

第 12 章

字串

到目前為止，我們已經看到了幾種資料型別：integer（整數）、long（常整數）、float（浮點數）、double（倍精度浮點數）、BOOL（布林），再加上上一章我們介紹的 pointer（指標）。我們之前接觸到字串（string）這個主題時，只有在講到 NSLog() 函式的時候提到一些。這個函式允許我們列印一個字串到螢幕上，也可以以一個值來置換以 % 符號開頭的特別碼，例如 %d。

[1]

```
float piValue = 3.1416;
NSLog(@"Here are three examples of strings printed to the screen.\n");
NSLog(@"Pi approximates %10.4f.\n", piValue);
NSLog(@"The number of eyes of a dice is %d.\n", 6);
```

我們之前沒有把字串作為一個資料型別來探討，因為我們有很好的理由。它們是物件，用 NSString 類別或 NSMutableString 類別來建立的。現在讓我們來探討這些類別，由 NSString 開始。

[2]

```
NSString *favoriteComputer;
favoriteComputer = @"Mac!";
NSLog(favoriteComputer);
```

您可能會發現第二個述句是很容易理解的，但是第一句 [2.1] 就需要一些解釋了。記得嗎？當我們宣告一個指標變數時，我們必須說明這個指標指向的是哪種資料型別。底下是第 11 章 [2] 的述句。

[3]

```
int *y;
```

我們告訴編譯器這個指標變數 y 含有可以找到一個整數的記憶體位址。在 [2.1] 中我們告訴編譯器這個指標變數 favoriteComputer 含有可以找到一個型別為 NSString 的物件的記憶體位址。我們使用指標來保存我們的字串，因為在 Objective-C 中，物件絕對不會被直接操縱，而是經由指向它們的指標來達到。

好，為什麼這個有趣的 @ 符號老是出現呢？嗯，那是因為 Objective-C 是一個 C 語言的擴充，而 C 有它自己處理字串的方式。而因為這個新的字串型別是個成熟的物件，為了區別，Objective-C 才會使用 @ 這個符號。

那麼 Objective-C 對於字串，相較於 C 語言改進了哪裡呢？嗯，Objective-C 字串是 Unicode（萬國碼）字串，而非 ASCII 字串。Unicode 字串可以顯示幾乎任何語言的字元，例如中文，就如同羅馬字母一樣。

當然，也可以同時為一個字串宣告並初始化指標變數 [4]。

[4]

```
NSString *favoriteActress = @"Julia";
```

這個指標變數 *favoriteActress* 指向一個記憶體位址，而它就是用來表示 "Julia" 這個字串的物件所儲存的地方。

一旦您初始化了這個變數，亦即 *favoriteComputer*，您就可以給予這個變數另一個值，但是您無法改變這個字串本身 [5]，因為它是一個 NSString 類別的實體。讓我們再多看一點點。

[5]

```
#import <Foundation/Foundation.h>

int main (int argc, const char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    NSString *x;
    x = @"iBook"; // [5.7]
    x = @"MacBook Pro Intel"; // 嘿，我只是試著讓這本書看起來能跟的上時代罷了！

    NSLog(x);
    [pool release];
    return 0;
}
```

執行之後，這個程式會印出：

```
MacBook Pro Intel
```

一個 NSString 類別的字串被稱為不可改變的（immutable），因為它不能被修改。

您不能修改字串有什麼好處呢？嗯，對作業系統來說，不能修改的字串比較容易處理，所以您的程式可以跑的比較快。事實上當您使用 Objective-C 來撰寫您的程式時，您會發現大部分的情況下您並不需要修改您的字串。

當然，有時候您還是會需要可以修改的字串，因此，還有另一種類別，而您用它所建立的字串物件則是可以用修改的。這個要用到的類別是 NSMutableString 類別。我們在本章稍後將會

討論到它。首先，讓我們確定您的確了解到了字串是物件。既然它們身為物件，我們就可以傳送訊息給它們。舉例來說，我們可以傳送 *length* 訊息給一個字串物件 [6]。

[6]

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int theLength;
    NSString * foo;

    foo = @"Julia!";
    theLength = [foo length]; // [6.10]
    NSLog(@"The length is %d.", theLength);

    [pool release];
    return 0;
}
```

執行之後，這個程式會印出：

The length is 6.

當要解釋事情的時候，程式設計師時常使用 *foo* 與 *bar* 作為變數名。事實上它們是不好的名字，因為它們沒有意義，就像 *x* 一樣。我們在這裡向您介紹它們，以後當您在網路上的討論看到它們時就不會被覺得奇怪了。

在第 [6.10] 行中我們向 *foo* 物件傳送了訊息 *length*。這個訊息 *length* 是定義在 `NSString` 類別中，如下所示：

```
- (unsigned int)length
Returns the number of Unicode characters in the receiver.
```

您可能也會想要把字串中的字元變成大寫 [7]。為了這個目的，請將適當的訊息，亦即 *uppercaseString*，傳送給這個字串物件，您應該可以在說明文件中自己找到它（請查看在 `NSString` 類別中可以使用的方法）。一旦接收到這個訊息，這個字串物件會建立並傳回一個動的字串物件，其中包含了相同的內容，但是每一個字元都已經轉換成相對應的大寫字元。

[7]

```
#import <Foundation/Foundation.h>
```

```
int main (int argc, const char * argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSString *foo, *bar;
    foo = @"Julia!";
    bar = [foo uppercaseString];
    NSLog(@"%@ is converted into %@.", foo, bar);

    [pool release];
    return 0;
}
```

執行之後，這個程式會印出：

```
Julia! is converted into JULIA!
```

有時候您可能會想要修改一個現存字串的內容，而非建立一個新的。這種情況下您必須使用 `NSMutableString` 類別的物件來表示您的字串。`NSMutableString` 提供了數種方法來讓您修改字串的內容。例如，`appendString:` 這個方法會把作為引數傳入的字串附加到接收者的尾端。

[8]

```
#import <Foundation/Foundation.h>
```

```
int main (int argc, const char * argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSMutableString *foo; // [8.7]
    foo = [@"Julia!" mutableCopy]; // [8.8]
    [foo appendString:@" I am happy"];
    NSLog(@"Here is the result: %@.", foo);

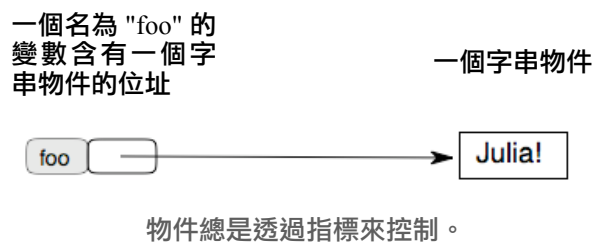
    [pool release];
    return 0;
}
```

執行之後，這個程式會印出：

```
Here is the result: Julia! I am happy.
```

在第 [8.8] 行，方法 `mutableCopy`（由 `NSString` 類別所提供）建立並傳回一個與接收者有相同內容的可改變（mutable）字串。這代表，在第 [8.8] 行執行之後，`foo` 會指向一個可改變的字串物件，而它含有字串 "Julia!"。

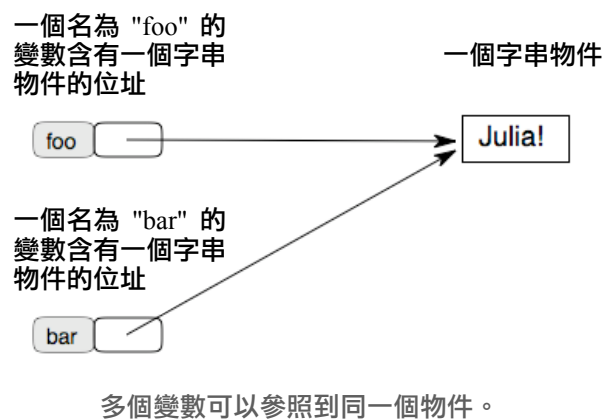
在本章前面我們已經說過，在 Objective-C 中物件絕對不會直接被操縱，而是經由指向它們的指標，這就是為什麼我們在上面的 [8.7] 行使用指標符號。事實上，當我們在 Objective-C 中使用 "物件" 這個字時，我們指的通常是 "指向一個物件的指標"。但是既然我們總是透過指標來使用物件，我們就用 "物件" 這個字來作為簡稱。而物件總是透過指標來使用的這個事實中包含了一個您必須要了解的暗示在裡面：數個變數可以同時參考到同一個物件。舉例來說，在執行完第 [8.7] 行之後，變數 `foo` 參考到一個用來表示字串 "Julia!" 的物件，我們可以用下面的圖片來表示：



現在假設我們像下面這樣把 `foo` 的值指定給變數 `bar`：

```
bar = foo;
```

這個動作的結果就是現在 `foo` 與 `bar` 都指向了同一個物件：



在這種情況下，使用 `foo` 作為接收者來傳送一個訊息給物件（亦即 `[foo dosomething];`）與使用 `bar` 來傳送訊息（亦即 `[bar dosomething];`）有相同的效果，就如下面這個例子所示：

```
[9]
```

```
#import <Foundation/Foundation.h>
```

```
int main (int argc, const char * argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    NSMutableString *foo = [@"Julia!" mutableCopy];
    NSMutableString *bar = foo;

    NSLog(@"foo points to the string: %@.", foo);
    NSLog(@"bar points to the string: %@.", bar);
    NSLog(@"-----");

    [foo appendString:@" I am happy"];

    NSLog(@"foo points to the string: %@.", foo);
    NSLog(@"bar points to the string: %@.", bar);

    [pool release];
    return 0;
}
```

執行之後，這個程式會印出：

```
foo points to the string: Julia!
bar points to the string: Julia!
-----
foo points to the string: Julia! I am happy
bar points to the string: Julia! I am happy
```

能夠同時由不同的地方參照到同一個物件對物件導向程式語言來說是一個不可缺少的功能。事實上，我們在之前的章節已經用過了這個功能。舉例來說，在第 8 章，我們就從兩個不同的按鈕物件參照到我們的 MAFoo 物件。