

1. Link-Time Optimization 原理介绍

Link-Time Optimization 是 LLVM 编译器的一个特性，用于在 link 中间代码时，对全局代码进行优化。这个优化是自动完成的，因此不需要修改现有的代码；这个优化也是高效的，因为可以在全局视角下优化代码。

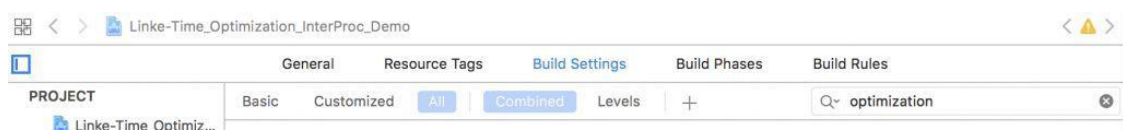
Xcode 目前使用的是 ld 链接器，链接器自带了 ThinLTO 框架可以进行 Link-Time Optimization。苹果在 WWDC 2016 中，明确提出了这个优化的概念，What's New in LLVM。并且说在苹果内部已经广泛地使用这个优化方法进行编译。

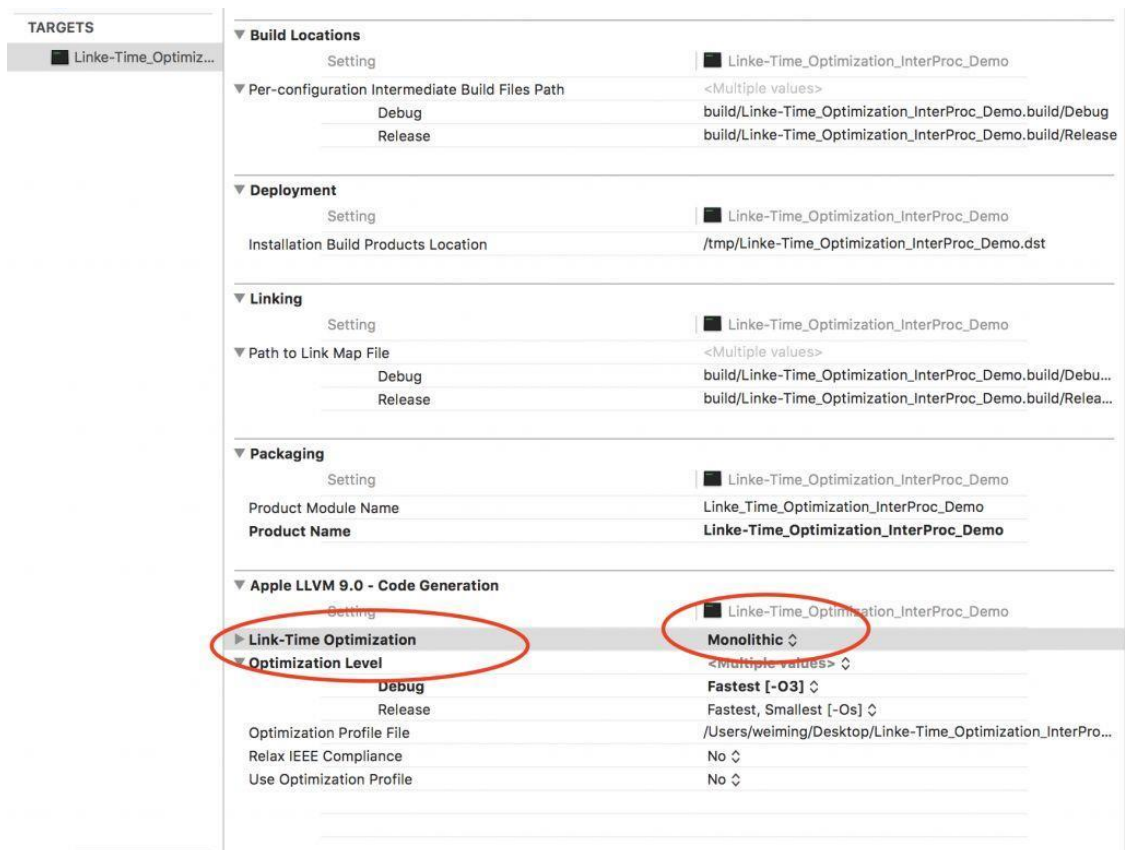
它的优化主要体现在如下几个方面：（后面的章节有例子，建议结合例子阅读）

1. 多余代码去除（Dead code elimination）：如果一段代码分布在多个文件中，但是从来没有被使用，普通的 -O3 优化方法不能发现跨中间代码文件的多余代码，因此是一个“局部优化”。但是 Link-Time Optimization 技术可以在 link 时发现跨中间代码文件的多余代码；
2. 跨过程优化（Interprocedural analysis and optimization）：这是一个相对广泛的概念。举个例子来说，如果一个 if 方法的某个分支永不可能执行，那么在最后生成的二进制文件中就不应该有这个分支的代码；
3. 内联优化（Inlining optimization）：内联优化形象来说，就是在汇编中不使用“call func_name”语句，直接将外部方法内的语句“复制”到调用者的代码段内。这样做的好处是不用进行调用函数前的压栈、调用函数后的出栈操作，提高运行效率与栈空间利用率。

2. 用例子展现优化效果

使用 Link-Time Optimization 的方法是在 Xcode 的 build settings 选项中开启。





以下的优化均使用 -O3 的优化

2.1 多余代码去除 (Dead code elimination)

这个例子主要由3个文件组成：

func1.m (用来定义func1)



```
#import <Foundation/Foundation.h>

int func1(int condition, int inputVar)
{
    int solution = 0;
    int unusedVar1 = 0;
    int unusedVar2 = -1;
    if(condition > 0)
    {
        solution = inputVar*inputVar;
        return solution;
    }
    else
    {
        solution = inputVar + inputVar;
        return solution;
    }
}
```

func2.m (用来定义func2)



```
#import <Foundation/Foundation.h>

extern int func1(int, int);

int func2(int inputVar)
{
    return func1(0, inputVar);
}
```

main.m (主函数)



```
#import <Foundation/Foundation.h>

extern func1(int, int);

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        // insert code here...
        func1(0, 5);
        NSLog(@"Hello, World!");
    }
    return 0;
}
```

编译后的汇编代码（未启用优化）：

在以下代码中，我们可以明显地发现在 `main` 函数中，进行了 `func1` 的调用，可

是 func1 并没有任何作用，属于需要去除的“Dead code”。

```
; Section .text
; Range: [0x10000ef0; 0x10000f53] (99 bytes)
; File offset : [3824; 3923] (99 bytes)
; Flags: 0x8000400
; S_REGULAR
; S_ATTR_PURE_INSTRUCTIONS
; S_ATTR_SOME_INSTRUCTIONS

; ===== BEGINNING OF PROCEDURE =====

func2:
0000000100000ef0    push    rbp
0000000100000ef1    mov     rbp, rsp
0000000100000ef4    mov     eax, edi
0000000100000ef6    xor     edi, edi
0000000100000ef8    mov     esi, eax
0000000100000efa    pop     rbp
0000000100000efb    jmp     _func1
; endp

; ===== BEGINNING OF PROCEDURE =====

_main:
0000000100000f00    push    rbp
0000000100000f01    mov     rbp, rsp
0000000100000f04    push    rbx
0000000100000f05    push    rax
0000000100000f06    call    imp__stubs_objc_autoreleasePoolPush
0000000100000f0b    mov     rbx, rax
0000000100000f0e    xor     edi, edi
0000000100000f10    mov     esi, 0x5
0000000100000f15    call    _func1
0000000100000f1a    lea     eax, dword [cfstring_Hello_World_]
0000000100000f21    xor     eax, eax
0000000100000f23    call    imp__stubs_NSLog
0000000100000f28    mov     rdi, rbx
0000000100000f2b    call    imp__stubs_objc_autoreleasePoolPop
0000000100000f30    xor     eax, eax
0000000100000f32    add     rsp, 0x8
0000000100000f36    pop     rbx
0000000100000f37    pop     rbp
0000000100000f38    ret
; endp
0000000100000f39    db      7 dup (0x90)

; ===== BEGINNING OF PROCEDURE =====

_func1:
0000000100000f40    push    rbp
0000000100000f41    mov     rbp, rsp
0000000100000f44    lea     eax, dword [rsi+rsi]
0000000100000f47    mov     ecx, esi
0000000100000f49    imul    ecx, ecx
0000000100000f4c    test    edi, edi
0000000100000f4e    cmovg   eax, ecx
0000000100000f51    pop     rbp
0000000100000f52    ret
; endp
0000000100000f53    db      0x90 : '.'
```

编译后的汇编代码（启用优化）：

启用优化后，我们可以看到，call func1 的代码已经被移除。同时，还对 func1 和 func2 进行了一些其他的优化。

```
; Section .text
; Range: [0x10000f00; 0x10000f53] (83 bytes)
; File offset : [3840; 3923] (83 bytes)
; Flags: 0x8000400
; S_REGULAR
; S_ATTR_PURE_INSTRUCTIONS
```

```

; S_ATTR_SOME_INSTRUCTIONS

; ===== BEGINNING OF PROCEDURE =====

_func2:
0000000100000f00    push    rbp
0000000100000f01    mov     rbp, rsp
0000000100000f04    lea     eax, dword [rdi+rdi]
0000000100000f07    pop     rbp
0000000100000f08    ret
; endp
0000000100000f09    nop     dword [rax]

; ===== BEGINNING OF PROCEDURE =====

_main:
0000000100000f10    push    rbp
0000000100000f11    mov     rbp, rsp
0000000100000f14    push    rbx
0000000100000f15    push    rax
0000000100000f16    call    imp__stubs__objc_autoreleasePoolPush
0000000100000f1b    mov     rbp, rax
0000000100000f1e    lea     rdi, qword [cfstring_Hello__World_] ; @"Hello, World!", argument "format" for method imp__stubs__
0000000100000f25    xor     eax, eax
0000000100000f27    call    imp__stubs__NSLog
0000000100000f2c    mov     rdi, rbx ; argument "pool" for method imp__stubs__objc_autoreleasePool
0000000100000f2f    call    imp__stubs__objc_autoreleasePoolPop
0000000100000f34    xor     eax, eax
0000000100000f36    add     rsp, 0x8
0000000100000f3a    pop     rbx
0000000100000f3b    pop     rbp
0000000100000f3c    ret
; endp
0000000100000f3d    nop     dword [rax]

; ===== BEGINNING OF PROCEDURE =====

_func1:
0000000100000f40    push    rbp
0000000100000f41    mov     rbp, rsp
0000000100000f44    lea     eax, dword [rsi+rsi]
0000000100000f47    mov     ecx, esi
0000000100000f49    imul    ecx, ecx
0000000100000f4c    test    edi, edi
0000000100000f4e    cmovg   eax, ecx
0000000100000f51    pop     rbp
0000000100000f52    ret
; endp
0000000100000f53    db      0x90 ; '.'

```

优化后，没有func1了

2.2 跨过程和内联优化

这个例子主要由 3 个文件组成

funcs.h: 声明了 extern 的函数名





```
#ifndef funcs_h
#define funcs_h

extern int func1(void);
extern int func2(void);
extern int func4(void);

#endif /* funcs_h */
```

funcs.m: 函数的具体实现



```
#import <Foundation/Foundation.h>

static int i = 0;

int func1()
{
    int data = 0;
    if (i < 0)
    {
        data = func3();
    }
}
```

```
    }  
    printf("func1 running.");  
    data += 10;  
    return data;  
}  
  
void func2()  
{  
    i = -1;  
}  
  
void func4()  
{  
    printf("func4 running.");  
}  
  
int func3()  
{  
    func4();  
    return 1;  
}
```

main.m: 主函数入口




```

#import <Foundation/Foundation.h>
#import "funcs.h"

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        // insert code here...
        func1();
        NSLog(@"Hello, World!");
    }
    return 0;
}

```

编译后的汇编代码（未启用优化）：

从下面的代码中，我们可以看出， `func1` 和 `func4` 是分布在不同的函数中的。

```

; Section: __text
; Range: [0x100000e60; 0x100000f29] (201 bytes)
; File offset: [3680; 3881] (201 bytes)
; Flags: 0x80000400
; S_REGULAR
; S_ATTR_PURE_INSTRUCTIONS

```

```

; S_ATTR_SOME_INSTRUCTIONS

; ===== BEGINNING OF PROCEDURE =====

_main:
0000000010000e60    push    rbp
0000000010000e61    mov     rbp, rsp
0000000010000e64    push    rbx
0000000010000e65    push    rax
0000000010000e66    call    imp__stubs_objc_autoreleasePoolPush
0000000010000e6b    mov     rbx, rax
0000000010000e6e    call    _func1
0000000010000e73    lea     rdi, qword [cfstring_Hello_World]    ; @"Hello, World!", argument "format" for method imp__stubs_
0000000010000e7a    xor     eax, eax
0000000010000e7c    call    imp__stubs_NSLog
0000000010000e81    mov     rdi, rbx
0000000010000e84    call    imp__stubs_objc_autoreleasePoolPop    ; argument "pool" for method imp__stubs_objc_autoreleasePool
0000000010000e89    xor     eax, eax
0000000010000e8b    add     rsp, 0x8
0000000010000e8f    pop     rbx
0000000010000e90    pop     rbp
0000000010000e91    ret
; endp
0000000010000e92    db      14 dup (0x90)

; ===== BEGINNING OF PROCEDURE =====

_func1:
0000000010000ea0    push    rbp                                ; CODE XREF=_main+14
0000000010000ea1    mov     rbp, rsp
0000000010000ea4    push    rbx
0000000010000ea5    push    rax
0000000010000ea6    mov     ebx, 0xa
0000000010000eab    cmp     byte [_i], 0x1
0000000010000eb2    jne     loc_10000ec7

0000000010000eb4    lea     rdi, qword [0x10000f99]
0000000010000ebb    xor     eax, eax
0000000010000ebd    call    imp__stubs_printf
0000000010000ec2    mov     ebx, 0xb

loc_10000ec7:
0000000010000ec7    lea     rdi, qword [0x10000f8a]
0000000010000ece    xor     eax, eax
0000000010000ed0    call    imp__stubs_printf
0000000010000ed5    mov     eax, ebx
0000000010000ed7    add     rsp, 0x8
0000000010000edb    pop     rbx
0000000010000edc    pop     rbp
0000000010000edd    ret
; endp
0000000010000ede    nop

```

编译后的汇编代码（启用优化）：

从下面的代码中，我们可以看出，`func1` 和 `func4` 是一起被“复制”到了 `main` 函数中。同时，`push` 和 `pop` 这类的压栈、出栈代码也都消失了，因为并没有出现类似“`call func_name`”的函数调用。

```

; Section __text
; Range: [0x10000e40; 0x10000f29] (233 bytes)
; File offset : [3648; 3881] (233 bytes)
; Flags: 0x80000400
; S_REGULAR
; S_ATTR_PURE_INSTRUCTIONS
; S_ATTR_SOME_INSTRUCTIONS

```

```

; ===== BEGINNING OF PROCEDURE =====

_main:
00000000100000e40    push    rbp
00000000100000e41    mov     rbp, rsp
00000000100000e44    push    rbx
00000000100000e45    push    rax
00000000100000e46    call    imp__stubs_objc_autoreleasePoolPush
00000000100000e4b    mov     rbx, rax
00000000100000e4e    cmp     byte [i], 0x1
00000000100000e55    jne     loc_100000e65

00000000100000e57    lea     rdi, qword [0x100000f8a]
00000000100000e5e    xor     eax, eax
00000000100000e60    call    imp__stubs_printf

loc_100000e65:
00000000100000e65    lea     rdi, qword [0x100000f99]
00000000100000e6c    xor     eax, eax
00000000100000e6e    call    imp__stubs_printf
00000000100000e73    lea     rdi, qword [cfstring_Hello_World_]
00000000100000e7a    xor     eax, eax
00000000100000e7c    call    imp__stubs_NSLog
00000000100000e81    mov     rdi, rbx
00000000100000e84    call    imp__stubs_objc_autoreleasePoolPop
00000000100000e89    xor     eax, eax
00000000100000e8b    add     rsp, 0x8
00000000100000e8f    pop     rbx
00000000100000e90    pop     rbp
00000000100000e91    ret
00000000100000e92    ; endp
                                nop     word [cs:rax+rax]

; func1 和 func4 代码被“复制”到main函数
; "func4 running.", argument "format" for method imp__stubs_
; "func1 running.", argument "format" for method imp__stubs_
; @"Hello, World!", argument "format" for method imp__stubs_
; argument "pool" for method imp__stubs_objc_autoreleasePool

```

2.3 总结

从上面的例子可以看出，开启这个优化后，一方面减少了汇编代码的体积，一方面提高了代码的运行效率。

3. QQ 音乐优化实例

在 QQ 音乐 iOS 版中，开启这项优化将二进制文件的体积从 130 MB 缩小至约 125 MB，缩减了约 4%。运行效率因为受网络等因素的影响，暂时无法具体测量。

从具体的文件来看，

优化前：

```

; ===== BEGINNING OF PROCEDURE =====

-[QQMusicAppDelegate application:didFinishLaunchingWithOptions:]
000000001014d6da0    push    rbp
000000001014d6da1    mov     rbp, rsp
; Objective C Implementation defined at 0x104c31a68 (instance

```

```

000000001014d6da4 sub     rsp, 0x70
000000001014d6da8 lea     rax, qword [rbp+var_18]
000000001014d6dac mov     qword [rbp+var_8], rdi
000000001014d6db0 mov     qword [rbp+var_10], rsi
000000001014d6db4 mov     rdi, rax
000000001014d6db8 mov     rsi, rdx
000000001014d6dbf mov     qword [rbp+var_30], rcx
000000001014d6dc2 call    imp__stubs_objc_storeStrong
000000001014d6dc6 lea     rax, qword [rbp+var_20]
000000001014d6dcb mov     qword [rbp+var_20], 0x0
000000001014d6dcf mov     rcx, qword [rbp+var_30]
000000001014d6ddb mov     rdi, rax
000000001014d6ddb mov     rsi, rcx
000000001014d6dde call    imp__stubs_objc_storeStrong
000000001014d6de6 mov     rax, qword [objc_cls_ref_QMLaunchManager]
000000001014d6ded mov     rsi, qword [0x104fbcc08]
000000001014d6df4 mov     rdi, rax
000000001014d6df7 call    imp__stubs_objc_msgSend
000000001014d6dfc mov     rdi, rax
000000001014d6dff call    imp__stubs_objc_retainAutoreleasedReturnValue
000000001014d6e04 lea     rcx, qword [cfstring_Launching_0]
000000001014d6e0b mov     rsi, qword [0x104fe50f8]
000000001014d6e12 mov     rdx, rax
000000001014d6e15 mov     rdi, rdx
000000001014d6e18 mov     rcx, rcx
000000001014d6e1b mov     qword [rbp+var_38], rax
000000001014d6e1f call    imp__stubs_objc_msgSend
000000001014d6e24 mov     rax, qword [rbp+var_38]
000000001014d6e28 mov     rdi, rax
000000001014d6e2b call    imp__stubs_objc_release
000000001014d6e30 mov     byte [rbp+var_21], 0x1
000000001014d6e34 mov     rax, qword [rbp+var_8]
000000001014d6e38 mov     rdx, qword [rbp+var_20]
000000001014d6e3c mov     rsi, qword [0x104fe0e50]
000000001014d6e43 mov     rdi, rax
000000001014d6e46 call    imp__stubs_objc_msgSend
000000001014d6e4b mov     rax, qword [objc_cls_ref_SetupHelper]
000000001014d6e52 mov     rsi, qword [0x104fe42b0]
000000001014d6e59 mov     rdi, rax
000000001014d6e5c call    imp__stubs_objc_msgSend
000000001014d6e61 mov     rcx, qword [rbp+var_8]
000000001014d6e65 mov     rdx, qword [OBJC_IVAR_$_QQMusicAppDelegate._launchMode]
000000001014d6e6c mov     qword [rcx+rdx], rax
000000001014d6e70 mov     rax, qword [objc_cls_ref_QMOpenAppManager]
000000001014d6e77 mov     rsi, qword [0x104fbcc08]
000000001014d6e7e mov     rdi, rax
000000001014d6e81 call    imp__stubs_objc_msgSend
000000001014d6e86 mov     rdi, rax
000000001014d6e89 call    imp__stubs_objc_retainAutoreleasedReturnValue
000000001014d6e8e mov     rdx, qword [rbp+var_20]
000000001014d6e92 mov     rsi, qword [0x104fe5160]
000000001014d6e99 mov     rcx, rax
000000001014d6e9c mov     rdi, rcx
000000001014d6e9f mov     qword [rbp+var_40], rax
000000001014d6ea3 call    imp__stubs_objc_msgSend
000000001014d6ea8 mov     rcx, qword [rbp+var_8]
000000001014d6eac mov     rdx, qword [OBJC_IVAR_$_QQMusicAppDelegate.eAppLaunchMode]
000000001014d6eb3 mov     dword [rcx+rdx], eax
000000001014d6eb6 mov     rcx, qword [rbp+var_40]
000000001014d6eba mov     rdi, rcx
000000001014d6ebd call    imp__stubs_objc_release
000000001014d6ec2 mov     edx, edx
000000001014d6ec4 mov     rcx, qword [rbp+var_8]
000000001014d6ec8 mov     rsi, qword [0x104fe5168]
000000001014d6ecf mov     rdi, rcx
000000001014d6ed2 call    imp__stubs_objc_msgSend
000000001014d6ed7 mov     rcx, qword [objc_cls_ref_QMFileManager]
000000001014d6ede mov     rsi, qword [0x104fe45d0]
000000001014d6ee5 mov     rdi, rcx
000000001014d6ee8 call    imp__stubs_objc_msgSend
000000001014d6eed mov     rdi, rax
000000001014d6ef0 call    imp__stubs_objc_retainAutoreleasedReturnValue
000000001014d6ef5 mov     rsi, qword [0x104fe5170]
000000001014d6efc mov     rcx, rax
000000001014d6eff mov     rdi, rcx
000000001014d6f02 mov     qword [rbp+var_48], rax
000000001014d6f06 call    imp__stubs_objc_msgSend
000000001014d6f0b mov     rax, qword [rbp+var_48]
000000001014d6f0f mov     rdi, rax
000000001014d6f12 call    imp__stubs_objc_release
000000001014d6f17 mov     rax, qword [objc_cls_ref_QMURLCache]
000000001014d6f1e mov     rsi, qword [0x104fb4cd8]
000000001014d6f25 mov     rdi, rax
000000001014d6f28 call    imp__stubs_objc_msgSend
000000001014d6f2d mov     rdi, rax
000000001014d6f30 call    imp__stubs_objc_retainAutoreleasedReturnValue

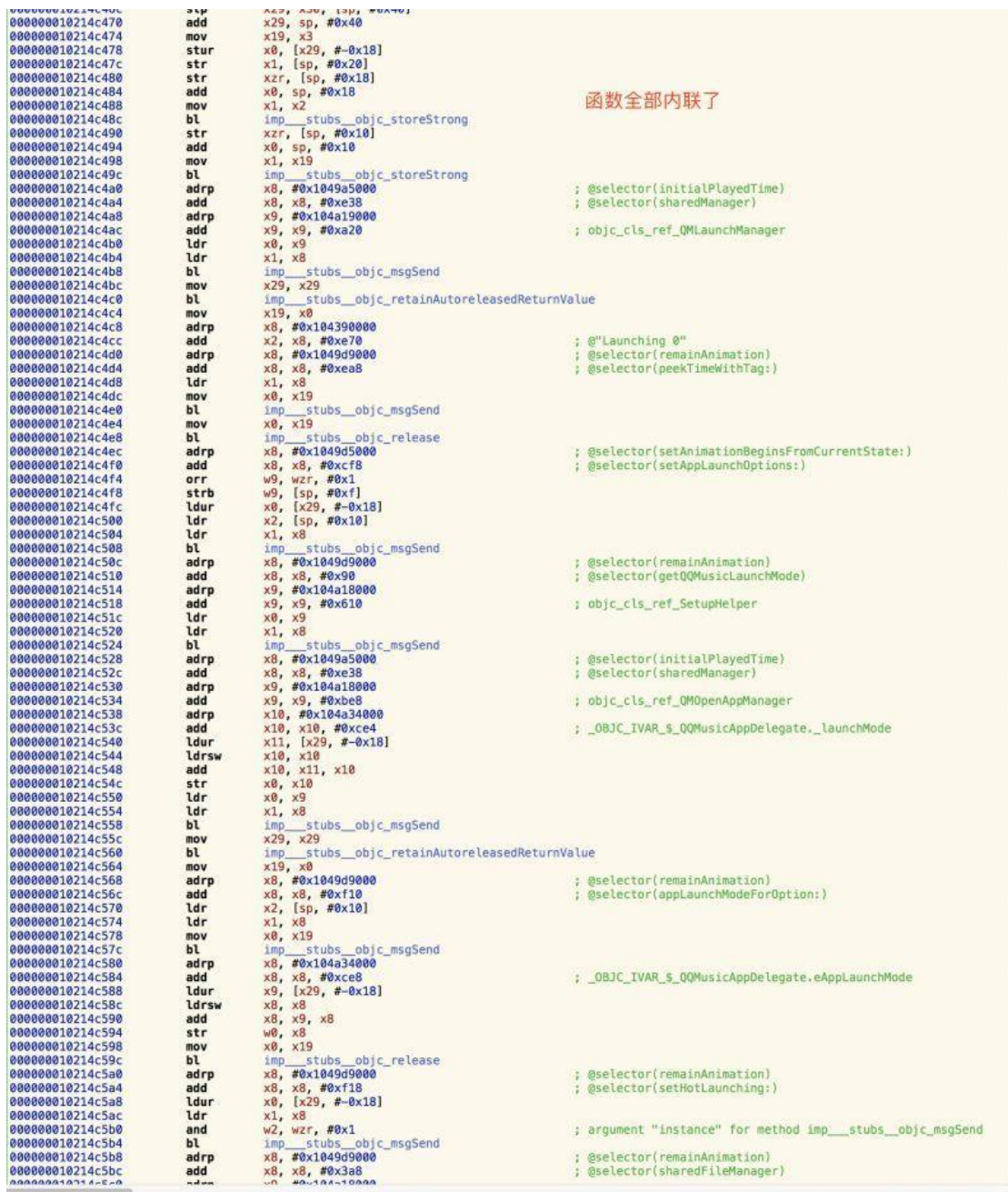
```

优化后：

```

; ===== BEGINNING OF PROCEDURE =====
- [QQMusicAppDelegate application:didFinishLaunchingWithOptions]:
0000000010214c464 sub     sp, sp, #0x50
0000000010214c468 stp     x20, x19, [sp, #0x30]
0000000010214c46c cmp     v30, v30, fcc #0x401

```

从代码行数上来看，汇编代码行数几乎保持不变，说明并没有很多“Dead code”。

4. 进一步拓展

前面在 build settings 中，我们设置了 Monolithic 的优化方式，这种方式并不支持多线程和增量链接。因此，在新的版本中，苹果使用了新的优化方式 Incremental，大大减少了链接的时间。建议开启。

小插曲：在开启 `Incremental` 的选项后，QQ音乐项目出现了“`duplicate symbols`”的错误，经查是之前代码不规范引起的。全局变量在定义时，必须使用 `static` 关键字或者单独在 `.m` 文件中定义，`.h` 文件中只能声明变量，而不应该定义变量。