









字节跳动--iPhone安装包的优化

背景

这是一次安装包大小优化的实践。

随着业务的增加，工程中引入越来越多的业务代码和第三方库，整个安装包越来越大。以今日头条5.7.5为例 最近几个版本的ipa大小如下：

- 5.7 -> 72.2M (+0.8M) 正常业务增量
- 5.6 -> 71.4M (+14M) 主要原因：接入某SDK后安装包的增加(约13M) 京东 SDK (约1M)
- 5.5 -> 57.4M (+1.7M) 正常业务增量
- 5.4 -> 55.1M (+28M) 主要原因：加入Swift 系统会加入语言库来支持 (约27M)
- 5.3 -> 27.5M 之前做过一次资源文件和类文件的清理

ne	Date Modified	Size	Packed
 close.png	16/8/11 下午3:34	4 KB	3 KB
 safe.jpg	16/8/11 下午3:34	16 B	19 B
 LaunchImage-700-568h@2x.png	16/8/11 下午3:34	72 KB	71 KB
 LaunchImage-700-Landscape@2x~ipad.png	16/8/11 下午3:34	128 KB	122 KB
 LaunchImage-700-Landscape~ipad.png	16/8/11 下午3:34	45 KB	43 KB
 LaunchImage-700-Portrait@2x~ipad.png	16/8/11 下午3:34	181 KB	178 KB
 LaunchImage-700-Portrait~ipad.png	16/8/11 下午3:34	66 KB	65 KB
 LaunchImage-700@2x.png	16/8/11 下午3:34	65 KB	64 KB
		00	07









优化安装包分为如下几个步骤：

1. 分析安装包的构成，一个安装包分为二进制代码文件，资源，配置文件。需要知道各个方面的占比。
2. 知道各个方向的优化策略，譬如二进制文件如何优化，资源文件如何优化
3. 执行优化，得出结果

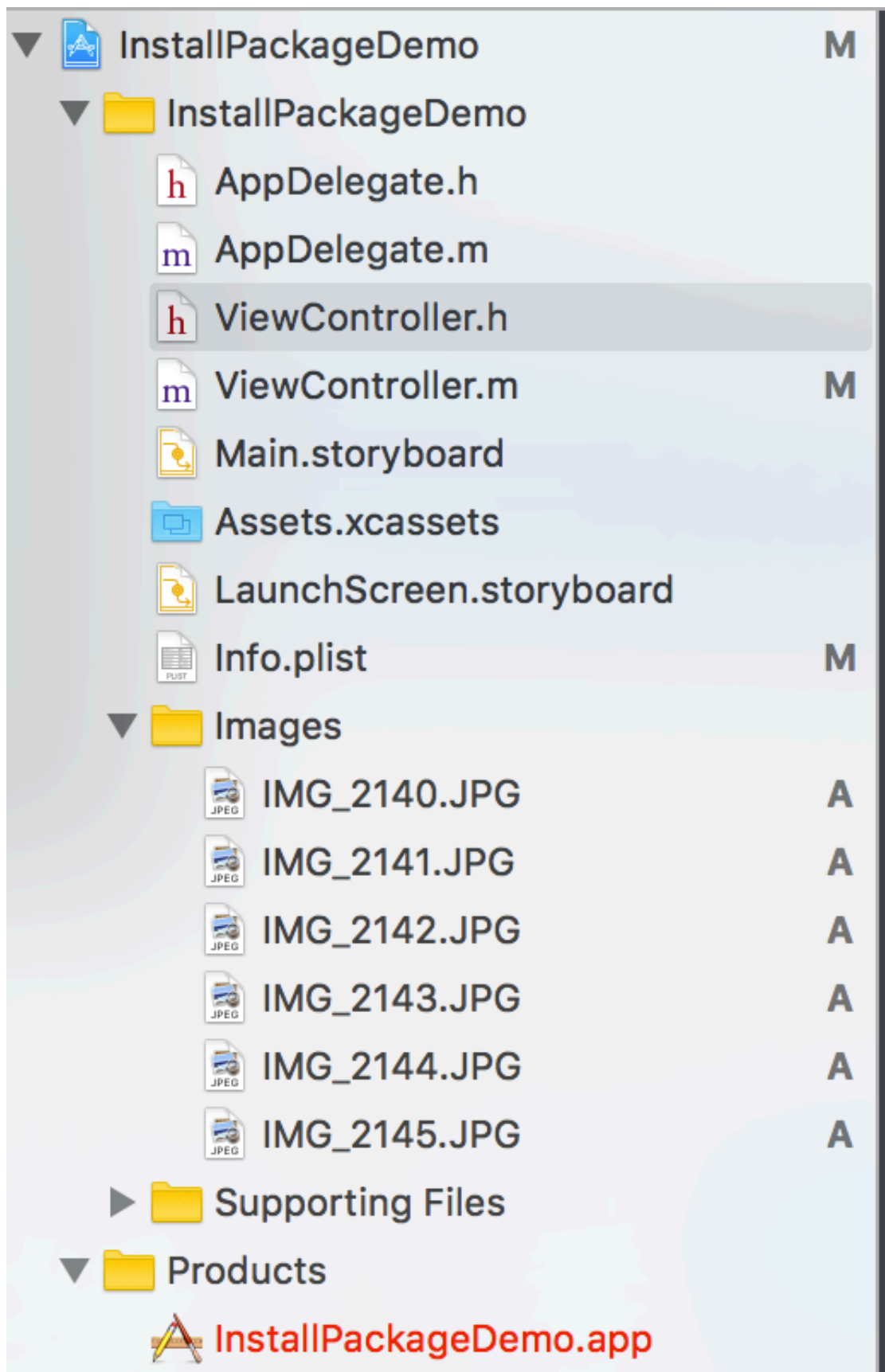
分析

首先进行第一步，分析安装包的构成： 88M的安装包解压后变成220MB。
ipa是一个压缩包， 安装包里的主要构成是（图片+文档+二进制文件）， 我们下面的分析

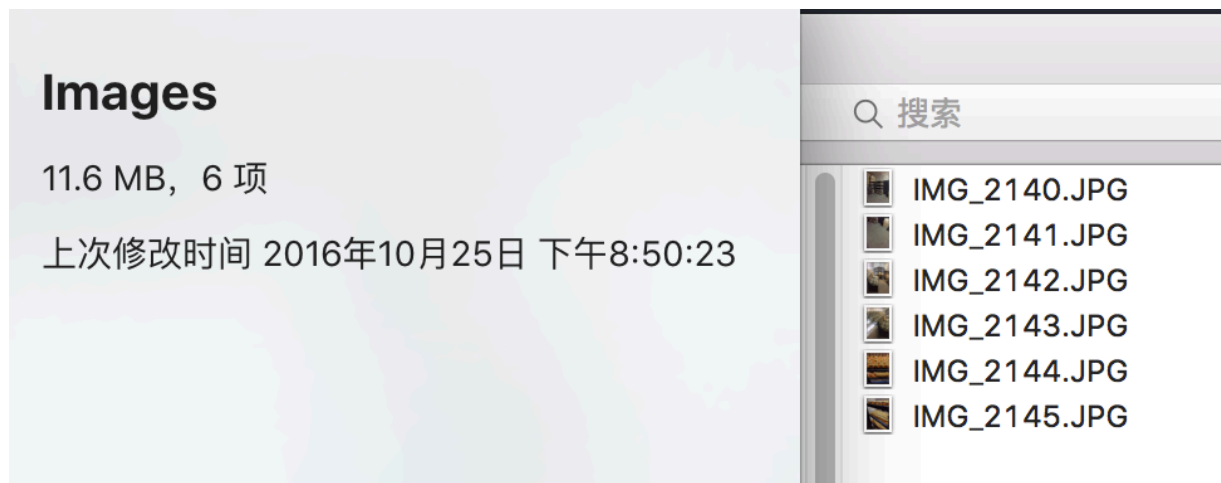
1.图片优化

ne	Date Modified	Size	Packed
 close.png	16/8/11 下午3:34	4 KB	3 KB
 safe.jpg	16/8/11 下午3:34	16 B	19 B
 LaunchImage-700-568h@2x.png	16/8/11 下午3:34	72 KB	71 KB
 LaunchImage-700-Landscape@2x~ipad.png	16/8/11 下午3:34	128 KB	122 KB
 LaunchImage-700-Landscape~ipad.png	16/8/11 下午3:34	45 KB	43 KB
 LaunchImage-700-Portrait@2x~ipad.png	16/8/11 下午3:34	181 KB	178 KB
 LaunchImage-700-Portrait~ipad.png	16/8/11 下午3:34	66 KB	65 KB
 LaunchImage-700@2x.png	16/8/11 下午3:34	65 KB	64 KB
		88	87

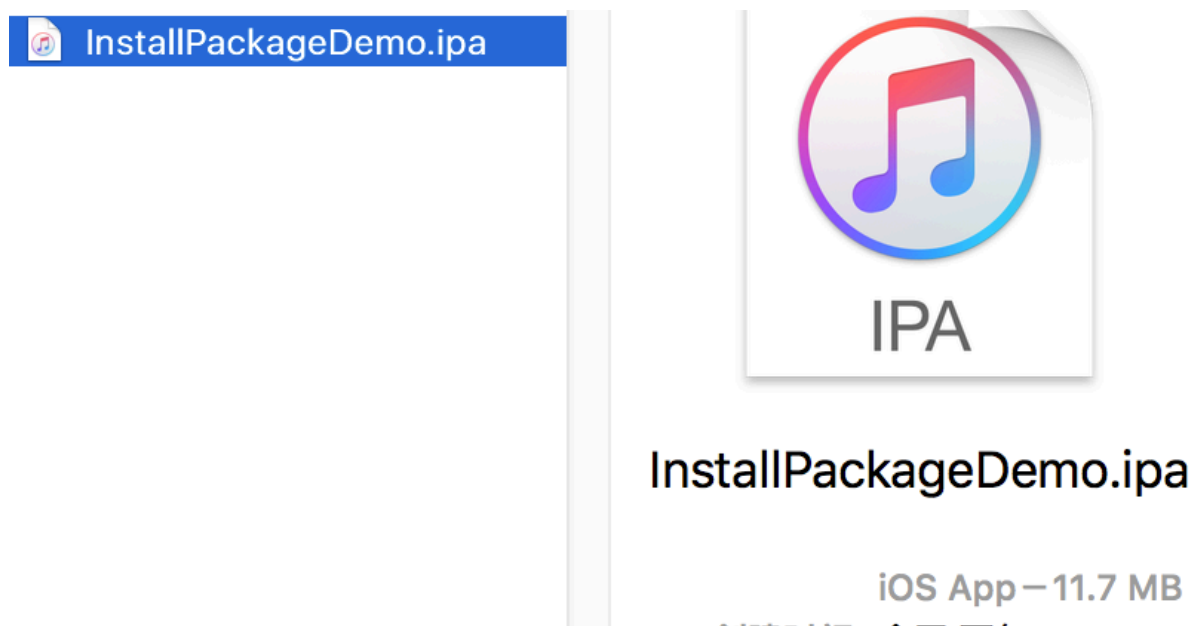
从上面来看，图片的压缩比最小。几乎没有压缩，这也说明每减少一张图片，就实实在在的减少了ipa的大小。 为了验证上面的数据，我们来做一些实验： 我们新建一个项目， 测试资源图片对安装包的大小的影响： 目录结构如下：



其中资源信息如下：



然后进行打包 Archive→Export. 得到 IPA 文件



从上面的结果来看，安装包的大小基本等于图片资源的大小，可以看一下IPA的内容详情视图（下图），发现图片确实没有怎么压缩：

ZIP		Extract with preset			
Name	Date Modified	Size	Packed	Kind	
▼ InstallPackageDemo.app	16/10/25 下午8:41	11.9 MB	11.7 MB	应用程序	
▼ _CodeSignature	16/10/25 下午8:41	7 KB	2 KB	文件夹	
CodeResources	16/10/25 下午8:41	7 KB	2 KB	文稿	
archived-expanded-entitlements.xcent	16/10/25 下午8:41	390 B	246 B	文稿	
▼ Base.lproj	16/10/25 下午8:41	6 KB	4 KB	文件夹	
▼ LaunchScreen.storyboardc	16/10/25 下午8:41	3 KB	2 KB	Interface Builder Compiled Storyboard Document	
01J-lp-oVM-view-Ze5-6b-2t3.nib	16/10/25 下午8:41	2 KB	1 KB	Interface Builder NIB Document	
Info.plist	16/10/25 下午8:41	258 B	149 B	Property List	
UIViewController-01J-lp-oVM.nib	16/10/25 下午8:41	832 B	552 B	Interface Builder NIB Document	
▼ Main.storyboardc	16/10/25 下午8:41	3 KB	2 KB	Interface Builder Compiled Storyboard Document	
BYZ-38-t0r-view-8bC-Xf-vdC.nib	16/10/25 下午8:41	2 KB	1 KB	Interface Builder NIB Document	
Info.plist	16/10/25 下午8:41	258 B	149 B	Property List	
UIViewController-BYZ-38-t0r.nib	16/10/25 下午8:41	916 B	603 B	Interface Builder NIB Document	
embedded.mobileprovision	16/10/25 下午8:41	9 KB	5 KB	Developer Provisioning Profile	
IMG_2140.JPG	16/10/25 下午8:41	1.9 MB	1.9 MB	预览 文稿	
IMG_2141.JPG	16/10/25 下午8:41	2.4 MB	2.4 MB	预览 文稿	
IMG_2142.JPG	16/10/25 下午8:41	1.6 MB	1.6 MB	预览 文稿	
IMG_2143.JPG	16/10/25 下午8:41	1.6 MB	1.6 MB	预览 文稿	
IMG_2144.JPG	16/10/25 下午8:41	2.1 MB	2.1 MB	预览 文稿	
IMG_2145.JPG	16/10/25 下午8:41	2.0 MB	2.0 MB	预览 文稿	
Info.plist	16/10/25 下午8:41	1 KB	696 B	Property List	
InstallPackageDemo	16/10/25 下午8:41	210 KB	52 KB	文稿	
PkgInfo	16/10/25 下午8:41	8 B	8 B	文稿	

结论1：JPG资源图片的压缩比很小，每减少一张图片，就实实在在的减少了ipa的大小。

下面我们进一步使用ImageOptim对图片进行压无损缩优化（如下图）。看能否优化下安装包大小。

ZIP		Extract with preset			
Name	Date Modified	Size	Packed	Kind	
▼ InstallPackageDemo.app	16/10/25 下午8:41	11.9 MB	11.7 MB	应用程序	
▼ _CodeSignature	16/10/25 下午8:41	7 KB	2 KB	文件夹	
CodeResources	16/10/25 下午8:41	7 KB	2 KB	文稿	
archived-expanded-entitlements.xcent	16/10/25 下午8:41	390 B	246 B	文稿	
▼ Base.lproj	16/10/25 下午8:41	6 KB	4 KB	文件夹	
▼ LaunchScreen.storyboardc	16/10/25 下午8:41	3 KB	2 KB	Interface Builder Compiled Storyboard Document	
01J-lp-oVM-view-Ze5-6b-2t3.nib	16/10/25 下午8:41	2 KB	1 KB	Interface Builder NIB Document	
Info.plist	16/10/25 下午8:41	258 B	149 B	Property List	
UIViewController-01J-lp-oVM.nib	16/10/25 下午8:41	832 B	552 B	Interface Builder NIB Document	
▼ Main.storyboardc	16/10/25 下午8:41	3 KB	2 KB	Interface Builder Compiled Storyboard Document	
BYZ-38-t0r-view-8bC-Xf-vdC.nib	16/10/25 下午8:41	2 KB	1 KB	Interface Builder NIB Document	
Info.plist	16/10/25 下午8:41	258 B	149 B	Property List	
UIViewController-BYZ-38-t0r.nib	16/10/25 下午8:41	916 B	603 B	Interface Builder NIB Document	
embedded.mobileprovision	16/10/25 下午8:41	9 KB	5 KB	Developer Provisioning Profile	
IMG_2140.JPG	16/10/25 下午8:41	1.9 MB	1.9 MB	预览 文稿	
IMG_2141.JPG	16/10/25 下午8:41	2.4 MB	2.4 MB	预览 文稿	
IMG_2142.JPG	16/10/25 下午8:41	1.6 MB	1.6 MB	预览 文稿	
IMG_2143.JPG	16/10/25 下午8:41	1.6 MB	1.6 MB	预览 文稿	
IMG_2144.JPG	16/10/25 下午8:41	2.1 MB	2.1 MB	预览 文稿	
IMG_2145.JPG	16/10/25 下午8:41	2.0 MB	2.0 MB	预览 文稿	
Info.plist	16/10/25 下午8:41	1 KB	696 B	Property List	
InstallPackageDemo	16/10/25 下午8:41	210 KB	52 KB	文稿	
PkgInfo	16/10/25 下午8:41	8 B	8 B	文稿	

压缩后，总文件大小为：屏幕快照 2016-10-25 下午8.58.24.png，优化掉了1MB的大小。我们然后进行打包操作，最终的安装包确实也小了0.8MB，从11.6MB变成了10.8MB，。还是有优化的，如下图所示

Images

10.7 MB, 6 项

上次修改时间 201



InstallPackageDemo.ipa

iOS App – 10.8 MB

创建时间 今天 下午8:59

此时我们看 xcode 里的工程配置，`COMPRESSPNGFILES` 是 YES 的，有一些说法是这个变量的设置和 `ImageOptim` 冲突，这里看起来不是如此。

结论2：ImageOptim有时候还是确实能优化资源大小，进而减少安装包大小的。

是否因此可以完全确定ImageOptim的优化能力，我觉得看情况而定，上面的几种图片都是我iPhone手机里的相片导出的。是JPG的格式。

我们再对PNG做一些测试，找一些资源图片放到工程中（我就不截图了，直截大小）：

▼ User-Defined

Setting

InstallPackageDemo

COMPRESS_PNG_FILES

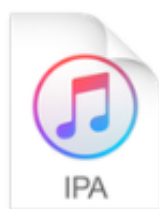
YES

打包后的大小是:

Images

3.3 MB, 20 项

系统帮我们优化掉1.1MB。同样我们对图片进行一轮无损压缩优化，经过ImageOptim优化后效果：



InstallPackageDemo.ipa

iOS App – 2.2 MB

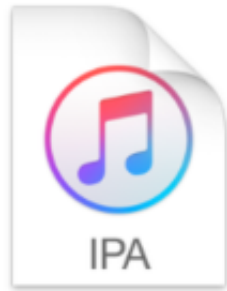
文件	保存	大小
✓ 11	60.3%	44,368
✓ 12	60.0%	45,027
✓ 13	37.8%	408,086
✓ 14	55.0%	34,328
✓ 15	43.5%	204,458
✓ 16	80.1%	10,184
✓ 17	83.0%	10,044
✓ 18	50.1%	61,747
✓ 19	45.8%	116,675
✓ 20	53.3%	53,491
+ 节约了1.7MB (总共3.3MB)。平均每个文件 55.9% (最多 83%)		
再次		

我们进行打包，得到的安装包的大小是:还是2.2MB（特意将系统优化关闭）：

Images

1.6 MB, 20 项

上次修改时间 2016年10月25日 下午9:33:22



InstallPackageDemo.ipa

iOS App – 2.2 MB

结论3： 对于我找的这几个png图片，ImageOptim的优化没有起到作用。

在我们项目中也对所有资源图片使用ImageOptim进行了优化，20多MB的资源最后优化掉1MB左右， 这里怀疑ImageOptim对PNG的优化能力一般。而且如果能优化的PNG资源，系统默认可以进行一些优化。

2.文档资源

 Model 10.mom	16/8/11 下午3:27	27 KB	16 KB	C
 Model 11.mom	16/8/11 下午3:27	28 KB	16 KB	C
 Model 12.mom	16/8/11 下午3:27	28 KB	16 KB	C
 Model 13.mom	16/8/11 下午3:27	32 KB	18 KB	C
 Model 14.mom	16/8/11 下午3:27	37 KB	21 KB	C
 Model 15.mom	16/8/11 下午3:27	39 KB	22 KB	C
 Model 16.mom	16/8/11 下午3:27	41 KB	23 KB	C
 Model 17.mom	16/8/11 下午3:27	44 KB	25 KB	C
 Model 18.mom	16/8/11 下午3:27	46 KB	26 KB	C

从上面来看，文档有一定的压缩比，大概40%，也就是如果工程里有40MB的文档，体验到压缩包里大概是 $40 \times 0.6 = 24$ MB。

3.二进制安装包

 News	16/8/25 上午10:42	125 MB	41.9 MB	文稿
 News_storyboard	16/8/25 上午10:42	5 KB	3 KB	Interface R

二进制代码的压缩率是最高的。

上面都是研究不同的资源对最后安装包大小的影响情况，现在有了一些理论依据，我们就可以开始对安装包进行优化工作了。

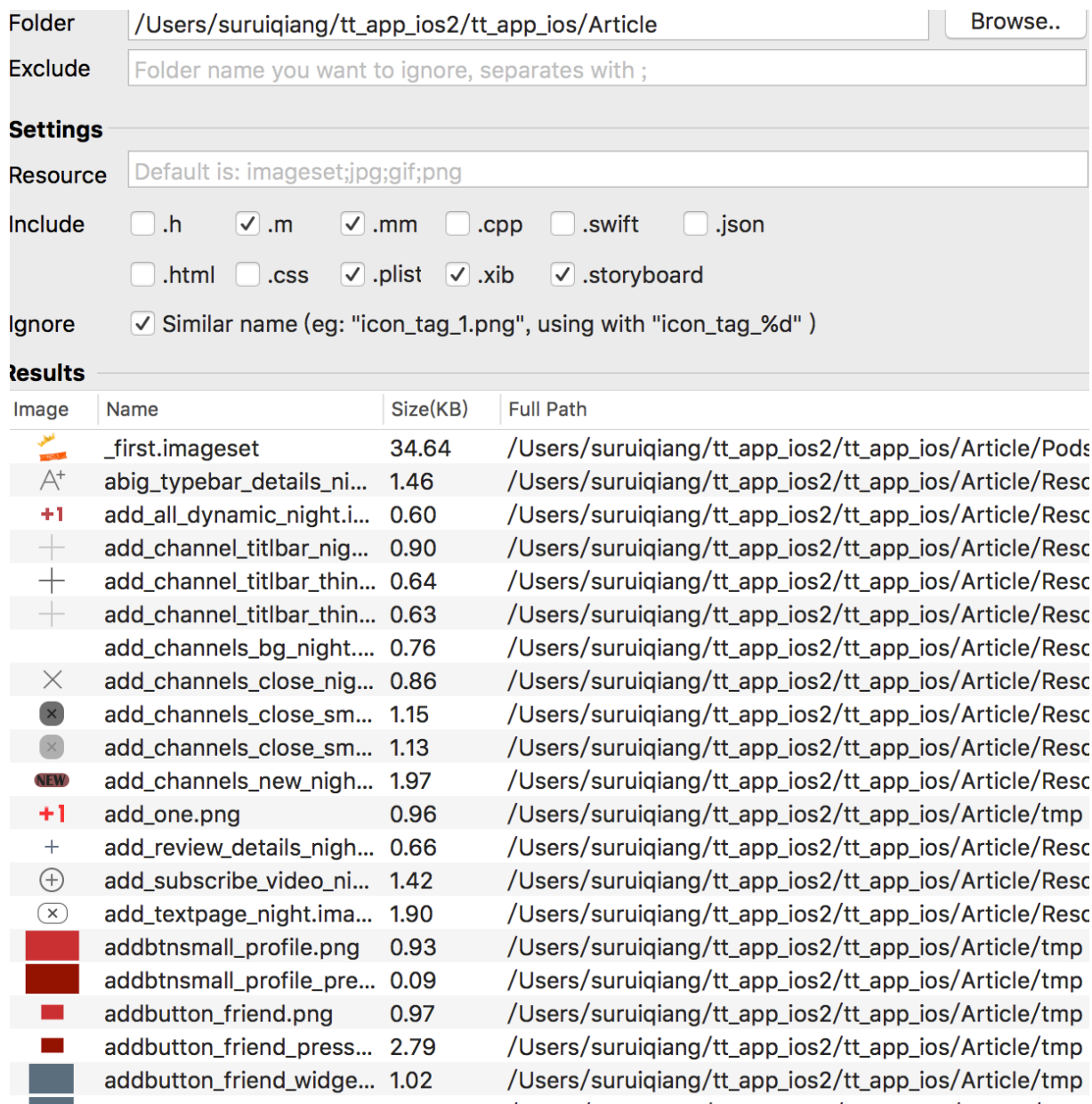
实践

1.如何优化图片

使用一些工具来检测unused 文件。比较推荐的是github上的一份开源代码：<https://github.com/tinymind/LSUnusedResource> 编译运行看到的页面是这样的：

[illegible]

指定搜索路径， 第二行 (EXclude) 指定哪些文件夹路径不被扫描。



然后这些资源可以清除掉了。但是存在着图片被误删除的可能性，譬如代码中使用图片的方式是：`[UIImage imageNamed:[NSString stringWithFormat:@"icon_%d.png",index]];` 这种情况下，图片可能被误删，所以删除的时候不妨10张一组的进行，用眼睛过滤一遍

2.删除完图片，是否还可以对已有的图片做优化呢

现在网上有非常多的关于 `ImageOptim` 对资源图片进行无损压缩的方式，在文章开头部分我们也对 `ImageOptim` 的优化能力进行了一些验证，通过上面的实验，我觉得结论不能确定，但值得一试。

2.文档资源的优化

文档资源主要是排查：

1. 是否有不必要的文档资源，如果过期的旧版本所需要的文档资源 清理即可。
2. 优化文档资源大小，主要是优化精简文档内容。

3.二进制包优化

二进制包是由各种代码文件，静态库 动态库 经过编译后生成的可执行文件。以头条二进制包125MB为例， 他是如何组成的？

Offset	Data	Description	Value
00000000	BEBAFECA	Magic Number	FAT_CIGAM
00000004	02000000	Number of Architecture	2
00000008	0C000000	CPU Type	CPU_TYPE_ARM
0000000C	09000000	CPU SubType	CPU_SUBTYPE_ARM_V7
00000010	00400000	Offset	16384
00000014	30D67803	Size	58250800
00000018	0E000000	Align	16384
0000001C	0C000001	CPU Type	CPU_TYPE_ARM64
00000020	00000000	CPU SubType	CPU_SUBTYPE_ARM64...
00000024	00407903	Offset	58277888
00000028	707CF603	Size	66485360
0000002C	0E000000	Align	16384

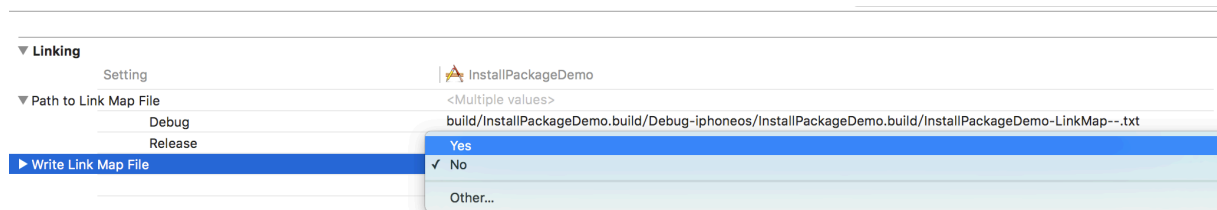
上图可以看到armv7 占可执行文件的58MB。 arm64占可执行文件的66MB。 加起来=125MB。 进一步分析

Fat Binary	pFile	Data LO	Data HI	Value
Fat Header	00000020	01 00 20 40 F9 F1 74 C0	4E F0 3B 00 C0 F2 FD 10	.n F...N.A ...
▼ Executable (ARM_V7)	00010830	78 44 01 68 20 46 F9 F1	6C C6 10 F0 FF 0F 44 D0	xD.h F...l...D.
Mach Header	00010840	4E F6 68 60 C0 F2 FD 10	78 44 01 68 20 46 F9 F1	N.h'...xD.h F..
► Load Commands	00010850	60 C6 10 F0 FF 0F 38 D0	4E F6 24 60 C0 F2 FD 10	`.....8.N.\$'....
Section (TEXT, _text)	00010860	4D F6 F2 52 C0 F2 FF 12	78 44 7A 44 01 68 10 68	M..R....xDzD.h.h
Section (TEXT, _picsymbolstub4)	00010870	F9 F1 4E C6 3F 46 F9 F1	54 C6 05 46 4E F6 30 60	..N.?F...T...FN.0'
Section (TEXT, _stub_helper)	00010880	C0 F2 FD 10 78 44 01 68	20 46 F9 F1 42 C6 3F 46xD.h F..B.?F
Section (TEXT, _cstring)	00010890	F9 F1 46 C6 06 46 4E F6	1A C0 C0 F2 FD 10 78 44	..F...FN...xD
Section (TEXT, _objc_methname)	000108A0	01 68 20 46 F9 F1 34 C6	03 46 4E F6 08 60 C0 F2	.h F..4...FN...'..
Section (TEXT, _cstring)	000108B0	FD 10 32 46 78 44 01 68	28 46 F9 F1 2A C6 30 46	..2FxD.h(F...0F
Section (TEXT, _objc_classname)	000108C0	F9 F1 2C C6 28 46 F9 F1	2A C6 02 B0 F0 BD 80 B5	... (F...*.....
Section (TEXT, _objc_methtype)	000108D0	6F 46 82 B0 41 F2 D4 11	C0 F2 00 21 4E F6 DA 53	oF...A.....!N..S
Section (TEXT, _gcc_except_tab)	000108E0	C0 F2 FD 13 79 44 7B 44	00 90 08 68 19 68 01 90yD{D...h.h..
Section (TEXT, _const)	000108F0	68 46 F9 F1 10 C6 02 B0	80 BD 80 B5 6F 46 82 B0	hF.....oF...
Section (TEXT, _swift2_proto)	00010900	41 F2 A8 11 C0 F2 00 21	4E F6 B2 53 C0 F2 FD 13	A.....!N..S....
Section (DATA, __nl_symbol_ptr)	00010910	79 44 7B 44 00 90 08 68	19 68 01 90 68 46 F9 F1	yD{D...h.h..hF..
Section (DATA, __la_symbol_ptr)	00010920	FA C5 02 B0 80 BD F0 B5	03 AF 82 B0 F0 46 41 F2FA.....
Section (DATA, __mod_init_func)	00010930	7A 10 C0 F2 00 20 4E F6	88 51 C0 F2 FD 11 78 44	z.... N..Q....xD
Section (DATA, _const)	00010940	79 44 00 94 00 68 09 68	01 90 68 46 F9 F1 E2 C5	yD...h.h..hF....
Section (DATA, _cfstring)	00010950	4E F6 52 50 00 22 C0 F2	FD 10 78 44 01 68 20 46	N.RP..."xD.h F
Section (DATA, _objc_classlist)	00010960	F9 F1 D6 C5 4E F6 1C 50	C0 F2 FD 10 78 44 01 68	...N..P....xD.h
Section (DATA, _objc_nclclist)	00010970	20 46 F9 F1 CE C5 10 F0	FF 0F 38 D0 4E F6 00 50	F.....8.N..P
Section (DATA, _objc_catlist)	00010980	C0 F2 FD 10 4D F6 CE 42	C0 F2 FF 12 78 44 7A 44	...M..B....xDzD
Section (DATA, _objc_nlcattlist)	00010990	01 68 10 68 F9 F1 BC C5	3F 46 F9 F1 C2 C5 05 46	.h.h....?F....F
Section (DATA, _objc_protolist)	000109A0	4E F6 0C 50 C0 F2 FD 10	78 44 01 68 20 46 F9 F1	N..P....xD.h F..
Section (DATA, _objc_imageinfo)	000109B0	B0 C5 3F 46 F9 F1 B4 C5	06 46 4E F6 F6 40 C0 F2	..?F.....FN...@..
Section (DATA, _objc_const)	000109C0	FD 10 78 44 01 68 20 46	F9 F1 A2 C5 03 46 4E F6	...xD.h F.....FN.
Section (DATA, _objc_selrefs)	000109D0	F4 40 C0 F2 FD 10 32 46	78 44 01 68 28 46 F9 F1	..@...2FxD.h(F...
Section (DATA, _objc_protorefs)	000109E0	98 C5 30 46 F9 F1 9A C5	28 46 F9 F1 98 C5 02 B0	..0F....(F.....
Section (DATA, _objc_classrefs)	000109F0	F0 BD F0 B5 03 AF 2D E9	00 05 87 B0 6C 46 6F F3LFo.....
Section (DATA, _objc_superrefs)	00010A00	02 04 A5 46 04 46 43 F6	8E 30 C0 F2 00 20 78 44	...F.FC..0... xD
Section (DATA, _objc_ivar)	00010A10	05 68 60 59 00 28 40 F0	C8 80 4E F6 52 10 C0 F2	.h`Y.(@...N.R...
Section (DATA, _objc_data)	00010A20	FD 10 4D F6 F0 32 C0 F2	FF 12 78 44 7A 44 01 68	..M..2....xDzD.h
Section (DATA, _data)	00010A30	10 68 F9 F1 6E C5 80 46	4E F6 C0 30 C0 F2 FD 10	.h..n...FN..0....
► Function Starts	00010A40	78 44 01 68 20 46 F9 F1	64 C5 3F 46 F9 F1 68 C5	xD.h F..d.?F..h.
► Data in Code Entries	00010A50	82 46 BA F1 00 0F 0A D0	4E F6 6E 40 51 46 C0 F2	.F.....N.n@QF..
► Symbol Table	00010A60	FD 10 78 44 02 68 02 A8	F9 F1 76 C5 04 E0 C0 EF	...xD.h....V.....
► Dynamic Symbol Table	00010A70	50 00 02 A8 40 F9 CF 0A	4E F6 EE 01 C0 F2 FD 11	P...@...N.....

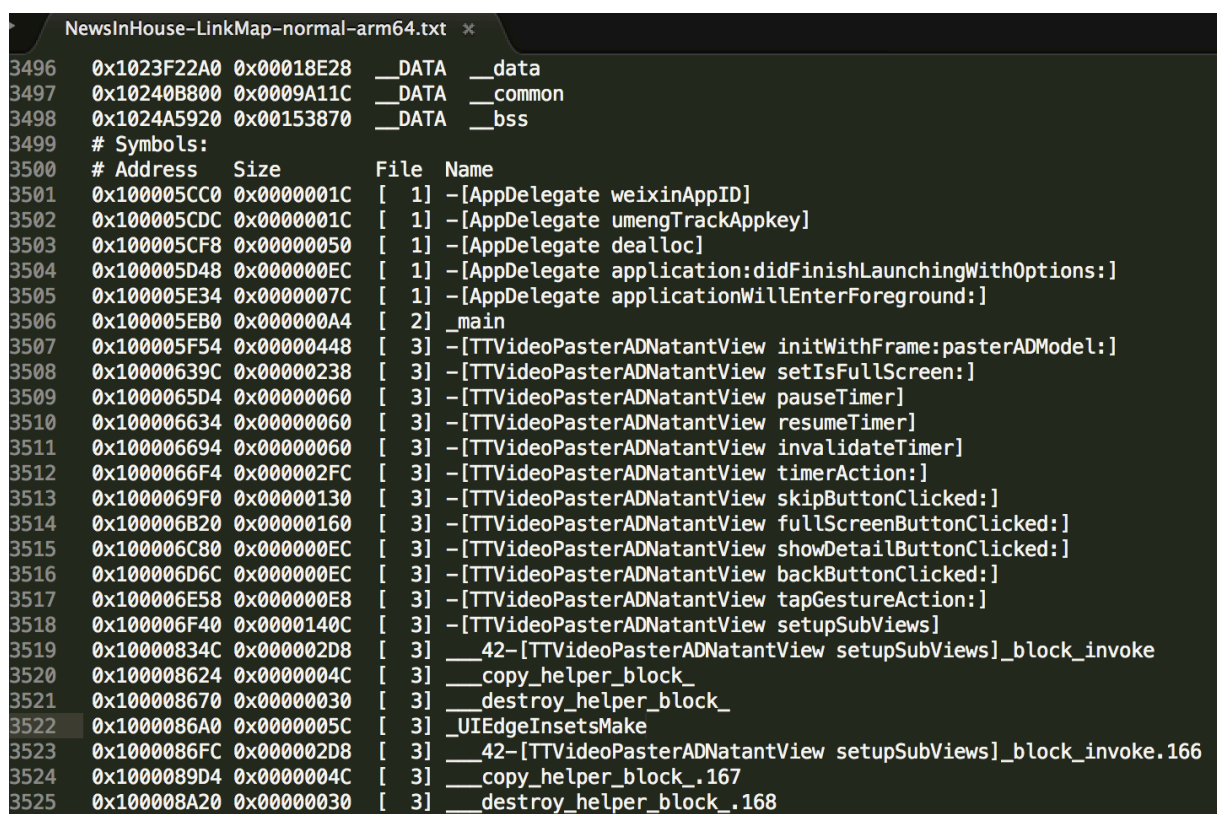
通过右侧的pfile偏移可以大概算出每个段的大小，但不直观， 我们可以通过开启

一些编译选项，生成可执行文件结构，然后借助一些工具生成更加直观的

- XCode开启编译选项Write Link Map File XCode -> target -> Build Settings -> 搜map -> 把Write Link Map File选项设为yes，并指定好linkMap的存储位置：



- 编译后到编译目录里找到该txt文件，文件名和路径就是上述的Path to Link Map File。~/Library/Developer/Xcode/DerivedData/XXX-eumsvrzbvgfobfbsoqokmjprvuh/Build/Intermediates/XXX.build/Debug-iphoneos/XXX.build/。这个LinkMap里展示了整个可执行文件的全貌，列出了编译后的每一个.o目标文件的信息（包括静态链接库.a里的），以及每一个目标文件的代码段，数据段存储详情



如上图，weinAppID这个函数占得内存大小为 $1 \times 16 + 13 = 29$ 字节。这样看下去也挺麻烦的，找个工具归类一下。

- 归类，去<https://github.com/huanxsd/LinkMap> 下载这个mac工程 然后运行。

/Users/suruiqiang/Desktop/NewsInHouse-LinkMap-normal-arm64.txt

选择文件

模块关键字

☐ 分组解析

开始

输出文件

使用方式：

- 1.在XCode中开启编译选项Write Link Map File

XCode -> Project -> Build Settings -> 把Write Link Map File选项设为yes, 并指定好linkMap的存储位置

- 2.工程编译完成后, 在编译目录里找到Link Map文件 (txt类型)

默认的文件地址: ~/Library/Developer/Xcode/DerivedData/XXX-xxxxxxxxxxxxx/Build/Intermediates/

XXX.build/Debug-iphoneos/XXX.build/

- 3.回到本应用, 点击“选择文件”, 打开Link Map文件

- 4.点击“开始”, 解析Link Map文件

- 5.点击“输出文件”, 得到解析后的Link Map文件

6. * 输入目标文件的关键字(例如: libIM), 然后点击“开始”。实现搜索功能

7. * 勾选“分组解析”, 然后点击“开始”。实现对不同库的目标文件进行分组

库大小 库名称

4.92M	MediaPlayer
1.82M	Live.a
1.47M	lapAPI
855.40K	_5.6.1.a
753.50K	Widget.a
674.34K	.a
654.66K	n
648.78K	veLib.a
639.91K	ocolBuffers.a
461.07K	veSDK.a
455.21K	aseLib.default-extension-only_no-arc.a
415.85K	DK
409.75K	erSDK
402.43K	tOpenAPI
395.20K	Model.o
356.14K	ClickLibrary.a
311.56K	etworking.a
292.46K	tics
248.28K	adAvatarView.o
234.90K	
217.36K	earchKit
216.94K	bp.a
197.58K	hatSDK.a
193.37K	MixedListBaseView.o
178.88K	ebImage.default-WebP.a
178.73K	oginSDK
169.28K	nAdsSDK
166.25K	tch.a
159.03K	epporter
152.84K	DetailView.o

譬如内部播放器sdk MediaPlayer在arm64架构下大小为4.92MB， armv7也可以分析另一份armv7 linkmap文件大概4.5MB 二者加起来就是在二进制占据的总大小—10MB左右。通过对上面的文件进行分析，就知道每个类在最终的可执行文件中占据的大小。然后有针对性的进行优化就可以了。

4.编译选项优化：

如果项目是很早之前（xcode4， 5）建立的，迭代到现在 的确可以检查一下有利于减少安装包的编译选项：

1. Optimization Level 使用Fastest, Smallest
2. 该选项对安装包大小影响几无，但可以提高app的性能。参考wwdc 2013-Session408 Optimize Your Code Using LLVM
3. Strip Linked Product 设置为YES 需要注意的是Strip Linked Product也受到Deployment Postprocessing设置选项的影响。在Build Settings中，我们可以看到， Strip Linked Product是在Deployment这栏中的，而Deployment Postprocessing相当于是Deployment的总开关。记得把Deployment Postprocessing也设置为YES， 该选项对安装包大小的影响非常大， 以头条客户端为例， 如果不开启此设置， ipa大小是48MB， 上线后appstore上显示的大小是65MB， 我们开启了此配置后， ipa大小变成40MB， appstore上显示45MB。优化效果还是非常明显的。PS： Deployment Postprocessing这个配置项如果使用xcode打包， xcode会默认把这个变量置为YES， 如果使用脚本打包，记得设置。
4. Symbols Hidden by Default设置为YES
5. Make Strings Read-Only 设置为YES

5. 可能无效的办法：

将Enable C++ Exceptions和Enable Objective-C Exceptions设为NO， 去掉异常支持； 如果你的项目比较大， 有很多try catch， 想去掉这些异常可能是一个比较大的工作量， 我在头条项目里尝试去掉了所有异常， 打包测试， 安装包大小没有变化， 因为只是一个项目的测试， 我只能比较怀疑去掉异常对最终安装包大小的优化能力。

6.一些额外的建议

iOS的项目， 在安装包没有大到一定程度之前， 研发和产品一般都关注较少， 我们在平时的开发中也会有一些不好的习惯， 在这里枚举一下：

1. 不要为了一片树叶 引入一片森林。有时候你只想接入一个base64编码的函数，结果接入了一个有几十个类文件的util库，除了你用到的这个函数，其余的可能再也不会有人用到， 另一个研发为了另一个RSA加密需求，结果又引入了另一个extensions库 。那些类文件都是成本，类文件，函数，甚至不同长度的函数名字都对最终的安装包大小产生影响。积少成多， 安装包会越来越大。而且代码量越大，app启动时候DYLD链接的工作量越大，启动耗时也会变长。
2. 对于要接入到app的资源文件，要check一下大小，一个100*100大小的图片如果几十MB 肯定设计师在切图的时候有些问题，打包后也要check一遍。对于集体打包到Assert.car里的文件，可以找工具解开看一下。我们遇到过打了一次包后 发现安装包大了几十MB，经过分析，发现是有一张图片意外的大，找人优化后 变成几百k。
3. 注意平时的开发习惯，废弃模块及早清理。
4. 同质的开源库（譬如AFnetworking vs ASIHttpRequest），只接入一种。
5. 建立预警机制，一般上线后，都是脚本打包，除了正常的生成ipa包之外，也要生成分析文件， 列出相对上一次上线包大了多少，类文件增加了多少。这样的机制也有助于防止安装包悄无声息的变成巨物。