筆數據處理，但是最後發現：反正程式碼都一樣！就多取回幾筆數據做分析，多筆可以完成解析，單筆也不會有問題！  
  
*/\*-/--\*-\*/\*/\*/\*/\*\*\*//-\*-\*-\*\*-\*/\*-\*-/\*/\*/\*-\*-/-////--/\*\*/\*\*--\*\*/--///--//\*\*----\*\*//--\*\*//\*\*----\*\*\*//\*-\*\*//\**  
[*JSON 函式庫下載*](https://github.com/bblanchon/ArduinoJson)  
*程式使用 5.0.2 版本編譯、測試通過！*  
 *要用哪一個版本由使用者自己選擇，但請注意的是！因為這個函式庫並不是專為 ESP8266 撰寫，因此存在一些隱藏的 BUG，使用者必須自己處理這個問題！*  
*/\*-/--\*-\*/\*/\*/\*/\*\*\*//-\*-\*-\*\*-\*/\*-\*-/\*/\*/\*-\*-/-////--/\*\*/\*\*--\*\*/--///--//\*\*----\*\*//--\*\*//\*\*----\*\*\*//\*-\*\*//\**  
  
  
**結論：**  
  
使用開發板在 Arduino IDE 做開發非常容易，只需要一條 MicroUSB 與電腦連接，就能做到程式碼的上傳以及 UART 除錯，使用上便利性很高！由於又開放多個 IO，可以瞬間將電子週邊瞬間升級為無線網路裝置，搭配上 IoT Server 則可說就是開始物聯網的應用。  
  
IoT Server 上資料的上傳與下載有其限制所在，不同的 IoT Server 之間要看該網站中的文件說明。以 ThingSpeak 為例，建議上傳間隔不小於 15 秒 ，可以使用 HTTP GET 或是 HTTP POST ( 又或是其他方法 )；下載資料則使用 HTTP GET ( 右或是有其他方法 )，根據所需要的回應方式而有不同的輸入格式 ( TEXT、JSON 或是 XML )。  
  
本篇屬於下載資料的方法與程式碼說明，雖然只針對其中兩種，但是其輸入的 HTTP GET 格式只存在於些許不同，可選擇的配合參數卻是大同小異，多或少而以！雖然其他的 HTTP GET 格式可以說就只是其擴充或是縮減的版本而以，沒那麼複雜！  
  
但對於程式碼來說，下載資料的格式以及所選用的 HTTP GET 格式，卻會影響到程式碼解析資料以及處理的複雜性。以 JSON 回應格式為例，即便回應的是 JSON 格式字串，但是因為所使用的 JSON 函式庫只接受某一格式類型的 JSON 格式輸入，所以在解析原始 JSON 回應字串時就必須再做簡化的處理，而不是一昧地將資料丟給 JSON 函式庫，因為這是行不通的！  
  
在這網頁中所提出的兩種下載方法，其處理的過程和程式碼，可以作為其他下載方法處理的過程與程式的依據。  
  
希望這篇網頁中的內容，對於有需要或是想要以其他下載方法取得 ThingSpeak IoT Server 資料的使用者，能夠起拋磚引玉的效果！

<< 部落格相關網頁連結 >>

* [初遇 IoT ( Internet of Thing, 物聯網 ) - 使用 Arduino 控制 ESP8266 無線模組上傳資料到 IoT Server ( ThingSpeak )](http://ruten-proteus.blogspot.com/2014/11/internet-of-thing-arduino-esp8266.html)
* [使用 Arduino IDE 開發 ESP8266 物聯網應用 - ThingSpeak, HTTP GET / POST 資料上傳方法](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-send.html)
* [使用 Arduino IDE 開發 ESP8266 物聯網應用 - 取回 ThingSpeak 特定 Channel 和 Field 最後一筆資料](http://ruten-proteus.blogspot.com/2015/09/iot-esp8266-arduino-retrieve.html)