2020-04-09

1：UI操作在主线程上进行

<https://www.jianshu.com/p/5ada82024d42>

2：紫色警告 子线程刷新UI了

2002-04-02

链接：https://juejin.im/post/5d8af88ef265da5b6e0a23ac

**1：iOS 13 支持适配的机型**

* iPhone 11、iPhone 11 Pro、iPhone 11 Pro Max
* iPhone X、iPhone XR、iPhone XS、iPhone XS Max
* iPhone 8、iPhone 8 Plus
* iPhone 7、iPhone 7 Plus
* iPhone 6s、iPhone 6s Plus
* iPhone SE
* iPod touch (第七代)
* 后续发布的新版本 iPhone

**2：《针对引导页不会全屏》automaticallyAdjustsScrollViewInsets，当设置为YES时（默认YES），如果视图里面存在唯一一个UIScrollView或其子类View，那么它会自动设置相应的内边距，这样可以让scroll占据整个视图，又不会让导航栏遮盖。**

当controller上的第一个子视图不是scrollview以及其子类的时候，就会取消内边距。此时原本全屏的scrollview设置的frame（0，0，self.view.frame.size.width，xx）就会从状态栏开始算起，如果应用有导航栏的话，那么就会遮盖住视图的64个高度

解决方案:

self.automaticallyAdjustsScrollViewInsets = NO；禁用掉自动设置的内边距，自行控制controller上index为0的控件以及scrollview控件的位置

self.edgesForExtendedLayout = UIExtendedEdgeNone;这种方式设置，不需要再重新设置index为0的控件的位置以及scrollview的位置，（0，0）默认的依然是从导航栏下面开始算起

链接：https://www.jianshu.com/p/75fd23bb5286

3：在 Xcode 中调试运行时，子线程修改界面会有紫色感叹号标出，注意修改成回到主线程即可。

* \* setNeedsLayout：告知页面需要更新，但是不会立刻开始更新。执行后会立刻调用layoutSubviews。  
  \* layoutIfNeeded：告知页面布局立刻更新。所以一般都会和setNeedsLayout一起使用。如果希望立刻生成新的frame需要调用此方法，利用这点一般布局动画可以在更新布局后直接使用这个方法让动画生效。  
    
  \* layoutSubviews：系统重写布局。
* \* setNeedsUpdateConstraints：告知需要更新约束，但是不会立刻开始。
* \* updateConstraintsIfNeeded：告知立刻更新约束。  
    
  \* updateConstraints：系统更新约束。
* 这么多方法中，目前使用比较多的是 layoutIfNeeded 。因为在Auto Layout 实现动画的时候，layoutIfNeeded 方法可以立刻生成新的frame特性是一大利器。

[在点击tableview上的一个cell后弹出alert，会发现有延迟的问题或者点击没有反应，随便再点击一下才会弹出](https://www.cnblogs.com/fendoulushangdefenqing/p/8359962.html)

将弹出的代码直接放到主线程执行

dispatch\_async(dispatch\_get\_main\_queue(), ^{

    [self presentViewController:AlertView animated:YES completion:nil];

    });

2020-04-09  <https://www.jianshu.com/p/5ada82024d42>

主线程更新ui:

UI必须放在主线程的原因很简单，因为UIKit为了提升性能，压根没加锁，它不是线程安全的，如果你不在主线程的里面操作，会出现什么样的UI，谁也不敢保证。

完全没有讲到重点啊，讲一些大家都知道的东西啊，UI必须放在主线程的原因很简单，因为UIKit为了提升性能，压根没加锁，它不是线程安全的，如果你不在主线程里面操作，会出现什么样的UI，谁也不敢保证。

bucket

英 [ˈbʌkɪt] 美 [ˈbʌkɪt]

n.

(有提梁的)桶;大桶状物;(起重机的)吊斗;(挖土机的)铲斗;一桶(的量)

v.

拼命划桨;挖掘铲斗;挑流鼻坎

### **4、上线前注意：**

1）、删掉代码中所有的测试代码  
2）、如果后台有审核模式，提醒后台开启此模式  
3）、主流程再跑一跑  
4）、全局搜索waring，检查所有标记waring的地方

//kDISPATCH\_MAIN\_THREAD(<#mainQueueBlock#>)

//    dispatch\_async(dispatch\_get\_main\_queue(), mainQueueBlock);

//   ^ mainQueueBlock

//    <#returnType#>(^<#blockName#>)(<#parameterTypes#>) = ^(<#parameters#>) {

//        <#statements#>

//    };

    #define kDISPATCH\_MAIN\_THREAD(mainQueueBlock) dispatch\_async(dispatch\_get\_main\_queue(), mainQueueBlock);

//    void(^myblock)() = ^(){

//    };

//    kDISPATCH\_MAIN\_THREAD(myblock);

//    dispatch\_async(dispatch\_get\_main\_queue(), ^{

//

//    });

#define kDISPATCH\_GLOBAL\_QUEUE\_DEFAULT(globalQueueBlock) dispatch\_async(dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0), globalQueueBlocl);

### 拿到当前正在显示的控制器，不管是push进去的，还是present进去的都能拿到

- (UIViewController \*)getVisibleViewControllerFrom:(UIViewController\*)vc {

if ([vc isKindOfClass:[UINavigationController class]]) {

return [self getVisibleViewControllerFrom:[((UINavigationController\*) vc) visibleViewController]];

}else if ([vc isKindOfClass:[UITabBarController class]]){

return [self getVisibleViewControllerFrom:[((UITabBarController\*) vc) selectedViewController]];

} else {

if (vc.presentedViewController) {

return [self getVisibleViewControllerFrom:vc.presentedViewController];

} else {

return vc;

}

}}

作者：隔墙送来秋千影  
链接：https://www.jianshu.com/p/3032a9d26a03  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

### **127、在非ViewController的地方弹出UIAlertController对话框**

// 最好抽成一个分类

UIAlertController \*alertController = [UIAlertController alertControllerWithTitle:@"Title" message:@"message" preferredStyle:UIAlertControllerStyleAlert];//...

id rootViewController = [UIApplication sharedApplication].delegate.window.rootViewController;if([rootViewController isKindOfClass:[UINavigationController class]]){

rootViewController = ((UINavigationController \*)rootViewController).viewControllers.firstObject;}if([rootViewController isKindOfClass:[UITabBarController class]]){

rootViewController = ((UITabBarController \*)rootViewController).selectedViewController;}[rootViewController presentViewController:alertController animated:YES completion:nil];

2020-04-17 AGAIN

19:<https://time.geekbang.org/column/article/91714> 极客时间

A:

Runloop <https://v.youku.com/v_show/id_XODgxODkzODI0.html>

卡顿监控系统 腾讯：<https://github.com/Tencent/matrix> 可以集中到工程中检测工程中卡顿算法。

退火算法 探寻最优解的方法

 模拟退火算法是一种随机算法，并不一定能找到全局的最优解，可以比较快的找到问题的近似最优解。 如果参数设置得当，模拟退火算法搜索效率比穷举法要高。

1：线程数过多，2：主线程堆栈 3：   dispatch\_async(dispatch\_get\_main\_queue(), ^{ });

4：崩溃监控 采集奔溃日志信息 如何捕获

被系统强杀

5：而崩溃率等技术指标，一般都是由崩溃监控系统来搜集。同时，崩溃监控系统收集到的堆栈信息，也为解决崩溃问题提供了最重要的信息。

6：KVO 问题、NSNotification 线程问题、数组越界、野指针等崩溃信息，是可以通过信号捕获的。但是，像后台任务超时、内存被打爆、主线程卡顿超阈值等信息，是无法通过信号捕捉到的。

7：handleSignalException 里通过backtrace\_symbols 方法就能获取到当前的堆栈信息。堆栈信息可以先保存在本地，下次启动时再上传到崩溃监控服务器就可以了。先将捕获到的堆栈信息保存在本地，是为了实现堆栈信息数据的持久化存储。那么，为什么要实现持久化存储呢？这是因为，在保存完这些堆栈信息以后，App 就崩溃了，崩溃后内存里的数据也就都没有了。而将数据保存在本地磁盘中，就可以在 App 下次启动时能够很方便地读取到这些信息。

监测卡顿：

1：复杂 UI 、图文混排的绘制量过大；2：在主线程上做网络同步请求；3：在主线程做大量的 IO 操作；4：运算量过大，5：CPU 持续高占用；6：死锁和主子线程抢锁。

Fps

那么，我们如何监控到什么时候会出现卡顿呢？是要监视 FPS 吗？以前，我特别喜欢一本叫作《24 格》的杂志，它主要介绍的是动画片制作的相关内容。那么，它为啥叫 24 格呢？这是因为，动画片中 1 秒钟会用到 24 张图片，这样肉眼看起来就是流畅的。FPS 是一秒显示的帧数，也就是一秒内画面变化数量。如果按照动画片来说，动画片的 FPS 就是 24，是达不到 60 满帧的。也就是说，对于动画片来说，24 帧时虽然没有 60 帧时流畅，但也已经是连贯的了，所以并不能说 24 帧时就算是卡住了。

由此可见，简单地通过监视 FPS 是很难确定是否会出现卡顿问题了，所以我就果断弃了通过监视 FPS 来监控卡顿的方案。

如何获取堆栈信息？Network？

获取堆栈信息的一种方法是直接调用系统函数

dSYM ？来获取具体是哪行代码出了问题

        dSYM是保存16进制函数地址映射信息的中转文件，我们调试的symbols都会包含在这个文件中，并且每次编译项目的时候都会生成一个新的dSYM文件，位于/Users/<用户名>/Library/Developer/Xcode/Archives目录下

作者：清夜银月  
链接：https://www.jianshu.com/p/72e16abc1350  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

第三方：获取堆栈信息

<https://opensource.plausible.coop/src/projects/PLCR/repos/plcrashreporter/browse>

1. RunLoop和线程是一一对应的关系，RunLoop负责帮线程处理事件，也可以让线程休眠或运行。主线程会默认开启RunLoop，而子线程的RunLoop需要我们手动开启。  
2. mach\_port可以简单理解为内核的端口，基于port的事件属于source1类型；RunLoop在创建时就开始跑起来，然后处理timer、source0、source1等事件，而不是由mach\_port触发的RunLoop。  
3. 相当于“你睡着了，有人喊你去吃小龙虾，你才愿意起来”。  
4. 每个线程都有RunLoop，包括主线程和子线程。

例子：

印象最深的一次卡顿：在TableView的Cell里访问了数据库。  
  
其次是在一个答题系统里，多选单选与判断题，一个Cell从最少两个选项、一个选项三到N行，到最多九个选项。然后使用了TableViewCell的高度自动计算，后来换成了自己手动计算并缓存。  
  
还有Cell的样式太多，后来把尽可能相同的都放到一个Cell里，也就是：一个Cell有七行，如果有四行不显示，那就移除四行，如果需要九行就再加两行。还有就是网络请求同步操作，结果数据过大，然后就做了异步和无数据的默认显示。  
  
其实我就是从这几个方面下手的：页面重叠部分、简单的动画与显示用CALayer自己画、对象释放与创建的次数、高度自动缓存、autoRelease使用、static使用部分（如单列对象）、UI刷新前的数据处理部分、页面刷新次数、通知和监听的移除、数据的不合理读取、复杂操作尽量异步、  
  
最后想说下：线程的消息传递依赖于NSRunLoop，函数的调用依赖runtime。感觉二者挺像的。

iOS通过堆栈管理所有的app进程，通过一个优先级最高的线程去监控系统内存的压力，还有一个快速对照表记录所有app的内存使用情况。如果内存有压力了，就按照优先级去释放优先级低还使用内存多的。  
所以app得内存阈值是没有固定大小的。我一直以为每个app可以使用的内存大小是固定的。

、

2020-04-20

Watchdog 看门狗

Fishhook 是facebook的源码

而链接器的作用，就是完成变量、函数符号和其地址绑定这样的任务。而这里我们所说的符号，就可以理解为变量名和函数名

链接的共用库分为静态库和动态库：静态库是编译时链接的库，需要链接进你的 Mach-O 文件里，如果需要更新就要重新编译一次，无法动态加载和更新；而动态库是运行时链接的库，使用 dyld 就可以实现动态加载。

dlopen 也可以选择是立刻解析所有引用还是滞后去做。dlopen 打开动态库后返回的是引用的指针，dlsym 的作用就是通过 dlopen 返回的动态库指针和函数符号，得到函数的地址然后使用。