Main函数之前做了什么?

dyld

dyld的作用

1. 从系统kernel留下的原始调用栈引导启动自己
2. 将程序依赖的动态链接库递归加载进内存，有缓存机制
3. non-lazy符号立即link到可执行文件，lazy的存表里
4. Runs static initializers for executable
5. 找到可执行文件的main函数，准备函数并调用
6. 程序执行中负责绑定lazy符号、提供runtime dynamic loading services、提供调试器接口
7. 程序main函数return后执行static terminator
8. 某系场景下main函数结束后调用libSystem的_exit函数

__dyld_start

a.调用dyldbootstrap::start()方法，b.通过上个方法返回main函数的地址，填入参数并调用main函数

imageLoader

image大概表示一个二进制文件(可执行文件或者so文件)，里面是被编译过的符号、代码等，所以ImageLoader作用是将这些文件加载进内存，且每一个文件对应一个ImageLoader实例来负责加载

1. 在程序运行时它先将动态链接的image递归加载
2. 再从可执行文件image递归加载所有符号

Runtime 与 +load

libSystem是若干个系统lib的集合，所以它只是一个容器而已; 由libSystem_initializer -> _objc_init，这里是objc和runtime的初始化入口

_obj_init中绑定了新image被加载后的callback

dyld_register_image_state_change_handler(dyld_image_state_bound, 1, &map_images);
dyld_register_image_state_change_handler(dyld_image_state_dependents_initialized, 0, &load_images);

dyld担当了runtime与ImageLoader中间的协调者，当新的image加载进来后交由runtime去解析这个二进制文件的符号和代码

1. dyld 开始将程序二进制文件初始化
2. 交由 ImageLoader 读取 image，其中包含了我们的类、方法等各种符号
3. 由于 runtime 向 dyld 绑定了回调，当 image 加载到内存后，dyld 会通知 runtime 进行处理
4. runtime 接手后调用 map_images 做解析和处理，接下来 load_images 中调用 call_load_methods 方法，遍历所有加载进来的 Class，按继承层级依次调用 Class 的 +load 方法和其 Category 的 +load 方法

- 加载可执行文件 （App的.o文件的集合）【Mach-O __TEXT被执行的代码和只读常量,__DATA包含全局变量和静态变量,__LINKEDIT程序元数据，比如函数的名称和地址】
- 加载动态链接库，进行rebase指针调整和bind符号绑定 【dyld】Load dylibs -> Rebase -> Bind -> Objc -> Initializers
- Objc运行时的初始处理，包括Objc相关类的 注册，category注册，selector唯一性检查等
- 初始化，包括执行+load方法，attribute（constructor）修饰的函数的调用，创建C++静态变量

启动时间优化

Xcode为我们提供了获取各个dylib加载时间的方法，在Xcode 中 Edit scheme -> Run -> Auguments 将环境变量 DYLD_PRINT_STATISTICS 设为 1

相应地，这个阶段对于启动速度优化来说，可以做的事情包括：

减少动态库加载。每个库本身都有依赖关系，苹果公司建议使用更少的动态库。静态库的加载时间更短，但是当我们的Extension和App需要使用同一部分代码时，我们需要将其封装动态库 (See this blog)。动态库同时也能解决二进制包文件过大的问题，苹果对app包大小判断是不将动态库包大小计算在内的。

check framework应设为optional和required，如果该framework在当前App支持的所有iOS系统版本都存在，那么就设为required，否则就设为optional，因为optional会有些额外的检查；

减少加载启动后不会去使用的类或者方法。

合并或者删减一些OC类，关于清理项目中没用到的类，可以借助AppCode代码检查工具：

删减一些无用的静态变量

删减没有被调用到或者已经废弃的方法

尽量不要用C++虚函数(创建虚函数表有开销)

避免使用 attribute((constructor))，可将要实现的内容放在初始化方法中配合 dispatch_once 使用。

+load()方法里的内容可以放到首屏渲染完成后再执行，或使用 +initialize()方法替换掉。因为，在一个 +load() 方法里，进行运行时方法替换操作会带来 4 毫秒的消耗。不要小看这4 毫秒，积少成多，执行+load()方法对启动速度的影响会越来越大。

控制C++ 全局变量的数量。

二进制重排

https://mp.weixin.qq.com/s?__biz=MzI1MzYzMjE0MQ==&mid=2247485101&idx=1&sn=abbbb6da1aba37a04047fc210363bcc9&scene=21#wechat_redirect

虚拟内存

物理内存

分页

缺页中断（Page Fault）：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

分配物理内存，有需要的话会从mmap读入数据

通过AppStore渠道分发的App，Page Fault还会进行签名验证，所以一次PageFault的耗时比想象的要多

分配物理内存->磁盘IO->验签

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

静态库文件.a是一组.o文件的ar包，可以用ar -t查看.a包含的所有的.o

Load -> C++静态初始化 ->didFinishLaunch-> UI Setup

System Trace