**每個軟體開發者都絕對一定要會的Unicode及字元集必備知識(沒有藉口！)**

作者：周思博 (Joel Spolsky)  
譯：Paul May 梅普華  
Wednesday, October 08, 2003  
屬於Joel on Software, [http://www.joelonsoftware.com](http://www.joelonsoftware.com/)

還搞不懂那個神秘的Content-Type tag嗎？你知道的，就是那個應該放在HTML裡卻又永遠不知道該設成什麼內容的標籤啊。

你曾經收到在保加利亞的朋友寄來，主題是「???? ?????? ??? ????」的電子郵件嗎？

很多軟體開發者並未真正完全理解字元集、字元編碼、Unicode等等的神秘世界，當我發現不懂的人那麼多時真的很失望。數年以前，某位beta測試人員想知道[FogBUGZ](http://www.fogcreek.com/FogBUGZ)是否能處理日文的電子郵件？他們竟然用日文寫電郵？我完全不知道耶。我們用了一個商用ActiveX控制元件來分析MIME電郵訊息，當我仔細調查這個元件時，才發現它對字元集的處理完全錯誤，所以我們還寫了些了不起的程式，把錯誤的轉換還原後再重做正確的轉換。我又去看看另一個商用程式庫，它的字元編碼實作也是完全不對。我聯絡該軟體的開發者，結果他似乎有點認為沒辦法改善。他跟很多程式師一樣，只希望這個問題能憑空消失。

不過問題並不會消失。PHP是個很普遍的web開發工具，不過它[完全忽略字元編碼問題](http://ca3.php.net/manual/en/language.types.string.php)，PHP很愉快地用8位元來處理字元，因此幾乎不可能開發好的國際化web應用程式。當我發現這件事時，覺得*真是夠了*。

所以我要做一個宣告：如果你在2003年還是個程式師，而你不知道字元、字元集、字元編碼、以及Unicode的基本知識，我就要去*抓*你，我會讓你在潛艇裡關6個月剝洋蔥。我發誓我一定會的。

另外還有一件事：

**這並沒有那麼難。**

我會在這篇文章中讓你確實瞭解*每個現役程式師*都應該知道的事情。所謂「純文字 = ascii = 字元都是8個位元」的說法不僅不對，而且還錯得離譜；如果你還是照這個想法寫程式，那麼你大概不會比不相信細菌的醫生好多少。在讀完這篇文章之前請暫時不要寫程式。

在我開始之前應該先提醒一下，如果你是極少數瞭解多國語言軟體製作的人，會發現我的討論有點過度簡化。我只是想設立一個底線，讓大家能瞭解這是怎麼一回事，而且寫出的程式有*希望*能處理任何語言的文字，而不是只認得沒有重音符號的英文。另外我也要提醒你，字元處理只佔建立多國語言軟體的一小部份，不過我一次只能寫一件事，所以今天只談字元集。

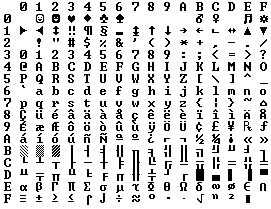
**歷史的觀點**

要瞭解這些事，最簡單的方法就是按年代來看。

你或許會認為我會講些EBCDIC之類很古老的字元集。我不會，EBCDIC跟你的生活無關。我們並不用回逆到那麼前面。

回到沒那麼古老的從前，Unix被發明而K&R正在寫[The C Programming Language](http://cm.bell-labs.com/cm/cs/cbook/)的那個時代，當時每件事都非常簡單。EBCDIC正在被淘汱。唯一重要的字元集就是古老美好的無重音英文字母，我們有一個對應的編碼系統叫做[ASCII](http://www.robelle.com/library/smugbook/ascii.html)，可以用32到127的數字表示每一個字元。空白是32，字母A是65，如此類推。這種方法可以把文字存成7個位元。當時大部份電腦的一個位元組都是8個位元，所以儲存全部ASCII字元之後有很多個位元沒用到。如果你夠邪惡，就會偷用這些空位元：事實上WordStar的壞蛋就用把最高位元設起來，代表一個單字中的最後一個字母。這是開玩笑的。空的位元被用來當控制字元，比如7會讓你的電腦發出嗶聲，而12會讓印表機把目前正在印的紙張送出並且捲入一張新紙。

所以天下太平，不過只限於英語系的人。

由於位元組有8個位元的空間，所以很多人就開始想啦：「對了，我們可以把128到255的碼拿來自己用。」問題是*很多*人同時都有這個想法，所以128到255的空間該怎麼用，大家都各自有自己的想法。IBM-PC用了一種名為OEM字元集的東西，提供了某些歐洲語言用的重音字母和[一堆線條繪圖字元](http://www.jimprice.com/ascii-dos.gif)：水平線、垂直線、右邊有個小吊釣的水平線等等。你可以用這些線條繪圖字元在螢幕上拼出很漂亮的方框和線條，在乾洗店裡的8088電腦還可以看到這種圖案。事實上當PC開始賣到美國以外時，各種不同的OEM字元就被憑空創造出來，大家都把上面這128個字元拿來自己用。舉例來說，字元碼130在某些PC上會顯示為é，不過在以色列賣的電腦上就變成希伯來文字母Gimel (?)，所以當美國人把履歷(résumé)寄到以色列就會變成r?sum?。 在很多情況下，比如說俄文好了，本身對於上面128個字元(值>127)就有很多不同的想法，所以什麼連俄文文件本身都不能可靠地轉換。

後來這段OEM亂用區終於在ANSI標準裡固定下來。在ANSI標準中，大家都同意小於128的字元定義(基本上和ASCII一致)，不過由128開始的字元就有很多不同的處理方法，會依照你住的地方而定。這些不同的系統就叫做*[頁碼(code page)](http://www.i18nguy.com/unicode/codepages.html" \l "msftdos)*。舉例來說以色列的DOS用叫862的頁碼，而希臘用戶則是用737。它們在128以下是一樣的，不過由128起就不同了，裡面充滿奇奇怪怪的字母。美國版本的MS-DOS有幾十種頁碼，由英文到冰島文都可以處理，甚至還有一些「多語」頁碼可以*在同一台電腦*上處理世界語和加利西亞語！了不起！不過要一台電腦同時處理希伯來文和希臘文是絕對不可能的，除非你自己寫程式自己用圖顯示所有文字。因為希伯來文和希臘文對128以上字元的解釋方法不同，必須用到不同的頁碼。

在同一時期亞洲發生的事情更誇張。由於亞洲的字母系統有幾千個字母，不可能用8個位元表示。通常是用一種叫DBCS的麻煩系統來處理。DBCS是雙位元組字元集(Double Byte Character Set)，字元集中的*某些*字母是一個位元組來存，其他字則要用兩個位元組。在DBCS的字串中要向後移到下一個字很容易，不過幾乎不可能往回移到前一個字。程式師被指示向後及往回移時不能用s++和s--，而是呼叫Windows的AnsiNext和AnsiPrev之類的函數，只有這些函數才知道怎麼處理這些麻煩。

不過大多數人還是假裝一個位元組就是一個字元，而一個字元就是8個位元。只要不會把字串在電腦間移動，或者只用一種語言，這種想法大致上還是能用。不過當Internet興起，在電腦間移動字串變成隨時都在做的事，整團麻煩自然就爆出來了。還好這時已經發明了Unicode。

**Unicode**

Unicode是個勇敢的嘗試，想用單一個字元集去涵括地球上所有合理的書寫系統，另外也要包括克林貢語等杜撰的語文。有些人誤認為Unicode只是個16位元碼，裡頭每個字都要佔16位元，所以總共有65,536個字元。**事實上這並不正確。**這是關於Unicode常見的誤解，所以如果你也這麼認為的話，不用難過。

事實上Unicode對字元有不一樣的想法，你必須瞭解Unicode的想法，否則是搞不懂的。

到目前為止，我們都假設一個字母會對映到某些位元，這些可以存在磁碟或記憶體中：

A -> 0100 0001

在Unicode裡一個字母是對映到一個叫*code point*的東西(還只是一個理論上的概念)。要如何在記憶體或是磁碟上表示code point就完全是另一回事。

在Unicode中，字母A是個精神上的觀念。它只會漂浮在天堂裡：

A

這個觀念上的A和B或者a都不一樣，不過A和***A***以及A都一樣。Times New Roman字型的A和Helvetica字型的A是相同的字元，但和小寫的"a"*不一樣*，這種想法似乎沒什麼好爭論的。不過在某些語言中，光是要決定一個字母*是*什麼就有得吵了。舉例來說，德文字母β究竟真正的字母還是ss(譯註：拉丁文的gei)的另一種特別寫法呢？如果字母的形狀在單字結束時會改變，改變之後要當作不同的字母嗎？希伯來文說是，阿拉伯文卻認為不是。不管如何，Unicode協會的聰明人已經在過去十年左右搞定了，雖然有一大堆政治爭論伴隨而來，不過你不用擔心。他們已經完全搞定了。

Unicode協會把所有字母系統中每一個觀念上的字母都分配一個魔術數字，這個數字的寫起來就像是：**U+0645**。這個魔術數字就叫一個*code point*。U+的意思是Unicode，數字則是用十六進位表示。**U+FEC9**就是阿拉拍文的字母Ain。英文字母A則是**U+0041**。你可以用Windows 2000/XP的**charmap**工具把這些數字全找出來，到[Unicode網站](http://www.unicode.org/)也可以找到。

Unicode可以定義的字母數量並沒有實質限制，事實上可以超過65,536個，所以並不是所有的Unicode字母都能擠進兩個位元組裡，不過反正那本來就是個迷思。

好吧，假設我們有個字串：

**Hello**

用Unicode來表示的話，這個字串會對映到下面五個code point：

U+0048 U+0065 U+006C U+006C U+006F.

就只是一堆code point。實際上也就是數字。不過我們還沒有提過要如何儲存到記憶體或在電郵訊息中表示。

**字元編碼**

這就是*字元編碼*上場的地方。

Unicode編碼最初的想法導致了兩個位元組的迷思，簡單說就是把那些數字都存成兩個位元組。所以Hello變成

00 48 00 65 00 6C 00 6C 00 6F

這樣對嗎？等一下！也有可能會是：

48 00 65 00 6C 00 6C 00 6F 00 ？

好吧，技術上是的，我的確相信可以這樣寫，而事實上早期的實作者希望能把Unicode碼存成high-endian或low-endian模式，可以依據CPU用哪一種最快來決定。於是就有*兩*種儲存Unicode的方法。所以人們被迫想出奇怪的作法，在每個Unicode字串的開頭存一個FE FF；稱之為[Unicode Byte Order Mark](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/intl/unicode_42jv.asp)。如果你把高低位元組對調，標記就會變成FF FE，讀字串的人就知道其他位元組都要對調。不過外面的Unicode字串開頭並不一定都會有這種位元組順序標記。

http://www.joelonsoftware.com/pictures/unicode/hummers.jpg

有一段時期這個方法好像還不錯，不過後來有程式師在抱怨了。他們說：「看看那些零」，因為他們都是美國人，看到的都是很少用到U+00FF以上code point的英文文字。何況他們還是注重*保育(哼)*又崇尚自由的加州嬉皮。如果他們是德州佬，才不會在意要花掉兩倍的位元組呢(譯註：指德州人少地方大，所以財大氣粗)。不過那些加州糊塗蛋受不了字串儲存空間會*倍增*的想法，而且外頭已經有太多文件是用各種ANSI和DBCS字元集寫的，要找誰來轉換這些文件？*新聞局嗎？*光是這個理由，就讓大多數人就決定不管Unicode，幾年下來情況就變得愈來愈糟了。

然後就有人[發明](http://www.cl.cam.ac.uk/~mgk25/ucs/utf-8-history.txt)了[UTF-8](http://www.utf-8.com/)這個絕佳的點子。UTF-8是另一個儲存系統，用8位元方式把Unicode code point(就是那些神秘的U+數字)存在記憶體中。在UTF-8中，由0-127的code point都存成*一個位元組*。只有128和更大的code point會存成2或3或個位元組，事實上最多可以用到6個位元組。

How UTF-8 works

這樣做有個很巧妙的副作用，就是英文文字*用UTF-8和用ASCII會完全一樣*，所以美國人根本不會覺得有啥不對。只剩世界上其他地方的人得跳火圈。具體來說Unicode為U+0048 U+0065 U+006C U+006C U+006F的**Hello**的，會被存成48 65 6C 6C 6F。看吧！這跟存成ASCII或ANSI或是地球上每一種OEM字元集的結果都一樣。這樣子一來，如果你斗膽敢用重音字母或希臘字母或是克林貢字母，就得用多個位元組來儲存一個code point，只是美國人永遠不會發現。(UTF-8還有一個蠻好的特性。由於舊的字串處理程式並不知道Unicode，在處理字串時會用一個值為零的位元組作為字串結尾。如果用UTF-8的話，這些舊程式不會中途截斷字串。)

到目前為止我說了*三*種Unicode編碼的方法。全部都存成兩個位元組的傳統作法叫做UCS-2(因為用兩個位元組)或是UTF-16(因為有16位元)，不過你還是得分辨是high-endian UCS-2還是low-endian UCS-2。再來是普遍使用的新UTF-8[標準](http://www.zvon.org/tmRFC/RFC2279/Output/chapter2.html)，這個標準有良好的特性，在用只認識ASCII的舊程式處理用英文文字還是一切正常。

Unicode其實還有其他多種編譯方式。其中之一叫UTF-7，非常像UTF-8不過保證最高位元一定是零，所以即使經過某個認為7位元*很足夠*的嚴苛警察國家電郵系統，還是能亳髮無傷地通過。另外也有每個code point都存成4個位元組的UCS-4，好處是每個code point都容量都一樣，不過連德州佬都不敢浪費*那麼多*記憶體。

實際上現在你正在以概念性的字母(表示成Unicode code point)來思考事情，這些Unicode code point也可以用任何老式的編碼方法來編碼！舉例來說，你可以把Unicode字串Hello(U+0048 U+0065 U+006C U+006C U+006F)編碼成ASCII或舊的OEM希臘編碼，也可以編成希伯來ANSI編碼或是到目前為止已發明的數百種編碼方式，不過*有一個陷阱*：某些字母可能會畫不出來！如果某個Unicode code point在你所用的編碼方式中沒有對應的字元，通常就會看到一個小問號?或是一個小方框。在箭頭後面你看到的是什麼呢？-> 嚙(譯註：這個字是十六進位EFBF，用UTF-8好像是沒有字，在大五碼裡就是「嚙」，因為譯文用big5編碼，所以看到的是「嚙」)？

多達數百種的傳統編碼方式都只能正確儲存*部份*的code point，而其他code point則是全部變成問號。常見的英文文字編碼有Windows-1252 (Windows 9x對西歐語言的標準)和[ISO-8859-1](http://www.htmlhelp.com/reference/charset/)又名Latin-1(也是用於西歐語言)，不過想要用這些編碼方式儲存俄文或希伯來文時就會得到一大堆的問號。而UTF 7, 8, 16和32通通都能正確的儲存*任何一個*code point。

**關於字元編碼最重要的一個事實**

如果你完全不記得我剛說的東西，請至少記住一件超級重要的事實。**光有字串卻不知道編碼方式是不行的**。你不能再把頭埋在沙裡假裝「純」文字就是ASCII。

**根本就沒有純文字這種東西。**

假設你有一個字串，不管是在記憶體或在檔案還是在電郵訊息裡，你都必須知道字串用的編碼方式，才能正確解譯出來並呈現給使用者。

「我的網站都是亂碼」或「她看不到我用重音符號寫的電郵」之類的笨問題，幾乎全部都是因為某位天真的程式師不瞭解一個單純的事實：如果不知道某個字串的編碼方式是UTF-8還是ASCII還是ISO 8859-1 (Latin 1)還是Windows 1252 (西歐)，根本不可能正確顯示出來，甚至連在哪結束可能都找不到。大於127的code point有上百種編碼方式，連猜都猜不到。

我們要如何保存某個字串的編碼資訊呢？好吧，是有一些標準方法可以用。以電子郵件來說，郵件表頭應該會有一個字串：

**Content-Type: text/plain; charset="UTF-8"**

如果是網頁的話，最原始的想法是在網頁之外，再讓web伺服器傳回一個類似的**Content-Type** http header。不是放在HTML裡面，而是在傳HTML網頁之前先送的header。

這樣做會有問題。假設你有一個很大的web伺服器，很多使用各種語言的人在裡面放了很多網站和網頁，所有網頁的編碼方式都是由微軟FrontPage自動產生。Web伺服器本身其實並不*知道*各個檔案的編碼方式，所以也沒法子傳出正確的Content-Type header。

利用某些特別的tag把HTML檔案的Content-Type放在HTML檔案裡比較方便。當然這會讓純粹主義者抓狂...你怎麼能在不知道編碼方式之前*讀*HTML檔案呢！？幸運的是，幾乎所有編碼方式由32到127的字元都是一樣的，所以不需用到怪字母就能在HTML網頁取到這些資訊：

**<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">**

不過這個meta tag一定得放在<head>段落非常前面的地方。因為網頁瀏覽器一看到這個tag就會停止分析，然後改用你指定的編碼方式重新解譯整個網頁。

如果瀏覽器在http header或meta tag都找不到Content-Type時會怎麼做呢？Internet Explorer會做一件很有趣的事：它會依據各位元組在各種常見語言編碼中出現的頻率，猜測網頁所用的語言及編碼方式。由於各種舊的8位元頁碼通常把該國的字母放在128到255範圍內不同的位置，而各種人類語言的字母使用頻率都有不同的分佈特性，所以這種做法的確有機會成功。這種做法真的很奇怪，不過似乎的確很有效。效果好到那些天真到不知道要用Content-Type header的網頁製作者根本不知道自己錯了，因為他們的網頁用瀏覽器來看時*一切正常*。等到某一天，當他們寫的內容不符合所用語言的字母頻率分佈時，Internet Explorer就會把它認成韓文來顯示。我認為這也證明Postel's Law中關於「發送時嚴謹，接收時寬鬆」的論點實在不是一個良好的工程原則。不管如何，當遇到這個用保加利亞文寫卻顯示成韓文(還不是有意義的韓文)的網頁時，可憐的讀者要怎麼辦呢？他會用由選單選 檢視|編碼，然後嘗試各種不同的編碼(裡面有十幾種東歐語言)直到看起來對為止。不過當然是要他會這招才行，不過大多數人都不會。



[我們公司](http://www.fogcreek.com/)有出一套網站管理軟體[CityDesk](http://www.fogcreek.com/CityDesk)，從上一版起我們決定內部全部使用UCS-2(2個位元組)的Unicode，它也是Visual Basic、COM、以及Windows NT/2000/XP的標準字串型別。寫C++程式時只要在字串宣告時用**wchar\_t**("wide char")代替**char**，再用**wcs**函數代替**str**函數(比如用**wcscat**和**wcslen**代替**strcat**和**strlen**)即可。要在C程式裡建立一個UCS-2字串常數，只要在字串前面加個L就好了，就是這樣：**L"Hello"**.

當CityDesk發行網頁時會把網頁轉成多年來廣受瀏覽器支援的UTF-8編碼。這也是*Joel on Software*上[29種語言版本](http://www.joelonsoftware.com/navLinks/OtherLanguages.html)編輯的方式，而且還沒有人跟我抱怨過有問題。

這篇文章寫到這裡已經很長了，反正我也不可能寫完所有關於字元編碼和Unicode的事情。不過我想你既然都讀到這裡了，應該也學夠了可以回去寫程式，這次別再用水螅和咒語了，改用現代的抗生素吧。這就是我留給你的工作。

程式碼:

: <http://ideone.com/PFYV3X>

我假設你是用了這個 lib <http://docs.python-requests.org/en/latest/>

Python 本身沒有內建 requests 這個 module

你應該要在原文內提到它的來源

: 使用版本是Python 3.3

: 在win7上執行會出現以下錯誤:

: Traceback (most recent call last):

: File "test.py", line 17, in <module>

: print (response.text.encode('cp950'))

: UnicodeEncodeError: 'cp950' codec can't encode character '\u30fc' in position

: 29222: illegal multibyte sequence

: 有查了一些資料 也知道是編碼的問題

: response.text的型態是str應該已經是utf-8

你的字元編碼概念需要砍掉重練

response.text 在 Python 3 是 str, 所以是 Unicode 序列

它本身沒有編碼概念

推薦閱讀

<http://www.csie.ntu.edu.tw/~p92005/Joel/Unicode.html>

如果你消化完上面那篇文章, 應該會瞭解到 Unicode 序列本身不可能會有錯誤[\*]

所以你的錯誤顯然在 encode

而如果你在上來發問前有進行過基本的嘗試

應該會發現 print(response.text.encode()) 不會發生錯誤

encode 不帶引數時預設是以 UTF-8 字元集進行編碼

所以問題在於你輸入的 Unicode 字元可以被 UTF-8 編碼, 但無法被 CP950 編碼

為什麼會這樣？

CP950 是 Windows 的 Big-5 擴充, 後者則是對 ASCII 的擴充

這些編碼（相對於 UTF-8）的共同點為擁有最大字元寬度

ASCII 是固定 8-bit, Big-5 和 CP950 則是 8 或 16-bit

所以它們可編碼的字元有限

而由於他們是針對（繁體）漢語世界發展的擴充

它們可編碼的字元以英數與繁體中文字符為主

回到你的程式本身

'\u30fc', 亦即 Unicode 字元編號 0x30fc, 是日語片平假名的長音符號

（正式名稱為 KATAKANA-HIRAGANA PROLONGED SOUND MARK）

CP950 不包含日語假名字符, 所以當然無法編碼這個字元

所以會產生你遇到的錯誤

解法？最好的解法就是不要用 CP950, 改用 UTF-8

或者如果你因為某種特殊原因非用 CP950 不可

可以多喂一個引數給 encode

根據 documentation

<http://docs.python.org/3/library/stdtypes.html#str.encode>

encode 可以接受第二個引數指明當無法編碼某個字元時應採取的行為

預設的 'strict' 會引發你看到的 exception

但其實有其他的選擇