

InlineHook新秀Dobby框架

[编译Dobby](#)

[导入Dobby.Framework](#)

[bitcode问题](#)

[拷贝问题](#)

[使用Dobby](#)

[HOOK前的准备](#)

[开始HOOK](#)

[参数解析](#)

由于最近研究InlineHook(内联钩子)，发现了一个不错的框架，[Dobby](#)（原名：HOOKZz）。这家伙是一个全平台的inlineHook框架，它用起来就和fishhook一样，我们先看一下如何使用。

内联钩子：所谓InlineHook就是直接修改目标函数的头部代码。让它跳转到我们自定义的函数里面执行我们的代码，从而达到Hook的目的。这种Hook技术一般用在静态语言的HOOK上面

编译Dobby

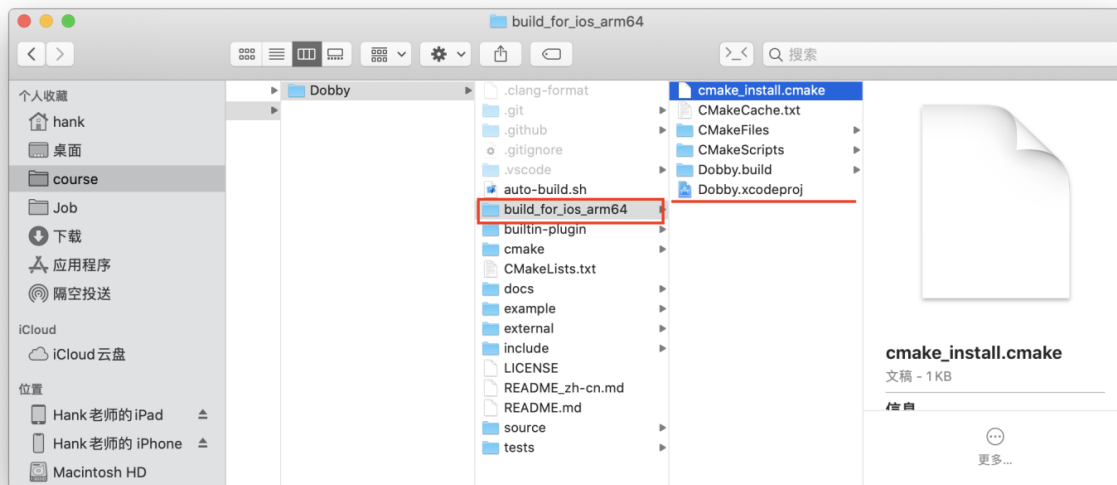
首先我们将代码clone下来

```
1 #depth用于指定克隆深度，为1即表示只克隆最近一次commit。
2 git clone https://github.com/jmpews/Dobby.git --depth=1
```

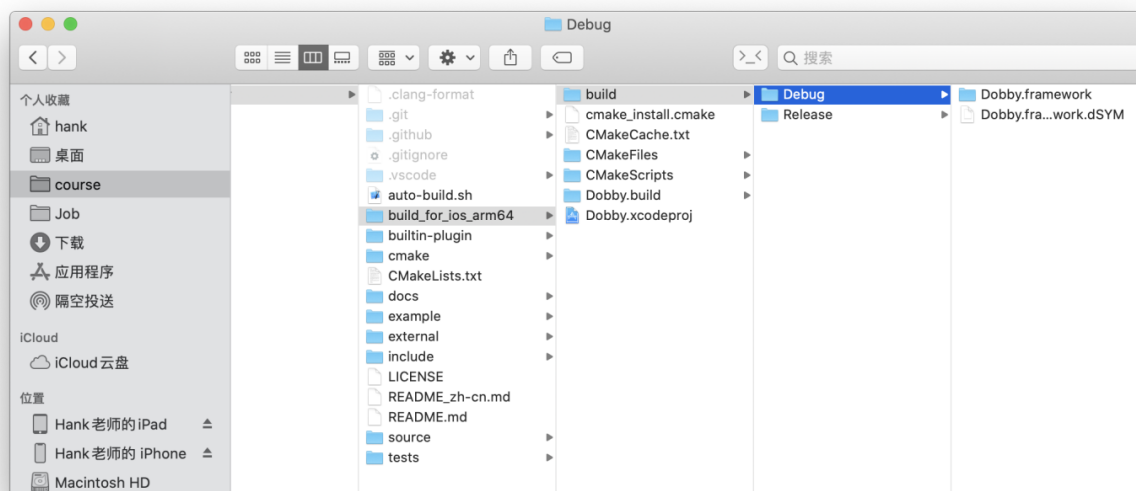
注意由于这家伙是跨平台的，所以项目并不是一个Xcode工程，我们要使用cmake将这个工程编译成为Xcode工程。进入Dobby目录，创建一个文件夹，然后cmake编译工程

```
1 cd Dobby && mkdir build_for_ios_arm64 && cd build_for_ios_arm64
2 cmake .. -G Xcode \
3 -DCMAKE_TOOLCHAIN_FILE=cmake/ios.toolchain.cmake \
4 -DPLATFORM=OS64 -DARCHS="arm64" -DCMAKE_SYSTEM_PROCESSOR=arm64 \
5 -DENABLE_BITCODE=0 -DENABLE_ARC=0 -DENABLE_VISIBILITY=1 -DDEPLOYMENT_TARGET=9.3 \
6 -DDynamicBinaryInstrument=ON -DNearBranch=ON -DPlugin.SymbolResolver=ON -DPlugin.Darwin.HideLibrary=ON -DPlugin.Darwin.ObjectiveC=ON
```

编译完成后，会生成一个Xcode工程



接下来编译Xcode工程。生成我们的Framework



导入Dobby.Framework

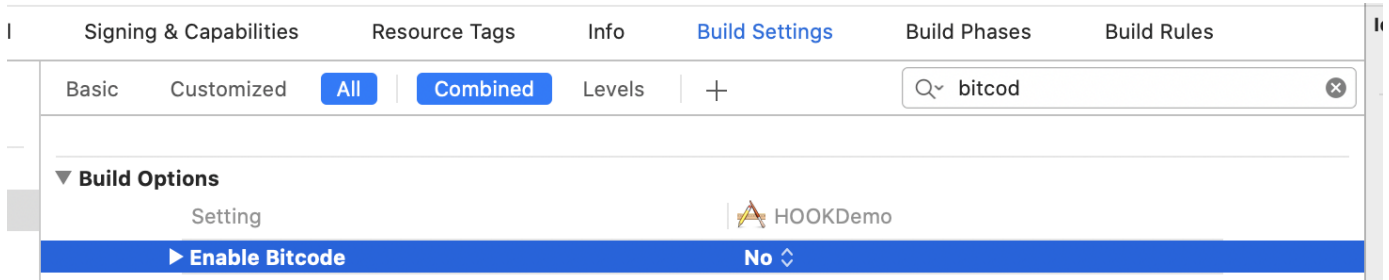
bitcode问题

我们新建一个工程开始使用Dobby。那么将Framework加入工程会有一个常见的问题，就是bitcode。

```
ld: '/Users/hank/Desktop/iOSHook/HOOKDemo/Dobby.framework/Dobby' does not contain bitcode. You must rebuild it with bitcode enabled (Xcode setting ENABLE_BITCODE), obtain an updated library from the vendor, or disable bitcode for this target. file '/Users/hank/Desktop/iOSHook/HOOKDemo/Dobby.framework/Dobby' for architecture arm64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

1. '/Users/hank/Desktop/iOSHook/HOOKDemo/Dobby.framework/Dobby' does not contain bitcode. You must rebuild it with bitcode enabled (Xcode setting ENABLE_BITCODE), obtain an updated library from the vendor, or disable bitcode for this target. file '/Users/hank/Desktop/iOSHook/HOOKDemo/Dobby.framework/Dobby' for architecture arm64
[less](#)

解决方案两种。1、解决Framework让他支持bitcode。2、工程关闭bitcode。

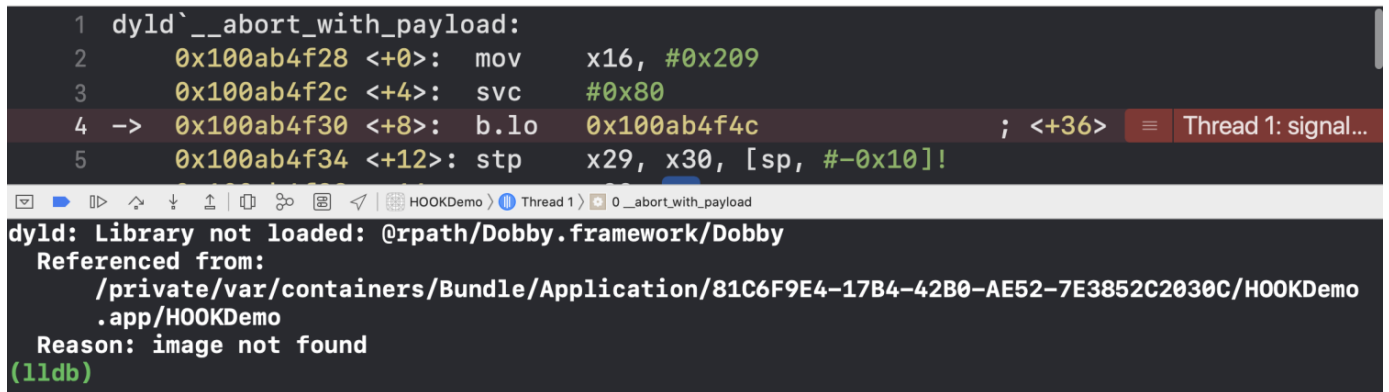


不了解的同学看下面的引用，其他的直接过。

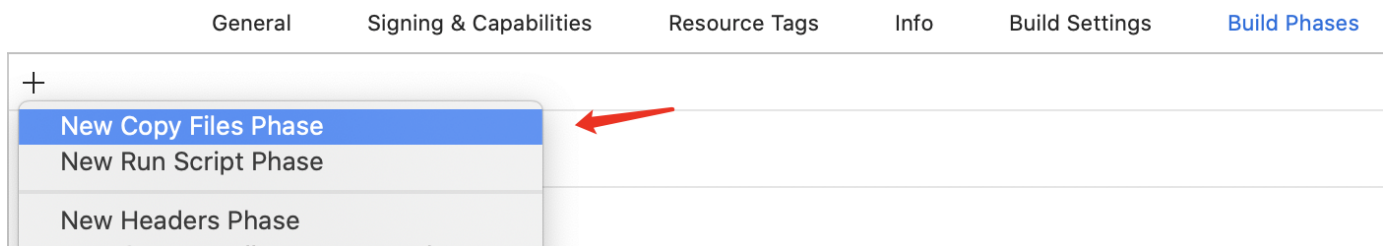
bitcode 是苹果独有的一层中间代码。包含 bitcode 配置的程序将会在 App Store 上被编译和链接。bitcode 允许苹果在后期重新优化我们程序的二进制文件，也就是苹果会将这个 bitcode 编译为可执行的64位或32位程序。

拷贝问题

Framework库首次拖入工程，Xcode不会自动帮你拷贝，运行时你会发现库没有打包进入App包。造成DYLD加载时找不到库的错误。




此刻需要手动添加拷贝



选择Framework

▼ Copy Files (1 item)

Destination	Frameworks
Subpath	
<input type="checkbox"/> Copy only when installing	
Name	
 Dobby.framework	

再次运行我们可以看到控制台的输出，说明引入成功了！

```
1 [*] [DobbySymbolResolver] resolve image: /usr/lib/system/libdyld.dylib
2 [*] Initialize DobbyInstrument => 0x100538c40 => 0x100490b60
3 [*] ===== DynamicBinaryInstrumentRouting Start =====
4 [*] Initialize assembler code buffer at 0x2822420a0
5 [*] Initialize assembler code buffer at 0x282242080
```

使用Dobby

引入成功之后，我们就可以开始玩一下了。迫不及待！首先我们来看看最关键的函数。

```
1 // replace function
2 /**
3     arg1:需要HOOK的函数地址
4     arg2:新函数地址
5     arg3:保留原始函数的指针的地址
6 */
7 int DobbyHook(void *function_address, void *replace_call, void **origin_call);
```

这个就是用来HOOK我们自定义函数的，那么我们来使用一下。你会发现这个函数的使用 and fishhook 非常像！

接下来，我们可以写一个Demo。比如我们有一个自定义的sum函数，明显的加法运算。

```
1 int sum(int a,int b){
2     return a + b;
```

```
3 }
```

接下来，我们在ViewController的ViewDidLoad中输出

```
1 - (void)viewDidLoad {
2     [super viewDidLoad];
3     NSLog(@"打印出: %d",sum(10, 20));
4 }
```

这个函数正常的执行结果应该是30。那么我们要在Load方法中去HOOK这个sum。

HOOK前的准备

首先我们要定义几个东西。

- 函数指针，用于保存被替换函数的地址

```
1 //函数指针用于保留原来的执行流程
2 static int(*sum_p)(int a,int b);
```

- 新函数（用这个函数替换你需要HOOK的函数，那么该函数的返回值以及参数要保持一致）

```
1 //新函数
2 int mySum(int a,int b){
3     NSLog(@"原有的结果是:%d",sum_p(a,b));
4     return a - b;
5 }
```

开始HOOK

使用DobbyHook来HOOK我们的函数。接下来，在Load方法中：

```
1 +(void)load
2 {
3     //Hook sum
4     DobbyHook(sum, mySum, (void *)&sum_p);
5 }
```

参数解析

- arg1(sum):需要HOOK的函数的地址，函数名称就是函数指针
- arg2(mySum):新函数的地址
- arg3(sum_p):将原来的sum函数的地址存放到sum_p这个函数指针中（因为要给指针赋值，所以取指针的地址，so：二级指针）

运行结果：

```
1 原有的结果是:30
2 打印出:-10
```

顺利HOOK成功！