orilme Lv2
2019年04月18日 阅读 696

关注

iOS Runloop

一、Runloop 简介

1. 简介

- RunLoop就是让线程随时处理事件但不退出的机制
- 每一个线程都有一个RunLoop
- RunLoop 实际上就是一个对象，这个对象管理了其需要处理的事件（比如button的点击、各种手势的的事件、定时器、tableView的代理方法）和消息，是iOS里的一种事件处理机制。
- 线程执行了这个函数后，就会一直处于这个函数内部 “接受消息->等待->处理” 的循环中，直到这个循环结束（比如传入 quit 的消息），函数返回。

2. 基本作用

- 保持程序的持续运行(比如主运行循环)
- 处理App中的各种事件（比如触摸事件、定时器事件、Selector事件）
- 节省CPU资源，提高程序性能：该做事时做事，该休息时休息

3. API

OSX / iOS 系统中，有2套API来访问和使用 `RunLoop` 。

- `CFRunLoopRef` 是在 `CoreFoundation` 框架内的，它提供了纯 C 函数的 API，所有这些 API 都是线程安全的。
- `NSRunLoop` 是基于 `CFRunLoopRef` 的封装，提供了面向对象的 API，但是这些 API 不是线程安全的。所以要了解 `RunLoop` 内部结构，需要多研究 `CFRunLoopRef` 层面的API（`Core Foundation` 层面）

`NSRunLoop` 和 `CFRunLoopRef` 都代表着 `RunLoop` 对象。

4. 存在价值

`main` 函数中的 `RunLoop`（主运行循环）：第14行代码的 `UIApplicationMain` 函数内部就启动了一个 `RunLoop`。所以 `UIApplicationMain` 函数一直没有返回，保持了程序的持续运行。这个默认启动的 `RunLoop` 是跟主线程相关联的。

```
12 int main(int argc, char * argv[]) {
13     @autoreleasepool {
14         return UIApplicationMain(argc, argv, nil, NSStringFromClass
               ([AppDelegate class]));
15     }
16 }
```

二、Runloop 解析

1. Runloop 运行模式

一种 Runloop 运行模式就是一个要监控的 Input 和 Timer 事件源的集合或者是一个要通知的 Runloop 观察者的集合。每次运 行Runloop，都要指定一个运行模式(显示地或者隐式地)。在 Runloop 的运行期间，只有和当前运行模式相关的源才能被监控和允许发送事件。相似的，只有和当前运行模式相关的观察者才会被通知 Runloop 的行为。和其他模式相关的源会保留新的事件直到 Runloop 运行在了合适的模式才会分发。

在我们的代码中，我们可以通过字符串来标识模式。Cocoa和Core Foundation定义了一个默认模式和几个普通的有用的模式，这些模式都是用字符串来标识的。我们可以用一个字符串当做名字来自定义一个模式，虽然我们自定义模式的名字是随意的，但是模式的内容不是随意的，在我们自己创建的要用的模式中至少要添加一个 Input 源、 Timer 源或者 Runloop 观察者。

在 Runloop 的特殊阶段我们可是使用运行模式来过滤我们不想要的源的事件，大多数的情况下，Runloop 都运行在系统提供的默认模式下，然而 Model Panel 可能运行在“模式”模式，当运行在这个模式期间，只有和这个模式相关的事件源才会发送事件到我们的线程。对于第二线程来说，我们通常使用自定义模式来阻止低优先级的事件源在其他关键处理的时间内发送事件。

注意：运行模式不是根据事件类型划分的，而是根据事件源划分的。我们不能通过模式来匹配鼠标按下事件或者键盘事件，但是我们可以用运行模式来监听一组不同的Port、暂时挂起Timers或者改变当前被监控的事件源和Runloop观察者。

下面列举了一些Cocoa和Core Foundation定义的标准模式：

- `NSModalPanelRunLoopMode`： 用于标明和Mode Panel相关的事件。
- `NSEventTrackingRunLoopMode`： 用于跟踪触摸事件触发的模式（例如UIScrollView上下滚动）。
- `NSRunLoopCommonModes`： 是一个模式集合，当绑定一个事件源到这个模式集合的时候就相当于绑定到了集合内的每一个模式。Cocoa 应用默认包含 Default、Panel、Event Tracking 模式，Core Foundation 只包含 Default 模式，我们可以通过 `CFRunLoopAddCommonMode` 添加模式。

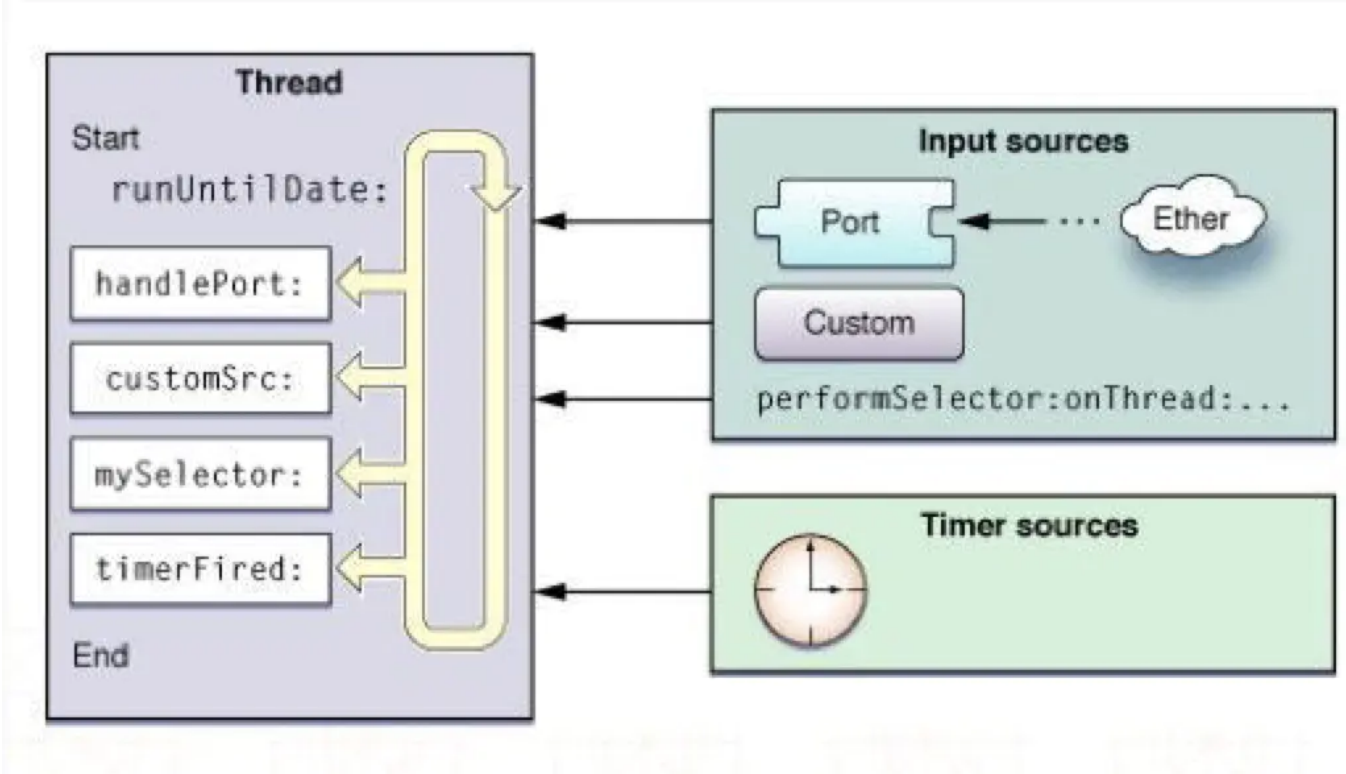
2. Runloop 处理逻辑

Runloop接收来自两种源的事件：

1. 输入源（Input sources）：传递异步消息，通常来自于其他线程或者程序。
2. 定时源（Timer sources）：传递同步消息，在设定好的时间或者循环间断地发生的事件。

这两种事件源都是使用应用指定的事件处理方法来处理到达的事件。

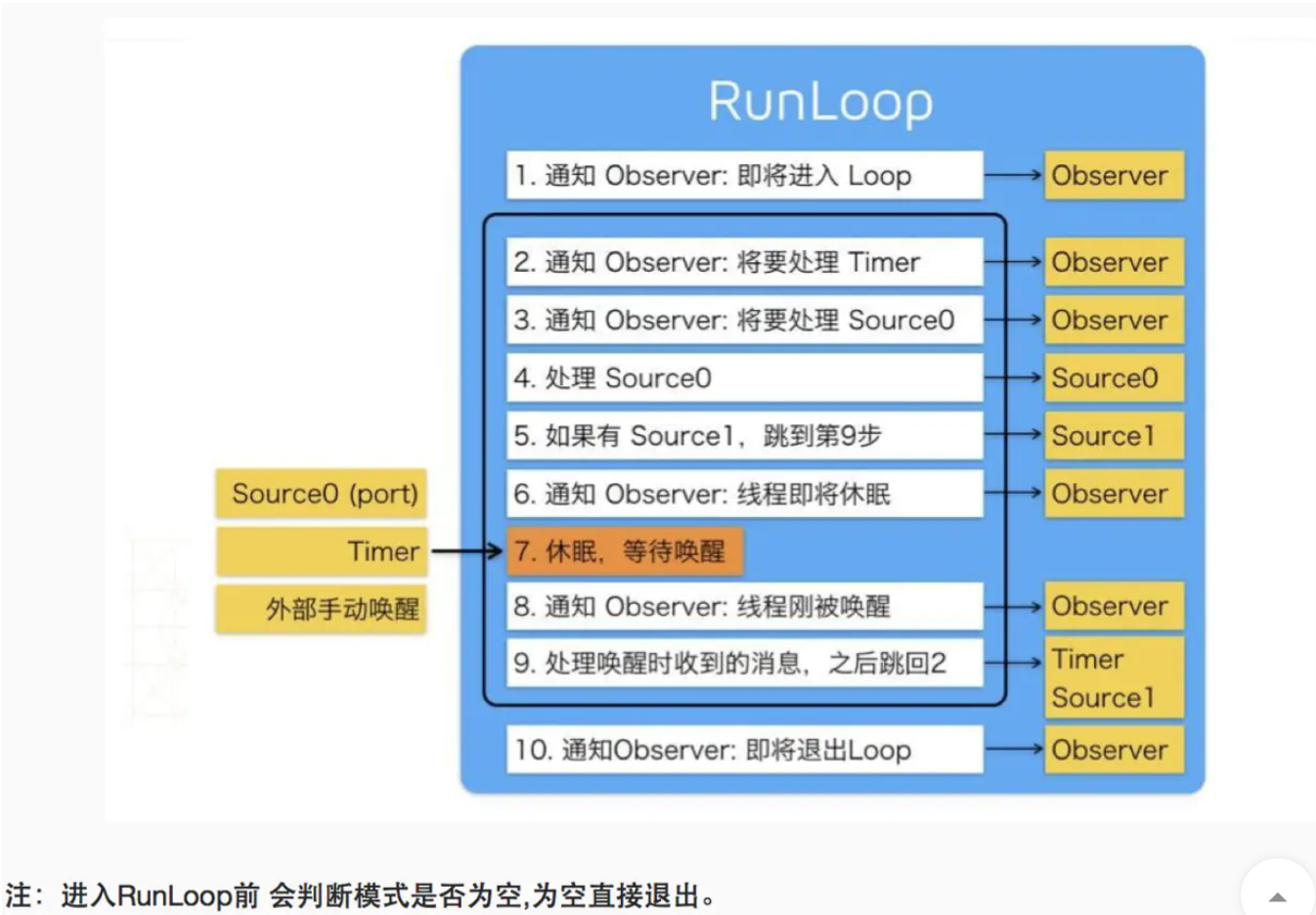
下面的图显示了Runloop和事件源的概念结构。 Input sources异步的分发事件到响应的处理器，然后引起runUntilDate:(由线程相关的Runloop对象调用)方法退出。 Timer sources同步分发事件到相应的处理器但是不会引起Runloop退出。



Run Loop的事件队列

每次运行run loop，你线程的run loop对会自动处理之前未处理的消息，并通知相关的观察者。具体的顺序如下：

1. 通知观察者run loop已经启动
2. 通知观察者任何即将要开始的定时器
3. 通知观察者任何即将启动的非基于端口的源
4. 启动任何准备好的非基于端口的源
5. 如果基于端口的源准备好并处于等待状态，立即启动；并进入步骤9。
6. 通知观察者线程进入休眠
7. 将线程置于休眠直到任一下面的事件发生：
    - 某一事件到达基于端口的源
    - 定时器启动
    - Run loop设置的时间已经超时
    - run loop被显式唤醒
8. 通知观察者线程将被唤醒。
9. 处理未处理的事件
    - 如果用户定义的定时器启动，处理定时器事件并重启run loop。进入步骤2
    - 如果输入源启动，传递相应的消息
    - 如果run loop被显式唤醒而且时间还没超时，重启run loop。进入步骤2
10. 通知观察者run loop结束。



注：进入RunLoop前 会判断模式是否为空,为空直接退出。

- 输入源：每一个需要Runloop处理事件的对象都有一个输入源（InputSource）,并且把这个输入源添加到Runloop里，每产生一个事件（比如用户做了一个手势、点了一个button、滑动了一下tableview、定时器到时）就把这个事件放到对应的输入源。Runloop运行时循环检查每一个输入源是否有事件需要处理，如果有事件要处理Runloop就就调用这个事件的处理方法（通过addTargetxxx指定的方法或者是代理的方法）。如果Runloop里所有的输入源都没有事件要处理，Runloop会休眠。如果Runloop里一个输入源都没有（对象销毁前会把它之前添加的那个输入源取消），Runloop（runUntilDate:这个方法）就退出来了。
- 除了处理输入源的事件，Runloop也会生成Runloop行为的通知。注册Runloop的观察者可以收到这些消息，然后在线程内用他们做一些额外的处理。我们只能使用Core Foundation接口来注册线程的Runloop观察者

3. Input Sources

Input Sources 异步地分发事件到线程。大概有两种类型的 Input Sources，Port-based类型的输入源监控着应用的Mach端口，自定义的输入源监控着自定义的事件源。NSRunloop不关心输入源的类型。两种输入源唯一的不同是输入源的触发方式，Port-based输入源是由系统内核触发的，而自定义的输入源要我们自己触发。创建输入源的时候我们就给给输入源添加指定的模式。下面是一些输入源：

- Port-Based Sources
  Cocoa 和 Core Foundation 提供了类和接口用来创建 Port-Based 源，Cocoa 只要创建 NSPort 对象，并添加到 NSRunloop 中就可以啦，NSPort负责输入源的创建和配置。Core Foundation 需要手动的常见 port 和输入源。

- Custom Input Sources
  我们要用到CFRunLoopSourceRef函数创建输入源，并定义几个回调函数用于配置输入源、处理事件和删除输入源。事件的触发机制要我们自己定义。

- Cocoa Perform Selector Sources
  Cocoa定义了可以在任何线程上执行方法的事件源，在想要执行的线程上执行方法是顺序执行的，避免了多个方法在线程上执行的同步问题。Perform Selector Sources在方法执行完之后就会自己从NSRunloop中删除。
  Perform Selector Sources要求目标线程的NSRunloop必须是运行的，主线程默认是运行的。NSRunloop在一次迭代过程中会处理所有的Perform Selector调用，而不是一次迭代处理一个Perform Selector调用。NSObject中定义的Perform Selector方法如下

  - `performSelectorOnMainThread:withObject:waitUntilDone:`
  - `performSelectorOnMainThread:withObject:waitUntilDone:modes:`
  - `performSelector:onThread:withObject:waitUntilDone:`

- `performSelector:withObject:afterDelay:inModes:`
- `cancelPreviousPerformRequestsWithTarget:`
- `cancelPreviousPerformRequestsWithTarget:selector:object:`

延迟执行是在NSRunloop的下一次迭代中过了指定的延迟事件才执行。取消操作是针对延迟执行方法的。

4. Timer Sources

Timer Sources 同步地在将来的一个确定的时间分发事件到我们的线程。Timers 可以让线程通知自己去处理一些事情。Timers 不是一个实时的机制，当 Timers 触发的时候 NSrunloop 刚好正在执行处理函数，Timer s会等待 NSRunloop 调用自己的处理函数。

Timers 可以创建一次性的和重复性的事件，当创建重复性的事件的时候，Timers 只会根据规划好的触发时间来重新规划触发时间，而不是根据确切的触发时间。而且由于延迟触发丢失了几次触发的话，Timers 只会补充一次触发。

5. NSRunloop 观察者

不像是事件源一样在事件触发的时候执行处理函数。NSRunloop 观察者是在 NSRunloop 几个执行的特定的点触发。NSRunloop 可以观察的几个事件是：

- 进入 NSRunloop
- NSRunloop 将要处理 Timer 事件
- NSRunloop 将要处理 Input 事件
- NSRunloop 将要进入睡眠
- NSRunloop 被唤醒，但是是在处理事件之前
- 退出 NSRunloop

创建观察者的方法是 CFRunLoopObserverRef，我们可以通 过Core Foundation 方法添加到指定的 NSRunloop。观察者也可以创建一次性的和重复性的。一次性的观察者触发之后就会从 NSRunloo p中删除。

三、RunLoop 相关类

`Core Foundation` 中关于 `RunLoop` 的5个类

- `CFRunLoopSourceRef`
- `CFRunLoopTimerRef`
- `CFRunLoopObserverRef` 注:RunLoop 如果没有这些东西会直接退出

1. 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 Source/Timer/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。

系统默认注册了5个Mode:(前两个跟最后一个常用)

- `kCFRunLoopDefaultMode`： App的默认Mode，通常主线程是在这个Mode下运行
- `UITrackingRunLoopMode`： 界面跟踪 Mode，用于 ScrollView 追踪触摸滑动，保证界面滑动时不受其他 Mode 影响
- `UIInitializationRunLoopMode`: 在刚启动 App 时第进入的第一个 Mode，启动完成后就不再使用
- `GSEventReceiveRunLoopMode`: 接受系统事件的内部 Mode，通常用不到
- `kCFRunLoopCommonModes`: 这是一个占位用的Mode，不是一种真正的Mode

2. CFRunLoopSourceRef 事件源（输入源）

按照官方文档的分类：

- Port-Based Sources (基于端口,跟其他线程交互,通过内核发布的消息)

按照函数调用栈的分类

- Source0：非基于Port的，event事件，只含有回调，需要先调用 CFRunLoopSourceSignal(source)，将这个 Source 标记为待处理，然后手动调用 CFRunLoopWakeUp(runloop) 来唤醒 RunLoop。
- Source1：基于Port的，包含了一个 mach_port 和一个回调，被用于通过内核和其他线程相互发送消息,能主动唤醒 RunLoop 的线程。

函数调用栈

3. CFRunLoopTimerRef

CFRunLoopTimerRef 是基于时间的触发器，基本上说的就是 NSTimer (CADisplayLink 也是加到 RunLoop),它受 RunLoop 的 Mode 影响。
GCD的定时器不受 RunLoop 的 Mode 影响。

4. 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

使用

```
- (void)observer {
    // 创建observer
    CFRunLoopObserverRef observer = CFRunLoopObserverCreateWithHandler(CFAllocatorGetDefau
        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);
}
```

特别注意

```
/*
    CF的内存管理（Core Foundation）
    1.凡是带有Create、Copy、Retain等字眼的函数，创建出来的对象，都需要在最后做一次release
    * 比如CFRunLoopObserverCreate
    2.release函数：CFRelease(对象);
 */
```

四、runloop应用

- NSTimer
- PerformSelector
- ImageView显示
- 需要让线程执行周期性的工作（常驻线程）
- 自动释放池
- 需要使用 Port 或者自定义 Input Source 与其他线程进行通讯

1. NSTimer (最常见RunLoop使用)

场景还原：拖拽时模式由 NSDefaultRunLoopMode 进入 UITrackingRunLoopMode ， NSTimer 不再响应图片停止轮播，将计时器改成 NSRunLoopCommonModes 模式下两种模式都可运行。

```
- (void)timer {
    NSTimer *timer = [NSTimer timerWithTimeInterval:2.0 target:self selector:@selector(run
    // 定时器只运行在NSDefaultRunLoopMode下，一旦RunLoop进入其他模式，这个定时器就不会工作
    // [[NSRunLoop currentRunLoop] addTimer:timer forMode:NSDefaultRunLoopMode];
    // 定时器只运行在UITrackingRunLoopMode下，一旦RunLoop进入其他模式，这个定时器就不会工作
    // [[NSRunLoop currentRunLoop] addTimer:timer forMode:UITrackingRunLoopMode];
    // 定时器会跑在标记为common modes的模式下
    // 标记为common modes的模式：UITrackingRunLoopMode和NSDefaultRunLoopMode兼容
    [[NSRunLoop currentRunLoop] addTimer:timer forMode:NSRunLoopCommonModes];
}
- (void)timer2 {
    // 调用了scheduledTimer返回的定时器，已经自动被添加到当前runLoop中，而且是NSDefaultRunLoopMode
    NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:2.0 target:self selector:@sele
    // 修改模式
    [[NSRunLoop currentRunLoop] addTimer:timer forMode:NSRunLoopCommonModes];
}
```

2. ImageView

需求:当用户在拖拽时(UI交互时)不显示图片,拖拽完成时显示图片

- 方法1 监听UIScrollerView滚动 (通过UIScrollViewDelegate监听,此处不再举例)
- 方法2 RunLoop 设置运行模式

```
// 只在NSDefaultRunLoopMode模式下显示图片
// inModes:设置运行模式
[self.imageView performSelector:@selector(setImage:) withObject:[UIImage imageNamed:@"pl
```

3. 常驻线程 (重要)

应用场景: 经常在后台进行耗时操作,如:监控联网状态，扫描沙盒等 不希望线程处理完事件就销毁，保持常驻状态

- 第一种(推荐)

```
- (void)run {
  //addPort:添加端口(就是source)  forMode:设置模式
   [[NSRunLoop currentRunLoop] addPort:[NSPort port] forMode:NSDefaultRunLoopMode];
  //启动RunLoop
    [[NSRunLoop currentRunLoop] run];
 /*
  //另外两种启动方式
    [NSDate distantFuture]:遥远的未来  这种写法跟上面的run是一个意思
    [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode beforeDate:[NSDate distant
    不设置模式
    [[NSRunLoop currentRunLoop] runUntilDate:[NSDate distantFuture]];
  */
}
```

退出-退出当前线程

```
[NSThread exit];
```

- 第二种(奇葩法)
  优点：退出RunLoop比较方便-定义个标记 while(flag){...}

```
- (void)run {
    while (1) {
        [[NSRunLoop currentRunLoop] run];
    }
}
```

4. 自动释放池

在休眠前(kCFRunLoopBeforeWaiting)进行释放，处理事件前创建释放池，中间创建的对象会放入释放池。
特别注意：在启动 RunLoop 之前建议用 @autoreleasepool {...} 包裹。
意义:创建一个大释放池,释放 {} 期间创建的临时对象，一般好的框架的作者都会这么做。

```
- (void)execute {
    @autoreleasepool {
        NSTimer *timer = [NSTimer timerWithTimeInterval:2.0 target:self selector:@selector(
        [[NSRunLoop currentRunLoop] addTimer:timer forMode:NSDefaultRunLoopMode];
        [[NSRunLoop currentRunLoop] run];
}
```

5. 补充: GCD定时器

一般的NSTimer定时器因为受到RunLoop，会存在时间不准时的情况。 上文有提到GCD不受RunLoop影响,下面简单的说一下它的使用

```
/** 定时器(这里不用带*，因为 dispatch_source_t 就是个类，内部已经包含了*) */
@property (nonatomic, strong) dispatch_source_t timer;
int count = 0;
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    // 获得队列
    // dispatch_queue_t queue = dispatch_get_global_queue(0, 0);
    dispatch_queue_t queue = dispatch_get_main_queue();
    // 创建一个定时器(dispatch_source_t本质还是个OC对象)
    self.timer = dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER, 0, 0, queue);
    // 设置定时器的各种属性（几时开始任务，每隔多长时间执行一次）
    // GCD的时间参数，一般是纳秒 NSEC_PER_SEC（1秒 == 10的9次方纳秒）
    // 何时开始执行第一个任务
    // dispatch_time(DISPATCH_TIME_NOW, 3.0 * NSEC_PER_SEC) 比当前时间晚3秒
    dispatch_time_t start = dispatch_time(DISPATCH_TIME_NOW, (int64_t)(1.0 * NSEC_PER_
    uint64_t interval = (uint64_t)(1.0 * NSEC_PER_SEC);
```

```
    dispatch_source_set_event_handler(self.timer, ^{
        NSLog(@"------------%@", [NSThread currentThread]);
        count++;
//        if (count == 4) {
//            // 取消定时器
//            dispatch_cancel(self.timer);
//            self.timer = nil;
//        }
    });
    // 启动定时器
    dispatch_resume(self.timer);
}
```

五、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;

- 获取RunLoop对象

```
// 工作线程 需要程序员手工写代码让runloop运行起来
[NSRunLoop currentLoop]runUntilDate:]
// 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
```

- 线程安全性
  基于 Cocoa 的接口不是线程安全的，基于 Core Foundation 的接口是线程安全的。

六、RunLoop 面试题

1. 什么是RunLoop?
    - 其实它内部就是do-while循环,在这个循环内部不断的处理各种任务(比如Source、Tim
      Observer)。

- RunLoop只能选择一个Mode启动,如果当前Mode中没有任何Soure、Timer、Observer,那么就直接退出RunLoop。

2. 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)

3. 在异步线程中下载很多图片。如果失败了，该如何处理？请结合runloop来谈谈解决方案？
   答：（提示：在异步线程中启动一个runloop重新发送网络图片）
   （1）重新下载图片
   （2）利用 runloop 的输入源回到主线程刷新 UIImageView。

相关链接

苹果官方文档
CFRunLoop官方文档
NSRunLoop官方文档
CFRunLoopRef
NSRunloop的使用

> **关注下面的标签，发现更多相似文章**

iOS

输入评论...

我不是黄花菜 Lv1

第五步的代码是这个:
if (MACH_PORT_NULL != dispatchPort && !didDispatchPortLastTime) {
#if DEPLOYMENT_TARGET_MACOSX || DEPLOYMENT_TARGET_EMBEDDED || DEPLOYMENT_TARGET_EMBEDDED_MINI...

展开

1年前 回复

我不是黄花菜 Lv1

执行source1的逻辑源码搜索下__CFRunLoopDoSource1这个函数的调用

1年前

ShiXianjun-2016 iOS开发 @ 北京地厚云...

够详细 收藏了

1年前 回复

CatalinaCore

每天一runloop/runtime压压惊

1年前 回复

相关推荐

橘子不酸丶 · 2天前 · iOS

iOS优化篇之App启动时间优化

32 1

老司机技术周报 · 1天前 · iOS / SwiftUI

【WWDC20】10037 - SwiftUI 中的 App 要领

8

FengyunSky · 4天前 · iOS

一文读懂iOS线程调用栈原理

12

ZenonHuang · 7天前 · iOS

iOS 的自动构建流程

首页

探索掘金

JeremyHuang37 · 1天前 · iOS

【译】自定义 Collection View Layout -- 一个简单的模板

2

掘金酱 · 27天前 · iOS / Android / 前端 / 后端 / 程序员

🏆 掘金征文 | 2020与我的年中总结

94 109

Chouee · 3天前 · iOS

造轮子 - UITableView字母索引条

7 1

路过看风景 · 2天前 · iOS

CocoaPods原理 及 组件化

3

阿里巴巴淘系技术 · 5天前 · iOS

Apple Widget：下一个顶级流量入口?

9

路过看风景 · 1天前 · iOS

iOS事件处理 UIResponder

2