多平台 Objective-C 程式設計 (試讀本)

發佈 1.0.0

Michael Chen

2020 年 11 月 04 日

本書目錄

版權聲明 (Copyright)

免責聲明 (Disclaimer)

本書的內容 (文字、圖片、電腦程式等) 僅為一般性質的資訊，而非正式的技術文件。我們致力於保持本書的內容是即時和正確的，但我們無法保證本書內容的完整性、即時性、正確性、可靠性。本書的內容仍可能因人為錯誤、技術性問題等因素造成錯誤。

此外，我們也無法擔保本書使用者因使用本書或直接或間接受到本書的內容所致的任何損失或傷害。本書使用者應自行評估、判斷本書的內容對自己的風險。

本書使用者可以透過本書的超連結前往外部網站，但我們無法控制外部網站的性質、內容和可得性。我們在本網站中加入這些連結不代表我們推薦這些連結的內容或為這些連結的內容背書。我們無法擔保這些連結的內容。

當你使用本書時，表示你同意本書的聲明，會自行評估、判斷使用本書所導致的風險。

版本演進 (History)

- 1.0.0
  - 首次發佈

本書所使用的記述方式

對於 C 和 Objective-C 程式碼，會以語法高亮來輔助閱讀：

```
#include <main>

int main(void)
{
    printf("Hello World\n");

    return 0;
}
```

同樣地，對於 C 和 Objective-C 程式碼片段，也會以語法高亮來輔助閱讀：

```
/* Excerpt */
assert(0 != strcmp("hello", "goodbye"));
```

為了便於在電子書閱讀器上閱讀本書，我們的範例程式碼不會完全遵守 K&R 風格。我們會儘可能地確保排列過的程式碼仍可正確運作。

按照 Unix 的慣例，終端機會以 `$` 符號來表示指令提示符：

```
$ cd path/to/project
```

當使用 `root` 操作 Unix 終端機時，則會改用 `#` 來表示指令提示符：

```
# apt install gcc
```

按照 Windows 的慣例，終端機會以 > 來表示指令提示符。為了簡化，不顯示工作目錄：

```
> cd path\to\project
```

前言

Objective-C 是基於 C 的物件導向編譯語言 (object-oriented programming language)。原本這個語言是開發蘋果軟體的唯一官方語言，在蘋果公司發布 Swift 後，Objective-C 的定位了出現微妙的變化。

我們在這裡先不動手寫程式。本文會對 Objective-C 做概念上的說明，討論該語言在蘋果系統及非蘋果系統的相關事項。

Objective-C 的當代意義

做為一門程式語言技術，目前 Objective-C 的存在具有以下意義：

- 蘋果生態圈的主力語言之一
- 非跨平台語言而是多平台語言
- 為 C 加上 Smalltalk 風格的物件系統

原本 Objective-C 是開發蘋果軟體的唯一標準，現在則和 Swift 並列為官方選項之一。按照目前蘋果公司規畫的方向，不太會繼續為 Objective-C 新增語法特性，但龐大的 Objective-C API 也不是說廢就廢的。所以 Objective-C 仍然是可用的技術。

在非蘋果平台上，可以用 GCC 搭配 GNUstep 函式庫撰寫 Objective-C 程式。但 Objective-C 的編譯器和標準函式庫在不同平台間的歧異性，讓 Objective-C 很難像 C++ 般成為實用的跨平台物件導向語言。

在非蘋果平台上推廣 Objective-C 對蘋果公司沒什麼實質好處，可想而知蘋果公司不會投入資源去做這件事。所以 Objective-C 開發工具在異質平台間的不相容現象也不會消除。

由此可知，學習在不同平台撰寫 Objective-C 的目的是延續其技術生命，在多種平台可重覆使用相同技術寫應用程式，而非完全重用程式碼。

Objective-C 的特色是為 C 帶來 Smalltalk 風格的物件系統。所以，可以把 Objective-C 當成強化版的 C，直接使用 C 生態圈的資源。但語法特性甚少成為程式語言成功的因素。程式語言的運行環境及其殺手級應用才會決定該語言是否受到程式設計者的青睞。

Objective-C 的用途

Objective-C 定位是應用程式語言，其用途如下：

- 蘋果手機軟體
- 蘋果智慧裝置軟體 (手錶、電視)
- 蘋果桌面軟體
- (少見) 命令列工具
- (少見) 非蘋果桌面軟體
- (少見) 網頁程式

前三者是 Objective-C 的標準應用，應該很容易理解。除此之外，Objective-C 也可以當成通用型程式語言，撰寫其他層面的應用程式。但 Objective-C 在非蘋果平台的社群資源甚少，所以這方面的應用相對也少。

學 Objective-C 需要買蘋果電腦嗎？

如果學習 Objective-C 的目的是寫蘋果軟體賺錢，當然就得買蘋果電腦。但蘋果電腦相對高價，只是要拿來學程式設計的話，有點太貴了。

如果只是想先學一下 Objective-C，暫時沒有要寫蘋果軟體的話，可以先用 GCC 搭配 GNUstep 來學這個語言。GNUstep 在 Windows 或 GNU/Linux 上皆可免費取得。由於可重用現有的電腦來學 Objective-C，無形中省下一筆費用。

Objective-C 落伍了？Swift 才是新鮮貨？

由於蘋果公司鼓勵蘋果開發者轉用 Swift，且電腦書籍出版商喜歡為新技術寫書，會讓人覺得 Objective-C 已經落伍了，Swift 才是新鮮貨。

然而，大量現存蘋果軟體依☒著蘋果系統的 Objective-C API，只為了換語言就重寫軟體是浪費時間。所以，會有很長一段時間是 Objective-C 和 Swift 並存的狀態。

此外，Swift 的語法特性尚未穩定。在 Swift 5 之前，每次的大版本號變動都帶來不相容的改變。Swift 程式設計者只得耗費無謂的時間修改程式碼。不穩定的語言也不利於推廣社群函式庫。

Swift 官方團隊承諾 Swift 5 的 ABI 穩定性 (出處[1])，可以觀望一下。相對於 Swift，Objective-C 是穩定、立即可用的技術，在蘋果平台上不會有相容性議題。

Objective-C 編譯器的歧異性

Objective-C 可用的編譯器是 Clang 和 GCC。在 Objective-C 的發展中，GCC 所提供的 Objective-C 特性落後於 Clang。這個現象應該會持續很長一段時間，不太會改變。

基本上，語法特性只是輔助寫程式的手段，這些特性本身不是目的，也不是最後的產出。不使用 Clang 所帶來的新特性的話，使用 Clang 或 GCC 的差異沒那麼大。

此外，只在蘋果平台寫 Objective-C 程式的話，一定是用 Clang 編譯 Objective-C 程式碼。這時候就可以放心地使用 Clang 所帶來的新特性。

Objective-C 的標準函式庫

原先 Objective-C 只是在 C 的基礎上外加物件系統，沒有自己的標準函式庫。但沒有標準函式庫的語言不太實用。現行實務是會搭配一套預寫好的 Objective-C 物件庫，不要重造輪子。

常見的 Objective-C 物件庫有三種：

- Cocoa
- GNUstep
- ObjFW

Cocoa 是蘋果系統提供的 Objective-C 物件庫。由於蘋果公司是 Objective-C 的實質維護者，Cocoa 可視為 Objective-C 的標準物件庫。

但 Cocoa 是僅限蘋果系統可用的專有 API。GNU 基金會在無法取得 Cocoa 原始碼的前提下，儘可能地仿作 Cocoa API，其實作品就是 GNUstep 專案。由於 GNUstep 沒有商業公司支援，完全由志願開發者維護，其 API 落後 Cocoa 甚多。

ObjFW 則是另一套獨立開發的 Objective-C 物件庫。由於 ObjFW 刻意和 Cocoa 或 GNUstep 使用相異的前綴 (prefix)，兩者可以在同專案中並存。但 ObjFW 相關的資料過少，而且其授權模式不利於商業軟體。本書不會介紹 ObjFW 相關的內容。

[1] https://swift.org/blog/swift-5-released/

在 Objective-C 程式中可用的 C 標準

在 Objective-C 現存的編譯器中，不論是 Clang 或 GCC，對於 C 標準都支援得不錯。因此，在寫 Objective-C 程式時，大可放心地使用現代 C 語言 (C99、C11 等) 的特性來寫程式，不用刻意守在 ANSI C (C89)。

Objective-C++

Objective-C++ 並不是新的語言，而是混合 Objective-C 和 C++ 程式碼的模式。這個 Objective-C 方言的資料很少，連官網都把相關資料給下架了，只能在網路上找到零散的資料。

實際上只有在寫 C++ 函式庫的 Objective-C binding 會用到 Objective-C++。不建議在 Objective-C 主程式中混入 C++ 程式碼。因為 Objective-C++ 和原生 C++ 有一些差異，在 Objective-C 中使用 C++ 會受到額外限制。

本書方向

本書會專注在 Objective-C 本身，不會介紹開發蘋果軟體的部分。因此，不會強調 Xcode 的使用。讀者也不需要買蘋果主機，用現有的電腦應該就可以學習。

由於 Objective-C 是 C 的嚴格超集合，在講 Objective-C 時還是會碰到純 C 的部分。為了讓讀者省下另外買 C 程式設計教材的費用，本書會涵蓋 C 的特性。

但本書會強調 Objective-C 可用的特性，也會儘早使用 Foundation 函式庫中的常見類別。相對來說，有些純 C 的手法，像是用結構體模擬物件導向程式，或是用 C 風格字串處理文字，在 Objective-C 中有更好的替代方案，就不會採用。

使用陣列 (Array)

前言

陣列是線性 (linear)、同質 (homogeneous)、連續的 (continuous) 容器。Objective-C 有兩種陣列，一種是原生 C 陣列，另一種是 `NSArray` 物件。一般情形下，應優先使用 `NSArray` 物件，C 陣列僅留在和 C 程式碼交互操作時使用。

使用 C 陣列

宣告陣列

以下指令宣告元素為 `double` 形態，長度為 `5` 的陣列 `arr`：

```
double arr[5];
```

這時候 `arr` 內的值尚未初始化，其值不可用。

以下指令在宣告陣列時一併初始化：

```
double arr[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
```

承上，陣列長度可省略。這時會自動由元素數量來決定陣列長度：

```
double arr[] = {1.1, 2.2, 3.3, 4.4, 5.5};
```

在 **C99** 後可以改用以下方式來初始化陣列：

```
double arr[] = {
    [0] = 1.1,
```

(下页⊠⊠)

(繼續上一頁)

```
    [1] = 2.2,
    [2] = 3.3,
    [3] = 4.4,
    [4] = 5.5
};
```

這樣寫可以快速得知元素的位置和值。如果專案不限於 ANSI C，建議使用這種方式來初始化陣列。

存取陣列元素

使用一對 [和] 可隨機存取陣列元素。參考以下例子：

```
#include <assert.h>

#define ABS(a) ((a) > 0.0 ? (a) : -(a))
#define IS_EQUAL(a, b, epsilon) \
    (ABS((a) - (b)) <= (epsilon))

int main(void)
{
    double arr[] = {1.1, 2.2, 3.3, 4.4, 5.5};

    assert(IS_EQUAL(1.1, arr[0], 0.00001));
    assert(IS_EQUAL(2.2, arr[1], 0.00001));
    assert(IS_EQUAL(3.3, arr[2], 0.00001));
    assert(IS_EQUAL(4.4, arr[3], 0.00001));
    assert(IS_EQUAL(5.5, arr[4], 0.00001));

    /* Mutate `arr[1]`. */
    arr[1] = 99.99;

    assert(IS_EQUAL(1.1, arr[0], 0.00001));

    /* `arr[1]` is changed. */
    assert(IS_EQUAL(99.99, arr[1], 0.00001));

    assert(IS_EQUAL(3.3, arr[2], 0.00001));
    assert(IS_EQUAL(4.4, arr[3], 0.00001));
    assert(IS_EQUAL(5.5, arr[4], 0.00001));

    return 0;
}
```

C 陣列的索引必需為自然數 (零或正整數)。此外，C 陣列不會自動檢查邊界，程式設計者

要負起相關的責任。

計算陣列長度

C 陣列沒有儲存陣列長度的資訊。可使用以下巨集可計算陣列長度：

```
#define ARRAY_SIZE(arr) \
    (sizeof(arr) / sizeof(arr[0]))
```

但這個巨集僅限於靜態或自動配置記憶體的陣列才能用。

走訪陣列

C 陣列沒有迭代器。走訪陣列的方式是直接以索引走訪：

```
#include <stdio.h>

#define ARRAY_SIZE(arr) \
    (sizeof(arr) / sizeof(arr[0]))

int main(void)
{
    double arr[] = {1.1, 2.2, 3.3, 4.4, 5.5};

    {
        size_t i;
        for (i = 0; i < ARRAY_SIZE(arr); ++i)
            printf("%.1f\n", arr[i]);
    }

    return 0;
}
```

這是因為 C 的設計以簡約為目標，不會放入迭代器等抽象度較高的特性。

宣告動態陣列

C 陣列的長度是固定的。如果想要使用動態陣列，得自行實作資料結構。以下結構體是以 `double` 為元素的動態陣列：

```
typedef struct array_double_t array_double_t;
```

(下页⊠⊠)

(繼續上一頁)

```
struct array_double_t {
    size_t size;
    size_t capacity;
    double *elements;
};
```

在此陣列型態中 size 及 capacity 和陣列長度相關。前者是動態陣列的當前大小，後者是動態陣列的最大容量。

以下函式展示建立此動態陣列的過程：

```
array_double_t * array_double_new()
{
    array_double_t *arr = \
        (array_double_t *) malloc(sizeof(array_double_t));
    if (!arr)
        return arr;

    arr->size = 0;
    arr->capacity = 16;  /* Arbitrary size. */
    arr->elements = \
        (double *) calloc(arr->capacity, sizeof(double));

    if (!(arr->elements)) {
        free(arr);
        arr = NULL;
        return arr;
    }

    return arr;
}
```

由於實作動態陣列屬於資料結構的範圍，這裡就不繼續展示其他的函式。此外，Objective-C 已經有 NSArray 類別了，自行手刻動態陣列的機會甚少。

使用 Objective-C 的陣列物件

陣列物件存取的元素也是物件

Objective-C 中所有的內建容器的元素皆為物件，包括陣列物件。若要存取來自 C 的基礎型態資料，要用 wrapper 將該資料轉為物件後才能做為陣列物件的元素。

從 NSArray 和 NSMutableArray 中擇一

根據其可變性，Objective-C 的陣列物件可分為 `NSArray` 及 `NSMutableArray` 兩種。由於前者的效能比較好，只有在需要可變動的陣列物件時才會使用後者。

宣告陣列物件

尚未建立陣列物件時，應該將其值指向 `nil`：

```
NSArray *arr = nil;
```

在 Clang 中可以使用陣列物件實字來建立陣列物件：

```
NSArray *arr = @[
    @(1.1), @(2.2), @(3.3), @(4.4), @(5.5)
];
```

在非蘋果平台上要考量編譯器相容性，較不建議用這種寫法。

使用 `+arrayWithObjects` 類別方法可建立陣列物件：

```
NSArray *arr = [NSArray arrayWithObjects:
    [NSNumber numberWithDouble:1.1],
    [NSNumber numberWithDouble:2.2],
    [NSNumber numberWithDouble:3.3],
    [NSNumber numberWithDouble:4.4],
    [NSNumber numberWithDouble:5.5],
    nil];
```

注意尾端要加上 `nil` 做為參數結訊的信號。

也可以使用 C 陣列做為參數來建立陣列物件：

```
NSNumber *cArr[] = {
    [NSNumber numberWithDouble:1.1],
    [NSNumber numberWithDouble:2.2],
    [NSNumber numberWithDouble:3.3],
    [NSNumber numberWithDouble:4.4],
    [NSNumber numberWithDouble:5.5]
};

NSArray *arr = \
    [NSArray arrayWithObjects:cArr count: 5];
```

由於 C 陣列本身沒有長度的資訊，得自行填入參數。

存取陣列物件內的元素

使用 -objectAtIndex: 方法可取出陣列物件中的元素，使用 -replaceObjectAtIndex:withObject: 方法可修改陣列物件中的元素。參考以下例子：

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

#ifdef __OBJC__
#undef ABS
#endif

#define ABS(a) ((a) > 0.0 ? (a) : -(a))
#define IS_EQUAL(a, b, epsilon) \
    (ABS((a) - (b)) <= (epsilon))

int main(void)
{
    NSAutoreleasePool* pool = \
        [[NSAutoreleasePool alloc] init];

    NSMutableArray *arr = \
        [NSMutableArray arrayWithObjects:
            [NSNumber numberWithDouble:1.1],
            [NSNumber numberWithDouble:2.2],
            [NSNumber numberWithDouble:3.3],
            [NSNumber numberWithDouble:4.4],
            [NSNumber numberWithDouble:5.5],
            nil];

    assert(IS_EQUAL(1.1, \
        [[arr objectAtIndex:0] doubleValue], 0.00001));
    assert(IS_EQUAL(2.2, \
        [[arr objectAtIndex:1] doubleValue], 0.00001));
    assert(IS_EQUAL(3.3, \
        [[arr objectAtIndex:2] doubleValue], 0.00001));
    assert(IS_EQUAL(4.4, \
        [[arr objectAtIndex:3] doubleValue], 0.00001));
    assert(IS_EQUAL(5.5, \
        [[arr objectAtIndex:4] doubleValue], 0.00001));

    /* Mutate the object at index 1. */
    [arr replaceObjectAtIndex:1 \
        withObject: [NSNumber numberWithDouble:99.99]];
```

(下页⊠⊠)

(繼續上一頁)

```
    assert(IS_EQUAL(1.1, \
        [[arr objectAtIndex:0] doubleValue], 0.00001));

    /* The object at index 1 is changed. */
    assert(IS_EQUAL(99.99, \
        [[arr objectAtIndex:1] doubleValue], 0.00001));

    assert(IS_EQUAL(3.3, \
        [[arr objectAtIndex:2] doubleValue], 0.00001));
    assert(IS_EQUAL(4.4, \
        [[arr objectAtIndex:3] doubleValue], 0.00001));
    assert(IS_EQUAL(5.5, \
        [[arr objectAtIndex:4] doubleValue], 0.00001));

    [pool release];

    return 0;
}
```

計算陣列物件長度

陣列物件建立時，就已經包含陣列長度的資訊。使用 `-count` 方法即可取出陣列物件的長度。參考以下程式碼片段：

```
NSArray *arr = [NSArray arrayWithObjects:
    @(1.1), @(2.2), @(3.3), @(4.4), @(5.5), nil
];

assert(5 == [arr count]);
```

走訪陣列物件

走訪陣列物件的方式有以下三種：

- 使用 `NSEnumerator` (迭代器)
- 使用 `for ...in` 敘述
- 使用 block 敘述 (Clang 限定)

以下例子使用 `NSEnummerator` 來走訪陣列物件：

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

int main(void)
{
    NSAutoreleasePool* pool = \
        [[NSAutoreleasePool alloc] init];

    NSArray *arr = [NSArray arrayWithObjects:
        [NSNumber numberWithDouble:1.1],
        [NSNumber numberWithDouble:2.2],
        [NSNumber numberWithDouble:3.3],
        [NSNumber numberWithDouble:4.4],
        [NSNumber numberWithDouble:5.5],
        nil];

    NSEnumerator *enumerator = \
        [arr objectEnumerator];

    {
        id obj;

        while (obj = [enumerator nextObject]) {
            printf("%.1f\n", \
                [obj doubleValue]);
        }
    }

    [pool release];

    return 0;
}
```

迭代器 (iterator) 是一種抽象物件。使用迭代器就可以在不碰觸容器內部的前提下走訪物件。

以下例子使用 `for ...in` 敘述來走訪容器：

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

int main(void)
{
    NSAutoreleasePool* pool = \
        [[NSAutoreleasePool alloc] init];

    NSArray *arr = [NSArray arrayWithObjects:
        [NSNumber numberWithDouble:1.1],
```

(下页□□)

(繼續上一頁)

```
        [NSNumber numberWithDouble:2.2],
        [NSNumber numberWithDouble:3.3],
        [NSNumber numberWithDouble:4.4],
        [NSNumber numberWithDouble:5.5],
        nil];

    for (id obj in arr)
        printf("%.1f\n", [obj doubleValue]);

    [pool release];

    return 0;
}
```

for ...in 敘述是 Objective-C 為容器所新增的迴圈語法。

Block 敘述是 Clang 限定的新語法。以下例子使用 block 敘述來走訪陣列元素：

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

int main(void)
{
    NSAutoreleasePool* pool = \
        [[NSAutoreleasePool alloc] init];

    NSArray *arr = [NSArray arrayWithObjects:
        @(1.1), @(2.2), @(3.3), @(4.4), @(5.5), nil
    ];

    [arr enumerateObjectsUsingBlock:\
        ^ (id obj, NSUInteger i, BOOL *stop) {
            if (!obj)
                *stop = YES;

            printf("%.1f\n", [obj doubleValue]);
        }];

    [pool release];

    return 0;
}
```

Block 敘述並非 Objective-C 限定的語法，而是 Apple Clang 新增的特性。要在非蘋果平台上的 Clang 使用 block 敘述的話，要經過額外的措施。詳見附錄的說明。