iOS NSTimer 最佳实践

2016-10-14 wison-苹果君 QQ空间开发团队

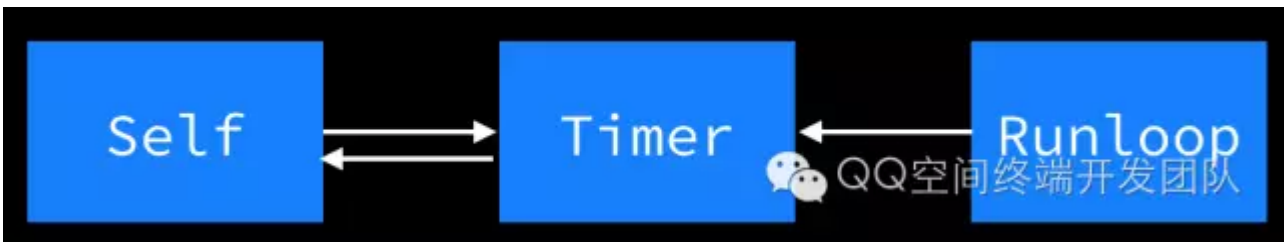
NSTimer是iOS上的一种计时器，通过NSTimer对象，可以指定时间间隔，向一个对象发送消息。NSTimer是比较常用的工具，比如用来定时更新界面，定时发送请求等等。但是在使用过程中，有很多需要注意的地方，稍微不注意就会产生bug, crash, 内存泄漏。本文讲解了使用NSTimer时需要注意的问题。

1. NSTimer 容易泄漏

比如以下代码创建了一个计时器：

```
self.timer =
  [NSTimer scheduledTimerWithTimeInterval:1
           target:self
           selector:@selector(update)
           userInfo:nil
           repeats:YES];
```

上述代码，将创建一个无限循环的timer, 并投入当前线程的Runloop中开始执行。此时，Runloop会引用住timer, timer会引用住self, self则保存了timer. 如下图所示：



需要注意的是，这种无限循环的timer, 会一直执行，需要调用`[timer invalidate]`显式停止。否则runloop会一直引用着timer, timer又引用了self, 导致self整个对象泄漏，实际情况中，这个self有可能是一个view, 甚至是一个controller.

那，`[timer invalidate]`要什么时候调用？有些人会在self的dealloc里面调用，这几乎可以确定是错误的。因为timer会引用住self，在timer停止之前，是不会释放self的，self的dealloc也不可能会被调用。

正确的做法应该是根据业务需要，在适当的地方启动timer和停止timer. 比如timer是页面用来更新页面内部的view的，那可以选择在页面显示的时候启动timer，页面不可见的时候停止timer. 比

如：

```
- (void)viewWillAppear
{
  [super viewWillAppear];  self.timer =
    [NSTimer scheduledTimerWithTimeInterval:1
             target:self
             selector:@selector(update)
             userInfo:nil
             repeats:YES];
}

- (void)viewDidDisappear
{
  [super viewDidDisappear];
  [self.timer invalidate];
}
```

2. 错误特征

实际开发中，或者Code Review的时候，可以通过一些特征初步判定可能会有问题。

错误特征 1:

```
- (void)dealloc
{
  [self.timer invalidate];
}
```

以上代码是有问题的。当timer没有停止的时候，self会被引用，也就没有机会走到dealloc. 同时，代码作者应该对timer没有正确的认识，所以需要review整个timer的使用情况。

错误特征 2:

```
[NSTimer scheduledTimerWithTimeInterval:1
         target:self
         selector:@selector(update)
```

```
            userInfo:nil
            repeats:YES];
```

以上代码创建了一个timer，但是没有保存起来，后续自然也没有机会停止这个 timer. 所以会导致timer泄漏。

错误特征 3:

```
- (void)viewDidAppear:(BOOL)animated
{
  [super viewDidAppear:animated];  self.timer =
    [NSTimer scheduledTimerWithTimeInterval:1
             target:self
             selector:@selector(update)
             userInfo:nil
             repeats:YES];
}
```

以上代码也是有问题的。因为我们要确保timer的创建和销毁必须是成对调用，否则会发生泄漏。而对于viewDidAppear其实很难找到一个准确的与之成对的方法（跟viewWillDisappear和viewDidDisappear都不是成对调用的），这里就需要检查timer有没有被重复创建和有没有在适当的时机销毁。

3. 停止 timer 可能会导致 self 对象销毁

值得注意的是，调用`[timer invalidate]`停止timer，此时timer会释放 target, 如果timer是最后一个持有target的对象，那么此次释放会直接触发target的 。比如：

```
- (void)onEnterBackground:(id)sender
{
    [self.timer invalidate];
    [self.view stopAnimation]; // dangerous!}
```

以上代码，加入第一行的invalidate之后，self被销毁了，那么第二行访问self.view时候，就会触发野指针crash。因为Objective-C的方法里面，self是没有被retain的。这种情况，有个临时的解决方案如下：

```
- (void)onEnterBackground:(id)sender
{
    __weak id weakSelf = self;
    [self.timer invalidate];
    [weakSelf.view stopAnimation]; // dangerous!}
```

将self改为弱引用。但是也是一个临时解决方案。正确解决方法是，查出其它对象没有引用self的时候，为什么timer还没停止。这个案例告诉大家，当见到 invalidate被调用之后很神奇地出现了self野指针crash的时候，不要惊讶，就是timer没处理好。

4. Perform Delay

`[NSObject performSelector:withObject:`

`afterDelay:]` 和 `[NSObject performSelector`

`:withObject:afterDelay:inMode:]` 我们简称

为Perform Delay, 他们的实现原理就是一个不循环（repeat 为 NO）的timer. 所以使用这两个接口的注意事项跟使用timer类似。所以使用这两个接口的注意事项跟使用 timer 类似。需要在适当的地方调用 `[NSObject cancelPreviousPerform`

`RequestsWithTarget:selector:object:]`

5. Runloop Mode

注意创建NSTimer或者调用Perform Delay方法，都是往当前线程的Runloop 中投递消息，大部分接口的默认投递模式是CFRunloopDefaultMode. 也就是说，Runloop不在DefaultMode下运行的时候（比如滚动列表的时候主线程的runloop mode是CFRunloopTrackingMode），消息将被暂时阻塞，不能及时处理。

6. Weak Timer

NSTimer之所以比较难用对，比较重要的原因主要是NSTimer对target是强引用的。这导致了target泄漏，或者生命周期超出开发者的预期。timer如果对target是弱引用的话，这些问题就不存在了，这就是Weak Timer.
Weak Timer的实现方式分为两种，第一种是在NSTimer和target中间加多一层代理(Proxy)，代理作为target被NSTimer强引用，同时弱引用真正的target，并对它转发消息。示例图如下：

```
+ (NSTimer *)qz_scheduledWeakTimerWithTimeInterval:
(NSTimeInterval)ti target:(id)target selector:
(SEL)selector userInfo:(id)userInfo repeats:(BOOL)repeats
{
    QzoneWeakProxy *proxy = [[QzoneWeakProxy weakProxyForObject:target];
    return [self scheduledTimerWithTimeInterval:
    ti target:proxy selector:aSelector userInfo:userInfo repeats:repeats];
}
```

第二种方案是用dispatch timer自己实现一遍timer, 具体实现里面，弱引用 target.

比如这个：

https://github.com/mindsnacks/MSWeakTimer

原文地址：

http://wisonlin.github.io/2016/05/14/NSTimer-使用进阶