原创 Tuski- 2020-08-02 00:17:59 1721 收藏 1 版权
分类专栏： iOS

文章目录

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

```
@implementation TAYThread

- (void)dealloc {
    NSLog(@"%s", __func__);
}

@end

```

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

```
@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    TAYThread *aThread = [[TAYThread alloc] initWithTarget:self selector:@selector(doSomething) object:nil
    [aThread start];

}

- (void)doSomething {
    NSLog(@"%s", __func__);
}

@end
```

```
22:14:59.923305+0800 RunLoopiOS[3527:109195] -[ViewController doSomething]
22:14:59.923597+0800 RunLoopiOS[3527:109195] -[TAYThread 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。
而我们有时会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：

```
 - (void)doSomething {
    NSLog(@"%s", __func__);
    [[NSRunLoop currentRunLoop] addPort:[[NSPort alloc] init] forMode:NSDefaultRunLoopMode];
    // run是RunLoop的启动方法
    [[NSRunLoop currentRunLoop] run];
    // 用于查看方法是否执行完毕
    NSLog(@"ok");
}
```

```
785696+0800 RunLoopiOS[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，也不能去执行别的任务。显而易见，这种办法是不够完美的。

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，但也最不推荐。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 以哪种模式运行

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，我们大概有这样一个思路：

1. 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
2. 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：

1. 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
2. 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：

```
@interface ViewController ()
@property (strong, nonatomic) TAYThread *aThread;
@property (assign, nonatomic, getter=isStoped) BOOL stopped;

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // 添加一个停止RunLoop的按钮
    UIButton *stopButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [self.view addSubview:stopButton];
    stopButton.frame = CGRectMake(180, 180, 100, 50);
    stopButton.titleLabel.font = [UIFont systemFontOfSize:20];
    [stopButton setTitle:@"stop" forState:UIControlStateNormal];
    stopButton.tintColor = [UIColor blackColor];
    [stopButton addTarget:self action:@selector(stop) forControlEvents:UIControlEventTouchUpInside];

    self.stopped = NO;
    __weak typeof(self) weakSelf = self;
    self.aThread = [[TAYThread alloc] initWithBlock:^{
        NSLog(@"go");
        [[NSRunLoop currentRunLoop] addPort:[[NSPort alloc] init] forMode:NSDefaultRunLoopMode];
        while (!weakSelf.isStoped) {
            [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode beforeDate:[NSDate distantFuture]];
        }
        NSLog(@"ok");
    }];
    [self.aThread start];
}

- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    [self performSelector:@selector(doSomething) onThread:self.aThread withObject:nil waitUntilDone:NO];
}

// 子线程需要执行的任务
- (void)doSomething {
    NSLog(@"%s %@", __func__, [NSThread currentThread]);
}

- (void)stop {
    // 在子线程调用stop
    [self performSelector:@selector(stopThread) onThread:self.aThread withObject:nil waitUntilDone:YES];
}

// 用于停止子线程的RunLoop
- (void)stopThread {
    // 设置标记为NO
    self.stopped = YES;

    // 停止RunLoop
    CFRunLoopStop(CFRunLoopGetCurrent());
    NSLog(@"%s %@", __func__, [NSThread currentThread]);
}

- (void)dealloc {
    NSLog(@"%s", __func__);
}

```

打印结果：

```
)0 RunLoopiOS[4791:167194] go
)0 RunLoopiOS[4791:167194] -[ViewController doSomething] <TAYThread: 0x6000035bb300>{number = 3, name = (null)}
)0 RunLoopiOS[4791:167194] -[ViewController doSomething] <TAYThread: 0x6000035bb300>{number = 3, name = (null)}
)0 RunLoopiOS[4791:167194] -[ViewController doSomething] <TAYThread: 0x6000035bb300>{number = 3, name = (null)}
)0 RunLoopiOS[4791:167194] -[ViewController stopThread] <TAYThread: 0x6000035bb300>{number = 3, name = (null)}
)0 RunLoopiOS[4791:167194] ok
```

留存的问题

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了。我这个线程是为了这个ViewController而活的，ViewController都死掉了，线程怎么能还活着呢，下面是修改后的

```
#import "NewViewController.h"
#import "TAYThread.h"

@interface NewViewController ()

@property (strong, nonatomic) TAYThread *aThread;
@property (assign, nonatomic, getter=isStoped) BOOL stopped;

@end

@implementation NewViewController

- (void)viewDidLoad {
    [super viewDidLoad];


    self.view.backgroundColor = [UIColor whiteColor];

    // 添加一个停止RunLoop的按钮
    UIButton *stopButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [self.view addSubview:stopButton];
    stopButton.frame = CGRectMake(180, 180, 100, 50);
    stopButton.titleLabel.font = [UIFont systemFontOfSize:20];
    [stopButton setTitle:@"stop" forState:UIControlStateNormal];
    stopButton.tintColor = [UIColor blackColor];
    [stopButton addTarget:self action:@selector(stop) forControlEvents:UIControlEventTouchUpInside];
    // 用于返回的按钮
    UIButton *backButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [self.view addSubview:backButton];
    backButton.frame = CGRectMake(180, 380, 100, 50);
    backButton.titleLabel.font = [UIFont systemFontOfSize:20];
    [backButton setTitle:@"back" forState:UIControlStateNormal];
    backButton.tintColor = [UIColor blackColor];
    [backButton addTarget:self action:@selector(back) forControlEvents:UIControlEventTouchUpInside];

    self.stopped = NO;
    __weak typeof(self) weakSelf = self;
    self.aThread = [[TAYThread alloc] initWithBlock:^{
        NSLog(@"go");
        [[NSRunLoop currentRunLoop] addPort:[[NSPort alloc] init] forMode:NSDefaultRunLoopMode];
        while (!weakSelf.isStoped) {
            [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode beforeDate:[NSDate distantFuture]];
        }
        NSLog(@"ok");
    }];
    [self.aThread start];
}

- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    [self performSelector:@selector(doSomething) onThread:self.aThread withObject:nil waitUntilDone:NO];
}

// 子线程需要执行的任务
- (void)doSomething {
    NSLog(@"%s %@", __func__, [NSThread currentThread]);
}

- (void)stop {
    if (self.aThread) {
        // 在子线程调用stop
        [self performSelector:@selector(stopThread) onThread:self.aThread withObject:nil waitUntilDone:YES
    }
}

// 用于停止子线程的RunLoop
- (void)stopThread {
    // 设置标记为NO
    self.stopped = YES;

    // 停止RunLoop
    CFRunLoopStop(CFRunLoopGetCurrent());
    NSLog(@"%s %@", __func__, [NSThread currentThread]);
    // 停止强引用
    self.aThread = nil;
}

- (void)dealloc {
    NSLog(@"%s", __func__);
}

- (void)back {
    // 调用线程停止方法
    [self stop];
    [self dismissViewControllerAnimated:NO completion:nil];
}

```

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