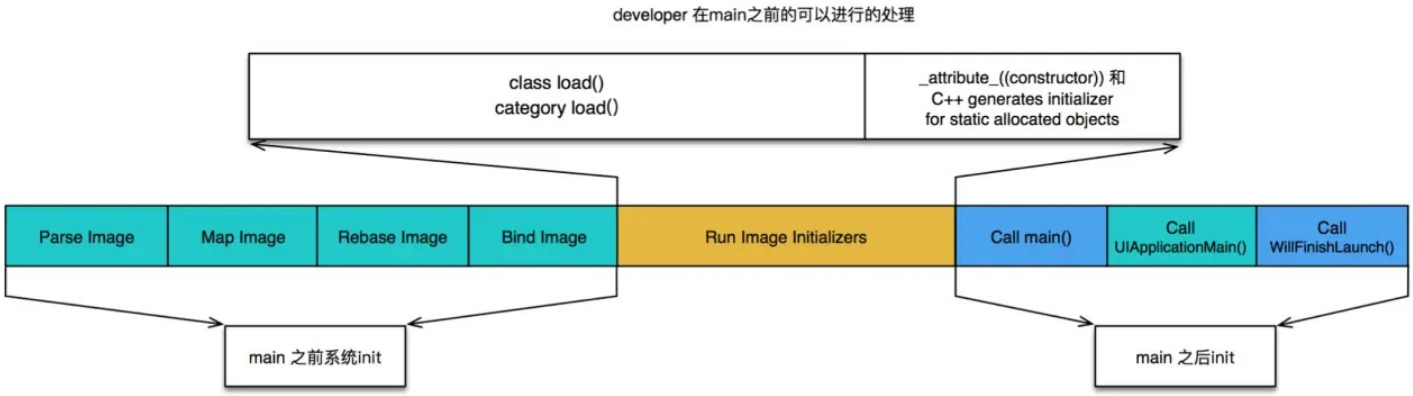
启动速度优化 —— 二进制重排



pre-main

> `SNKClangTraceGuard`

1. 耗时监测

```
选择 `Edit Scheme` - `Arguments` - `Environment Variables`

添加 name `DYLD_PRINT_STATISTICS` value :  `${DEBUG_ACTIVITY_MODE}`
```

2. 代码及工程优化

2.1 dylib-loading

> 加载可执行文件（App 的.o 文件的集合），加载动态链接库:
>
> 1. 使用更少的动态库，并且建议在使用动态库的数量较多时，尽量将多个动态库进行合并。数量上，苹果公司建议最多使用 `6 个非系统动态库` 。

2.2 rebase/binding

> 对动态链接库进行 rebase 指针调整和 bind 符号绑定;
>
> 1. 减少OC类的数量
> 2. 减少分类的数量
> 3. 减少静态变量
> 4. 减少重复、无用以及失效的函数数量

2.3 objc-setup

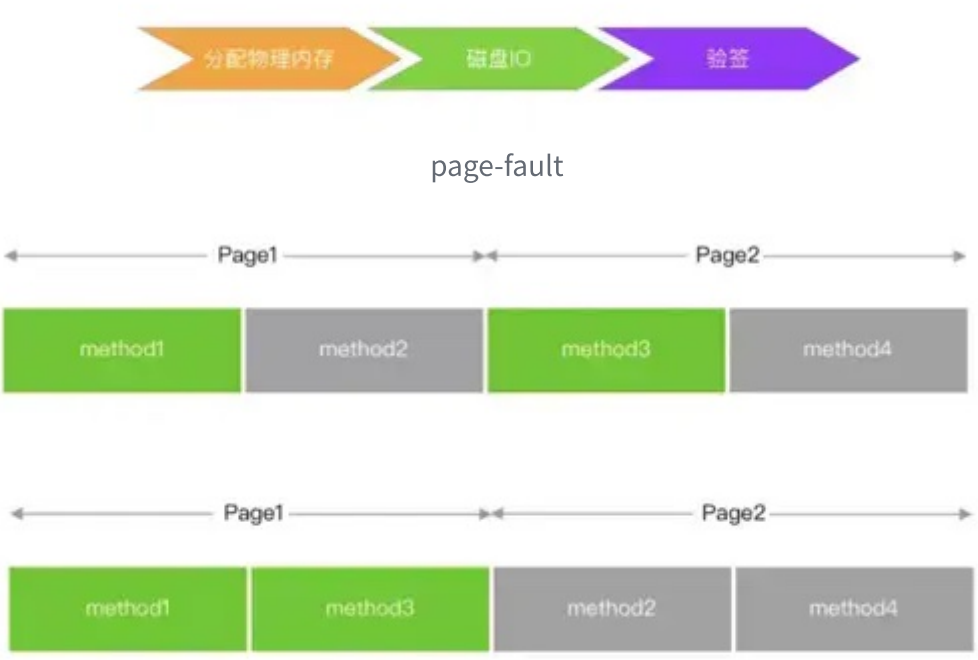
> Objc 运行时的初始化处理，包括 Objc 相关类的注册、category 注册、selector 唯一性检查等；
>
> 和上一步优化方案一致

2.4 initializer

> 初始化，包括了执行 +load() 方法、attribute((constructor)) 修饰的函数的调用、创建 C++ 静态全局变量。
>
> 1. 尽量使用 `+initialize()` 替代 `+load()`
> 2. 减少使用c/c++的attribute((constructor))；推荐使用dispatch_once(),peathrd_once(), std:once()等方法
> 3. 不要在初始化中创建线程
> 4. 推荐使用swift类替代OC类

3. mach-o重排

3.1 重排原因



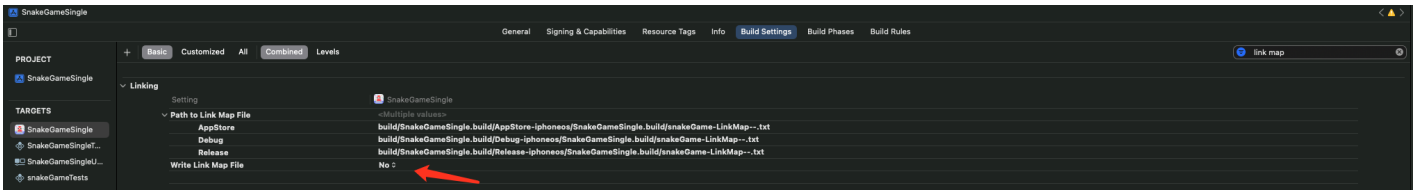
> 1. 进程通过虚拟内存访问物理内存，系统对虚拟内存进行分页以提高效率，当进程访问虚拟内存分页（每页16kb），而此分页对应的物理内存又不存在时会触发page-fault，系统对此虚拟内存分页重新分配物理内存，有需要的话会从磁盘mmap读取数据。

2. 编译器在生成二进制代码的时候，默认按照链接的Object File(.o)顺序写文件，按照Object File内部的函数顺序写函数。
3. 而项目启动期间的函数的实际调用顺序很可能与之不一致，因此需要在加载文件及函数期间，尽量将临近的函数调用分配到一个page。

3.2 重排前准备

3.3 拿到link-map

`LinkMap` 是iOS编译过程的中间产物，记录了二进制文件的布局，需要在Xcode的Build Settings里开启Write Link Map File：



选中编译后的 app，Show In Finder -- 找到build目录 -- 具体路径如下:
/Build/Intermediates.noindex/SnakeGameSingle.build/Debug-iphoneos/SnakeGameSingle.build/snakeGame-LinkMap-normal-arm64.txt

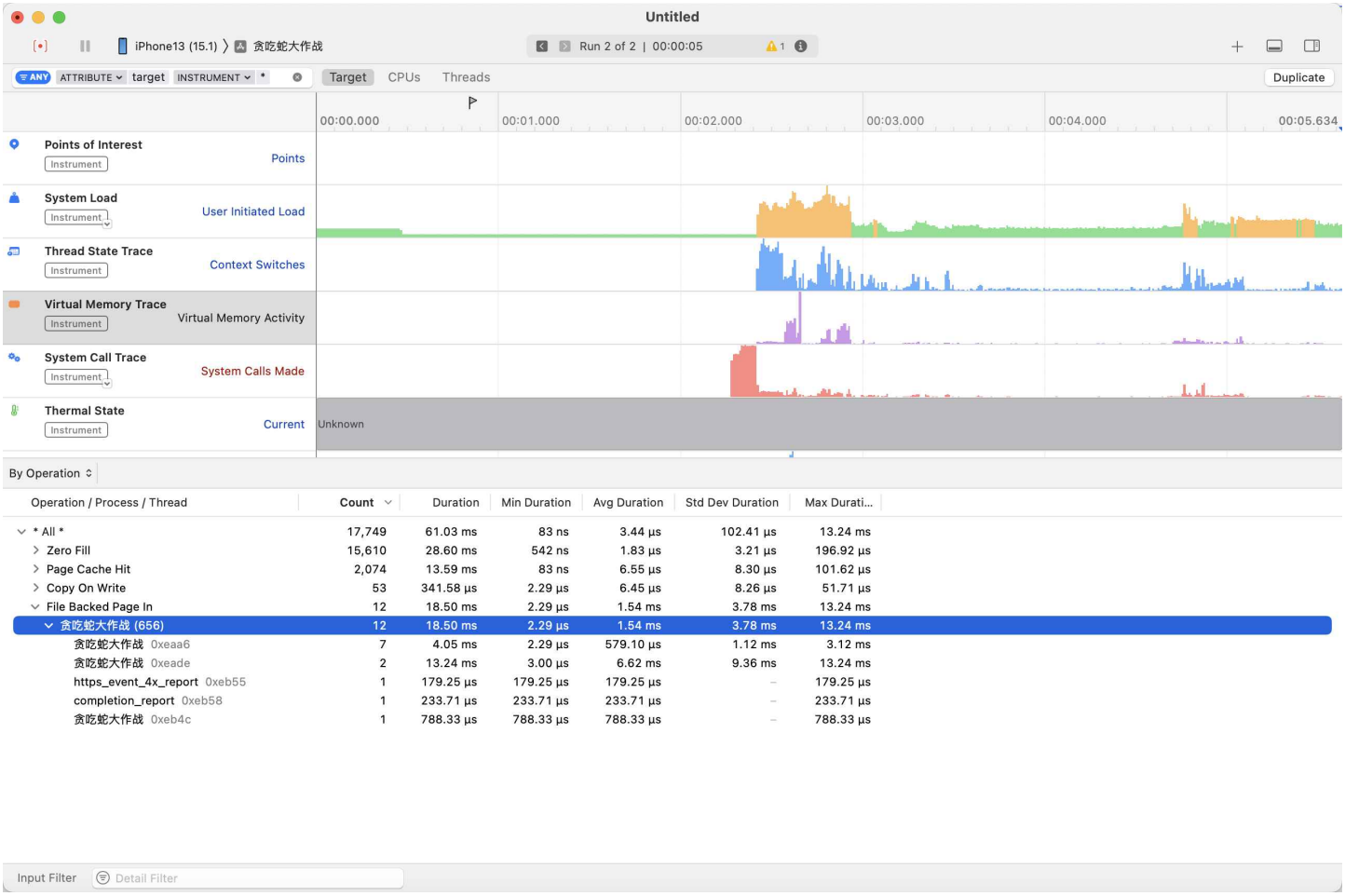
根据芯片类型、系统以及xcode版本不同，位置也有所不同。

主要包含：

1. Object Files 生成二进制用到的link单元的路径和文件编号
2. Sections 记录Mach-O每个Segment/section的地址范围
3. Symbols 按顺序记录每个符号的地址范围

3.3.1 获取启动阶段的page-fault次数

Instruments => System Trace => Run with current device & process => Stop => VM Trace => By Operation => File Backed Page In



3.3.2 获取启动调用的函数符号(Clang插桩)

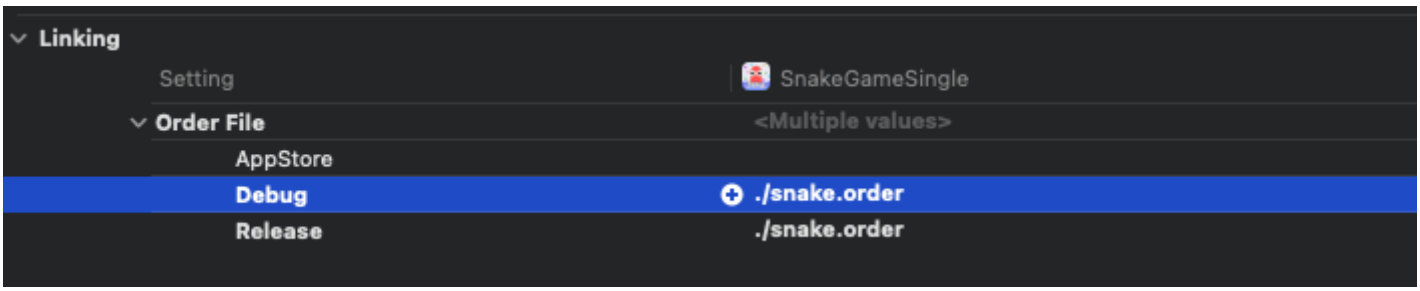
- 在App 的 `Target` - `Build Settings` - `Other C Flags` /Other C++ Flags Debug 添加 `-fsanitize-coverage=func,trace-pc-guard`
- OC - Swift 混编，则在 `Other Swift Flags` Debug 添加 `-sanitize-coverage=func` 和 `-sanitize=undefined`
- 配置Pod编译选项

```
post_install do |installer|
    # 二进制重排设置
  installer.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      if config.name == 'Debug'
        config.build_settings['OTHER_CFLAGS'] ||= '$(inherited)'
        config.build_settings['OTHER_CFLAGS'] << ' '
        config.build_settings['OTHER_CFLAGS'] << '-fsanitize-coverage=func,trace-pc-guard'
      end
    end
  end
end
```

- 通过clang的回调进行函数转化并存储到沙盒目录（`SNKClangTraceGuard`）

3.4 让链接器按照指定顺序生成Mach-O（Clang插桩）

- 配置`app.order`，为链接器指定函数链接顺序



- 移除之前关于clang的所有配置（Other C/C++ Flags/ Other Swift Flags/ Pods）
- 运行项目，对比前后`link-map`内容、`page-fault`效果

4. 总结

> mach-o二进制重排需要解决的问题：
>
> - 如何重排
>   - 通过clang插桩，获取当前项目的函数调用顺序，通过插桩转化后的`Order Files`指定给链接器，链接器会按顺序将`Order Files`中的符号插入到`link-map`中对应`section`的头部
> - 重排的内容
>   - 通过Clang插桩，获取当前项目的函数调用顺序，保存至沙盒目录中
> - 重排效果怎么样
>   - 使用`Instruments`工具获取启动阶段的page fault次数
> - 重排成功了没
>   - 对比指定`Order Files`链接前后的`link-map`是否一致
>
> **为配合优化二进制重排，还需要对4.1.2进行足够的约束/优化操作，具体方案待定**