苹果 Cocoa 编程规范

中文版

翻译：罗朝辉（kesalin@gmail.com）

2011/11/3

本文档翻译自苹果官方的 Cocoa 编码规范。

目录

苹果 Cocoa 编码规范 3
代码命名基础 3
一般性原则 3
前缀 5
书写约定 5
类与协议命名 5
头文件 6
方法命名 8
一般性规则 8
访问方法 9
委托方法 10
集合方法 11
方法参数 12
私有方法 13
函数命名 14
实例变量与数据类型命名 15
实例变量 15
常量 15
异常与通知 16
可接受的缩略语 18
常见的缩写 18
常见的略写 18
框架开发者小贴士与技巧 19
初始化 19

苹果 Cocoa 编码规范

代码命名基础

在面向对象软件库的设计过程中，开发人员经常忽视对类，方法，函数，常量以及其他编程接口元素的命名。本节讨论大多数 Cocoa 接口的一些命名约定。

一般性原则

清晰性

- 最好是既清晰又简短，但不要为简短而丧失清晰性

代码	点评
insertObject:atIndex:	good
insert:at:	不清晰；要插入什么？“at”表示什么？
removeObjectAtIndex:	good
removeObject:	这样也不错，因为方法是移除作为参数的对象
remove	不清晰；要移除什么？

- 名称通常不缩写，即使名称很长，也要拼写完全

代码	点评
destinationSelection	good
destSel	不清晰
setBackgroundColor:	good
setBkgdColor:	不清晰

你可能会认为某个缩写广为人知，但有可能并非如此，尤其是当你的代码被来自不同文化和语言背景的开发人员所使用时。

- 然而，你可以使用少数非常常见，历史悠久的缩写。请参考：”可接受的缩略名“一节

- 避免使用有歧义的 API 名称，如那些能被理解成多种意思的方法名称

代码	点评
sendPort	是发送端口还是返回一个发送端口？
displayName	是显示一个名称还是返回用户界面中控件的标题？

一致性

- 尽可能使用与 Cocoa 编程接口命名保持一致的名称。如果你不太确定某个命名的一致性，请浏览一下头文件或参考文档中的范例

- 在使用多态方法的类中，命名的一致性非常重要。在不同类中实现相同功能的方法应该具有相同的名称

代码	点评
- (int) tag	在 NSView, NSCell, NSControl 中有定义
- (void) setStringValue:(NSString *)	在许多 Cocoa classes 中有定义

请参考“方法参数”一节。

不要自我指涉

- 不要名称自我指涉

代码	点评
NSString	okey
NSStringObject	自我指涉

- 掩码（可使用位操作进行组合）和用作通知名称的常量不受该约定限制

代码	点评
NSUnderlineByWordMask	okey
NSTableViewColumnDidMoveNotification	okey

前缀

前缀是名称的重要组成部分。它们可以区分软件的功能范畴。通常，软件会被打包成一个框架或多个紧密相关的框架（如 Foundation 和 Application Kit 框架）。前缀可以防止第三方开发者与苹果公司之间的命名冲突（同样也可防止苹果内部不同框架之间的命名冲突）

- 前缀有规定的格式。它由两到三个大写字符组成，不能使用下划线与子前缀

前缀	Cocoa 框架
NS	Foundation
NS	Application Kit
AB	Address Book
IB	Interface Builder

- 命名 class，protocol，structure，函数，常量时使用前缀；命名成员方法时不使用前缀，因为方法已经在它所在类的命名空间种；同理，命名结构体字段时也不使用前缀

书写约定

在为 API 元素命名时，请遵循如下一些简单的书写约定

- 对于包含多个单词的名称，不要使用标点符号作为名称的一部分或作为分隔符（下划线，破折号等）；此外，大写每个单词的首字符并将这些单词连续拼写在一起。请注意以下限制：

    - 方法名小写第一个单词的首字符，大写后续所有单词的首字符。方法名不使用前缀。如：fileExistsAtPath:isDirectory: 如果方法名以一个广为人知的大写首字母缩略词开头，该规则不适用，如：NSImage 中的 TIFFRepresentation
    - 函数名和常量名使用与其关联类相同的前缀，并且要大写前缀后面所有单词的首字符。如：NSRunAlertPanel，NSCellDisabled
    - 避免使用下划线来表示名称的私有属性。苹果公司保留该方式的使用。如果第三方这样使用可能会导致命名冲突，他们可能会在无意中用自己的方法覆盖掉已有的私有方法，这会导致严重的后果。请参考“私有方法”一节以了解私有 API 的命名约定的建议

类与协议命名

类名应包含一个明确描述该类（或类的对象）是什么或做什么的名词。类名要有合适的前缀（请参考“前缀”一节）。Foundation 及 Application Kit 有很多这样例子，如：NSString, NSData, NSScanner, NSApplication, NSButton 以及 NSEvent。

协议应该根据对方法的行为分组方式来命名。

- 大多数协议仅组合一组相关的方法，而不关联任何类，这种协议的命名应该使用动名词(ing)，以不与类名混淆。

代码	点评
NSLocking	good
NSLock	糟糕，它看起来像类名

- 有些协议组合一些彼此无关的方法（这样做是避免创建多个独立的小协议）。这样的协议倾向于与某个类关联在一起，该类是协议的主要体现者。在这种情形，我们约定协议的名称与该类同名。NSObject 协议就是这样一个例子。这个协议组合一组彼此无关的方法，有用于查询对象在其类层次中位置的方法，有使之能调用特殊方法的方法以及用于增减引用计数的方法。由于 NSObject 是这些方法的主要体现者，所以我们用类的名称命名这个协议。

头文件

头文件的命名方式很重要，我们可以根据其命名知晓头文件的内容。

- 声明孤立的类或协议：将孤立的类或协议声明放置在单独的头文件中，该头文件名称与类或协议同名

头文件	声明
NSApplication.h	NSApplication 类

- 声明相关联的类或协议：将相关联的声明（类，类别及协议) 放置在一个头文件中，该头文件名称与主要的类/类别/协议的名字相同。

头文件	声明
NSString.h	NSString 和 NSMutableString 类
NSLock.h	NSLocking 协议和 NSLock, NSConditionLock, NSRecursiveLock 类

- 包含框架头文件：每个框架应该包含一个与框架同名的头文件，该头文件包含该框架所有公开的头文件。

头文件	框架
Foundation.h	Foundation.framework

- 为已有框架中的某个类扩展 API：如果要在一个框架中声明属于另一个框架某个类的范畴类的方法，该头文件的命名形式为：原类名+“Additions”。如 Application Kit 中的 NSBundleAdditions.h

- 相关联的函数与数据类型：将相联的函数，常量，结构体以及其他数据类型放置到一个头文件中，并以合适的名字命名。如 Application Kit 中的 NSGraphics.h

方法命名

一般性规则

为方法命名时，请考虑如下一些一般性规则：

- 小写第一个单词的首字符，大写随后单词的首字符，不使用前缀。请参考“书写约定”一节。有两种例外情况：1，方法名以广为人知的大写字母缩略词（如 TIFF or PDF）开头；2，私有方法可以使用统一的前缀来分组和辨识，请参考“私有方法”一节

- 表示对象行为的方法，名称以动词开头：

- (void) invokeWithTarget:(id)target:

- (void) selectTabViewItem:(NSTableViewItem *)tableViewItem

名称中不要出现 do 或 does，因为这些助动词没什么实际意义。也不要在动词前使用副词或形容词修饰

- 如果方法返回方法接收者的某个属性，直接用属性名称命名。不要使用 get，除非是间接返回一个或多个值。请参考“访问方法”一节。

- (NSSize) cellSize;	对
- (NSSize) calcCellSize;	错
- (NSSize) getCellSize;	错

- 参数要用描述该参数的关键字命名

- (void) sendAction:(SEL)aSelector to:(id)anObject forAllCells:(BOOL)flag;	对
- (void) sendAction:(SEL)aSelector :(id)anObject :(BOOL)flag;	错

- 参数前面的单词要能描述该参数。

- (id) viewWithTag:(int)aTag;	对
- (id) taggedView:(int)aTag;	错

- 细化基类中的已有方法：创建一个新方法，其名称是在被细化方法名称后面追加参数关键词

- (id) initWithFrame:(NSRect)frameRect;	NSView
- (id) initWithFrame:(NSRect)frameRect mode:(ind)aMode cellClass:(Class)factoryId numberOfRows:(int)rowsHigh numberOfColumns:(int)colsWide;	NSMatrix - NSView 的子类

- 不要使用 and 来连接用属性作参数关键字

- (int) runModalForDirectory:(NSString *)path file:(NSString *)name types:(NSArray *)fileTypes;	对
- (int) runModalForDirectory:(NSString *)path addFile:(NSString *)name addTypes:(NSArray *)fileTypes;	错

虽然上面的例子中使用 add 看起来也不错，但当你方法有太多参数关键字时就有问题。

- 如果方法描述两种独立的行为，使用 and 来串接它们

- (BOOL) openFile:(NSString *)fullPath withApplication:(NSString *)appName andDeactivate:(BOOL)flag;	NSWorkspace

访问方法

访问方法是对象属性的读取与设置方法。其命名有特定的格式依赖于属性的描述内容。

- 如果属性是用名词描述的，则命名格式为：

- (void) setNoun:(type)aNoun;

- (type) noun;

例如：

- (void) setgColor:(NSColor *)aColor;

- (NSColor *) color;

- 如果属性是用形容词描述的，则命名格式为：

- (void) setAdjective:(BOOL)flag;

- (BOOL) isAdjective;

例如：

- (void) setEditable:(BOOL)flag;

- (BOOL) isEditable;

- 如果属性是用动词描述的，则命名格式为：（动词要用现在时时态）

- (void) setVerbObject:(BOOL)flag;

- (BOOL) verbObject;

例如：

- (void) setShowAlpha:(BOOL)flag;

- (BOOL) showsAlpha;

- 不要使用动词的过去分词形式作形容词使用

- (void) setAcceptsGlyphInfo:(BOOL)flag;	对
- (BOOL) acceptsGlyphInfo;	对
- (void) setGlyphInfoAccepted:(BOOL)flag;	错
- (BOOL) glyphInfoAccepted;	错

- 可以使用情态动词（can, should, will 等）来提高清晰性，但不要使用 do 或 does

- (void) setCanHide:(BOOL)flag;	对
- (BOOL) canHide;	对
- (void) setShouldCloseDocument:(BOOL)flag;	对
- (void) shouldCloseDocument;	对
- (void) setDoseAcceptGlyphInfo:(BOOL)flag;	错
- (BOOL) doseAcceptGlyphInfo;	错

- 只有在方法需要间接返回多个值的情况下，才使用 get

- (void) getLineDash:(float *)pattern count:(int *)count phase:(float *)phase;	NSBezierPath

像上面这样的方法，在其实现里应允许接受 NULL 作为其 in/out 参数，以表示调用者对一个或多个返回值不感兴趣。

委托方法

委托方法是那些在特定事件发生时可被对象调用，并声明在对象的委托类中的方法。它们有独特的命名约定，这些命名约定同样也适用于对象的数据源方法。

- 名称以标示发送消息的对象的类名开头，省略类名的前缀并小写类第一个字符

- (BOOL) tableView:(NSTableView *)tableView shouldSelectRow:(int)row;
- (BOOL)application:(NSApplication *)sender openFile:(NSString *)filename;

- 冒号紧跟在类名之后（随后的那个参数表示委派的对象）。该规则不适用于只有一个 sender 参数的方法

- (BOOL) applicationOpenUntitledFile:(NSApplication *)sender;

- 上面的那条规则也不适用于响应通知的方法。在这种情况下，方法的唯一参数表示通知对象

```
- (void) windowDidChangeScreen:(NSNotification *)notification;
```

- 用于通知委托对象操作即将发生或已经发生的方法名中要使用 did 或 will

```
- (void) browserDidScroll:(NSBrowser *)sender;
- (NSUndoManager *) windowWillReturnUndoManager:(NSWindow *)window;
```

- 用于询问委托对象可否执行某操作的方法名中可使用 did 或 will，但最好使用 should

```
- (BOOL) windowShouldClose:(id)sender;
```

集合方法

管理对象（集合中的对象被称之为元素）的集合类，约定要具备如下形式的方法：

```
- (void) addElement:(elementType)adObj;
- (void) removeElement:(elementType)anObj;
- (NSArray *)elements;
```

例如：

```
- (void) addLayoutManager:(NSLayoutManager *)adObj;
- (void) removeLayoutManager:(NSLayoutManager *)anObj;
- (NSArray *)layoutManagers;
```

集合方法命名有如下一些限制和约定：

- 如果集合中的元素无序，返回 NSSet，而不是 NSArray
- 如果将元素插入指定位置的功能很重要，则需具备如下方法：

```
- (void) insertElement:(elementType)anObj atIndex:(int)index;
- (void) removeElementAtIndex:(int)index;
```

集合方法的实现要考虑如下细节：

- 以上集合类方法通常负责管理元素的所有者关系，在 add 或 insert 的实现代码里会 retain 元素，在 remove 的实现代码中会 release 元素

- 当被插入的对象需要持有指向集合对象的指针时，通常使用 set... 来命名其设置该指针的方法，且不要 retain 集合对象。比如上面的 insertLayerManager:atIndex: 这种情形，NSLayoutManager 类使用如下方法：

```
- (void) setTextStorage:(NSTextStorage *)textStorage;
- (NSTextStorage *)textStorage;
```

通常你不会直接调用 setTextStorage:，而是覆写它。

另一个关于集合约定的例子来自 NSWindow 类：

```
- (void) addChildWindow:(NSWindow *)childWin ordered:(NSWindowOrderingMode)place;
- (void) removeChildWindow:(NSWindow *)childWin;
- (NSArray *)childWindows;
- (NSWindow *) parentWindow;
- (void) setParentWindow:(NSWindow *)window;
```

方法参数

命名方法参数时要考虑如下规则：

- 如同方法名，参数名小写第一个单词的首字符，大写后继单词的首字符。如：removeObject:(id)anObject
- 不要在参数名中使用 pointer 或 ptr，让参数的类型来说明它是指针
- 避免使用 one， two，...，作为参数名
- 避免为节省几个字符而缩写

按照 Cocoa 惯例，以下关键字与参数联合使用：

```
...action:(SEL)aSelector
..alignment:(int)mode
...atIndex:(int)index
...content:(NSRect)aRect
...doubleValue:(double)aDouble
...floatValue:(float)aFloat
...font:(NSFont *)fontObj
...frame:(NSRect)frameRect
...intValue:(int)anInt
```

...keyEquivalent:(NSString *0charCode
...length:(int)numBytes
...point:(NSPoint)aPoint
...stringValue:(NSString *)aString
...tag:(int)anInt
...target:(id)anObject
...title:(NSString *)aString

私有方法

大多数情况下，私有方法命名相同与公共方法命名约定相同，但通常我们约定给私有方法添加前缀，以便与公共方法区分开来。即使这样，私有方法的名称很容易导致特别的问题。当你设计一个继承自 Cocoa framework 某个类的子类时，你无法知道你的私有方法是否不小心覆盖了框架中基类的同名方法。

Cocoa framework 的私有方法名称通常以下划线作为前缀（如：_fooData），以标示其私有属性。基于这样的事实，遵循以下两条建议：

- 不要使用下划线作为你自己的私有方法名称的前缀，Apple 保留这种用法。
- 若要继承 Cocoa framework 中一个超大的类（如：NSView），并且想要使你的私有方法名称与基类中的区别开来，你可以为你的私有方法名称添加你自己的前缀。这个前缀应该具有唯一性，如基于你公司的名称，或工程的名称，并以“XX_”形式给出。比如你的工程名为"Byte Flogger"，那么就可以是“BF_addObject:”

尽管为私有方法名称添加前缀的建议与前面类中方法命名的约定冲突，这里的意图有所不同：为了防止不小心地覆盖基类中的私有方法。

函数命名

Objective-C 允许通过函数（C 形式的函数）描述行为，就如成员方法一样。 你应当优先使用函数，而不是类方法，如果隐含的类为单例或在处理函数子系统时。

函数命名应该遵循如下几条规则：

- 函数命名与方法命名相似，但有两点不同：

  1，它们有前缀，其前缀与你使用的类和常量的前缀相同

  2，大写前缀后紧跟的第一个单词首字符

- 大多数函数名称以动词开头，这个动词描述该函数的行为

```
NSHighlightRect
```

```
NSDeallocateObject
```

查询属性的函数有个更多的规则要遵循：

- 查询第一个参数的属性的函数，省略动词

```
unsigned int NSEventMaskFromType(NSEventType type)
```

```
float NSHeight(NSRect rect)
```

- 返回值为引用的方法，使用 Get

```
const char *NSGetSizeAndAlignment(const char *typePtr, unsigned int
*sizep, unsigned int *alignp)
```

- 返回 boolean 值的函数，名称使用判断动词 is/does 开头

```
BOOL NSDecimalIsNotANumber(const NSDecimal *decimal)
```

实例变量与数据类型命名

这一节描述实例变量，常量，异常以及通知的命名约定。

实例变量

在为类添加实例变量是要注意：

- 避免创建 public 实例变量
- 使用 @private，@protected 显式限定实例变量的访问权限
- 确保实例变量名简明扼要地描述了它所代表的属性

如果实例变量别设计为可被访问的，确保编写了访问方法

常量

常量命名规则根据常量创建的方式不同而大不同。

枚举常量

- 使用枚举来定义一组相关的整数常量
- 枚举常量与其 typedef 命名遵守函数命名规则。如：来自 NSMatrix.h 中的例子：（本例中的 typedef tag（_NSMatrixMode）不是必须的）

```
typedef enum _NSMatrixMode {
    NSRadioModeMatrix       = 0,
    NSHighlightModeMatrix = 1;
    NSListModeMatrix          = 2,
    NSTrackModeMatrix       = 3
} NSMatrixMode;
```

- 位掩码常量可以使用不具名枚举。如：

```
enum {
    NSBorderlessWindowMask          = 0,
    NSTitledWindowMask                = 1 << 0,
    NSClosableWindowMask            = 1 << 1,
    NSMiniaturizableWindowMask   = 1 << 2,
    NSResizableWindowMask          = 1 << 3
};
```

常量

- 使用 const 来修饰浮点常量或彼此没有关联的整数常量
- 枚举常量命名规则与函数命名规则相同。const 常量命名范例：

```
const float NSLightGray;
```

其他常量

- 通常不使用 #define 来创建常量。如上面所述，整数常量请使用枚举，浮点数常量请使用 const
- 使用大写字母来定义预处理编译宏。如：#ifdef DEBUG
- 编译器定义的宏名首尾都有双下划线。如：__MACH__
- 为 notification 名及 dictionary key 定义字符串常量，从而能够利用编译器的拼写检查，减少书写错误。Cocoa 框架提供了很多这样的范例：

```
APPKIT_EXTERN NSString *NSPrintCopies;
```

实际的字符串值在实现文件中赋予。（注意：APPKIT_EXTERN 宏等价于 Objective-C 中 extern）

异常与通知

异常与通知的命名遵循相似的规则，但是它们有各自推荐的使用模式。

异常

虽然你可以处于任何目的而使用异常（由 NSException 类及相关类实现），Cocoa 通常不使用异常来处理常规的，可预料的错误。在这些情形下，使用诸如 nil, NULL, NO 或错误代码之类的返回值。异常的典型应用类似数组越界之类的编程错误。

异常由具有如下形式的全局 NSString 对象标识：

[Prefix] + UniquePartOfName + Exception

UniquePartOfName 部分是有连续的首字符大写的单词组成。例如：

```
NSColorListIOException
NSColorListNotEditableException
NSDraggingException
NSFontUnavailableException
NSIllegalSelectorException
```

通知

如果一个类有委托，那它的大部分通知可能由其委托的委托方法来处理。这些通知的名称应该能够反应其响应的委托方法。比如，当应用程序提交 NSApplicationDidBecomeActiveNotification 通知时，全局 NSApplication 对象的委托会注册从而能够接收 applicaitonDidBecomeActive: 消息。

通知由具有如下形式的全局 NSString 对象标识：

[相关联类的名称] + [Did 或 Will] + [UniquePartOfName] + Notification

例如：

NSApplicationDidBecomeActiveNotification
NSWindowDidMiniaturizeNotification
NSTextViewDidChangeSelectionNotification
NSColorPanelColorDidChangeNotification

可接受的缩略语

在设计编程接口时，通常名称不要缩写。然而，下面列出的缩写要么是固定下来的要么是过去被广泛使用的，所以你可以继续使用。关于缩写有一些额外的注意事项：

- 标准 C 库中长期使用的缩写形式是可以接受的。如："alloc"，"getc"
- 你可以在参数名中更自由地使用缩写。如：imageRep，col（column），obj，otherWin

常见的缩写

缩写	含义	缩写	含义
alloc	Allocate	msg	Message
alt	Alternate	nib	Interface Builder archive
app	Application	pboard	Pasteboard
calc	Calculate	rect	Rectangle
dealloc	Deallocate	Rep	Representation
func	Function	temp	Temporary
horiz	Horizontal	vert	Vertical
info	Information	init	Initialize
max	Maximum		

常见的略写

ASCII，PDF，XML，HTML，URL，RTF，HTTP，TIFF
JPG，GIF，LZW，ROM，RGB，CMYK，MIDI，FTP

框架开发者小贴士与技巧

初始化

类初始化

在 initialize 类方法中，能够编写实现一些延迟执行且只被一次的代码，initialize 类方法是由运行时系统在该类响应任何其他消息之前调用的。典型的应用是在其中设置类的版本信息。运行时系统向每个类发送 initialize 消息，即使该类没有实现 initialize，也会调用其基类的某个 initialize 方法。因此一个类的 initialize 方法可能会因为存在继承类的缘故被执行多次。因此有必要使用一定的技巧来防止只执行一次的代码被多次执行。如：NSFoo 类的 initialize 方法实现可能如下：

```
+ (id) initialize
{
    if (self == [NSFoo class])
    {
        // 初始化代码
    }
    return self;
}
```

不应当显式调用 initialize 方法。如果你需要激活 initialize 方法，使用 [NSFoo self] 形式的调用。