橘子不酸丶 Lv1
2020年08月17日 阅读 1025

关注

iOS优化篇之App启动时间优化

> 原文：橘子不酸丶 [www.zyiner.com/article/5]

前言

最近由于体验感觉我们的app启动时间过长，因此做了APP的启动优化。本次优化主要从三个方面来做了启动时间的优化，**main之后的耗时方法优化**、**premain的+load方法优化**、**二进制重排优化premain时间**。

通常我们对于启动时间的定义为从用户点击app到看到首屏的时间。因此对于启动时间优化就是遵循一个原则：尽早让用户看到首页内容。

app启动过程

iOS应用的启动可分为pre-main阶段和main()阶段，pre-main阶段为main函数执行之前所做的操作，main阶段为main函数到首页展示阶段。其中系统做的事情为：

premain

- 加载所有依赖的Mach-O文件（递归调用Mach-O加载的方法）
- 加载动态链接库加载器dyld（dynamic loader）
- 定位内部、外部指针引用，例如字符串、函数等
- 加载类扩展（Category）中的方法
- C++静态对象加载、调用ObjC的 +load 函数
- 执行声明为**attribute**((constructor))的C函数

main

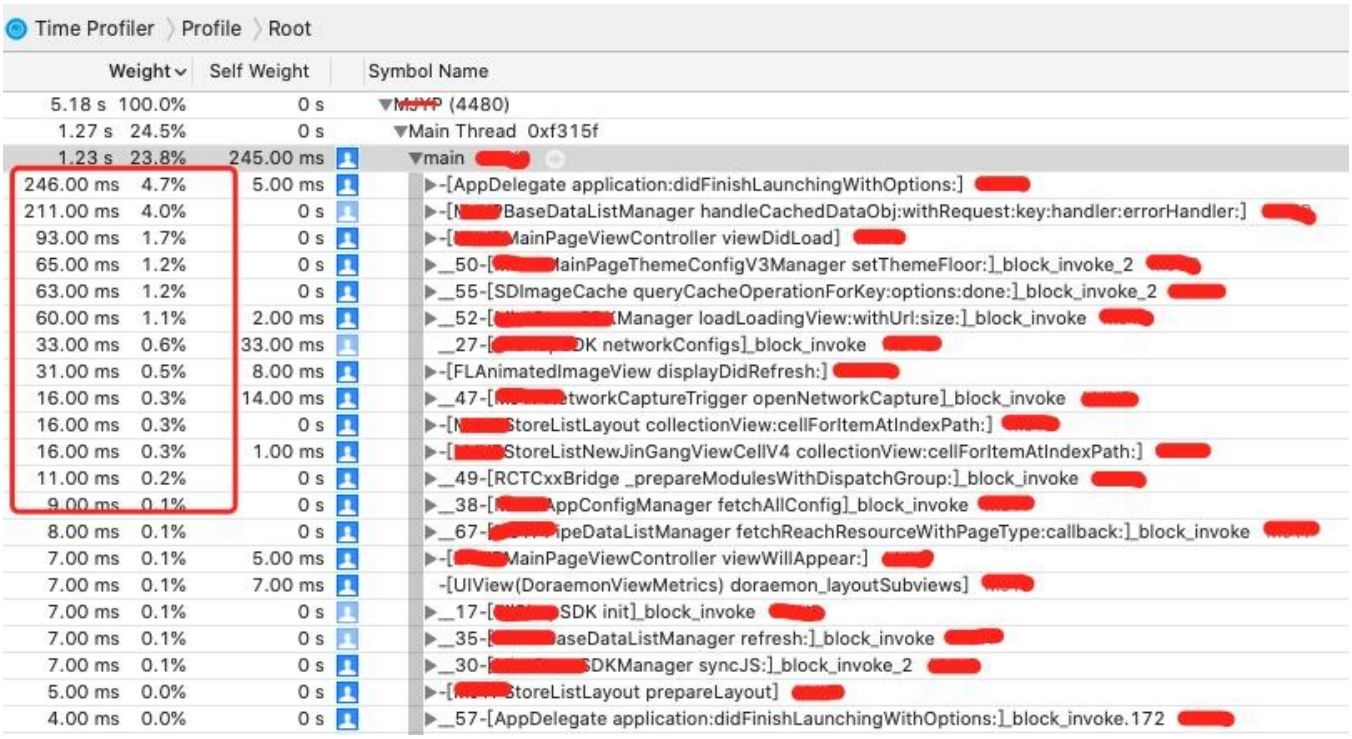
- 调用applicationWillFinishLaunching

通常的premain阶段优化即为删减无用的类方法、减少+load操作、减少**attribute**((constructor))的C函数、减少启动加载的动态库。而main阶段的优化为将启动时非必要的操作延迟到首页显示之后加载、统计并优化耗时的方法、对于一些可以放在子线程的操作可以尽量不占用主线程。

一、耗时方法优化

1.统计启动时的耗时方法

我们可以通过Instruments的TimeProfile来统计启动时的主要方法耗时，Call Tree->Hide System Libraries过滤掉系统库可以查看主线程下方法的耗时。



也可以通过打印时间的方式来统计各个函数的耗时。

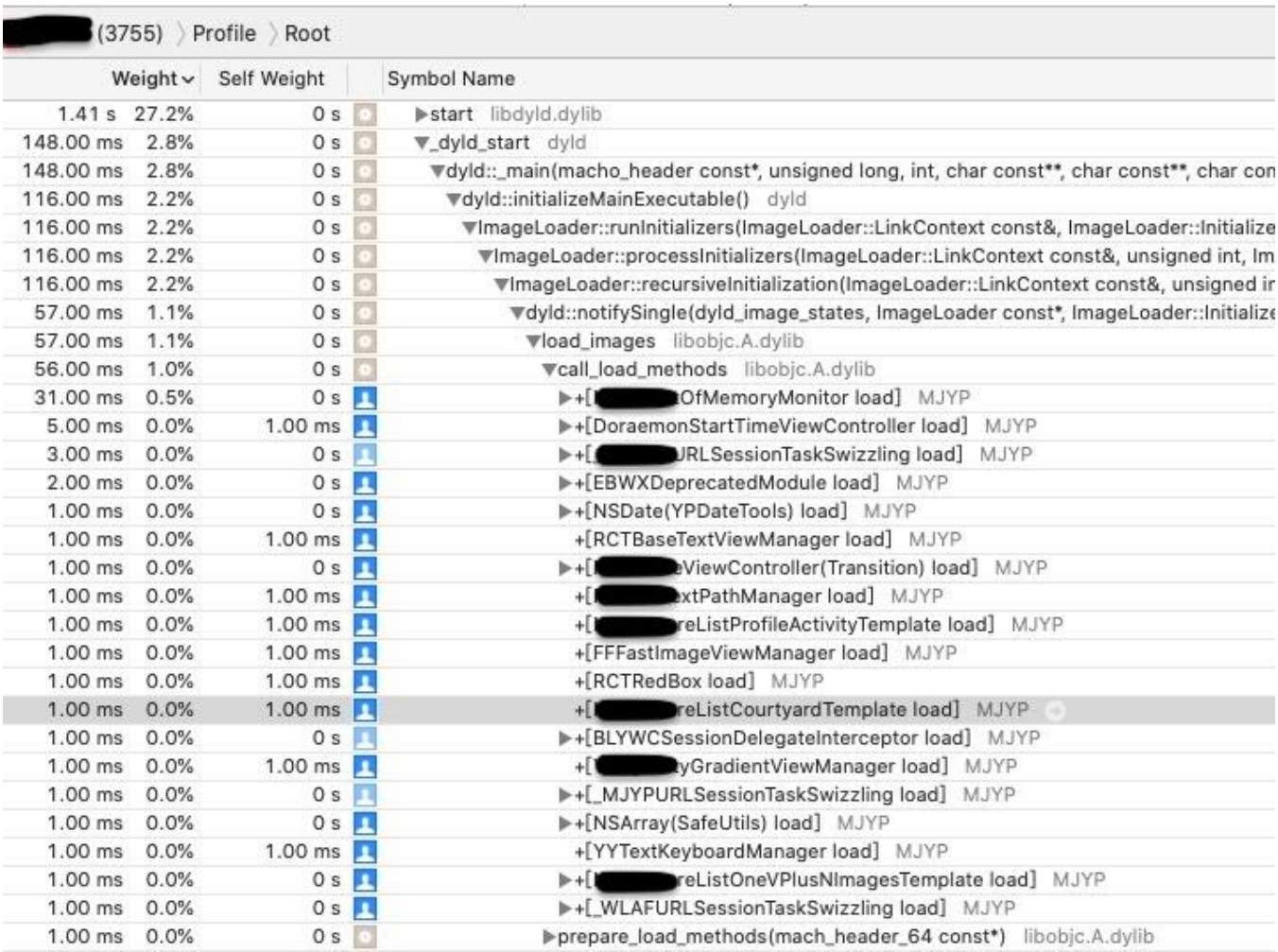
```
double launchTime = CFAbsoluteTimeGetCurrent();
[SDWebImageManager sharedManager];
NSLog(@"launchTime = %f秒", CFAbsoluteTimeGetCurrent() - launchTime);
```

这一阶段就是需要对启动过程的业务逻辑进行梳理，确认哪些是可以延迟加载的，哪些可以放在子线程加载，以及哪些是可以懒加载处理的。同时对耗时比较严重的方法进行review并提出优化策略进行优化

二、+load方法优化以及删减不用的类

2.1 +load方法统计

同样的我们可以通过Instruments来统计启动时所有的+load方法，以及+load方法所用耗时



我们可以对不必要的+load方法进行优化，比如放在+initialize里。不必要的+load进行删减。

2.2 使用__attribute优化+load方法

由于在我们的工程中存在很多的+load方法，而其中一大部分为cell模板注册的+load方法(我们的每一个cell对应一个模板，然后该模板对应一个字符串，在启动时所有的模板方法都在+load中注册对应的字符串即在字典中存储字符串和对应的cell模板，然后动态下发展示对应的cell)。

即存在这种场景，在启动时需要大量的在+load中注册key-value。

此时可以使用__attribute((used, section("__DATA,"#sectname" ")))的方式在编译时写

```
#ifndef ZYStoreListTemplateSectionName
#define ZYStoreListTemplateSectionName "ZYTempSection"
#endif

#define ZYStoreListTemplateDATA(sectname) __attribute((used, section("__DATA,"#sectname" ")

#define ZYStoreListTemplateRegister(templatename,templateclass) \
class NSObject; char * k##templatename##_register ZYStoreListTemplateDATA(ZYTempSection) = '
/**
通过ZYStoreListTemplateRegister(key,classname)注册处理模板的类名(类必须是ZYStoreListBaseTemplate
【注意事项】
该方式通过__attribute属性在编译期间绑定注册信息，运行时读取速度快，注册信息在首次触发调用时读取，不影响pre-
该方式注册时‘key’字段中不支持除下划线'_'以外的符号
【使用示例】
注册处理模板的类名：@ZYStoreListTemplateRegister(baseTemp,ZYStoreListBaseTemplate)
**/
```

在使用时@ZYStoreListTemplateRegister(baseTemp,ZYStoreListBaseTemplate)即为在编译期间绑定注册信息。

读取使用__attribute在编译期间写入的key-value字符串。 关于__attribute详情可以参考__attribute黑魔法

```
#pragma mark - 第一次使用时读取ZYStoreListTemplateSectionName的__DATA所有数据
+ (void)readTemplateDataFromMach0 {
    //1.根据符号找到所在的mach-o文件信息
    Dl_info info;
    dladdr((__bridge void *)[self class], &info);

    //2.读取__DATA中自定义的ZYStoreListTemplateSectionName数据
    #ifndef __LP64__
        const struct mach_header *mhp = (struct mach_header*)info.dli_fbase;
        unsigned long templateSize = 0;
        uint32_t *templateMemory = (uint32_t*)getsectiondata(mhp, "__DATA", ZYStoreListTemp
    #else /* defined(__LP64__) */
        const struct mach_header_64 *mhp = (struct mach_header_64*)info.dli_fbase;
        unsigned long templateSize = 0;
        uint64_t *templateMemory = (uint64_t*)getsectiondata(mhp, "__DATA", ZYStoreListTemp

    #endif /* defined(__LP64__) */

    //3.遍历ZYStoreListTemplateSectionName中的协议数据
    unsigned long counter = templateSize/sizeof(void*);
```

```
        NSString *str = [NSString stringWithUTF8String:string];
        if(!str)continue;

        //NSLog(@"config = %@", str);
        NSData *jsonData = [str dataUsingEncoding:NSUTF8StringEncoding];
        NSError *error = nil;
        id json = [NSJSONSerialization JSONObjectWithData:jsonData options:0 error:&error];
        if (!error) {
            if ([json isKindOfClass:[NSDictionary class]] && [json allKeys].count) {
                NSString *templatesName = [json allKeys][0];
                NSString *templatesClass  = [json allValues][0];
                if (templatesName && templatesClass) {
                    [self registerTemplateName:templatesName templateClass:NSClassFromStrin
                }
            }
        }
    }
}
```

这样我们就可以优化大量的重复+load方法。而且使用__attribute属性为编译期间绑定注册信息，运行时读取速度快，注册信息在首次触发调用时读取，不影响pre-main时间。

三、二进制重排

自从抖音团队分享了这篇 抖音研发实践：基于二进制文件重排的解决方案 APP启动速度提升超15% 启动优化文章后，二进制重排优化 pre-main 阶段的启动时间自此被大家广为流传。

当进程访问一个虚拟内存Page而对应的物理内存却不存在时，会触发一次 缺页中断（Page Fault）。

二进制重排，主要是优化我们启动时需要的函数非常分散在各个页，启动时就会多次Page Fault造成时间的损耗。

3.1 获取Order File

本次主要是通过Clang静态插桩的方式，获取到所有的启动时调用的函数符号，导出为OrderFile。

`Target -> Build Setting -> Custom Complier Flags -> Other C Flags` 添加 `-fsanitize-coverage=func,trace-pc-guard` 参数

然后实现hook代码获取所有启动的函数符号。启动后在首页显示之后，可以通过触发下边-

```
#import "dlfcn.h"
#import <libkern/OSAtomic.h>
```

```
void __sanitizer_cov_trace_pc_guard_init(uint32_t *start,
                                         uint32_t *stop) {
    static uint64_t N;  // Counter for the guards.
    if (start == stop || *start) return;  // Initialize only once.
    printf("INIT: %p %p\n", start, stop);
    for (uint32_t *x = start; x < stop; x++)
        *x = ++N;  // Guards should start from 1.
}

//原子队列
static OSQueueHead symboList = OS_ATOMIC_QUEUE_INIT;
static BOOL isEnd = NO;
//定义符号结构体
typedef struct{
    void * pc;
    void * next;
}SymbolNode;

void __sanitizer_cov_trace_pc_guard(uint32_t *guard) {
    //if (!*guard) return;  // Duplicate the guard check.
    if (isEnd) {
        return;
    }
    void *PC = __builtin_return_address(0);

    SymbolNode * node = malloc(sizeof(SymbolNode));
    *node = (SymbolNode){PC,NULL};

    //入队
    // offsetof 用在这里是为了入队添加下一个节点找到 前一个节点next指针的位置
    OSAtomicEnqueue(&symboList, node, offsetof(SymbolNode, next));
}

- (void)getAllSymbols {
    isEnd = YES;
    NSMutableArray<NSString *> * symbolNames = [NSMutableArray array];
    while (true) {
        //offsetof 就是针对某个结构体找到某个属性相对这个结构体的偏移量
        SymbolNode * node = OSAtomicDequeue(&symboList, offsetof(SymbolNode, next));
        if (node == NULL) break;
        Dl_info info;
        dladdr(node->pc, &info);
```

```
        // 添加 _
        BOOL isObjc = [name hasPrefix:@"+["] || [name hasPrefix:@"-["];
        NSString * symbolName = isObjc ? name : [@"_" stringByAppendingString:name];

        //去重
        if (![symbolNames containsObject:symbolName]) {
            [symbolNames addObject:symbolName];
        }
    }

    //取反
    NSArray * symbolAry = [[symbolNames reverseObjectEnumerator] allObjects];
    NSLog(@"%@",symbolAry);

    //将结果写入到文件
    NSString * funcString = [symbolAry componentsJoinedByString:@"\n"];
    NSString * filePath = [NSTemporaryDirectory() stringByAppendingPathComponent:@"linkSymbo
    NSData * fileContents = [funcString dataUsingEncoding:NSUTF8StringEncoding];
    BOOL result = [[NSFileManager defaultManager] createFileAtPath:filePath contents:fileCon
    if (result) {
        NSLog(@"linkSymbol result %@",filePath);
    }else{
        NSLog(@"linkSymbol result文件写入出错");
    }
}
```

由于我们的工程为pod工程，如果只在主工程里添加other c flags只能获取到主工程层下的所有启动函数，如果要获取所有的包含依赖pod中启动函数符号则需要在每一个pod target设置other c flags参数。

我们可以通过添加pod脚本来对每一个target添加other c flags参数。

在podfile最后添加脚本来为每一个target添加编译参数。注意可以过滤掉Debug环境才加载的库。

```
post_install do |installer|
    pods_project = installer.pods_project
    build_settings = Hash[
    'OTHER_CFLAGS' => '-fsanitize-coverage=func,trace-pc-guard'
#    ,'OTHER_SWIFT_FLAGS' => '-sanitize=undefined -sanitize-coverage=func'
    ]

    pods_project.targets.each do |target|
#      if !target.name.include?('Pods-')
      if !target.name.include?('Pods-') and target.name != 'LookinServer' and target.nam
```

```
            build_settings.each do |pair|
                key = pair[0]
                value = pair[1]
                if config.build_settings[key].nil?
                    config.build_settings[key] = ['']
                end
                if !config.build_settings[key].include?(value)
                    config.build_settings[key] << value
                end
            end
        end

        puts '[Other C Flags]: ' + target.name + ' success.'
      end
    end
end
```

重新install之后所有的pod target都会添加上other c flags参数。然后就可以获取到所有的函数符号(注意如果是二进制库则还是会获取不到)。

3.1 设置Order File

通过objc的源码可以看到objc也是通过设置order file设置编译顺序的。

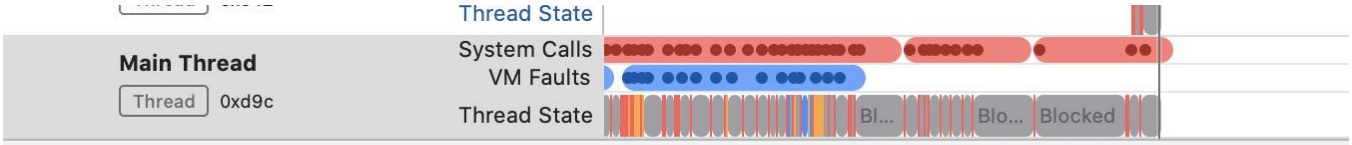
我们可以在主工程的`Target -> Build Setting -> Linking -> Order File`添加上述步骤导出的函数符号列表linkSymbols.order。

`$(SRCROOT)/linkSymbols.order` 这里可以根据根目录路径然后寻找，不必把orderfile添加到工程bundle里。如果添加到工程里则会被打包到ipa里。我们可以只是放在工程文件夹下，只在编译的时候根据路径引用就可以了。

设置完orderfile之后我们可以通过设置write link map file属性为YES来找到编译时生成的符号`($Project)-LinkMap-normal-arm64.txt`。 修改完毕后 clean 一下 , 运行工程 , Products - show in finder, 找到 macho 的上上层目录。 找到结尾为arm64.txt的文件并打开。

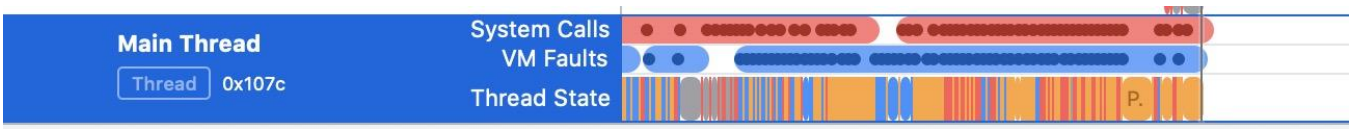
`Intermediates -> project_ios.build -> Debug-iphoneos -> project_ios.build -> project_ios-LinkMap-normal-arm64.txt`

`($Project)-LinkMap-normal-arm64.txt`文件里在`#Symbols`之后为函数符号链接的顺序，可以验证一下重排是否成功。



Main Thread (0xd9c) 〉Summary: Virtual Memory

Operation	Count	Duration	Min Duration	Avg Duration	Std Dev Du...	Max Durati...
▼ * All *	9,573	533.32 ms	625 ns	55.71 µs	191.34 µs	7.21 ms
File Backed Page In	2,326	505.57 ms	1.38 µs	217.35 µs	339.11 µs	7.21 ms
Zero Fill	2,449	10.75 ms	2.29 µs	4.39 µs	5.01 µs	86.67 µs
Page Cache Hit	3,857	8.92 ms	625 ns	2.31 µs	6.64 µs	407.54 µs
Copy On Write	941	8.08 ms	3.21 µs	8.59 µs	51.90 µs	1.20 ms



Main Thread (0x107c) 〉Summary: Virtual Memory

Operation	Count	Duration	Min Duration	Avg Duration	Std Dev Du...	Max Durati...
▼ * All *	8,897	413.31 ms	709 ns	46.45 µs	181.51 µs	9.59 ms
File Backed Page In	2,014	388.71 ms	1.25 µs	193.00 µs	342.04 µs	9.59 ms
Zero Fill	2,300	9.45 ms	1.92 µs	4.11 µs	4.28 µs	91.17 µs
Copy On Write	1,101	8.33 ms	2.58 µs	7.57 µs	38.01 µs	942.67 µs
Page Cache Hit	3,482	6.82 ms	709 ns	1.96 µs	1.54 µs	44.08 µs

总结

最后在看一下本次优化的效果。图中为iPhone6s Plus重启后第一次启动的优化前后截屏。

参考文章:

iOS 优化篇 - 启动优化之Clang插桩实现二进制重排

iOS App启动优化

> **关注下面的标签，发现更多相似文章**

iOS

输入评论...

用户3661151249884

mark

12小时前 回复

相关推荐

老司机技术周报 · 1天前 · iOS / SwiftUI

【WWDC20】10037 - SwiftUI 中的 App 要领

8

FengyunSky · 4天前 · iOS

一文读懂iOS线程调用栈原理

12

ZenonHuang · 7天前 · iOS

iOS 的自动构建流程

61 7

JeremyHuang37 · 1天前 · iOS

【译】自定义 Collection View Layout -- 一个简单的模板

2

掘金酱 · 27天前 · iOS / Android / 前端 / 后端 / 程序员

🏆 掘金征文 | 2020与我的年中总结

94 109

Chouee · 3天前 · iOS

造轮子 - UITableView字母索引条

7 1

路过看风景 · 2天前 · iOS

CocoaPods原理 及 组件化

3

首页

探索掘金

9

路过看风景 · 1天前 · iOS

iOS事件处理 UIResponder

2