

文章目录

- 为什么需要线程保活

线程保活的操作

关于run方法的疑问

RunLoop的启动方法

RunLoop的关闭方法

线程保活的分析及代码

留存的问题
- 参考文献

为什么需要线程保活

在iOS项目中，有时会有一些花费时间较长的操作阻塞主线程，我们通常为了防止界面卡顿，将其放入子线程中运行。根据线程知识，如果子线程执行完分配的任务后，就会自动销毁。比如我们现在定义一个线程，改写它的dealloc方法，观察它什么时候销毁

```
1 @implementation TAThread
2
3 -(void)dealloc {
4     NSLog(@"%s", __func__);
5 }
6
7 @end
```

在ViewController里新建一个TAThread子线程，让它去执行任务

```
1 @implementation ViewController
2
3 -(void)viewDidLoad {
4     [super viewDidLoad];
5
6     TAThread *aThread = [[TAThread alloc] initWithTarget:self selector:@selector(doSomething) object:nil];
7     [aThread start];
8
9 }
10
11 -(void doSomething {
12     NSLog(@"%s", __func__);
13 }
14
15 @end
```

```
22:14:59.923305+0800 RunLoopIO[3527:109195] -[ViewController doSomething]
22:14:59.923597+0800 RunLoopIO[3527:109195] -[TAThread dealloc]
```

根据打印结果我们可以看到，在子线程执行完任务后线程自动销毁。而我们有时需要经常在一个子线程中执行任务，频繁的创建和销毁线程就会造成资源浪费，这时候就要用到RunLoop来使线程长时间存活了，我们称之为线程保活

线程保活的操作

关于run方法的疑问

我们首先需要创建一个RunLoop，注意：RunLoop直接获取就可以，没有的话会在第一次获取时创建；如果RunLoop中的modes为空，或者要执行的mode里没有item，那么RunLoop会直接在当前RunLoop中返回，并进入睡眠状态。即doSomething改为：

```
1 -(void doSomething {
2     NSLog(@"%s", __func__);
3     [[NSRunLoop currentRunLoop] addPort:[(NSPort alloc) init] forMode:NSDefaultRunLoopMode];
4     // 添加RunLoop的回调方法
5     [[NSRunLoop currentRunLoop] run];
6     // 用于查看方法是否执行完毕
7     NSLog(@"ok");
8 }
```

```
785696+0800 RunLoopIO[3752:119512] -[ViewController doSomething]
```

根据打印结果可以知道，方法并没有执行完毕，我们希望的是在子线程执行完任务后就睡觉等待被唤醒，这样子写相当于虽然该任务执行完了但是RunLoop一直卡在这里，也不能去执行别任务。显而易见，这种方法是不够完美的。

RunLoop的启动方法

但是为什么执行了run方法以后就会一直卡在这里呢，我查了一些关于run方法的资料：RunLoop官方文档
官方文档中提到有三种启动RunLoop的方法：

方法名	介绍	中文翻译
run	Unconditionally	无条件
runUntilDate	With a set time limit	设定时间限制
runMode:beforeDate:	In a particular mode	在特定模式下

对于三种方法介绍总结如下：

- 无条件进入是最简单的做法，但也不推荐。这会使线程进入死循环，从而不利于控制RunLoop，结束RunLoop的唯一方式是kill它
- 如果我们设置了超时时间，那么RunLoop会在处理完事件或超时后结束，此时我们可以选择重新开启RunLoop。这种方式要优于前一种
- 这是相对来说最优秀的方式，相比于第二种启动方式，我们可以指定RunLoop以睡眠模式运行

查看run方法的文档还可以知道，它的本质就是无限调用runMode:beforeDate:方法，同样地，runUntilDate:也会重复调用runMode:beforeDate:，区别在于它超时后就不会再调用

也就是说，只有runMode:beforeDate:方法是单次调用，其他两种都是循环调用

RunLoop的关闭方法

在处理事件之前，有两种方法可以使运行循环退出：

- 设置超时时间
- 手动结束

如果你使用方法二或三来启动RunLoop，那么在启动的时候就可以设置超时时间，然而考虑到我们的目标是：“利用RunLoop进行线程保活”，所以我们希望对线程和它的RunLoop有最精确的控制，比如在完成任务后立刻结束，而不是依赖于超时机制

根据文档描述，我们有一个CFRunLoopStop()方法来手动结束一个RunLoop

注意：CFRunLoopStop()方法只会结束当前的runMode:beforeDate:调用，而不会结束后续的调用

线程保活的分析及代码

通过以上关于RunLoop启动和关闭的方法分析，我们大概有这样思路：

- 我们想要控制RunLoop，就需要使用runMode:beforeDate:方法，因为其他两种方法一个无法停止一个只能依赖超时机制
- CFRunLoopStop()方法只会结束当前的一次的runMode:beforeDate:方法调用，我们必须再做点什么

针对以上疑问，有以下解答：

- 首先，因为runMode:beforeDate:方法是单次调用，我们需要给它加上一个循环，否则调用一次就over了，和不使用RunLoop的效果大同小异
- 这个循环的条件可以默认设置为YES，当调用stop方法时，执行CFRunLoopStop()方法并且将循环条件改为NO，就可以使循环停止，RunLoop退出

代码长这样：

```
1 @interface ViewController ()
2 @property (strong, nonatomic) TAThread *aThread;
3 @property (assign, nonatomic, getter=IsStopped) BOOL stopped;
4
5 @end
6
7 @implementation ViewController
8
9 -(void)viewDidLoad {
10     [super viewDidLoad];
11     // 添加一个停止RunLoop的按钮
12     UIButton *stopButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
13     [self.view addSubview:stopButton];
14     stopButton.frame = CGRectMake(180, 100, 100, 50);
15     stopButton.titleLabel.font = [UIFont systemFontOfSize:20];
16     [stopButton setTitle:@"stop" forState:UIControlStateNormal];
17     stopButton.tintColor = [UIColor blackColor];
18     [stopButton addTarget:self action:@selector(stop) forControlEvents:UIControlEventTouchUpInside];
19
20     self.stopped = NO;
21     __weak typeOf(self) weakSelf = self;
22     self.aThread = [[TAThread alloc] initWithBlock:^(
23         NSLog(@"go");
24         [[NSRunLoop currentRunLoop] addPort:[(NSPort alloc) init] forMode:NSDefaultRunLoopMode];
25         while ([weakSelf.isStopped] == NO) {
26             [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode beforeDate:[NSDate distantFuture]];
27         }
28         NSLog(@"ok");
29     )];
30     [self.aThread start];
31 }
32
33 -(void)touchesBegan:(NSSet<UITouch> *)touches withEvent:(UIEvent *)event {
34     [self performSelector:@selector(doSomething) onThread:self.aThread withObject:nil waitUntilDone:NO];
35 }
36
37 // 子线程需要执行的任务
38 -(void)doSomething {
39     NSLog(@"%s %s", __func__, [NSThread currentThread]);
40 }
41
42 -(void)stop {
43     // 在子线程调用stop
44     [self performSelector:@selector(stopThread) onThread:self.aThread withObject:nil waitUntilDone:YES];
45 }
46
47 // 用于停止子线程的RunLoop
48 -(void)stopThread {
49     // 设置标志位
50     self.stopped = YES;
51
52     // 停止RunLoop
53     CFRunLoopStop(CFRunLoopGetCurrent());
54     NSLog(@"%s %s", __func__, [NSThread currentThread]);
55 }
56
57 -(void)dealloc {
58     NSLog(@"%s", __func__);
59 }
60 }
```

打印结果：

```
30 RunLoopIO[4791:167494] go
30 RunLoopIO[4791:167494] -[ViewController doSomething] <TAThread: 0x6000035b3300> (number = 3, name = (null))
30 RunLoopIO[4791:167494] -[ViewController doSomething] <TAThread: 0x6000035b3300> (number = 3, name = (null))
30 RunLoopIO[4791:167494] -[ViewController doSomething] <TAThread: 0x6000035b3300> (number = 3, name = (null))
30 RunLoopIO[4791:167494] -[ViewController stopThread] <TAThread: 0x6000035b3300> (number = 3, name = (null))
30 RunLoopIO[4791:167494] ok
```

后来又发现还存在一些问题，如果将上述代码写入一个NewViewController，当该ViewController已经dealloc时，线程并没有死，这就造成了内存泄漏。我这个线程是为了这个ViewController而活的，ViewController都死了，线程怎么能还活着呢，下面是修改后的

```
1 #import "NewViewController.h"
2 #import "TAThread.h"
3
4 @interface NewViewController ()
5
6 @property (strong, nonatomic) TAThread *aThread;
7 @property (assign, nonatomic, getter=IsStopped) BOOL stopped;
8
9 @end
10
11 @implementation NewViewController
12
13 -(void)viewDidLoad {
14     [super viewDidLoad];
15
16     self.view.backgroundColor = [UIColor whiteColor];
17
18     // 添加一个停止RunLoop的按钮
19     UIButton *stopButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
20     [self.view addSubview:stopButton];
21     stopButton.frame = CGRectMake(180, 100, 100, 50);
22     stopButton.titleLabel.font = [UIFont systemFontOfSize:20];
23     [stopButton setTitle:@"stop" forState:UIControlStateNormal];
24     stopButton.tintColor = [UIColor blackColor];
25     [stopButton addTarget:self action:@selector(stop) forControlEvents:UIControlEventTouchUpInside];
26
27     // 添加退出的按钮
28     UIButton *backButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
29     [self.view addSubview:backButton];
30     backButton.frame = CGRectMake(180, 380, 100, 50);
31     backButton.titleLabel.font = [UIFont systemFontOfSize:20];
32     [backButton setTitle:@"back" forState:UIControlStateNormal];
33     backButton.tintColor = [UIColor blackColor];
34     [backButton addTarget:self action:@selector(back) forControlEvents:UIControlEventTouchUpInside];
35
36     self.stopped = NO;
37     __weak typeOf(self) weakSelf = self;
38     self.aThread = [[TAThread alloc] initWithBlock:^(
39         NSLog(@"go");
40         [[NSRunLoop currentRunLoop] addPort:[(NSPort alloc) init] forMode:NSDefaultRunLoopMode];
41         while ([weakSelf.isStopped] == NO) {
42             [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode beforeDate:[NSDate distantFuture]];
43         }
44         NSLog(@"ok");
45     )];
46     [self.aThread start];
47
48     (void)touchesBegan:(NSSet<UITouch> *)touches withEvent:(UIEvent *)event {
49         [self performSelector:@selector(doSomething) onThread:self.aThread withObject:nil waitUntilDone:NO];
50     }
51
52 // 子线程需要执行的任务
53 -(void)doSomething {
54     NSLog(@"%s %s", __func__, [NSThread currentThread]);
55 }
56
57 -(void)stop {
58     if (self.aThread) {
59         // 在子线程调用stop
60         [self performSelector:@selector(stopThread) onThread:self.aThread withObject:nil waitUntilDone:YES];
61     }
62 }
63
64 // 用于停止子线程的RunLoop
65 -(void)stopThread {
66     // 设置标志位
67     self.stopped = YES;
68
69     // 停止RunLoop
70     CFRunLoopStop(CFRunLoopGetCurrent());
71     NSLog(@"%s %s", __func__, [NSThread currentThread]);
72 }
73 // 退出线程
74 self.aThread = nil;
75
76 -(void)dealloc {
77     NSLog(@"%s", __func__);
78 }
79
80 -(void)back {
81     [self stop];
82     [self dismissViewControllerAnimated:NO completion:nil];
83 }
84
85 }
```

参考文献

深入研究RunLoop与线程保活

iOS开发-使用RunLoop实现线程保活、线程常驻

打赏

文章很值，打赏犒劳作者一下

RunLoop和线程的关系 it5809284891的专栏 · 7301
RunLoop与线程是一一对应的，一个RunLoop对应一个核心的线程。为什么说核心的，是因为RunLoop是可以嵌套的，但是核心的只能有一个。对于主线程...
RunLoop完全指南 无话无话 · 8394

iOS的RunLoop，相信每个iOS开发都或多或少的听过，面试时也会经常作为压轴题的问题来问。之前写过RunLoop相关的内容，但是那写的太浅，基...
优质评论可以帮助作者获得更高权重 抢沙发 评论

相关推荐

RunLoop六在实际开发中的应用之 控制线程生命周期(线程保活) 二 6-9
而在selfThread线程使用的CFRunLoopStop(CFRunLoopGetCurrent());方法,不是停止run方法,而是 停止run方法里的一次循环(当前的RunLoop)...
基于RunLoop的线程保活、销毁与通信_Rio的博客 6-30

首先我们要明确一个概念,线程一般都是一次执行完任务,就销毁了。而添加了RunLoop,并运行起来,实际上是添加了一个do-while循环,这样这个线程...
RunLoop和多线程总结 大飞的专栏 · 2120
苹果用RunLoop实现的功能(部分): 1. AutoreleasePool 2. 监听和响应事件,如事件响应、手势识别、网络事件 3. UI更新 4. 定时器 5. PerformSele...

RunLoop与线程保活 a3031618的博客 · 63
代码: #import "ViewController.h" @interface ViewController () @property (nonatomic, weak) NSThread *thread; @end @implementation ViewController...

深入研究RunLoop与线程保活_g359798678的博客 6-28
在讨论RunLoop相关的文章,以及分析AFNetworking(2.x)源码的文章中,我们经常会看到关于利用RunLoop进行线程保活的分析,但如果不求甚解的话,极...

深入研究RunLoop与线程保活_aaa12317222的博客 7-9
在讨论RunLoop相关的文章,以及分析AFNetworking(2.x)源码的文章中,我们经常会看到关于利用RunLoop进行线程保活的分析,但如果不求甚解的话,极...

如何实现线程保活 wangshuo的博客 · 714
有两种方法: 第一种:提升优先级,降低进程被杀死的概率 线程的优先级 a.前台进程 b.可见进程 c.后台进程 d.后台进程 e.空进程 1.利用Activity提升权限...
RunLoop应用-线程保活 weixin_44824605的博客 · 114
在开发场景中,有可能需要对某条线程保活,让这些线程在有事情做的时候进行工作,没有事情做的时候进行休眠。线程保活创建线程和在线程获得ru...

RunLoop六在实际开发中的应用之 控制线程生命周期(线程保活) 6-7
OC的程序员大多数用过的AFNetwork这个网络请求框架。这个框架中就使用了RunLoop技术,去控制子线程的生命周期。相当于它创建了一个子线程...

基于RunLoop的线程保活、销毁与通信_Lu_Ca的博客 6-11
首先我们要明确一个概念,线程一般都是一次执行完任务,就销毁了。而添加了RunLoop,并运行起来,实际上是添加了一个do-while循环,这样这个线程...

iOS开发-使用RunLoop实现线程保活、线程常驻 u012338916的专栏 · 436
【源代码】保证线程的长时间存活。在iOS开发过程中,有时一些花费时间比较长的操作阻塞主线程,导致界面卡顿,那么咱们就会创建一个线程...

基于RunLoop的线程保活、销毁与通信 Phalm_JOS的博客 · 978
基于RunLoop的线程保活、销毁与通信 原文地址: http://www.jianshu.com/p/4d5b6fc33519 首先看一段AF2.x经典代码: + (NSThread *)networkReques...

iOS线程保活 kangpp的博客 · 1441
iOS线程保活的三种方案。目前从网上找到三种线程保活方案,前两种通过RunLoop实现,第三种通过条件锁实现。首先我们自己去创建KeyListener...

RunLoop-线程保活_zip 03-04
保持线程demo 利用RunLoop机制休眠机制来保持线程活跃 并且开发者可以自己控制线程dealloc

深入研究RunLoop与线程保活 weixin_342757348的博客 · 65
在讨论RunLoop相关的文章,以及分析AFNetworking(2.x)源码的文章中,我们经常会看到关于利用RunLoop进行线程保活的分析,但如果不求甚解的...

你了解RunLoop线程保活吗?已封装好2个代码直接使 Lucky_Deng的博客 · 62
如果没读过RunLoop相关的文章,以及分析AFNetworking的一些基础,建议你看看这篇2篇博客,使用RunLoop本质理解有很大帮助 中高iOS必备知识点之RunLoop(一) 源码解读RunLoop...

保活一个线程 ifranklin1374的博客 · 46
BOOL __block isAuthalFinish = NO; [BHCreateService(YDContactProtocol) requestContactAuthal:@"(BOOL success) { if (success) { } isAuthalFinish...

常驻线程的创建-线程不死之谜 peyou1239的博客 · 4432
主线程不死是因为主线程里面有一个RunLoop,RunLoop里面有一个do while死循环,保证了程序的不停退出 那么如果我们有一个需求,需要一直在后台运行...

Android进程保活(杀驻内存) ShareU的专栏 · 7688
Android进程分为6个等级,它们按优先级顺序由高到低依次是:1.前台进程(FOREGROUND_APP); 2.可进程(VISIBLE_APP); 3.次要服务进程(SEC...

Android 进程保活, Service进阶 u014302433的专栏 · 9908
关于 Android 平台的线程保活,这一直,想必是所有 Android 开发者瞩目的内容之一。你到网上搜 Android 进程保活,可以搜出各种各样神乎其技的魔法...

关于 Android 进程保活,你所需要知道的 不患无往,不患后患 · 421
http://www.jianshu.com/p/3aaf3c12af 作者 D_clock 2016.04.17 17:04 早前,我在知乎上回答了这样一个问题: 怎么让 Android 程序一直后台运行, ...

iOS 高级开发面试题面试进阶 05-13
<> 本篇旨在帮助读者面试 iOS 开发高级工程师为目标,从 runtime、RunLoop、消息转发、UI 绘制原理、算法策略和设计模式等角度,对面试中常...

© 2020 CSDN 皮肤主题: 像素格子 设计师:CSDN官方博客 返回顶部

关于我们 招贤纳士 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00
公安备案号11010502030143 京ICP备19004658号 京网文(2020) 1029-165号 经营性网站备案信息
北京互联网违法和不良信息举报中心 网络110报警服务 中国互联网举报中心 家长监护 Chrome商店下载

CSDN 首页 博客 专栏课程 下载 问答 社区 插件 认证 idea
登录/注册 会员中心 收藏 消息 创作中心
Tuski- 关注
1199 24 31 21 61
积分 粉丝 获赞 评论 收藏
私信 关注
热评文章
GCD简介 4593
iOS聊天室 简单的对话聊天界面 (Cell自适应高度) 2249
Pycharm Available Package无法显示/安装包的问题Error Loading Package List解决 2165
C语言-简易表达式求值 (栈的初步应用) 1855
UIImagePickerController (图像选取器) 使用方法 1813
最新评论
Pycharm Available Package无法显示/安装... weixin_44813194: 好文,感谢感谢
Runtime学习之weak原理 Tuski- 感谢推荐
Runtime学习之weak原理 金鸡路菜鸟: 太棒了
最新文章
NSNotificationCenter底层探究
URL Scheme delegate、NSNotification、KVO、block
2021年 6篇 2020年 34篇
2019年 42篇