编译速度优化

1. 测试环境

- M1 Pro + 32G
- Xcode 14.2 (14C18)

2. 获取构建用时

2.1 非 `Running` 状态下可以在活动查看器中显示出构建时间

> `defaults write com.apple.dt.Xcode ShowBuildOperationDuration YES`

2.2 查看Xcode构建时间摘要

2.2.1 通过命令行编译

> `xcodebuild -buildWithTimingSummary`

2.2.2 通过Xcode编译

> `Product->Perform Action->Build With Timing Summary`
>
> `⌘9` 在编译日志中查看 `Editor > Open Timeline`

Xcode编译时间线

> 11个线程，其中：
>
> - 预编译7.4s
> - 三方库 27s
> - copy非 `xcasset` 中保存的静态资源文件（png、MP4、xib等）4.5s

- `xcasset` 压缩 147s
- 其余源码文件，由于swift存在类型推断、模块化处理等特性所以编译时间相对于OC会显著变长，但一般无需考虑优化

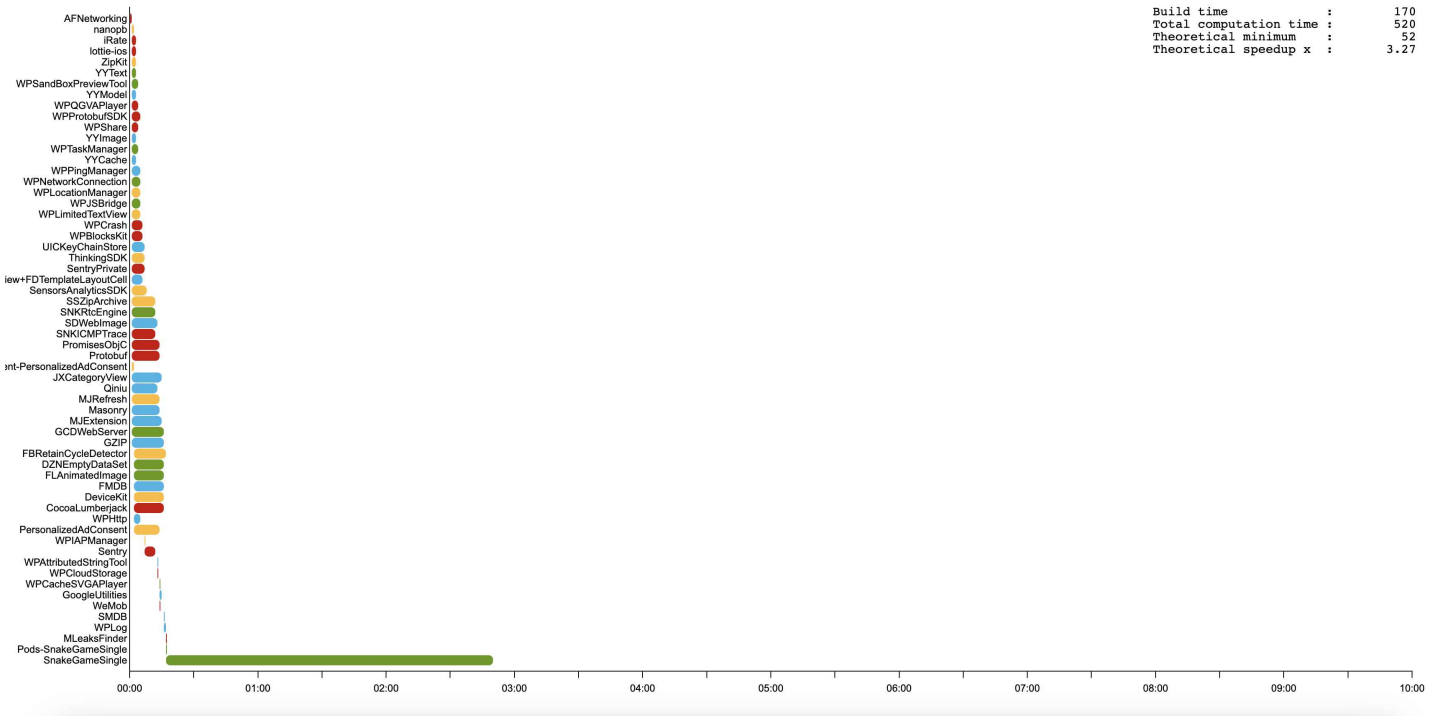
2.3 xcode-build-times-rendering

2.3.1 安装

- `gem install xcode-build-times`
- 工程目录 `xcode-build-time install .`

2.3.2 查看

- 编译项目后的分析产物路径 `/xcode-build-times-chart/gantt.html`



> 太过于笼统，忽略掉此方案

2.4 XCLogParser

2.4.1 安装

```
1 git clone https://github.com/spotify/XCLogParser
2 rake build
3 rake install
```

2.4.2 添加Swift编译项

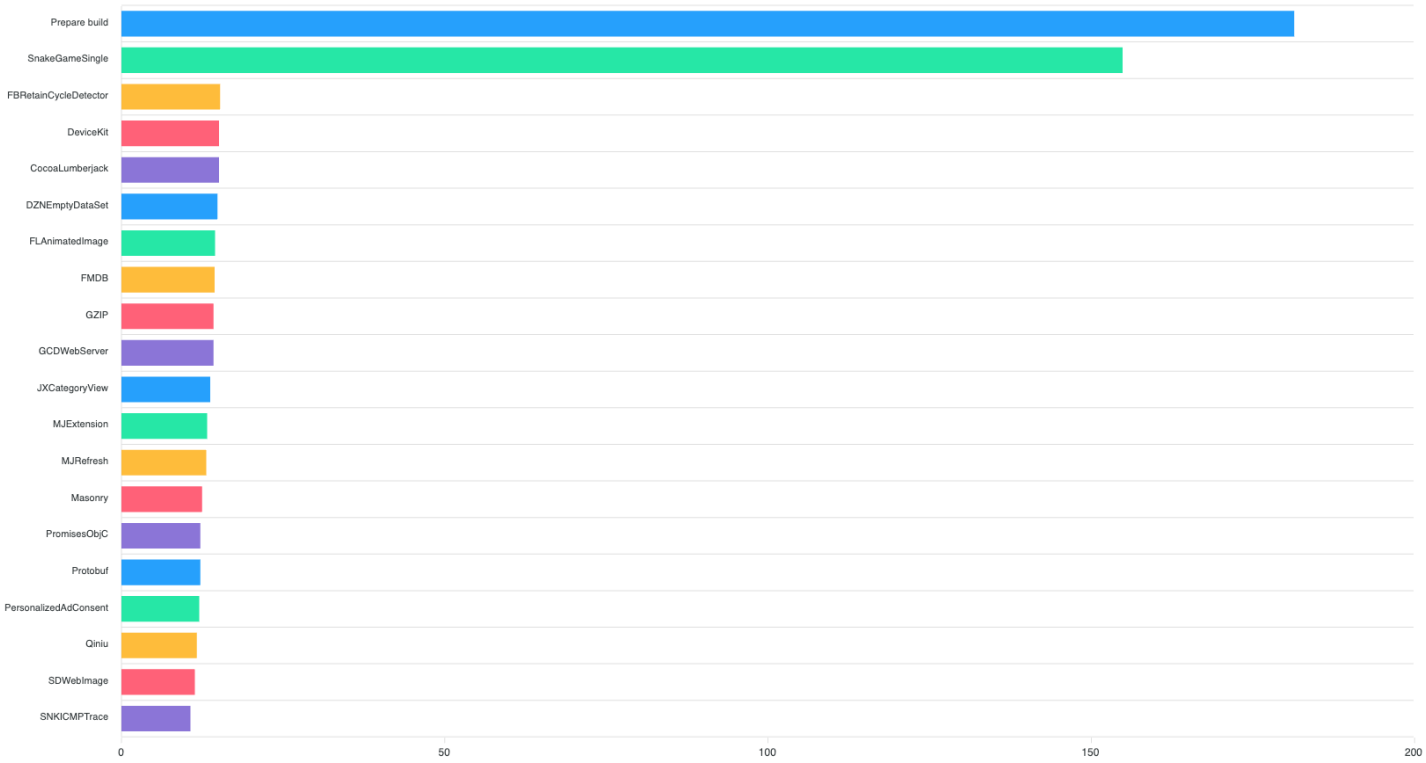
```
// Swift function compilation times
-Xfrontend -debug-time-function-bodies
// Swiftc type checks times
-Xfrontend -debug-time-expression-type-checking
```

2.4.3 解析

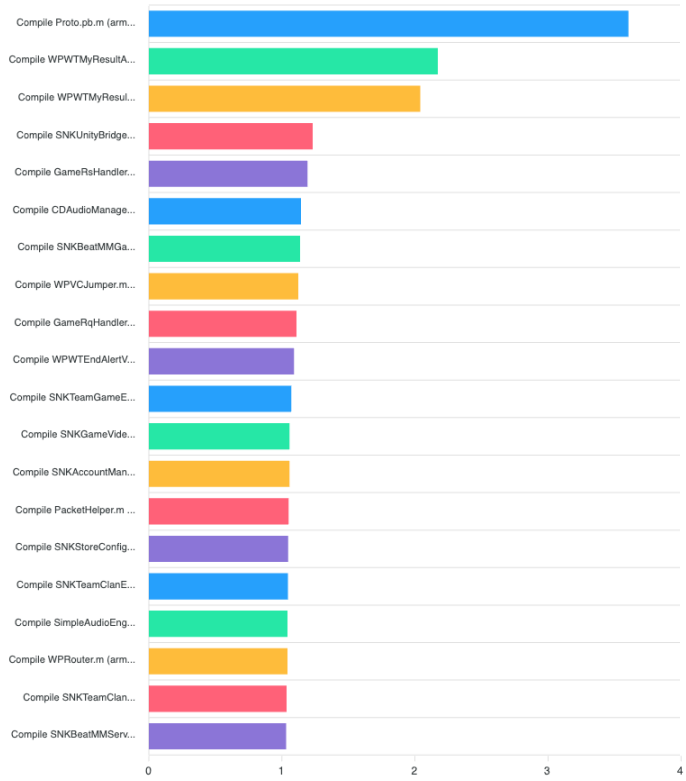
> 删除`DerivedData`缓存后执行一次全量编译，最后在命令行执行：

`xclogparser parse --project SnakeGameSingle --reporter html`

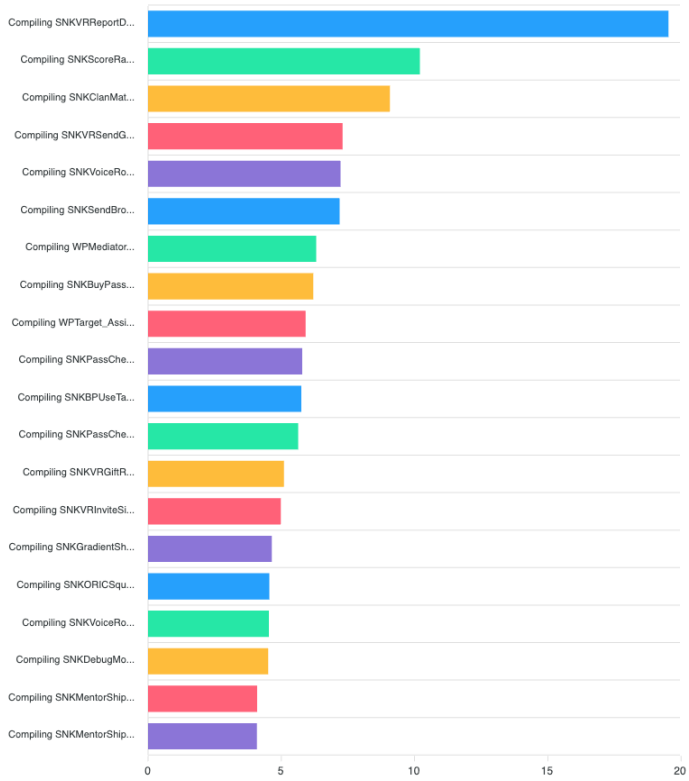
2.4.4 查看



Target编译耗时



OC/C编译耗时



Swift编译耗时

2.5 总结

- 使用Xcode自带的分析工具`RBT`，初步判断是在`Compile asset catalogs`生成`Asset.car`的过程耗时过长，开发过程中不需要频繁的对资源文件做压缩操作
- 三方库大多源码引入，导致项目全量编译时需要对每个target都进行编译。

3. 解决方案

3.1 编译流程跳过Compile Asset Catalogs

> Debug环境下直接将Assets中的资源Copy到包里，进行一次全量预编译，跳过将Assets资源编译压缩为Asset.car文件的过程。

3.1.1 脚本

3.1.1.1 校验并修正Assets和资源名称

```
import os
import re

# 定义排除列表文件（每行一个目录）
exclude_list = "exclude_path.txt"
# 非正常匹配的文件列表（需要手动更改）
unexpected_path = "unexpected_path.txt"
```

```
8
9  # 读取排除列表并存入一个集合
10 with open(exclude_list, "r") as f:
11     exclude_set = set([line.strip() for line in f])
12
13 # 在当前目录及其子目录中查找所有PNG文件，排除在排除列表中的目录
14 exceptFileList = []
15 for root, dirs, files in os.walk("/Users/jocer/Developer/SnakeGame_New",
   topdown=True):
16     # 从搜索中移除排除的目录
17     dirs[:] = [d for d in dirs if os.path.join(root, d) not in exclude_set]
18
19     for filename in files:
20         if not filename.endswith(".png"):
21             continue
22         # 检查父目录是否在排除列表中
23         if os.path.basename(root) in exclude_set:
24             continue # 如果目录在排除列表中，则跳过此文件
25
26         fPath = os.path.join(root, filename)
27         skip = False
28         for line in exclude_set:
29             if line in fPath:
30                 skip = True
31                 break;
32         # print("!!!skip path:" + fPath)
33         if "xcasset" not in fPath:
34             skip = True
35         if skip :
36             continue
37
38
39         # 获取父目录的名称并移除 .xcasset 后缀
40         parent = os.path.basename(root).replace(".imageset", "")
41
42         # 使用正则表达式提取PNG文件的前缀（不包括 @2x/@3x 后缀）
43         match = re.match(r"^(.+)/([^/@]+)@[0-9]+x\.png$", filename)
44         if match:
45             prefix = match.group(2)
46         else:
47             prefix = os.path.splitext(filename)[0]
48
49         # 从前缀中移除 @2x/@3x 后缀（如果存在）
50         # prefix = prefix.replace("@2x", "").replace("@3x", "")
51
52         prefix2x = parent + "@2x"
53         prefix3x = parent + "@3x"
```

```
        # 比较PNG文件的前缀与父目录的名称
        if prefix != prefix2x and prefix != prefix3x and prefix != parent:
            # 使用正确的前缀生成新文件名
            new_filename = ""
            newPrefix = ""
            if "@2x" in prefix:
                newPrefix = prefix2x
                new_filename = os.path.join(root, prefix2x + ".png")
            if "@3x" in prefix:
                newPrefix = prefix3x
                new_filename = os.path.join(root, prefix3x + ".png")
            if newPrefix == "" :
                exceptFileList.append(os.path.join(root, filename))
                break
            # 重命名文件
            os.rename(os.path.join(root, filename), new_filename)
            print(f"已重命名\n{os.path.join(root, filename)}\n{new_filename}")
            contentJsonPath = os.path.join(root, "Contents.json")
            retLines = []
            with open(contentJsonPath, 'r') as contentJson:
                lines = contentJson.readlines()
            for line in lines:
                if prefix in line:
                    # print(line)
                    # print(newPrefix)
                    retLines.append(line.replace(prefix, newPrefix))
                else :
                    retLines.append(line)
            with open(contentJsonPath, 'w') as contentJson:
                contentJson.writelines(retLines)

with open(unexpected_path, "w") as f:
    for file in exceptFileList:
        f.write(file + "\n")
        print("例外文件，请手动修改这些文件: "+ file)
```

exclude_paths.txt

```
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/AppIcon_dev.appiconset/
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/AppIcon.appiconset/
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/OLGame/ol_team_match_bg.imageset
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/OLGame/ol_match_team_left_bg.imageset
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/EatClub/eat_club_head_finish.imageset
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/EatClub/eat_club_head.imageset
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/Common/common_merge_light_bg.imageset
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/Store/store_skin_preview.imageset
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/OLMatchTeam/ol_match_team_left_bg.imageset
```

过滤非标准名称路径

unexpected_path.txt

```
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/4.3.28/ol_game_message_btn_dark.imageset/ol_game_message_btn.png
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/Start/homeview_team_light.imageset/light.png
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/5.5.8/wedding_alert_select_btn_purple.imageset/wedding_alert_select_btn.png
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/5.5.8/wedding_alert_empty_icon.imageset/webdding_alert_empty_icon.png
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/5.5.8/wedding_alert_confirm1_btn.imageset/wedding_propose_alert_confirm_btn.png
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/5.5.8/wedding_alert_cancel1_btn.imageset/wedding_propose_alert_reject_btn.png
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/5.5.8/NewsList/proposal_background_img.imageset/proposal_background_img 1.png
/Users/jocer/Developer/SnakeGame_New/SnakeGameSingle/Assets.xcassets/WonTreasure/Team/teamGame_end_rank1.imageset/teamGame_end_rank1@1x.png
```

需要手动更改名称的资源

> 将Assets中的每个imageset名称和资源名进行统一，输出需要手动统一的文件列表（`exclude_paths.txt`），以便后续使用。

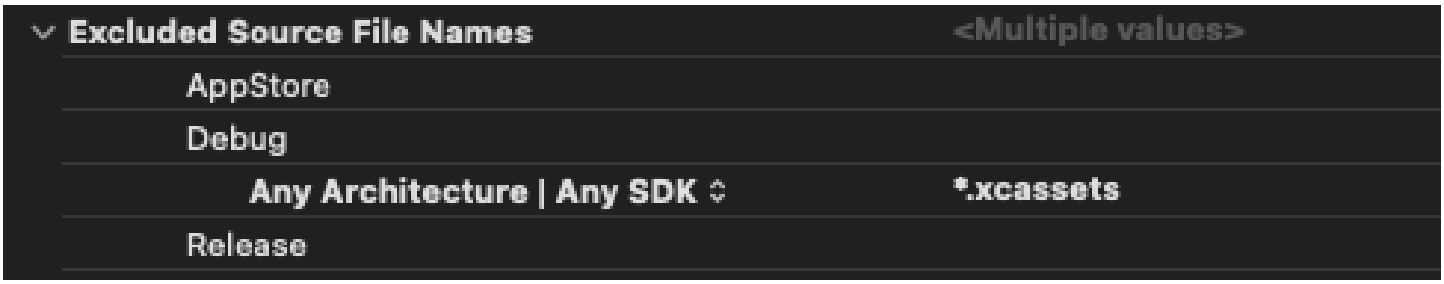
3.1.1.2 Build Phases添加Run Script

> 将Assets中经过`exclude_paths.txt`过滤后的资源copy到编译产物.app中，否则运行时无法正确读取对应的资源

```
# CopyAssetsToAppBundleUnderDebugEnv
#!/bin/bash

# 检查当前构建配置是否为 DEBUG
if [ "$CONFIGURATION" != "Debug" ]; then
  echo "This script is only intended for DEBUG configuration. Skipping..."
  exit 0
fi

echo "😜😜😜😜😜"

# 设置目标文件夹，这里使用了一些环境变量来构建目标路径
DESTINATION=$BUILT_PRODUCTS_DIR"/"$PRODUCT_NAME".app/"

# 使用 find 命令查找项目中所有的 PNG 文件，排除了指定的排除目录
find "$SRCROOT/$TARGETNAME/Assets.xcassets" -type f -name '*.png' \
  | grep -Fvf "$SRCROOT/exclude_paths.txt" \
  | xargs -I {} -P 4 cp {} $DESTINATION

# 退出脚本，表示成功完成
echo "🥳🥳🥳🥳🥳"
exit 0
```

3.1.2 配置Build Setting

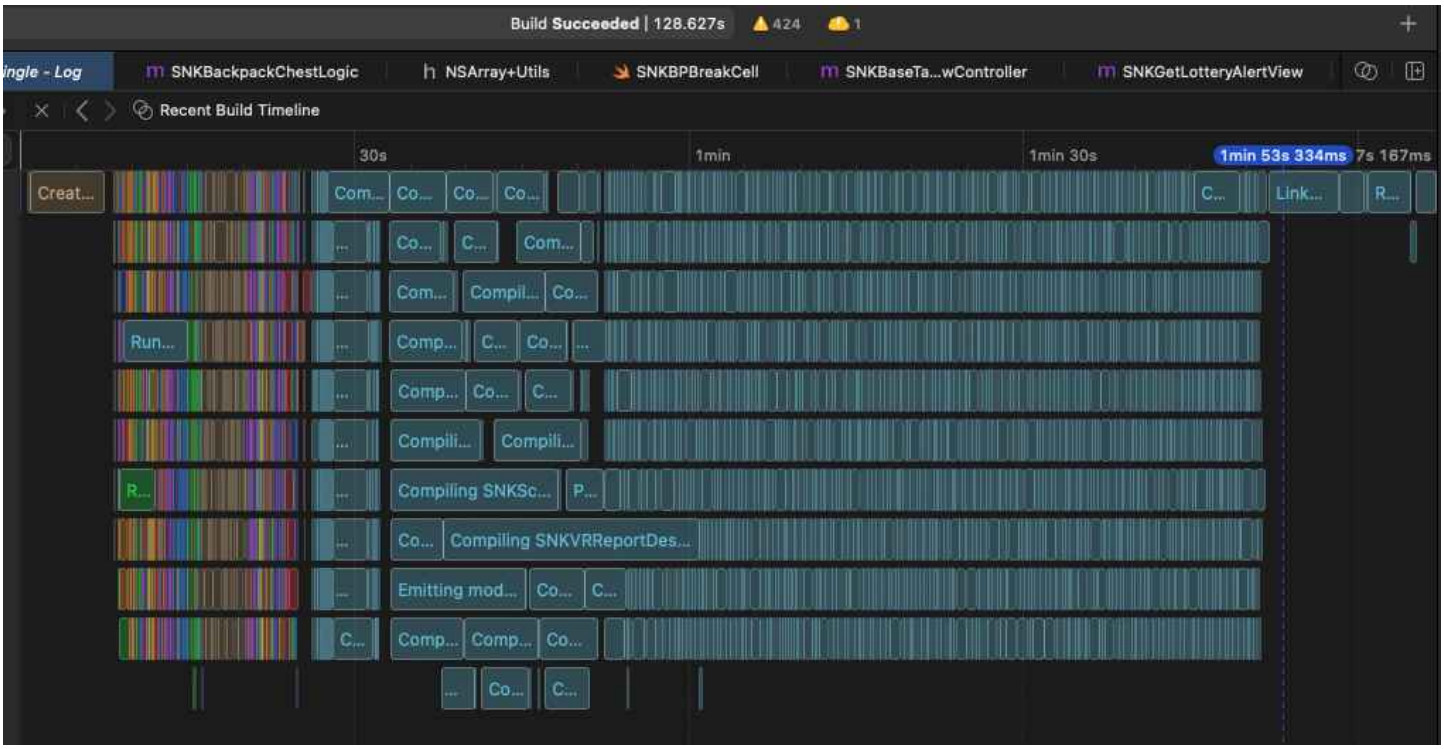
> 配置`*.xcassets`，编译过程忽略`Compile Asset Catalogs`



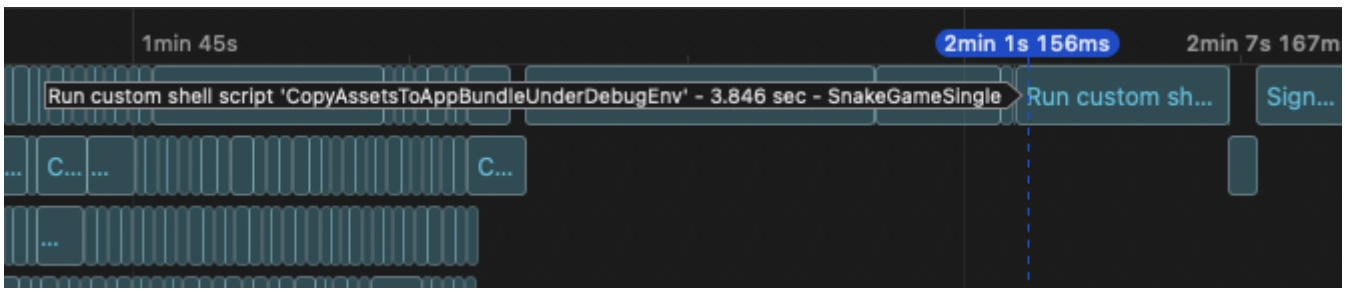
源文件排除xcassets

将特殊asset添加进编译进程，不知为何不起作用😂

3.1.3 效果对比



全量编译耗时128s



复制资源文件脚本耗时3.8s

> 做完以上操作之后进行一次全量编译，结果显示编译阶段不再有`Compile Asset Catalogs`。
>
> 优化前耗时：191s上下浮动，一般情况下浮动不大
>
> 优化后耗时：129s ~ 141s
>
> 总计节省时长62s ~ 50s之间（根据机器状态会上下浮动）

3.2 三方库二进制化

> 使用cocoapods插件`cocoapods-binary-bel`将podfile中的依赖项选择性的进行二进制化，预编译为binary，省去源码编译阶段而直接进行链接

3.2.1 安装

3.2.1.1 GemFile

```
# GemFile中添加依赖项
gem "cocoapods-binary", '0.5.4'
```

> 执行`bundle install`安装依赖项

3.2.1.2 Podfile

```
plugin 'cocoapods-binary-bel'

all_binary!

# 打开use_frameworks
use_frameworks!
pre_install do |installer|
  Pod::Installer::Xcode::TargetValidator.send(:define_method, :verify_no_static_framework_transitive_dependencies){}
end

target 'SnakeGameSingle' do

  pod 'AnyThinkiOS','6.2.36', :binary => false
  pod 'AnyThinkBaiduAdapter','6.2.36.1', :binary => false #5.310 - 5.313
  pod 'AnyThinkiOS/AnyThinkKuaiShouAdapter','6.2.36', :binary => false #3.3.47
  pod 'AnyThinkiOS/AnyThinkSigmobAdapter','6.2.36', :binary => false #4.9.1
  pod 'AnyThinkiOS/AnyThinkVungleAdapter','6.2.36', :binary => false # 6.12.1
```

```
  pod 'AnyThinkiOS/AnyThinkUnityAdsAdapter','6.2.36', :binary => false #4.7.1
  pod 'AnyThinkiOS/AnyThinkTTAdapter','6.2.36', :binary => false #5.4.0.5
  pod 'AnyThinkiOS/AnyThinkAdmobAdapter','6.2.36', :binary => false #10.6.0
  pod 'AnyThinkiOS/AnyThinkApplovinAdapter','6.2.36', :binary => false #11.9.0
  pod 'AnyThinkiOS/AnyThinkMintegralAdapter','6.2.36', :binary => false #7.3.9
  pod 'AnyThinkGDTAdapter','6.2.36.1', :binary => false #4.14.31 - 4.14.40


  pod 'MLeaksFinder',  :configuration => 'Debug', :binary => false
  pod 'FBRetainCycleDetector', :configuration => 'Debug', :binary => false

  pod 'WPShare', '2.3.1', :binary => false

  pod 'WPHttp', '0.5.5', :binary => false

end
```

3.2.2 使用

3.2.2.1 打开依赖项二进制化

> 需要关闭二进制化的组件

- MLeaksFinder/FBRetainCycleDetector，存在编译错误代码
- WPShare/WPHTTP，架构配置项缺失
- AnyThinkiOS.framework，AnyThinkiOS.framework/LICENSE，LICENSE缺失
- QIYU_iOS_SDK，libqysdk.a, libcrypto.a, and libevent.a静态库冲突

```
pod install
```

3.2.2.2 关闭依赖项二进制化

```
# 不需要改动Podfile文件
pod install --hsource

```

3.2.3 注意事项

1. 二进制化后无法在看到开源库的内部实现

2. 需要注意Pod二进制污染: 即部分库的实现修改了，而其他引用到该库的关联库因为并不知道这个修改， 还在使用之前遗留的引用，可能会导致出现Bug, 比如库B之前调用了库A的run方法，但是库A经过了一次迭代，删除了run方法，如果没有重新编译一次库B, 那么它在项目编译阶段并不会有报错和提示，但是真正调用run方法时候，会直接异常

4. 其它

4.1 Build Settings

4.2 二进制重排