Runloop

什么是 Runloop?

- 从字面上讲就是运行循环。
- 它内部就是do-while循环，在这个循环内部不断地处理各种任务。
- 一个线程对应一个RunLoop，主线程的RunLoop默认已经启动，子线程的RunLoop得手动启动（调用run方法）
- RunLoop只能选择一个Mode启动，如果当前Mode中没有任何Source(Sources0、Sources1)、Timer，那么就直接退出RunLoop
- 基本的作用就是保持程序的持续运行，处理app中的各种事件。通过runloop，有事运行，没事就休息，可以节省cpu资源，提高程序性能。

Runloop对象

iOS中有2套API来访问和使用RunLoop

- Foundation：NSRunLoop
- Core Foundation：CFRunLoopRef
- NSRunLoop和CFRunLoopRef都代表着RunLoop对象
- NSRunLoop是基于CFRunLoopRef的一层OC包装，所以要了解RunLoop内部结构，需要多研究CFRunLoopRef层面的API。

Runloop与线程

- 每条线程都有唯一的一个与之对应的RunLoop对象
- 主线程的RunLoop已经自动创建好了，子线程的RunLoop需要主动创建
- RunLoop在第一次获取时创建，在线程结束时销毁

获得RunLoop对象

- Foundation

[NSRunLoop currentRunLoop]; // 获得当前线程的RunLoop对象

[NSRunLoop mainRunLoop]; // 获得主线程的RunLoop对象

- Core Foundation

CFRunLoopGetCurrent(); // 获得当前线程的RunLoop对象

CFRunLoopGetMain(); // 获得主线程的RunLoop对象

RunLoop相关类

```
Core Foundation中关于RunLoop的5个类
CFRunLoopRef
CFRunLoopModeRef
CFRunLoopSourceRef
CFRunLoopTimerRef
CFRunLoopObserverRef
```

CFRunLoopModeRef

```
CFRunLoopModeRef代表RunLoop的运行模式。
一个RunLoop包含若干个Mode，每个Mode又包含若干个(set)Source/(array)Timer/(array)Observer

每次RunLoop启动时，只能指定其中一个 Mode，这个Mode被称作CurrentMode

如果需要切换Mode，只能退出Loop，再重新指定一个Mode进入

这样做主要是为了分隔开不同组的Source/Timer/Observer，让其互不影响

mode主要是用来指定事件在运行循环中的优先级的，分为：
•   NSDefaultRunLoopMode（kCFRunLoopDefaultMode）：默认，空闲状态
•   UITrackingRunLoopMode：ScrollView滑动时会切换到该Mode
•   UIInitializationRunLoopMode：run loop启动时，会切换到该mode
•   NSRunLoopCommonModes（kCFRunLoopCommonModes）：Mode集合
苹果公开提供的Mode有两个：
•   NSDefaultRunLoopMode（kCFRunLoopDefaultMode）
•   NSRunLoopCommonModes（kCFRunLoopCommonModes）
```

CFRunLoopTimerRef

- CFRunLoopTimerRef是基于时间的触发器

- CFRunLoopTimerRef基本上说的就是NSTimer，它受RunLoop的Mode影响

- GCD的定时器不受RunLoop的Mode影响

CFRunLoopSourceRef

```
CFRunLoopSourceRef是事件源（输入源）
按照官方文档，Source的分类
Port-Based Sources
Custom Input Sources
Cocoa Perform Selector Sources
按照函数调用栈，Source的分类
Source0：非基于Port的
Source1：基于Port的，通过内核和其他线程通信，接收、分发系统事件
```

CFRunLoopObserverRef

- CFRunLoopObserverRef是观察者，能够监听RunLoop的状态改变
- 可以监听的时间点有以下几个
  - kcfRunLoopEntry(即将进入loop)//1
  - kcfRunLoopBeforeTimers(即将处理timer)//2
  - kcfRunLoopBeforeSources(即将处理source)//4
  - kcfRunLoopBeforeWaiting(即将进入休眠)//32
  - kcfRunLoopAfterWaiting(刚从休眠中唤醒)//64
  - kcfRunLoopExit(即将退出loop)//128
- 添加Observer

```
    CFRunLoopObserverRef observer = CFRunLoopObserverCreateWithHandler(CFAllocatorGetDefault(), kCFRunLo
    NSLog(@"----监听到RunLoop状态发生改变---%zd", activity);
});

// 添加观察者：监听RunLoop的状态
CFRunLoopAddObserver(CFRunLoopGetCurrent(), observer, kCFRunLoopDefaultMode);

// 释放Observer
CFRelease(observer);
```

RunLoop处理逻辑

- 通知Observer:即将进入Loop（1）
- 通知Observer：将要处理Timer（2）
- 通知Observer：将要处理Source0（3）
- 处理Source0（4）
- 如果有Source0，跳到第9步（5）
- 通知Observer：线程即将休眠（6）
- 休眠，等待唤醒：（7）
  - Source0(port)。
  - timer启动
  - RunLoop设置的timer已经超时
  - Runloop被外部手动唤醒
- 通知Observer：线程将被唤醒（8）
- 处理未处理的时间（9）
  - 如果用户定义的定时器启动，处理定时器事件并重启Runloop。进入步骤2.
  - 如果输入源启动，传递相应的消息。
  - 如果RunLopp被显式唤醒而且时间还没超时，重启RunLoop，进入步骤2.
- 通知Observer：即将退出Loop

Runloop的应用

- NSTimer
- ImageView显示
- PerformSelector
- 常驻线程
- 自动释放池

runloop定时源和输入源

image

- Runloop处理的输入事件有两种不同的来源：输入源（input source）和定时源（timer source）
- 输入源传递异步消息，通常来自于其他线程或者程序。
- 定时源则传递同步消息，在特定时间或者一定的时间间隔发生

NSRunLoop的实现机制,及在多线程中如何使用

```
- 实现机制：回答runloop的基本作用，处理逻辑，前面都有。

- 程序创建子线程的时候，才需要手动启动runloop。主线程的runloop已经默认启动。

- 在多线程中，你需要判断是否需要runloop。如果需要runloop，那么你要负责配置runloop并启动。你不需要在任何情况下都
```

runloop和线程有什么关系?

```
- 主线程的run loop默认是启动的。
iOS的应用程序里面，程序启动后会有一个如下的main()函数
( argc,  * argv[]) {
@autoreleasepool {
    return UIApplicationMain(argc, argv, , NSStringFromClass([AppDelegate class]));
}
}
重点是UIApplicationMain()函数，这个方法会为main thread设置一个NSRunLoop对象，这就解释了：为什么我们的应用可以在

- 对其它线程来说，runloop默认是没有启动的，runloop只在你要和线程有交互时才需要。

- 在任何一个 Cocoa 程序的线程中，都可以通过以下代码来获取到当前线程的 run loop 。
NSRunLoop *runloop = [NSRunLoop currentRunLoop];
```

autorelease 对象在什么情况下会被释放?

- 分两种情况：手动干预释放和系统自动释放
- 手动干预释放就是指定autoreleasepool,当前作用域大括号结束就立即释放
- 系统自动去释放:不手动指定autoreleasepool,Autorelease对象会在当前的 runloop 迭代结束时释放
- kCFRunLoopEntry(1):第一次进入会自动创建一个autorelease
- kCFRunLoopBeforeWaiting(32):进入休眠状态前会自动销毁一个autorelease,然后重新创建一个新的autorelease
- kCFRunLoopExit(128):退出runloop时会自动销毁最后一个创建的autorelease

对于runloop的理解不正确的是

```
A 每一个线程都有其对应的RunLoop
B 默认非主线程的RunLoop是没有运行的
C 在一个单独的线程中没有必要去启用RunLoop
D 可以将NSTimer添加到runloop中
```

- 参考答案：C
- 理由：说到RunLoop，它可是多线程的法定。通常来说，一个线程一次只能执行一个任务，执行完任务后就会退出线程。但是，对于主线程是不能退出的，因此我们需要让主线程即时任务执行完毕，也可以继续等待是接收事件而不退出，那么RunLoop就是关键法宝了。但是非主线程通常来说就是为了执行某一任务的，执行完毕就需要归还资源，因此默认是不运行RunLoop的。NSRunLoop提供了一个添加NSTimer的方法，这个方法是在应用正常状态下会回调。

runloop的mode作用是什么?

```
mode主要是用来指定事件在运行循环中的优先级的，分为：
•   NSDefaultRunLoopMode（kCFRunLoopDefaultMode）：默认，空闲状态
•   UITrackingRunLoopMode：ScrollView滑动时会切换到该Mode
•   UIInitializationRunLoopMode：run loop启动时，会切换到该mode
•   NSRunLoopCommonModes（kCFRunLoopCommonModes）：Mode集合
苹果公开提供的Mode有两个：
•   NSDefaultRunLoopMode（kCFRunLoopDefaultMode）
•   NSRunLoopCommonModes（kCFRunLoopCommonModes）
如果我们把一个NSTimer对象以NSDefaultRunLoopMode（kCFRunLoopDefaultMode）添加到主运行循环中的时候，ScrollView
```

请写出NSTimer使用时的注意事项（两项即可）

```
思路和上一题一样，如果想要销毁timer，则必须先将timer置为失效，否则timer就一直占用内存而不会释放。造成逻辑上的内存
参考答案：
•   注意timer添加到runloop时应该设置为什么mode
•   注意timer在不需要时，一定要调用invalidate方法使定时器失效，否则得不到释放
```

UITableViewCell上有个UILabel，显示NSTimer实现的秒表时间，手指滚动cell过程中，label是否刷新，为什么?

```
和上一题一样的思路，如果要cell滚动过程中定时器正常回调，UI正常刷新，那么要将timer放入到CommonModes下，因为是NSDe
```

为什么 UIScrollView 的滚动会导致 NSTimer 失效?

- 思路和上一题一样，解决办法有2个,一个是更改mode为NSRunLoopCommonModes(无论runloop运行在哪个mode,都能运行),还有种办法是切换到主线程来更新UI界面的刷新

```
//将timer添加到NSDefaultRunLoopMode中
[NSTimer scheduledTimerWithTimeInterval: target: selector:@selector(timerTick:) userInfo: repeats:];
 //然后再添加到NSRunLoopCommonModes里
  NSTimer *timer = [NSTimer timerWithTimeInterval: target: selector:@selector(timerTick:) userInfo: rep
   [[NSRunLoop currentRunLoop] addTimer:timer forMode:NSRunLoopCommonModes];
```

在滑动页面上的列表时，timer会暂定回调，为什么？如何解决?

- 思路和上一题一样

在开发中如何使用RunLoop？什么应用场景?

- 开启一个常驻线程（让一个子线程不进入消亡状态，等待其他线程发来消息，处理其他事件）
- 在子线程中开启一个定时器
- 在子线程中进行一些长期监控
- 可以控制定时器在特定模式下执行
- 可以让某些事件（行为、任务）在特定模式下执行
- 可以添加Observer监听RunLoop的状态，比如监听点击事件的处理（在所有点击事件之前做一些事情）

文章如有问题，请留言，我将及时更正。

满地打滚卖萌求赞，如果本文帮助到你，轻点下方的红心，给作者君增加更新的动力。