iOS屏幕共享预研

这里只记录在iOS 12及以上版本的屏幕共享预研，之前版本的信息，后续更新。

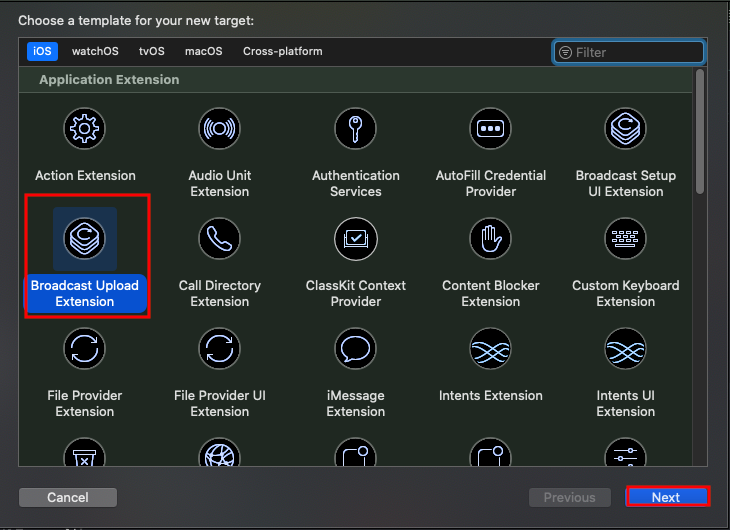
想实现屏幕共享，需要用到ReplayKit（iOS 9）框架，且需要创建App Extension进行录制及广播。

1. Extension是什么？

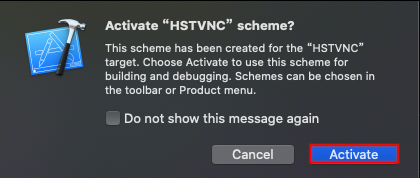
<https://juejin.im/post/5acefc0e6fb9a028e1205688>

1. 集成Extension
2. 新建target

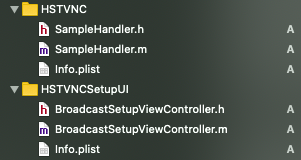
在现有工程选择File->New->Target，选择Broadcast Upload Extension，如下图:



配置好Product Name，Bundle id（在宿主App的基础上加.xxx，目的是为了App Groups，详见配置App Group）单击Finish，就弹出一个框，选择Activate，如下图：



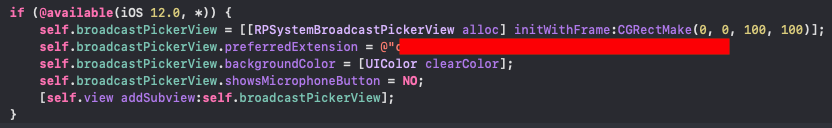
然后你会发现项目里会新增两个目录，如下图：



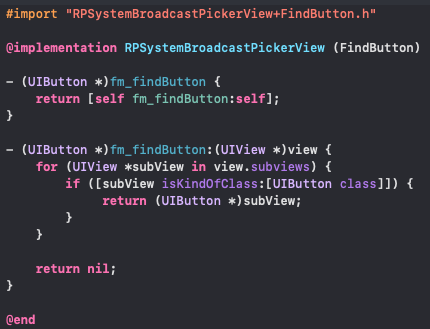
创建target时，iOS12上可以不选UI是没问题的。

1. RPSystemBroadcastPickerView

iOS12之前，用户只能在应用外部启动系统录屏，操作较复杂。iOS12开始，就可以直接在应用内部发起录屏，RPSystemBroadcastPickerView就是用来展示启动录屏界面的入口view，包含一个黑色圆点按钮，点击黑色按钮就会显示系统默认启动录屏界面。通过将该view添加app内部view上，就有操作录屏的能力了，具体实现如下：



短短几行代码就可以启动录屏，但这样的流程还是有点繁琐，用户必须点击黑点才能真正打开启动录屏界面，实际需求往往也是不需要中间黑点按钮点击过程，所以就有了隐藏黑点按钮的需求：首先从RPSystemBroadcastPickerView找到按钮，然后将按钮保存，每次点击，主动触发按钮点击时间，具体实现如下：





以上代码就完成了弹出启动录屏界面，但这里会有两个问题（详情见问题总结2与3），最终完整代码应该是：



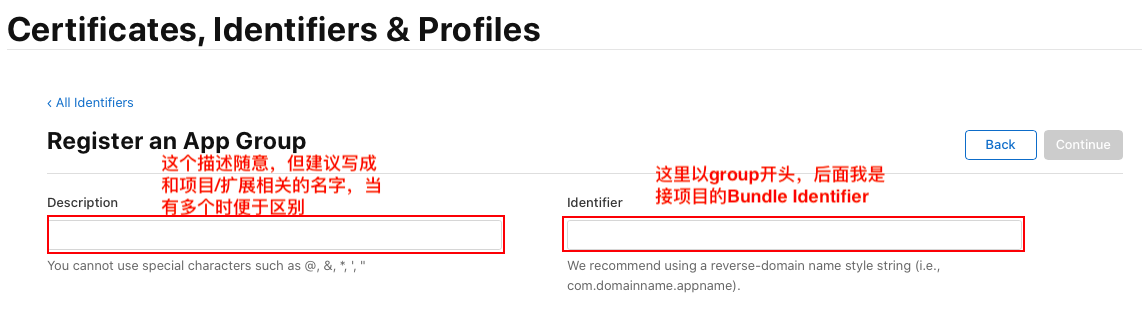
由于可能会多次发起录屏与停止录屏，于是将相关对象通过属性方式来处理存储，就不用每次创建一个新对象。

1. 配置App Groups

想实现Extension App与Container App间的通信（eg：发送开始录屏、停止录屏等通知）、数据交互，可以通过App Groups实现。接下来就看看怎么创建App Group。

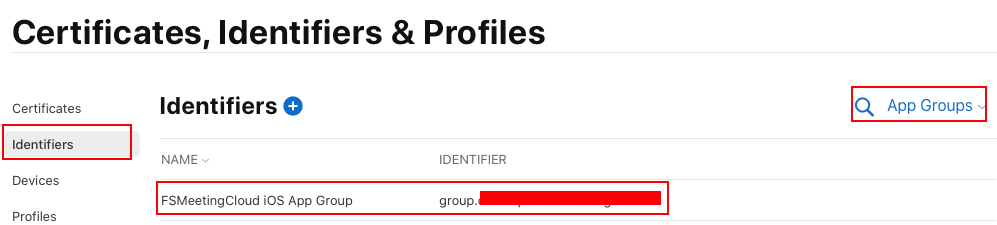
1. 创建App Groups

登录苹果开发者后台，来到证书配置处，选择indentifiers->点击左上角＋->选择 App Groups -> 进行App Group相关信息填写（如下图）



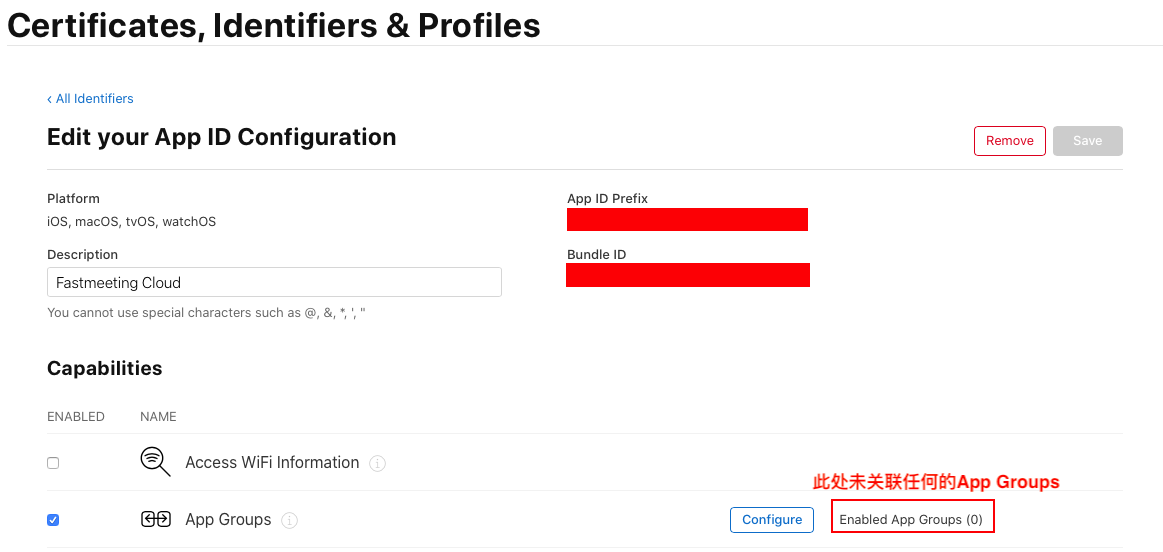
以group开头的indentifier，后期不能修改，程序中也会使用到。

创建好之后，就可以在indentifiers的列表里看到刚才创建的App Group。

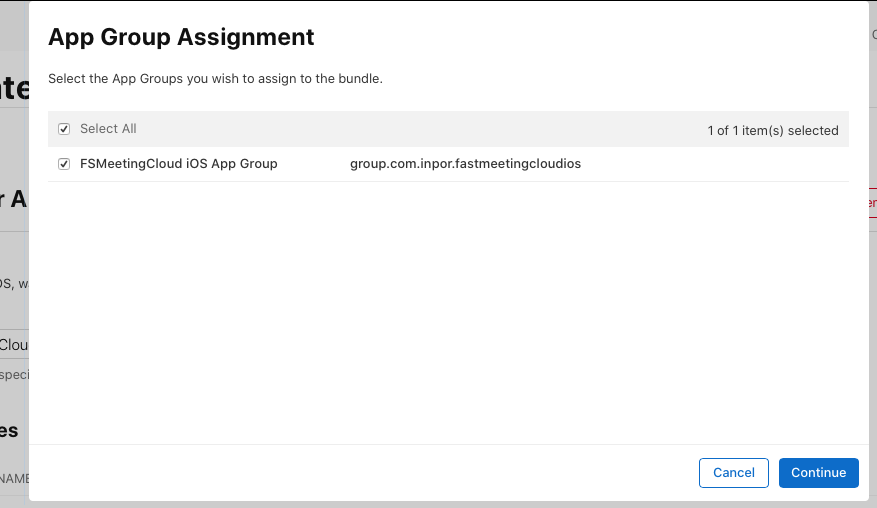


1. 创建（编辑）App ID

对于没有创建过App ID的项目，需要新建App ID（此处省略）。如果已经创建过，在App IDs列表里直接找到项目对应的App ID，选择Edit -> 勾选App Groups -> 点击Configure进行配置。



刚创建的App Group会在列表里，可能还会包含其他App Group，选中刚才创建的App Group。



配置完后，就能看到该App Groups配置了一个App Group，如下图：

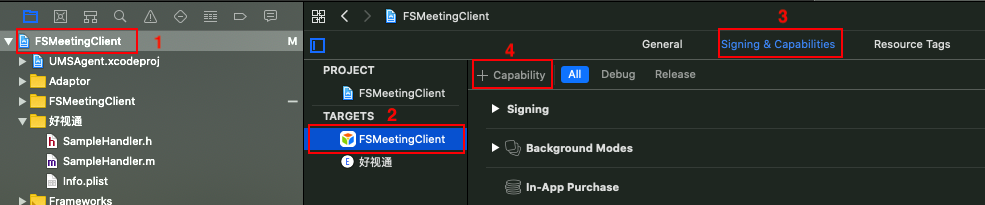
截屏2020-04-07 上午11.18.26

1. Provisoning Profiles配置

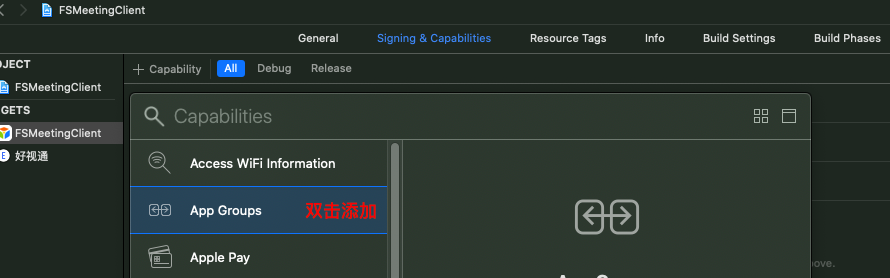
由于App ID进行了修改，所以证书状态变成了Invalid状态，所以需要重新配置证书。这里省略修改步骤。

1. 项目配置

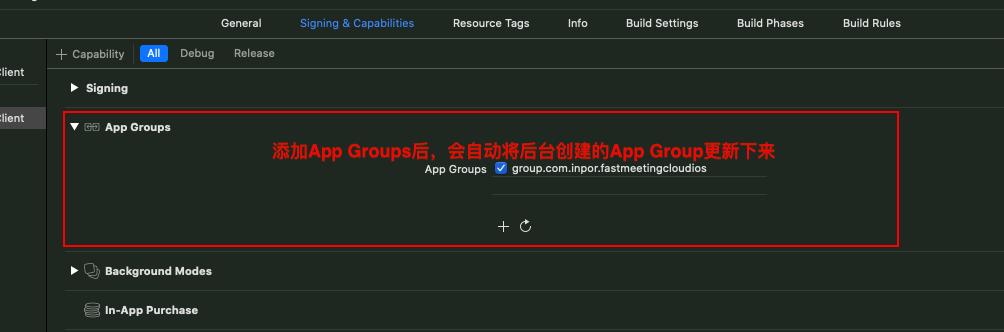
配置好App Groups之后，需回到项目工程文件进行App Groups配置。



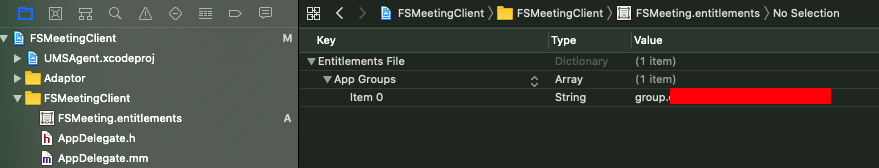
按照上图的1->2->3->4步骤之后，会弹出一个新窗口，双击App Groups进行添加（如下图）：



添加后，之前在苹果后台创建的App Group会自动同步下来，然后进行勾选，后期有新增App Group，手动刷新。



当完成这些配置后，在项目文件夹下会新增一个.entitlements的文件。

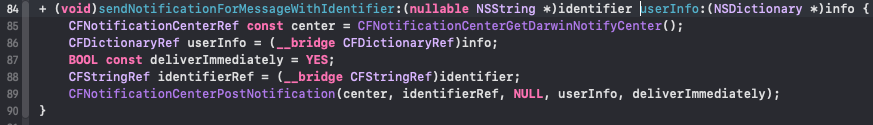


1. 宿主App与Extension间事件传递
2. CFNotificationCenterGetDarwinNotifyCenter

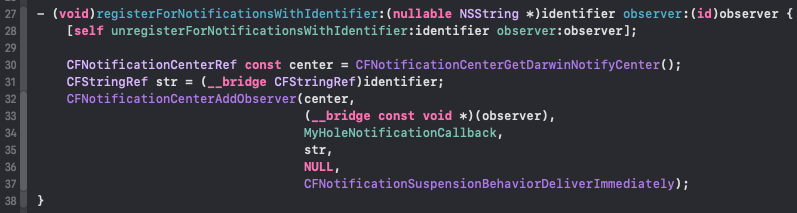
平时我们发送通知是通过NSNotificationCenter实现，但它仅限于App内部通知，想要跨进程发通知，需要使用CFNotificationCenterRef。

通过CFNotificationCenterGetDarwinNotifyCenter可以拿到CFNotificationCenterRef对象。它的逻辑和NSNotificationCenter差不多，监听者需要进行注册通知，处理通知（回调方法）、移除通知，通过identifier来辨别是哪一条通知；发送者只负责发。

1. 发送通知



1. 接收通知
2. 需要接收通知，首先得注册这条通知：

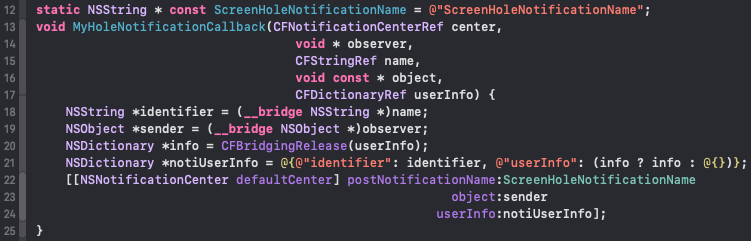


1. 注册对应事件的通知



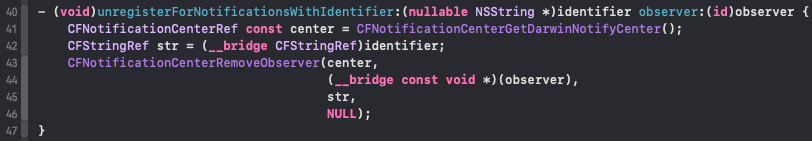
如果该方法是container App和Extentision App都会用到，identifier最好是通过参数传进来，通过for循环一一注册事件。

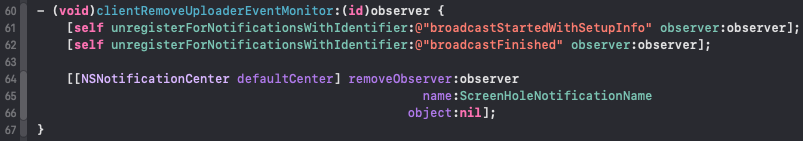
1. 通知的block回调



当接收到通知时，就会回调这个block，将通知做进一步的转发，将消息分发出去（这里做一层转发的好处就是便于在需要的地方监听需要的消息，所以注册App内部通知，不一定是在注册进程间通知处注册，可以在你需要监听这个通知的地方注册），在App内部发送通知（App内部需通过NSNotificationCenter处理通知）。

1. 当不需要时，进行对应事件通知移除





1. 宿主App与Extension间数据传递
2. 数据共享方式（共享内存传数据）
3. NSUserDefault
4. NSFileManager
5. 网络通信传数据（socket）
6. 宿主App与Extension间代码共享

宿主App与Extension属于两个不同进程，而在实际开发中，他们直接往往包含一些相同逻辑，部分代码需要共用（eg：进程间事件传递、进程间数据传递、协议宏定义、类型定义等等）。

宿主App与Extension属于两个不同的target，共用代码的方式有以下几种：

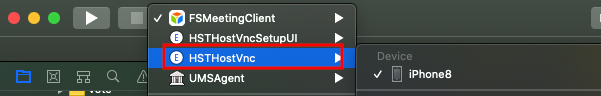
* 同一份代码创建两份，分别添加到宿主App和Extension两个target中。不建议使用这种方法，一旦逻辑有修改，还得改两份，如果改动较大，有点受不了。
* 抽离公共代码，将该部分代码同时引用的两个target中，这样有逻辑修改时，只需修改一份代码（但这里需要注意，也是我当时犯的一个错误，虽然代码是共用的，但是数据的销毁，得自己处理自己的，一开始将搭建的socket服务器与客户端弄成单例，初始化都放在了SampleHandler文件里，关闭录屏销毁单例时也是在该文件，但是后面发现客户端单例并没有真正意义上销毁，宿主App里还是在正常打印socket连接失败的log，这里的建议是能分开则分开引用，减少文件之间的依赖）
* 将抽离的公共代码，放到独立的framework，然后两个target都依赖该framework，这是比较标准的做法
* 使用CocoaPods或者Carthage管理，只需要在Podfile里分别写两个target所依赖的pod即可（Carthage的Cartfile类似）。

我自己项目里用的第二种，方便快捷。

1. App Extention debug

宿主App与Extension之间不能同时进行调试，如何调试Extension呢？

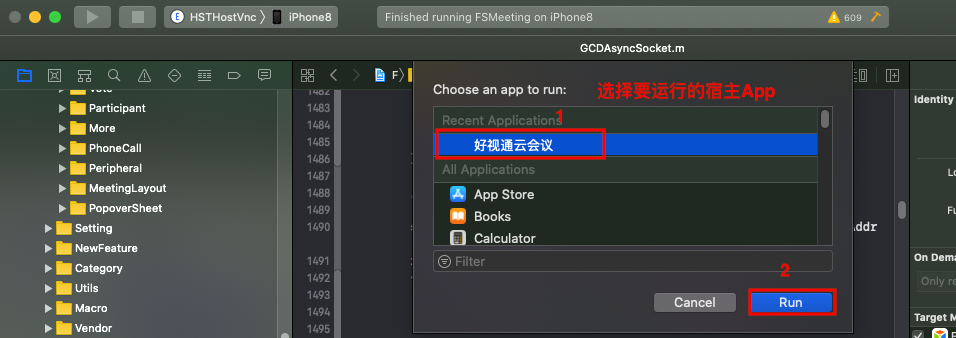
1. 将target切换到Extension，如下图：



切换后结果如下：

app extention调试2

1. Command+R，会弹出一个应用列表框，然后选择要运行的宿主App，再点击run，如下图：



1. 然后就可以在插件里进行调试。但当你debug时，很快就会发现一个问题：通过打断点进行数据调试，Extension很快就结束并退出。这是系统限制了它的最大运行事件，以保持系统的流畅。
2. 共享屏幕方案

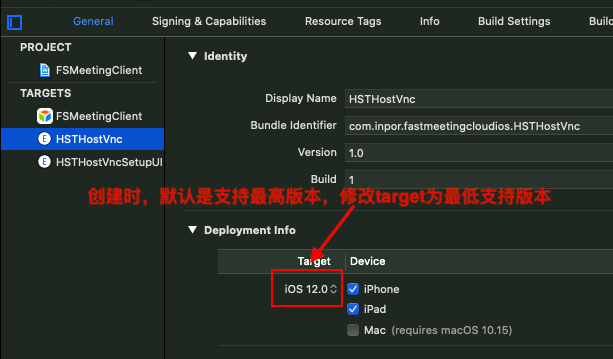
前面准备工作差不多了，接下来说说具体的实现方案以及踩过的坑

1. 数据处理
2. 降帧
3. H264编解码
4. Metal
5. OpenGL
6. Libyuv
7. 问题总结
8. 缺少Broadcast Upload Extension App List

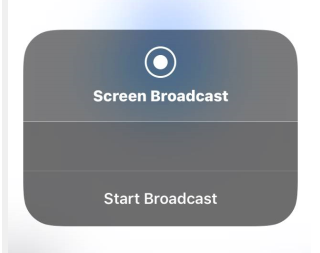
进行弹出系统启动录制弹框时，在Xcode支持的最新系统以下的设备，弹框中均缺少app extension，现象如下图：



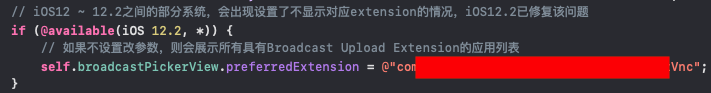
进行定位发现，是由于Extension的target创建时默认是支持最新系统iOS 13.4.1，低于该系统的，都不能正常加载Extension，将target改为最低支持版本iOS 12，就成功解决该问题。



1. iOS 12.1.4的设备不能加载出app extention，现象如下图：



将preferredExtension赋值那句代码注视掉，是能够加载出所有具有Broadcast Upload Extension的应用列表，所以猜测应该是系统bug，后面查到12.2的系统已修复该问题。由于测试设备有限，也不知道具体那个系统引入，为了保险起见，将12.0～12.2之间的版本，都设置为加载所有应用列表，12.2及以上的只加载指定的app extention。代码如下图：



1. 处理黑点按钮的点击事件时，发现iOS 12的系统上点击不响应

通过将事件改为UIControlEventAllTouchEvents发现，又可以触发点击事件，所以当时猜测应该是iOS 12与iOS 13内部按钮点击事件不同造成的。后面经过尝试，代码应像下面这样写：

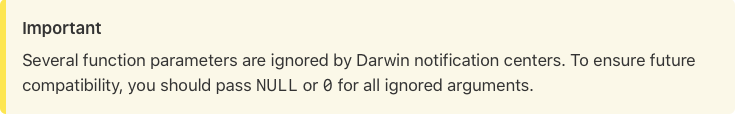
39E9EA09-0E8F-44A5-BBEF-1EA944497058

1. 内存泄漏

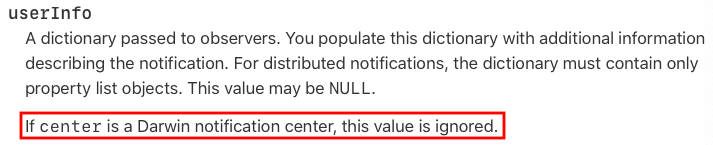
数据处理过程中，数据处理不当，一不小心就会造成内存泄露，每秒10帧左右，内存几十兆几十兆的增加

1. 进程间事件传递里的userInfo一直为空

后期修改录屏方案为直接从插件与音视频库交互时，需要将客户端的一些数据传给Extension App，想着通过userInfo来处理，但发现不管userInfo怎么处理，一直为空。后来通过查看官方文档发现有这样一条警告：

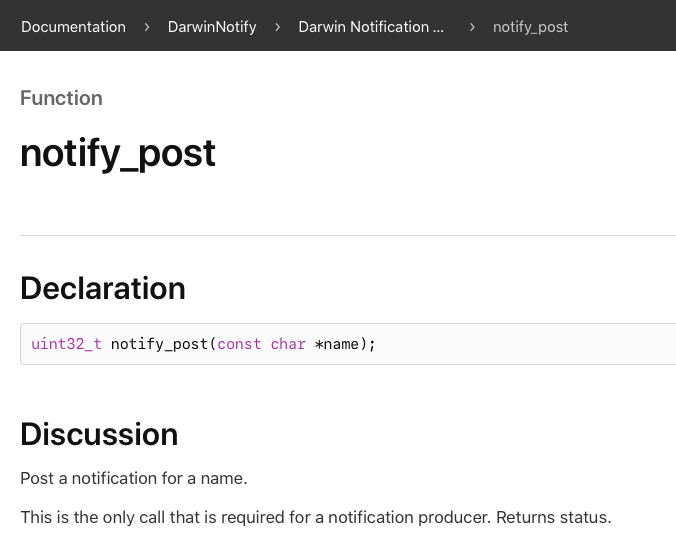


意思就是Darwin通知中心会忽略几个功能参数，应该设置所有被忽略的参数为NULL或者0。再查看CFNotificationCenterPostNotification对userInfo参数的描述，也是一样的说明：



为什么是这样？

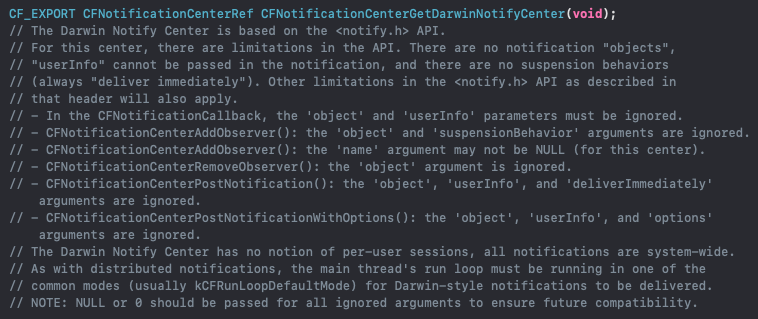
文档中提到Darwin通知中心是建立在notify.h中声明的核心操作系统通知机制之上，通过这一点，再深入就会发现post是通过调用notify\_post()来发送通知，该方法的声明如下：



它只有一个参数，无法向通知中心添加其他数据，这是一个非常古老和基本的IPC机制，它不知道任何关于Core Foundation数据结构的东西。

所以，是不能通过userInfo来传递一些数据的。本以为换了方案可以把socket丢弃，没想到还是需要它。

更直观的，可以通过CFNotificationCenterGetDarwinNotifyCenter头文件里的注释得到同样的结论：



本问题参考连接：

<https://hant-kb.kutu66.com/ios/post_12830568>

<https://developer.apple.com/documentation/darwinnotify/1433472-notify_post>

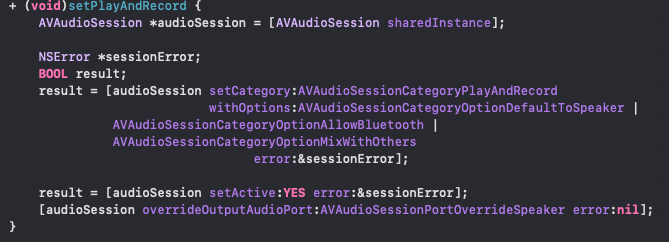
<https://developer.apple.com/library/archive/documentation/Darwin/Reference/usr_APIs/notify/index.html>

1. 录屏后退后台，播放视频，应用就被挂起

现象：插件里的数据不能传到宿主App，远端卡住

底层音视频库默认用的AudioUnit引擎，退后台后，音频设备会被其他播放视频的App（如：腾讯视频，爱奇艺等）夺走，咱们的App从而被挂起，将引擎换为AudioQueue就可以了，AudioQueue对AudioUnit做了一层封装，具体是封装了些什么，没去进一步研究（后续有研究，再更新具体原因）。

虽然上述修改可以应用不再挂起，但是首次录屏后再去打开其他App播放视频，仍然会出现视频数据传不出去问题，主要还是因为音频被中断：



上图中的修改就可以保证App与其他App共存（混音），支持录制音频。

1. 实现Extension App名称多语言不生效
2. 注意事项