iOS

Learn Objective C

Nguyễn Giang Nam
namng174@gmail.com
092.881.3468

Giới thiệu

- Ngôn ngữ lập trình Objective C (ObjC) là một ngôn ngữ lập trình được xây dựng dựa trên nền tảng ngôn ngữ C. Mọi thứ làm việc trong C đều làm việc được trong ObjC. ObjC thêm vào nhiều tính năng mới từ C, thêm "lập trình hướng đối tượng" (Object-Oriented Programming, OOP) và Smalltalk – message passing.
- Objective C là ngôn ngữ được biên dịch thành mã máy (native machine code) giống với C/C++, và do đó nó nhanh hơn so với những ngôn ngữ chạy trên máy ảo như Java, Python hoặc Ruby.
- Mặt khác, ObjC có nhiều tính năng làm cho nó linh động hơn C++. ObjC hỗ trợ cả "định kiểu tĩnh– static typing" và "định kiểu động – dynamic typing", tuỳ thuộc vào biến được khai báo. Nhưng về hiệu năng làm việc thì ObjC kém hơn C++.
- Ngôn ngữ Objective C được ra mắt vào năm 1983 bởi Brad Cox và Tom Love.

Source Code

- Trong Objective C không sử dụng namespaces giống như các ngôn ngữ khác, thay vào đó để phân biệt các class đến từ các frameworks khác nhau ObjC sử dụng name prefixes ở tên của class.
- Các tập tin được tổ chức bằng cách tương tự như với C/C++: tập tin tiêu đề (*.h) - chứa các khai báo, tập tin mã nguồn (*.m) - thực hiện viết các dòng lệnh triển khai. (*.h) nghĩa là header, (*m) nghĩa là “message” hoặc là “methods”.
- Chương trình thực thi bắt đầu với hàm int main().
- Để có thể sử dụng frameworks hoặc các class tự định nghĩa cần sử dụng từ khoá #include.

Class & Object

- Chúng có vẻ giống nhau, nhưng **không phải là một**. Class là một định nghĩa, trong khi Object là một thể hiện của Class đã tạo. Class như một bản vẽ kĩ thuật trong khi Objects là đối tượng tồn tại thực sự.
- Cú pháp để tạo ra một class trong Objective C là rất đơn giản. Nó bao gồm hai phần:
  - Class interface thường được lưu trữ trong tập tin ClassName.h, định nghĩa các instance variables và public method.
  - Mã nguồn thực thi được viết ở trong tập tin ClassName.m.
- Objective C sử dụng đơn kế thừa.
- Sau đây là cú pháp thường dùng để tạo ra class:

Class & Object

Class Header (.h)

```
#import "AnyHeaderFile.h"
@interface ClassName : SuperClass {
// declare instance variables
}
// define properties
// define methods (including any // custom initializers)
@end
```

Class Implementation (.m)

```
#import "YourClassName.h”
@implementation ClassName
// synthesize properties
// implement methods (including any custom initializers, and dealloc)
@end
```

Class & Object

- Cách khởi tạo Object (Class instance):

```
//cách thứ nhất:
ClassName * myClass = [[ClassName alloc] init];

//cách thứ hai:
ClassName * myClass = [ClassName new];

//cách thứ ba:
ClassName * myClass = [[ClassName alloc] initWithSomething:something];

//myClass được sử dụng cho đến khi gọi release:
[myClass release];  // không cần thiết với ARC
```

Variable & Property

Variable

- Variable access (quyền truy cập) được thiết lập bởi các từ khoá **@public**, **@private**, **@protected**; các từ khoá sử dụng được đặt trước nhóm variable cần thiết lập. Thiết lập mặc định là **@protected**.
- Để truy cập instance variable sử dụng "->" ví dụ: person->name .
- Nếu bạn muốn sử dụng static variable trong class, khi khai báo bạn sử dụng từ khoá static trong header file, bên ngoài vùng của @interface (ví dụ: **static anytype staticVariable;**).
- Cách khai báo variables: **anytype myVariable;**

```
@public
anytypeA myVariableA;
anytypeB myVariableB;
```

Variable & Property

Property

- Trong @interface, khai báo như sau **@property anytype age;**. Tương đương với việc khai báo getter: (anytype) age; *(trong ObjC getter không có tiền tố "get")* và setter: (void) setAge: (anytype)age.
- Trong @implementation, sử dụng **@synthesize age;**, tương đương với tạo ra các lệnh thực thi getter và setter mặc định. Có thể tạo ra các lệnh thực thi tuỳ biến cho getter và setter bằng việc overide lại setter và getter mà không sử dụng **@synthesize**.
- Để tránh nhầm lẫn giữa variable và property, các property không thể sử dụng "->" để truy cập mà sử dụng **obj.age** hoặc **[obj age]**.
- Các property có thể có các thuộc tính tuỳ chọn bằng cách thêm vào sau từ khoá @property, như là: @property (readonly) anytype age; Có thể tìm hiểu thêm các thuộc tính từ tài liệu apple cung cấp.

Method

- Cách định nghĩa một method:

```
- (void)doIt;  hoặc  - (anytype)doIt;
+ (void)doIt;  hoặc  + (anytype)doIt; //class method
- (anytype)doItWithA:(anytype)a;
- (anytype)doItWithA:(anytype)a  andB:(anytype)b;
```

- Cách triển khai method:

```
- (anytype)doItWithA:(anytype)a  andB:(anytype)b {

// Do something with a and b... return retVal;

}
```

Method

- Một điều khác biệt về method trong Objective C là các tham số được đưa ra cùng với khái niệm label, được sử dụng trong việc gọi hàm (với Objective C việc gọi hàm được định nghĩa là gửi một message tới object – kế thừa từ Smalltalk).
- Label thực sự được hình thành từ tên đầy đủ của method. Bởi vậy method trước có label là "**doItWithA:andB:**".
- Method trong Objective C không thể overload bằng kiểu dữ liệu của tham số như trong các ngôn ngữ thông dụng, nên không thể cùng lúc có hai method **doIt(typeA)** và **doIt(typeB)**.
- Trong Objective C, khi gửi một message (gọi một method) không tồn tại sẽ không đưa ra lỗi lúc biên dịch, nhưng sẽ đưa ra exception trong lúc thực thi. Có thể bắt và xử lý với cú pháp try...catch.
- Method đôi khi được sử dụng với khái niệm là "**selector**".

Category

- Trong ObjC, ngoài việc kế thừa từ các class, bạn cũng có thể thêm code vào các class hiện có. Có thể phát triển các class hiện có mà không cần phải thay đổi code trong class đó, tuy nó không mạnh mẽ như Ruby (ví dụ như bạn không thể thêm các instance varibale mới) nhưng điều này thực sự rất tuyệt vời.
- Ở đây nó được gọi là “category” bởi vì nếu có một class nhiều method, có thể phân chia các method của nó thành nhiều loại và đặt chúng trong các file khác nhau.
- Dưới đây là cách để khai báo một category:

```
@interface ExistingClass (SomeNewName)
// methods
@end
```

- Sau đó, include cả header cũ và header mới vào nơi bạn muốn sử dụng, và bạn đã có một class với các method mới được thêm vào.

Data types

- ObjC có cả hai kiểu dữ liệu là primitive và object. Với kiểu object chỉ có kiểu con trỏ và không có kiểu giá trị.
- Các kiểu dữ liệu primitive (cơ bản): char, int, float, double, short, long, BOOL. ObjC không khuyến khích sử dụng các kiểu dữ liệu primitive, có các kiểu thay thế ví dụ như NSInteger, NSUInteger, hoặc CGFloat (Đây là các phiên bản có kiến trúc an toàn của int, unsigned int, float). Ngoài ra còn có một class là NSNumber sử dụng cho việc boxing/unboxing kiểu pimitive khi cần chuyển thành kiểu dữ liệu object.
- Kiểu dữ liệu logic được gọi là BOOL, và giá trị không giống như các ngôn ngữ khác True/False mà ObjC sử dụng YES/NO ☺.
- Kiểu Null gọi là nil. Đối với kiểu con trỏ, nil được viết là Nil. Tương tự như NSNumber cho int, class dùng cho nil là NSNull.
- Kiểu chuỗi trong ObjC là NSString.

Loops

- Cấu trúc của lệnh vòng lặp trong ObjC không khác biệt nhiều với các ngôn ngữ khác. ObjC có các lệnh lặp for, while, do-while, for-each.
- Sau đây là cú pháp của các vòng lặp:

```
for (<#initialization#>; <#condition#>; <#increment#>) {
        <#statements#>
}

for (int i=0; i<n; i++) {
        //Statements here...
}
```

Loops

```
while (<#condition#>) {
        <#statements#>
}
```

```
while (a<b) {
        //Statements here..
}
```

```
do {
        <#statements#>
} while (<#condition#>);
```

```
do {
        //Statements here..
    } while (a<b);
```

```
for (<#type *object#> in <#collection#>) {
        <#statements#>
}
```

```
for (anyType *obj in anyArray) {
        //Statements here..
}
```

Conditions

- ObjC có các lệnh điều kiện: if, switch.
- Sau đây là cú pháp của các lệnh điều kiện:

```
if (<#condition#>) {
        <#statements#>
    }
```

```
if (a<b) {
        //Statements here..
    }
```

```
switch (<#expression#>) {
        case <#constant#>:
            <#statements#>
            break;
        default:
            break;
}
```

```
switch (i) {
        case 1:
            //Statements here..
            break;
        default:
            break;
}
```

Protocols

- Protocol định nghĩa một danh sách các method bắt buộc hoặc tùy chọn mà các class chấp nhận (adopt) protocol phải thực thi.
- Các method khai báo trong protocol cũng có thể là các khai báo property. Các từ khóa @optional và @required thể hiện theo đúng ý nghĩa của nó, nếu không sử dụng từ khóa thì mặc định là @required.
- Protocol được sử dụng trong các trường hợp:
  - Khai báo các method dự kiến sẽ được thực thi.
  - Khai báo một interface cho một object trong khi ẩn đi class của nó.
  - Khảo sát tương đồng giữa các class mà không phải liên quan đến cấu trúc thứ bậc.

Protocols

- Cách khai báo một protocol

```
@protocol FooInterface
- (int) foo;
- (void) bar;
@end
```

- Cách class chấp nhận (adopt) protocol

```
@interface MyClass: NSObject <FooInterface>
```

Các method khai báo trong protocol không cần thiết phải khai báo lại trong @interface của class.

Reflection

- ObjC có khả năng động hơn C và C++, mặc dù thực tế rằng nó thực sự là một ngôn ngữ biên dịch trực tiếp mã máy mạnh mẽ. Dưới đây là một số ví dụ về các chức năng liên quan đến reflection:
  - isKindOfClass:[MyClass class] - cho biết rằng object có class thừa kế từ MyClass hay không.
  - isMemberOfClass:[MyClass class] - cho bết rằng class của object có phải là MyClass hay không.
  - respondsToSelector:@selector(doIt) - cho biết rằng object có method doIt: hay không.
  - performSelector:@selector(doIt) - gọi method được chỉ định doIt: của object.

Exceptions & debugging

- Exceptions làm việc giống như trong tất cả các ngôn ngữ hiện đại khác: chúng được đại diện bởi các class kế thừa (thông thường) từ NSException, và sử dụng các từ khoá @try, @catch và @finally để chặn các exception rồi ném ra bên trong một block (khối) và xử lý chúng. Sử dụng @throw e để ném ra exception e.
- Hai method có thể hữu ích trong debug (gỡ lỗi):
  - NSAssert(x != 0, @"X must not be zero"); - đưa ra exception nếu x bằng 0.
  - NSLog(@”result: x = %@", x); - ghi ra màn hình console chuỗi, và cũng ghi một vài dòng vào /var/log/system.log.

Memory management

- Các object trong ObjC là reference counted(tham chiếu đếm).
  - Các object bắt đầu với reference là 1.
  - Tăng reference với method retain.
  - Giảm reference với method release, autorelease.
  - Khi số lượng reference là 0, runtime (trình biên dịch) sẽ gọi method dealloc.
- Khi bạn nắm quyền sở hữu một object, khởi tạo object bằng các method mà trong tên bắt đầu với “alloc”, “new” hoặc “copy” (ví dụ, alloc, allocWithZone, newObject hoặc mutableCopy…) hoặc gửi một message retain, bạn phải có trách nhiệm giải phóng quyền sở hữu object đó bằng cách sử dụng release hoặc autorelease. Bất kỳ khi nào bạn nhận được một object (không phải tự mình khởi tạo), bạn không được release nó.
- Từ iOS 4.3 về sau đã hỗ trợ quản lý bộ nhớ tự động (ARC) với garbage collector. ARC thực sự giúp developer dễ dàng loại bỏ memory leaks(thất thoát bộ nhớ).

retain/release/autorelease

```
@implementation BankAccount
- (NSString *)accountNumber {
        return [[accountNumber retain] autorelease];
}
- (void)SetAccountNumber {
        if (accountNumber != newNumber) {
                [accountNumber release];
                accountNumber = [newNumber retain];
        }
}
- (void)dealloc {
        [self setAccountNumber:nil];
        [super dealloc];
}
@end
```