Daniel's Objective-C Coding Style Guidelines

From: 杨德升 / http://desheng.me / desheng.young@gmail.com
Date: 2010.10.31

参考资料:
- Apple: Coding Guidelines for Cocoa
- Google: Objective-C Style Guide
- Three20: Source code style guildelines

正文:
- 格式化代码
  - 指针“*”号的位置
    - 如: `NSString *varName;`
  - 空格 VS tabs
    - 只允许使用空格，将编辑器设置为1个TAB = 2个字符缩进
  - 每行的长度
    - 每行最多不得超过100个字符
    - 以15寸Macbook Pro的大小，每行100个字符时能最大化地同时容下编辑器和iPhone模拟器
    - Google的80字符的标准有点少，这导致过于频繁的换行（Objectve-C的代码一般都很长）
    - 通过 “Xcode => Preferences => TextEditing => 勾选Show Page Guide / 输入 100 => OK” 来设置提醒
  - 方法的声明和定义
    - 在 - OR + 和返回值之间留1个空格，方法名和第一个参数间不留空格。如:
      ```
      - (void)doSomethingWithString:(NSString *)theString {
        ...
      }
      ```
    - 当参数过长时，每个参数占用一行，以冒号对齐。如:
      ```
      - (void)doSomethingWith:(GTMFoo *)theFoo
                         rect:(NSRect)theRect
                     interval:(float)theInterval {
        ...
      }
      ```
    - 如果方法名比参数名短，每个参数占用一行，至少缩进4个字符，且为垂直对齐（而非使用冒号对齐）。如:
      ```
      - (void)short:(GTMFoo *)theFoo
          longKeyword:(NSRect)theRect
          evenLongerKeyword:(float)theInterval {
        ...
      }
      ```
  - 方法的调用
    - 调用方法沿用声明方法的习惯。例外：如果给定源文件已经遵从某种习惯，继续遵从那种习惯。
    - 所有参数应在同一行中，或者每个参数占用一行且使用冒号对齐。如:
      ```
      [myObject doFooWith:arg1 name:arg2 error:arg3];
      ```
      或
      ```
      [myObject doFooWith:arg1
                     name:arg2
                    error:arg3];
      ```
    - 和方法的声明一样，如果无法使用冒号对齐时，每个参数一行、缩进4个字符、垂直对其（而非使用冒号对齐）。如:
      ```
      [myObj short:arg1
          longKeyword:arg2
          evenLongerKeyword:arg3];
      ```
  - @public 和 @private
    - @public 和 @private使用单独一行，且缩进1个字符

- Protocals
  - 类型标示符、代理名称、尖括号间不留空格。
  - 该规则同样适用于：类声明、实例变量和方法声明。如：
    ```
    @interface MyProtocoledClass : NSObject<NSWindowDelegate> {
     @private
      id<MyFancyDelegate> _delegate;
    }
    - (void)setDelegate:(id<MyFancyDelegate>)aDelegate;
    @end
    ```
  - 如果类声明中包含多个protocal，每个protocal占用一行，缩进2个字符。如：
    ```
    @interface CustomViewController : ViewController<
      AbcDelegate,
      DefDelegate
    > {
      ...
    }
    ```
- 命名
  - 类名
    - 类名（及其category name 和 protocal name）的首字母大写，写使用首字母大写的形式分割单词
    - 在面向特定应用的代码中，类名应尽量避免使用前缀，每个类都使用相同的前缀影响可读性。
    - 在面向多应用的代码中，推荐使用前缀。如：GTMSendMessage
  - Category Name
    - 待完善
  - 方法名
    - 方法名的首字母小写，且使用首字母大写的形式分割单词。方法的参数使用相同的规则。
    - 方法名+参数应尽量读起来像一句话（如：）。在这里查看苹果对方法命名的规范。
    - getter的方法名和变量名应相同。不允许使用“get”前缀。如：
      ```
      - (id) getDelegate; // 禁止
      - (id)delegate;     // 对头
      ```
    - 本规则仅针对Objective-C代码，C++代码使用C++的习惯
  - 变量名
    - 变量名应使用容易意会的应用全称，且首字母小写，且使用首字母大写的形式分割单词
    - 成员变量使用“_”作为前缀（如：“NSString *_varName;”。虽然这与苹果的标准（使用“_”作为后缀）相冲突，但基于以下原因，仍使用“_”作为前缀。
      - 使用“_”作为前缀，更容易在有代码自动补全功能的IDE中区分“属性(self.userInfo)”和“成员变量(_userInfo)”
    - 常量（#define, enums, const等）使用小写“k”作为前缀，首字母大写来分割单词。如：
      ```
      kInvalidHandle
      ```
- 注释
  - 待完善
- Cocoa 和 Objective-C特有的规则
  - 成员变量使用 @private。如：
    ```
    @interface MyClass : NSObject {
     @private
      id _myInstanceVariable;
    }
    // public accessors, setter takes ownership
    - (id)myInstanceVariable;
    - (void)setMyInstanceVariable:(id)theVar;
    @end
    ```
  - Indentify Designated Initializer
    - 待完善
  - Override Desingated Initializer

    - 待完善
- 初始化
    - 在初始化方法中，不要将变量初始化为“0”或“nil”，那是多余的
    - 内存中所有的新创建的对象（isa除外）都是0，所以不需要重复初始化为“0”或“nil”
- 避免显式的调用 +new 方法
    - 禁止直接调用 NSObject 的类方法 +new，也不要在子类中重载它。使用alloc和init方法
- 保持公共API的简洁性
    - 待完善
- #import VS #include
    - 使用 #import 引入Ojbective-C和Ojbective-C++头文件，使用 #include 引入C和C++头文件
- import根框架（root frameworks），而非各单个文件
    - 虽然有时我们仅需要框架（如Cocoa 或 Foundation）的某几个头文件，但引入根文件编译器会运行的更快。因为根框架（root frameworks）一般会预编译，所以加载会更快。再次强调：使用 #import 而非 #include 来引入Objective-C框架。如：

```
#import <Foundation/NSArray.h>    // 禁止
#import <Foundation/NSString.h>
...
#import <Foundation/Foundation.h> // 对头
```

- 创建对象时尽量使用autorelease
    - 创建临时对象时，尽量同时在同一行中 autorelease 掉，而非使用单独的 release 语句
    - 虽然这样会稍微有点慢，但这样可以阻止因为提前 return 或其他意外情况导致的内存泄露。通盘来看这是值得的。如：

```
// 避免这样使用（除非有性能的考虑）
MyController* controller = [[MyController alloc] init];
// ... 这里的代码可能会提前return ...
[controller release];
// 这样更好
MyController* controller = [[[MyController alloc] init] autorelease];
```

- 先autorelease，再retain
    - 在为对象赋值时，遵从“先autorelease，再retain”
    - 在将一个新创建的对象赋给变量时，要先将旧对象release掉，否则会内存泄露。市面上有很多方法来handle这种情况，这里选择“先autorelease，再retain”的方法，这种方法不易引入error。注意：在循环中这种方法会“填满”autorelease pool，稍稍影响效率，但是Google和我(:P)认为这个代价是可以接受的。如：

```
- (void)setFoo:(GMFoo *)aFoo {
  [foo_ autorelease];  // 如果foo_和aFoo是同一个对象（foo_ == aFoo），
dealloc不会被调用
  foo_ = [aFoo retain];
}
```

- dealloc的顺序要与变量声明的顺序相同
    - 这有利于review代码
    - 如果dealloc中调用其他方法来release变量，将被release的变量以注释的形式标注清楚
- NSString的属性的setter使用“copy”
    - 禁止使用retain，以防止意外的修改了NSString变量的值。如：

```
- (void)setFoo:(NSString *)aFoo {
  [foo_ autorelease];
  foo_ = [aFoo copy];
}
或
@property (nonatomic, copy) NSString *aString;
```

- 避免抛出异常（Throwing Exceptions）
    - 待完善
- 对 nil 的检查
    - 仅在有业务逻辑需求时检查 nil，而非为了防止崩溃

    - 向 nil 发送消息不会导致系统崩溃，Objective-C运行时负责处理。
- BOOL陷阱
    - 将int值转换为BOOL时应特别小心。避免直接和YES比较
    - Objective-C中，BOOL被定义为unsigned char，这意味着除了 YES (1) 和 NO (0)外它还可以是其他值。禁止将int直接转换（cast or convert）为BOOL。
    - 常见的错误包括：将数组的大小、指针值或位运算符的结果转换（cast or convert）为BOOL，因为该BOOL值的结果取决于整型值的最后一位
    - 将整型值转换为BOOL的方法：使用三元运算符返回YES / NO，或使用位运算符（&&, ||, !）
    - BOOL、_Bool和bool之间的转换是安全的，但是BOOL和Boolean间的转换不是安全的，所以将Boolean看成整型值。
    - 在Objective-C中，只允许使用BOOL
    - 如：

```
// 禁止
- (BOOL)isBold {
  return [self fontTraits] & NSFontBoldTrait;
}
- (BOOL)isValid {
  return [self stringValue];
}
// 对头
- (BOOL)isBold {
  return ([self fontTraits] & NSFontBoldTrait) ? YES : NO;
}
- (BOOL)isValid {
  return [self stringValue] != nil;
}
- (BOOL)isEnabled {
  return [self isValid] && [self isBold];
}
```

    - 禁止直接将BOOL和YES/NO比较，如：

```
// 禁止
BOOL great = [foo isGreat];
if (great == YES)
  ...
// 对头
BOOL great = [foo isGreat];
if (great)
  ...
```

- 属性
    - 命名：与去掉“_”前缀的成员变量相同，使用@synthesize将二者联系起来。如：

```
// abcd.h
@interface MyClass : NSObject {
 @private
  NSString *_name;
}

@property (copy, nonatomic) NSString *name;

@end

// abcd.m
@implementation MyClass
@synthesize name = _name;
@end
```

        - 位置：属性的声明紧随成员变量块之后，中间空一行，无缩进。如上例所示
        - 严把权限：对不需要外部修改的属性使用readonly
        - NSString使用copy而非retain
        - CFType使用@dynamic，禁止使用@synthesize
        - 除非必须，使用nonatomic
- Cocoa Pattern
    - Delegate Pattern（委托）
        - delegate对象使用assign，禁止使用retain。因为retain会导致循环索引导致内存泄露，并且此类型的内存泄露无法被Instrument发现，极难调试
        - 成员变量命名为_delegate，属性名为delegate
    - Model/View/Controller
        - Model和View分离
            - 不多解释
        - Controller独立于View和Controller
            - 不要在与view相关的类中添加过多的业务逻辑代码，这让代码的可重用性很差
            - Controller负责业务逻辑代码，且Controller的代码与view尽量无关
        - 使用 @protocal 定义回调APIs，如果并非所有方法都是必须的，使用 @optional 标示
- 其他
    - init方法和dealloc方法是是最常用的方法，所以将他们放在类实现的开始位置
    - 使用空格将相同的变量、属性对齐，使用换行分组