iOS10定时消息的改动

作者 Delpan (/u/3e54419b722d) + 关注
2016.10.08 13:44* 字数 2616 阅读 1496 评论 17 喜欢 23
(/u/3e54419b722d)

前言

iOS10已经发布了一段时间，iOS10的各种适配相信大家已经完成。本文将讲述的是关于iOS10内核的一个小改动，惯例，本文属于进阶性技术文，不会讲解API的使用，要求读者对RunLoop有一定的认知，感谢网友@送你的独白么 提供的SDK。

定时器

当我们的程序需要定时处理一些事件时，我们就会用到定时器，常用的定时器有NSTimer，CADisplayLink，GCD Timer，本文主要针对NSTimer和CADisplayLink进行讲述，因为这两者跟你的Application更为密切。

NSTimer和CADisplayLink都是建立在CFRunLoopTimer之上的抽象物，但有趣的是，苹果只提供了NSTimer和CFRunLoopTimer互转的Toll-Free Bridge，并没有提供CADisplayLink和CFRunLoopTimer互转的接口，因此一些开发者对此产生了一些猜想，有的人认为，CADisplayLink是用GCD Dispatch Source来实现的，有的人认为，CADisplayLink是用RunLoopSource来实现的，但这些猜想的依据都太容易被推翻了。如果CADisplayLink是用GCD Dispatch Source来实现的，那么CADisplayLink是怎么在你所创建的子线程中工作的呢？如果CADisplayLink是用RunLoopSource来实现的，会不会多此一举?

CFRunLoopTimer是RunLoop的定时源，与Source1(Port)一样，都属于端口事件源，但不同的是，每一个Source1都有与之对应的端口，而一个RunLoopMode中的所有CFRunLoopTimer共用一个端口(Mode Timer Port)，CFRunLoopTimer在RunLoop中的工作原理如下图。

定时源工作

从定时源在RunLoop中的工作原理我们得知，只要符合条件的定时器都会被触发，也就是说，在同一次Loop中，可能会执行几个定时器的回调。

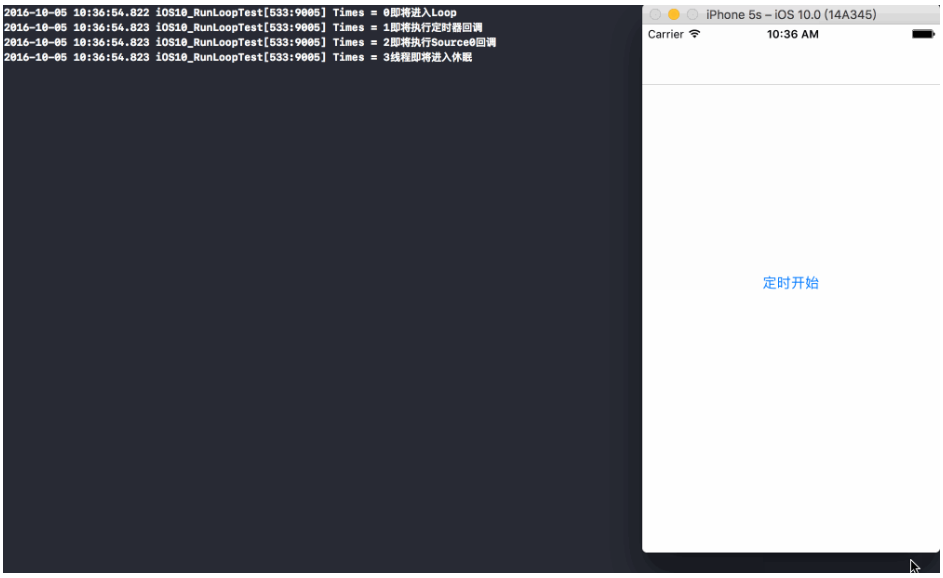
很多讲述定时器的技术文中都有这么一个观点，如果一个定时器错过了本次可以触发的时间点，那么定时器将跳过这个时间点，等待下一个时间点的到来，这个观点似乎是从官方文档中得来的，但这个观点跟定时器在RunLoop中的工作原理并不符。定时消息从内核发出，消息在消息中心等待被处理，RunLoop每次Loop都会去消息中心查找相应的端口消息，若找到相应的端口消息就会进行处理，所以，即使当前RunLoop正在执行一个耗时很长的任务，当任务执行完进入下一次Loop时，那些未被处理的消息仍然会被处理。经过大量测试表明，定时消息并不会因延迟而掉失。

关于RunLoop，官方文档在这一部份的勘误比较多，经常会出现文档的介绍跟源码不同的情况，所以想学习RunLoop的同学，建议看源码和自己做测试，特别是自己做测试。

NSTimer和CADisplayLink最大的区别在于信号的发射频率不同，CADisplayLink的发射频率固定在16.67ms一次，而NSTimer则可以自由定义。我在页面间跳转的性能优化(一) (http://www.jianshu.com/p/77847c0027c9)中曾经提到过，不是必要的情况下，都不要选择使用CADisplayLink作为定时器，因为它会使目标RunLoop一直处理活跃状态。下面通过一个例子来看看实际的效果，创建一个CADisplayLink定时器，设置为100秒后触发，然后观察目标RunLoop的状态。

CADisplayLink

从实际效果我们可以看到，目标RunLoop一直处于活跃状态，不断地处理内核发出的信号，直到RunLoop Stop或CADisplayLink定时器被移除。同样的条件，我们把定时器换成NSTimer来观察实际情况。



NSTimer

与CADisplayLink的固定信号不同，NSTimer的信号间隔完全是由使用者来定义。所以，除非你需要实现定时动画，不然都不要选择使用CADisplayLink作为定时器，它不仅会损耗大量的CPU资源，还会影响目标RunLoop处理其它事件源。

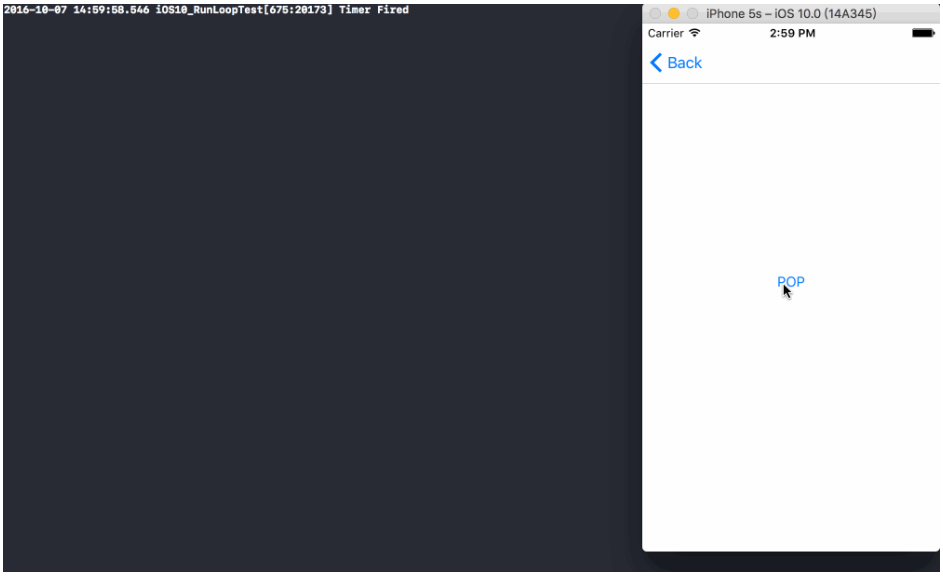
改动

前面介绍了定时器的工作原理，现在来看看实际的改动，从一个例子入手进行讲述。现在有页面A，B，页面A，B各有一个按钮，页面A的按钮用来进入页面B，进入页面B后创建一个子线程，然后向子线程添加一个定时器并启动RunLoop，页面B的按钮用于停止

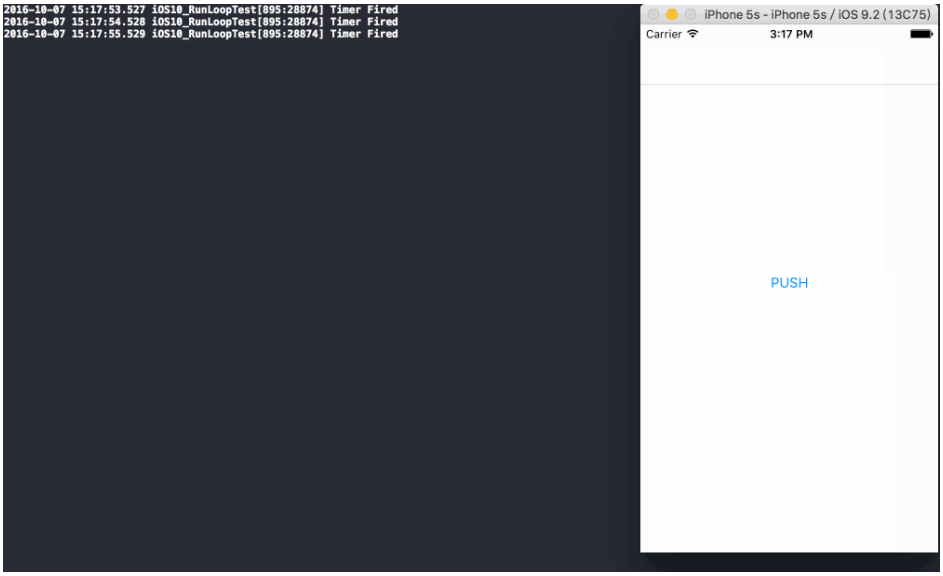
定时器，并返回页面A，页面B被释放时会在dealloc方法里输出dealloc，编译环境是ARC，下图为页面B的代码，Gif图分别是iOS10与iOS9的实际运行效果。

```
    UIButton *button = [UIButton buttonWithType:UIButtonTypeSystem];
    button.frame = CGRectMake(110, 260, 100, 35);
    [button setTitle:@"POP" forState:UIControlStateNormal];
    [button addTarget:self action:@selector(buttonAction:) forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:button];

    [NSThread detachNewThreadSelector:@selector(secondaryThreadMain) toTarget:self withObject:nil];
}

#pragma mark - SecondaryFunction
- (void)secondaryThreadMain
{
    _timer = [NSTimer scheduledTimerWithTimeInterval:1.f
                                              target:self
                                            selector:@selector(timerAction:)
                                            userInfo:nil
                                             repeats:YES];
    CFRunLoopRun();
}

#pragma mark - Action
#pragma mark -Button Action
- (void)buttonAction:(UIButton *)sender
{
    [_timer invalidate];
    [self.navigationController popViewControllerAnimated:YES];
}

#pragma mark -Timer Action
- (void)timerAction:(id)sender
{
    NSLog(@"Timer Fired");
}

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

页面B代码



iOS10



iOS9

一般情况下，从页面B返回到页面A后，页面B会被释放，页面B的dealloc方法会输出dealloc，但从实际的运行效果可以看到，在iOS10环境下页面B并没有被释放，WTF，为什么iOS10环境下会这样？要回答这个问题，我们需要先知道iOS10的改动是什么。

若目标RunLoop当前没有定时源需要处理（像上面的例子那样，子线程RunLoop只有一个定时器，该定时器移除后，则子线程RunLoop没有定时源需要处理），则通知内核不需要再向当前Timer Port发送定时消息并移除该Timer Port。在iOS10环境下，当移除Timer Port后，内核会把消息列表中与该Timer Port相应的定时消息移除，而iOS10以前的环境下，当移除Timer Port后，内核不会把消息列表中与该Timer Port相应的定时消息移除。iOS10的处理是更为合理的，iOS10以前的处理可能是历史遗留问题吧。

看回上面的例子，例子中遇到的问题是页面B返回后并没有被释放，即页面B的内存被强制保留了，所以我们现在需要知道的是页面B为什么被强制保留了。在页面B中我们创建了一个子线程，子线程的主函数是页面B的对象函数，这可能是导致页面B被强制保留的原因，所以，我们需要知道子线程开启前后，页面B对象的引用计数是否有增加。

```
[NSThread detachNewThreadSelector:@selector(secondaryThreadMain) toTarget:self withObject:nil];
```

创建并开启子线程

```
2016-10-08 11:04:26.658 iOS10_RunLoopTest[638:20280] 页面B引用计数: 8
2016-10-08 11:04:26.659 iOS10_RunLoopTest[638:20280] Secondary Thread Started
2016-10-08 11:04:26.659 iOS10_RunLoopTest[638:20280] 页面B引用计数: 9
```

页面B的引用计数

从输出的信息我们得知，创建子线程后，Target会被强制保留，直到子线程的主函数返回。引用计数在很多时候可以帮助我们了解内存的使用情况，但在ARC编译环境下，我们无法直接使用retainCount方法来获取一个对象的引用计数，所以，我们需要做额外的处理。

```
#pragma mark - 获取当前对象的引用计数
- (NSInteger)currentRetainCount
{
    return CFGetRetainCount((__bridge void *)self);
}
```

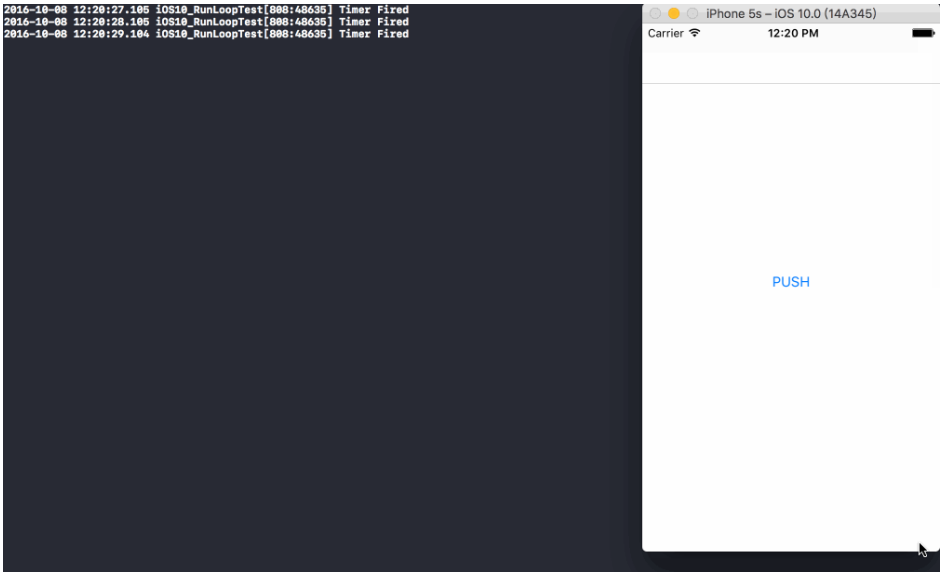
获取对象的引用计数

回到例子中，我们知道了页面B被强制保留的原因后，就知道了怎么解决，只需要退出子线程即可，子线程之所以可以一直存活，是因为启动了RunLoop，所以，我们只需要退出RunLoop，子线程的主函数就会返回。例子中涉及到线程异步的问题，定时器是在子线程RunLoop中注册的，但定时器的移除操作却是在主线程，由于子线程RunLoop处理完一次定时信号后，就会进入休眠状态。在iOS10以前的环境下，定时器被移除后，内核仍然会向对应的Timer Port发送一次信号，所以子线程RunLoop接收到信号后会被唤醒，由于没有定时源需要处理，所以RunLoop会直接跳转到判断阶段，判断阶段会检测当前RunLoopMode是否有事件源需要处理，若没有事件源需要处理，则会退出RunLoop。由于例子中子线程RunLoop的当前RunLoopMode只有一个定时器，而定时器被移除后，RunLoopMode就没有了需要处理的事件源，所以会退出RunLoop，子线程的主函数也因此返回，页面B对象被释放。

但在iOS10环境下，当定时器被移除后，内核不再向对应的Timer Port发送任何信号，所以子线程RunLoop一直处于休眠状态并没有退出，而我们只需要手动唤醒RunLoop即可。

```
    self.view.backgroundColor = [UIColor whiteColor];

    UIButton *button = [UIButton buttonWithType:UIButtonTypeSystem];
    button.frame = CGRectMake(110, 260, 100, 35);
    [button setTitle:@"POP" forState:UIControlStateNormal];
    [button addTarget:self action:@selector(buttonAction:) forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:button];

    [NSThread detachNewThreadSelector:@selector(secondaryThreadMain) toTarget:self withObject:nil];
}

#pragma mark - SecondaryFunction
- (void)secondaryThreadMain
{
    _timer = [NSTimer scheduledTimerWithTimeInterval:1.f
                                              target:self
                                            selector:@selector(timerAction:)
                                            userInfo:nil
                                             repeats:YES];

    _runLoop = CFRunLoopGetCurrent();
    CFRunLoopRun();
}

#pragma mark - Action
#pragma mark -Button Action
- (void)buttonAction:(UIButton *)sender
{
    [_timer invalidate];
    CFRunLoopWakeUp(_runLoop);

    [self.navigationController popViewControllerAnimated:YES];
}

#pragma mark -Timer Action
- (void)timerAction:(id)sender
{
    NSLog(@"Timer Fired");
}
```

更改页面B代码



iOS10

例子中所遇到的问题已经解决，但看完这个例子，可能你会有疑问，这个例子讲述的情况有实战意义？这个例子是从一个国外成熟产品所提供的配套SDK中简化而来，配套的SDK用于与产品进行对接。额.....实话说，当我看到这个处理方式的时候，我被震惊了，没想到一个成熟产品所提供的配套SDK会出现这样的问题，让我更震惊的是，随后在其它SDK中也发现了这个问题，这.....

我们回头来看看例子中的处理方式，例子中，子线程RunLoop的退出依赖于RunLoopMode的事件源为空，这种RunLoop的退出方式是极不稳定的，因为系统有很多API会向目标RunLoopMode添加额外的事件源来处理系统事件的，所以这种方式是不能确保一定可以退出RunLoop的。正确的方式应该是配对调用CFRunLoopRun( )，CFRunLoopStop( )来启动和退出RunLoop，需要注意的是，除非你要创建一个单例线程，不然不要使用[runloop run]方法来启动RunLoop，因为使用run方法启动RunLoop后，唯一退出RunLoop的方式是当前RunLoopMode的事件源为空，而我们知道这种方式本身是极不稳定的。

送你的独白么 (/u/5427e81393ab)
3楼 · 2016.10.08 13:57
(/u/5427e81393ab)
我也来

赞 回复

880bc798dc5f (/u/880bc798dc5f)
4楼 · 2016.10.08 14:15
(/u/880bc798dc5f)
果然是大雕！

赞 回复

sindri的小巢 (/u/0cf7d455eb9e)
5楼 · 2016.10.08 14:45
(/u/0cf7d455eb9e)
表哥出手，不同凡响

赞 回复

sindri的小巢 (/u/0cf7d455eb9e)
6楼 · 2016.10.08 14:54
(/u/0cf7d455eb9e)
换句话说，实际上在定时器启动的时候，引用target的并不是定时器，而是NSRunLoop

赞 回复

姜谷子的姜 (/u/a3bbd5a74b8b)
7楼 · 2016.10.08 17:27
(/u/a3bbd5a74b8b)
渣神你好

赞 回复

> Delpan (/u/3e54419b722d)： @姜谷子的姜 (/users/a3bbd5a74b8b) 姜大神好
> 2016.10.08 21:21 回复
>
> 添加新评论

陈阿票 (/u/2db58466d30f)
8楼 · 2016.10.08 17:30
(/u/2db58466d30f)
子线程RunLoop的退出依赖于RunLoopMode的事件源为空，这种RunLoop的退出方式是极不稳定的。正确的方式应该是配对调用CFRunLoopRun( )，CFRunLoopStop( )来启动和退出RunLoop。所以最终是要把 buttonAction 中的 CFRunloopWakeUp(_runloop); 替换为CFRunLoopStop( ) 么？但是如果系统真的向目标RunLoopMode添加额外的事件源来处理系统事件，那么不是会造成其他意想不到的后果？

赞 回复

> Delpan (/u/3e54419b722d)： @陈阿票 (/users/2db58466d30f) 看你本身是否要退出线程，如果是，系统事件对于你当前线程而言已经没有意义
> 2016.10.08 17:46 回复
>
> 陈阿票 (/u/2db58466d30f)： @Delpan (/users/3e54419b722d) 恩，是的。其实就算使用 CFRunloopWakeUp(_runloop);也没有什么关系，系统添加的额外的事件执行完之后，runloop 里也就没有了其他 source，自然也会自动销毁。除非系统的事件一直占用着。
> 2016.10.08 18:00 回复

> Delpan (/u/3e54419b722d)： @陈阿票 (/users/2db58466d30f) 主要看你为什么要自己创建子线程并开启RunLoop，系统添加的事件源大多数是不会被移除的，但是在你目的达到后，子线程就没有存在的意义了，所以系统的事件源完全可以不用管了
> 2016.10.08 19:03 回复
>
> 添加新评论

y_xh (/u/d8035de43505)
10楼 · 2016.11.01 23:54
(/u/d8035de43505)
学习了👍

赞 回复

DanTeBao (/u/39ca768d0039)
11楼 · 2016.11.09 13:47
(/u/39ca768d0039)
啊欧表哥！mark！

赞 回复

老司机Wicky (/u/a56ec10f6603)
12楼 · 2016.11.22 02:43
(/u/a56ec10f6603)
真正的大神，受益匪浅，求群号

赞 回复

> Delpan (/u/3e54419b722d)： @老司机Wicky (/users/a56ec10f6603) 小菜B的我没开群😂😂
> 2016.11.22 08:51 回复
>
> 添加新评论

ljysdfz (/u/9658b4413ae7)
13楼 · 2016.11.29 15:36
(/u/9658b4413ae7)
实战经验丰富

赞 回复

厨神小W空男子O (/u/479e60e40fc5)
14楼 · 2017.01.17 15:31
(/u/479e60e40fc5)
大表哥666

赞 回复

被以下专题收入，发现更多相似内容