### 关于 iOS 64位编程的建议

Apple下发了支持64位的最后通牒：

As we announced in October, beginning February 1, 2015 new iOS apps submitted to the App Store must include 64-bit support and be built with the iOS 8 SDK. Beginning June 1, 2015 app updates will also need to follow the same requirements.

官网文档地址：https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaTouch64BitGuide/ConvertingYourAppto64-Bit/ConvertingYourAppto64-Bit.html#//apple\_ref/doc/uid/TP40013501-CH3-SW1

早应该做的适配终于要开始动工了，苦了64位的CPU运行了这么久32位的程序。前段时间公司项目完成了64-bit包的适配，本没那么复杂的事被无数不标准的老代码搅和的不轻，总结几个Tip共勉。

**Tips**

**拒绝基本数据类型和隐式转换**

首当其冲的就是基本类型，比如下面4个类型在32-bit和64-bit下分别是多长呢？

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2  3  4 | size\_t s1 = sizeof(int);  size\_t s2 = sizeof(long);  size\_t s3 = sizeof(float);  size\_t s4 = sizeof(double); |

32-bit下：4, 4, 4, 8；64-bit下：4, 8, 4, 8  
（PS： 这个结果随编译器，换其他平台可不一定）  
它们的长度变化可能并非我们对64-bit长度加倍的预期，所以说，程序中出现sizeof的代码多看两眼。而且，除非你明确知道自己在做什么，应该使用下面的类型代替基本类型：

* int -> NSInteger
* unsigned -> NSUInteger
* float -> CGFloat
* 动画时间 -> NSTimeInterval
* …

这些都是SDK中定义的类型，而我们大部分时间都在跟SDK的API们打交道，使用它们能将类型转换的影响降低很多。

再比如说下面的代码：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2  3  4 | NSArray \*items = @[<a href="<http://www.jobbole.com/members/1/>">@1</a>, @2, @3];  for (int i = -1; i < items.count; i++) {      NSLog(@"%d", i);  } |

结果是，for循环一次都没有进。  
数组的count是NSUInteger类型的，-1与其比较时隐式转换成NSUInteger，变成了一个很大的数字：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2  3  4 | (lldb) p i  (int) $0 = -1  (lldb) p (NSUInteger)i  (NSUInteger) $1 = 18446744073709551615 |

这和64-bit到没啥关系，想要说明的是，这种隐式转换也需要小心，一定要注意和这个变量相关的所有操作（赋值、比较、转换）  
老式for循环可以考虑写成：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1 | for (NSUInteger index = 0; index < items.count; index++) {} |

当然，数组遍历还是更推荐用for-in或block版本的，它们之间的比较可以回顾下[这篇文章](http://blog.sunnyxx.com/2014/04/30/ios_iterator/" \t "_blank)。

**使用新版枚举**

和上面的原因差不多，枚举应该使用新版的写法：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | typedef NS\_ENUM(NSInteger, UIViewAnimationCurve) {      UIViewAnimationCurveEaseInOut,      UIViewAnimationCurveEaseIn,      UIViewAnimationCurveEaseOut,      UIViewAnimationCurveLinear  }; |

不仅能为枚举值指定类型，而且当赋值赋错类型时，编译器还会给出警告，没理由不用这种写法。

**替代Format字符串**

适配64-bit时，你是否遇到了下面的恶心写法：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2 | NSArray \*items = @[<a href="<http://www.jobbole.com/members/1/>">@1</a>, @2, @3];  NSLog(@"数组元素个数：%lu", (unsigned long)items.count); |

一般情况下，利用NSNumber的@语法糖就可以解决：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2 | NSArray \*items = @[<a href="<http://www.jobbole.com/members/1/>">@1</a>, @2, @3];  NSLog(@"数组元素个数：%@", @(items.count)); |

同理，int转string也可以：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2 | NSInteger i = 10086;  NSString \*string = @(i).stringValue; |

当然，如需要%.2f这种Format就不适用了。

**64-bit下的BOOL**

32-bit下，BOOL被定义为signed char，@encode(BOOL)的结果是'c'  
64-bit下，BOOL被定义为bool，@encode(BOOL)结果是'B'  
更直观的解释是：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2  3  4 | (lldb) p/t (signed char)7  (BOOL) $0 = 0b00000111 (YES)  (lldb) p/t (bool)7  (bool) $1 = 0b00000001 (YES) |

32-bit版本的BOOL包括了256个值的可能性，还会引起一些坑，像[这篇文章](http://www.bignerdranch.com/blog/bools-sharp-corners/" \t "_blank)所说的。而64-bit下只有0（NO），1（YES）两种可能，终于给BOOL正了名。

**不直接取isa指针**

编译器已经默认禁用了这种使用，isa指针在32位下是Class的地址，但在64位下利用bits mask才能取出来真正的地址，若真需要，使用runtime的object\_getClass和object\_setClass方法。关于64位下isa的讲解可以看[这篇文章](http://www.sealiesoftware.com/blog/archive/2013/09/24/objc_explain_Non-pointer_isa.html" \t "_blank)

**解决第三方lib依赖和lipo命令**

以源码形式出现在工程中的第三方lib，只要把target加上arm64编译就好了。  
恶心的就是直接拖进工程的那些静态库(.a)或者framework，就需要重新找支持64-bit的包了。这时候就能看出哪些是已无人维护的lib了，是时候找个替代品了（比如我全网找不到工程中用到的一个音频库的64位包，终于在一个哥们的github上找到，哭着给了个star- -）

**打印Mach-O文件支持的架构**

如何看一个可执行文件是不是支持64-bit呢？

使用lipo -info命令，比如看看UIKit支持的架构：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2  3 | // 当前在Xcode Frameworks目录  sunnyxx$ lipo -info UIKit.framework/UIKit  Architectures in the fat file: UIKit.framework/UIKit are: arm64 armv7s |

想看的更详细的信息可以使用lipo -detailed\_info：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | sunnyxx$ lipo -detailed\_info UIKit.framework/UIKit  Fat header in: UIKit.framework/UIKit  fat\_magic 0xcafebabe  nfat\_arch 2  architecture arm64      cputype CPU\_TYPE\_ARM64      cpusubtype CPU\_SUBTYPE\_ARM64\_ALL      offset 4096      size 16822272      align 2^12 (4096)  architecture armv7s      cputype CPU\_TYPE\_ARM      cpusubtype CPU\_SUBTYPE\_ARM\_V7S      offset 16826368      size 14499840      align 2^12 (4096) |

当然，还可以使用file命令：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1  2  3  4 | sunnyxx$ file UIKit.framework/UIKit  UIKit.framework/UIKit: Mach-O universal binary with 2 architectures  UIKit.framework/UIKit (for architecture arm64):Mach-O 64-bit dynamically linked shared library  UIKit.framework/UIKit (for architecture armv7s):Mach-O dynamically linked shared library arm |

上述命令对Mach-O文件适用，静态库.a文件，framework中的.a文件，自己app的可执行文件都可以打印下看看。

**合并多个架构的包**

如果，我们有MyLib-32.a和MyLib-64.a，可以使用lipo -create命令合并：

[?](http://www.open-open.com/lib/view/open1420697675843.html)

|  |  |
| --- | --- |
| 1 | sunnyxx$ lipo -create MyLib-32.a MyLib-64.a -output MyLib.a |

**支持64-bit后程序包会变大么？**

会，支持64-bit后，多了一个arm64架构，理论上每个架构一套指令，但相比原来会大多少还不好说，我们这里增加了大概50%，还有听说会增加一倍的。

**一个lib包含了很多的架构，会打到最后的包里么？**

不会，如果lib中有armv7, armv7s, arm64, i386架构，而target architecture选择了armv7s, arm64，那么只会从lib中link指定的这两个架构的二进制代码，其他架构下的代码不会link到最终可执行文件中；反过来，一个lib需要在模拟器环境中正常link，也得包含i386架构的指令。

**Checklist**

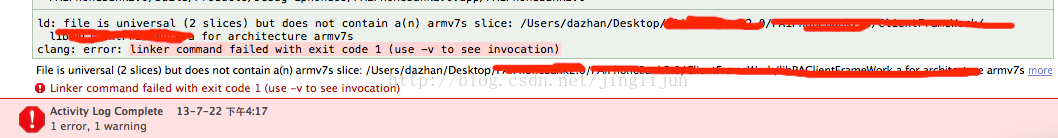
最后列一下官方文档中的注意点：

* 不要将指针强转成整数
* 程序各处使用统一的数据类型
* 对不同类型的整数做运算时一定要注意
* 需要定长变量时，使用如int32\_t, int64\_t这种定长类型
* 使用malloc时，不要写死size
* 使用能同时适配两个架构的格式化字符串
* 注意函数和函数指针（类型转换和可变参数）
* 不要直接访问Objective-C的指针（isa）
* 使用内建的同步原语（Primitives）
* 不要硬编码虚存页大小
* [Go Position Independent](https://developer.apple.com/library/ios/qa/qa1788/_index.html" \l "/apple_ref/doc/uid/DTS40013354" \t "_blank)

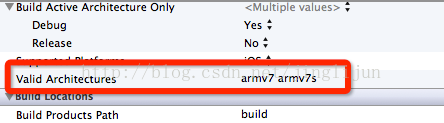
## 二：让你的iOS静态库支持armv7s

我们在用到第三方的静态库或者自己以前写的静态库时，在iPhone5，ipad4 上面经常会报错，下面

ld: file is universal (3 slices) but does not contain a(n) armv7s slice: /zhangyg/XXX/XXX/libs/libxxx.a for architecture armv7sclang: error: linker command failed with exit code 1 (use -v to see invocation)



这个解决方法有几种，有山寨的，有专业的。

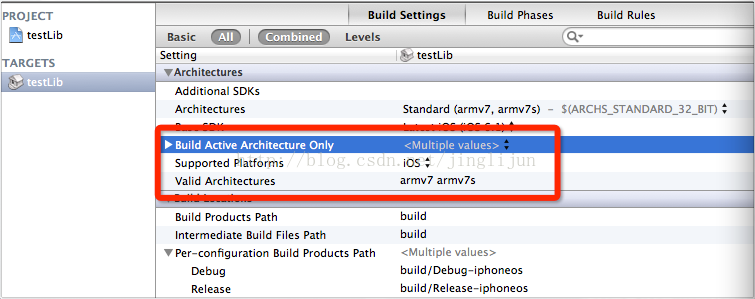


山寨的就是在你引用静态库的工程设置里面（上图）设置下不要添加armv7s，这个方法是治标不治本，所以建议不要使用。

我们还是彻底来解决他吧。

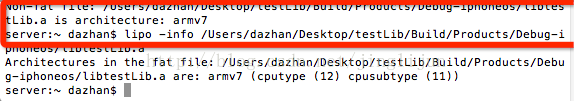
下面就是具体方法：

 我们新建的静态库工程，设置为



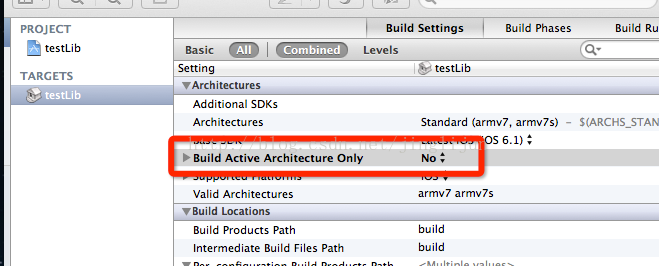
然后这样我们bulid一次，生成的xxx.a文件我们用命令看一下是：

用命令查看下：lipo -info xxxx.a



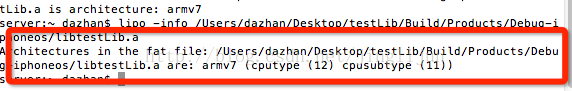
这样只有armv7，在新的apple  设备上肯定会报错，不支持armv7s。

那我们简单的，做一个设置，也是最重要的设置就是Build Active Architecture Only的值设置为NO



这样我们继续在bulid一次，然后在看一下

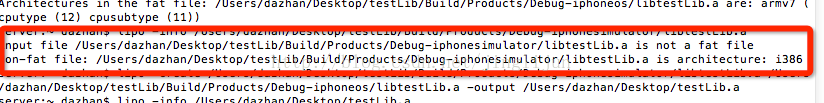
lipo -info xxxx.a



这样就多了一个 (cputype (12)  cpusubtype(11) ),看见这个你可能还很纳闷，为什么还没有armv7s，其实这里的 (cputype  
(12)  cpusubtype(11) )就是armv7s。

这样我们在选择模拟器bulid一次。

lipo -info  xxxx.a



这次目标结构变为了 ：i386，ok了。

然后我们用命令合一下这二个静态库 就ok了

 lipo -create 模拟器.a  真机.a -output 目标地址/最终.a



我们可以查看下:lipo -info  最终.a



这样就包含了模拟器和真机都可以运行的静态库了。

然后最终.a这个静态库发布出去或者给第三方用就没问题了。