# Learning Objective-C: A Primer (中文版)

Objective-C 语言是一门简单的计算机语言，它设计的目标是帮助开发者精巧熟练地进行面向对象编程。Objective-C 是 ANSI 标准 C 语言的扩展，通过提供定义类、方法及其他结构的语法，实现基于类的动态扩展。

> 重要：本文档不传授任何 C 语言本身的知识。如果您尚不熟悉 C 语言，请先进行相关的基础学习。
>
> 在开始之前，您还需要对面向对象编程概念有基本的理解。对象的使用和面向对象设计模式是设计 Cocoa 应用程序的基础，理解对象之间的交互方式对于创建你自己的应用程序也是至关重要的。要获得相关概念的概览，可以阅读 Object-Oriented Programming with Objective-C。另外，阅读 Cocoa Fundamentals Guide 可以了解到 Cocoa 中使用到的设计模式。

如果您熟悉 C 语言，并且曾经使用面向对象的语言编写程序，那么本文中的内容将帮助您学习 Objective-C 的基本语法。很多面向对象的传统概念如封装、继承、多态，都会出现在 Objective-C 中。尽管也有些显著的差异，但本文都略有涉及，如果需要更详细的信息可以从其他资料中获取。

要详细了解 Objective-C 语言和语法，请参阅 The Objective-C Programming Language。

内容：



## Objective-C：C 语言的超集

Objective-C 是 ANSI 版本 C 编程语言的一个超集，它的基本语法与 C 语言相同。在编写 C 代码时，一般通过定义头文件和源文件来将代码的公开声明和具体实现分隔开。Objective-C 头文件使用表 1-1 中列出的扩展名。

表 1-1 Objective-C 代码文件扩展名

| 扩展名 | 源代码类型 |
| --- | --- |
| `.h` | 头文件。头文件包含类（class）、类型（type）、函数（function）和常量（constant）的声明。 |
| `.m` | 源文件。这是源文件使用的典型扩展名，可以同时包含 Objective-C 和 C 代码。 |
| `.mm` | 源文件。这种类型的源文件不但可以包含 Objective-C 和 C 代码，而且可以包含 C++ 代码。只有确实在 Objective-C 代码中引用了 C++ 类或特性才使用这个扩展名。 |

当你希望在源代码中包含头文件时，一般使用 `#import` 指令。它和 `#include` 相似，不同点在于它会确保同一个文件不会被包含多次。Objective-C 示例和文档中都使用 `#import`，你编写的代码中也应该使用它。

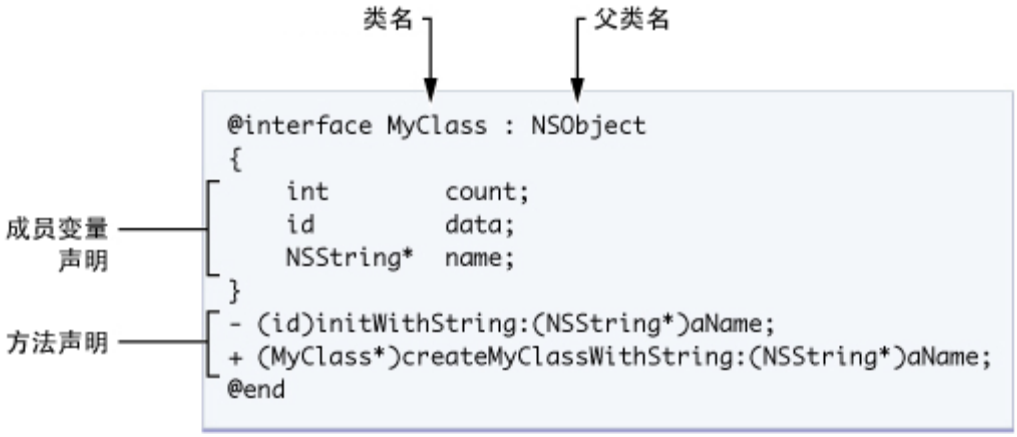
## 类（Classes）

和其他面向对象语言类似，Objective-C 中的类提供了封装数据和操作这些数据的动作的基本构造。对象是类在运行时的实例，它包含由类声明指定的实例变量和指向类方法的指针的内存拷贝。

Objective-C 规范要求一个类由两部分组成：接口（interface）和实现（implementation）。接口部分包含类的声明，并定义和类相关的实例变量与方法。接口通常写在 `.h` 文件中。实现部分则包含类的方法对应的实际代码。 实现通常写在 `.m` 文件中。

图 1-1 显示了声明一个类用到的语法，这个类的名称是 `MyClass`，它继承了 Cocoa 中的基础类 `NSObject`。类声明以 `@interface` 编译指令开始，以 `@end` 指令结束。紧随在类名之后（并以一个冒号分隔）的是父类名称。类的实例变量（有时称为“ivars”，在其他语言中也称为“成员变量”）在以花括号（`{` 和 `}`）标示的代码块中声明。在实例变量代码块之后是类声明的方法的列表。在每个实例变量和代码声明末尾都要有一个分号。

图 1-1 类的声明




> 注意：这个接口只声明了方法；除此之外，类还可以声明属性。关于属性的更多信息请查看本文的“属性（Properties）声明”一节。

Objective-C 对于包含对象的变量既支持强类型也支持弱类型。强类型变量在声明中包含类名。弱类型变量则使用 `id` 作为对象的类型。弱类型变量经常用在集合类等场合，因为集合内的对象具体为何种类型也许不可知。如果你习惯了强类型语言，也许会觉得使用弱类型变量会有问题，但是它们提供了强大的灵活性，给 Objective-C 程序带来显著的动态性。

下面是声明强类型和弱类型变量的例子：

```
MyClass *myObject1;  // 强类型声明
id       myObject2;  // 弱类型声明
```

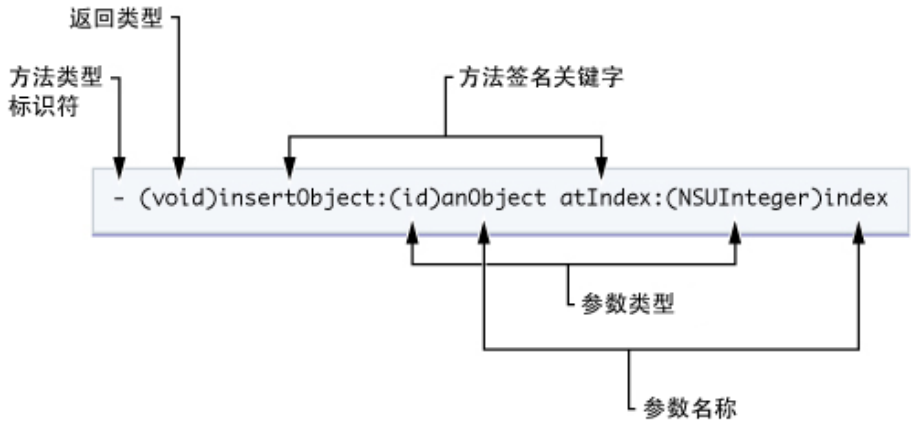
请留意第一句声明中的 `*`。在 Objective-C 语言中，对象是通过指针引用的。如果你不太明白也没有关系——在 Objective-C 编程中无需精通指针。你只需要记住在强类型变量对象名之前加上 `*`。`id` 类型则隐含了指针。

## 方法（Methods）和消息（Messaging）

在 Objective-C 中，类可以声明两种类型的方法：实例方法和类方法。实例方法的执行范围仅限于类的一个具体的实例之内。也就是说，如果你要调用一个实例方法，那么必须首先创建一个类的实例。类方法则不需要你创建实例，详情稍后再作讨论。

方法的声明包含方法类型标识符、返回类型、一个或多个签名关键字（signature keywords），以及参数类型和参数名称等信息。图 1-2 显示了 `insertObject:atIndex:` 实例方法的声明。

图 1-2 方法声明语法




声明以一个减号（-）开始，表示这是一个实例方法。方法的实际名称（`insertObject:atIndex:`）由所有签名关键字连接起来构成，包括冒号。冒号用来声明参数的存在。如果一个方法没有参数，则只写明第一个（也是仅有的一个）签名关键字，并忽略冒号。在这个例子中，方法有两个参数。

当你想调用一个方法时，需要向对象发送消息。消息由方法签名，和方法所需的参数信息构成。所有发送到对象的消息都是被动态调度，Objective-C 以此实现类的多态行为。

消息被括在方括号（`[` 和 `]`）中。在方括号中，左边是接收消息的对象，右边是消息（以及消息所需的参数）。举例来说，要发送 `insertObject:atIndex:` 消息到 `myArray` 变量中的一个对象，你需要使用下面的语法：

```
[myArray insertObject:anObject atIndex:0];
```

为了避免声明过多本地变量来存储临时结果，Objective-C 允许嵌套消息。每个嵌套消息的返回值可以被用作其他消息的参数或目标。举例来说，你可以在前面的例子中用消息代替任何一个变量来提供值。也就是说，如果你有另外一个对象叫做 `myAppObject`，它有适当的方法用来访问表示数组的对象和即将被插入到数组的对象，那么你可以在前面例子的基础上写出类似下面的代码：

```
[[myAppObject theArray] insertObject:[myAppObject objectToInsert] atIndex:0];
```

Objective-C 还提供句点（.）语法来调用访问器方法（accessor methods）。访问器方法可以获得和设置对象的状态，典型的形式是 -(type)propertyName 和 -(void)setPropertyName:(type)。您可以借助句点语法把前面的例子改写成：

```
[myAppObject.theArray insertObject:[myAppObject objectToInsert] atIndex:0];
```

句点语法可用在赋值中：

```
myAppObject.theArray = aNewArray;
```

这相当于：

```
[myAppObject setTheArray:aNewArray];
```

虽然在前面的例子中，消息被发送到类的实例，但你也可以向类本身发送消息。向类发送消息时，接收消息的方法必须是类方法，而不能是实例方法。

类方法经常被用作工厂方法来创建类的新实例，或者用来访问和类相关的某些共享信息。声明类方法的语法和声明实例方法的语法非常相似，只有一点不同。类方法声明以加号（+）开头，而不是减号。

下面的例子描述了如何使用类方法作为一个工厂方法，array 方法 是 NSArray 类中的一个类方法，从 NSMutableArray 继承，这个方法分配和初始化类的新实例并把它返回给你的代码。

```
NSMutableArray *myArray = nil;  // nil 基本上等同于 NULL

// 创建新的数组并将其赋予 myArray 变量
myArray = [NSMutableArray array];
```

列表 1-1 给出了图 1-1 中 MyClass 的实现。和声明类似，类的实现也由两个编译指令标示，@implementation 和 @end。这些指令告诉编译器，在指令范围内的方法是和哪个类相对应的。因此方法定义要符合接口中的声明，并包含代码块。

列表 1-1　类的实现

```
@implementation MyClass

- (id)initWithString:(NSString *)aName{

    self = [super init];
    if (self) {
        name = [aName copy];
    }
    return self;
}

+ (MyClass *)createMyClassWithString: (NSString *)aName{

    return [[[self alloc] initWithString:aName] autorelease];
}
@end
```

## 属性（Properties）声明

属性声明是一种简便的标注，它可以代替访问器方法的声明和部分实现。

属性的声明和方法的声明一样，都是在类接口定义中进行。基本的方式是使用 `@property` 编译指令，接着写出类型信息和属性名称。此外还有一些附加选项，用来定义访问器方法特别的行为。

下面的示例显示了几个简单的属性声明：

```
@property BOOL flag;
@property (copy) NSString *nameObject; // 赋值时拷贝对象
@property (readonly) UIView *rootView; // 只声明 getter 方法
```

每个可读属性将指定一个与属性同名的方法。

每个可写属性将指定一个形如 `set`PropertyName`:` 的方法，在其中属性名的第一个字母被转换为大写。

在类实现中，你可以使用 `@synthesize` 编译指令告诉编译器根据声明的规范来生成对应的方法。

```
@synthesize flag;
@synthesize nameObject;
@synthesize rootView;
```

你还可以把多个属性的 `@synthesize` 语句放置在同一行：

```
@synthesize flag, nameObject, rootView;
```

从实践角度讲，应用属性可以减少你需要写的代码量。由于大多数访问器方法都用同样的方式实现，因此你不用去为类中每一个暴露的属性都去编写一次 getter 和 setter 的实现。而是使用属性声明指定属性的行为，然后在编译时基于声明生成实际的 getter 和 setter 方法。

要获得更多关于如何在你自己的类中声明属性的信息，请阅读 The Objective-C Programming Language 中的“Declared Properties”一章。

## 字符串（Strings）

作为 C 的超集，Objective-C 支持与 C 语言相同的指定字符串的规则。即单个字符括在单引号中，字符串括在双引号中。但是，Objective-C 框架通常不使用 C 风格的字符串。而是使用 `NSString` 对象传递字符串。

`NSString` 类是一个面向字符串的对象包装器，它拥有你期望的一切优点，包括用于存储任意长度字符串的内建内存管理，Unicode 支持，`printf` 风格的格式化功能，还有更多。由于字符串非常常用，因此 Objective-C 提供一种从常量值创建 `NSString` 对象的简便方式。要使用这种简便方式，你只需要在双引号括起来的字符串之前加上一个 `@` 符号，就像下面的示例：

```
NSString *myString = @"My String\n";
NSString *anotherString =
  [NSString stringWithFormat:@"%d %@", 1, @"String"];

// 从 C 字符串创建 Objective-C 字符串
NSString *fromCString =
  [NSString stringWithCString:"A C string" encoding:NSASCIIStringEncoding];
```

## 协议（Protocols）

协议用来声明可以被任何类实现的方法。协议并不是类。在协议中只需定义由其他对象来负责实现的接口。当你在一个类中实现了一个协议中的方法时，我们说这个类符合（conform）这个协议。

协议经常用于为委托（delegate）对象指定接口。要了解协议、委托和其他对象的关系，最好的方法是看看下面这个例子。

`UIApplication` 类实现了一个应用程序必需的行为。要接收一些关于应用程序当前状态的简单通知，你没有必要去使用 `UIApplication` 的子类，`UIApplication` 类通过调用赋予到它的委托对象的特定方法来传递这些通知。那些实现了 `UIApplicationDelegate` 协议中的方法的对象均可接收这些通知并作出适当响应。你可以通过在接口块中被继承的类名之后加上“<>”括起来的协议名称来指定你的类符合或采用（adopts）一个协议。在这里并没有必要声明将要实现的协议方法。

```
@interface MyClass : NSObject <UIApplicationDelegate, AnotherProtocol> { }

@end
```

协议声明和类的接口声明类似，不同点在于协议没有父类，也不定义实例变量。下面的示例显示了一个简单的协议声明，它只有一个方法：

```
@protocol MyProtocol
- (void)myProtocolMethod;
@end
```

在很多使用委托协议的情况下，采用一个协议只需将协议定义的方法实现即可。有一些协议需要你显式说明你支持这个协议，协议可以指定必需的和可选的方法。在你进行更深入开发之前，你有必要花一点时间看一看 The Objective-C Programming Language 中“Protocols”的章节，学习如何使用协议。

## 更多信息

本文中的信息供您大致了解 Objective-C 语言的基础。涉及到的主题反映了您阅读后续文档中可能会遇到的语言特性。这些特性并不代表语言的所有特性，建议您继续阅读 The Objective-C Programming Language 获得更多信息。

---

# 译者声明

您看到的这篇中文文档（译文）翻译自苹果公司文章：Learning Objective-C: A Primer（原文）。译者翻译原文的目的仅为方便中文读者理解原文。原文所有权利均属于苹果公司。译者不为译文的准确性负责。使用本译文造成的后果均与苹果公司和译者无关。

本译文最近更新于 2011-07-17。

如果有任何建议请发送邮件至：objc.primer@gmail.com。