```
程序main函数主线程在RunLoop中,一直运
                                           行,没有事件就休眠,等待被唤醒
                         保持程序持续运行
                                           内部是do while 循环
                                              触摸事件,定时器事件,Selector事件
                        处理App中的各种事件
                作用
                                                    一直处于循环中,当有事件触发
                                                   时就会开始工作
                         节省CPU资源,提高程序性能
                                                    要处理时CPU调度资源,没有时
                                                    释放资源去做其它的事情
                            APP启动进入下面mian函数,只会打印开始,不会打印结
                            束,说明UIApplicationMain一直运行着没有返回
                            int main(int argc, char * argv[]) {
                              @autoreleasepool {
                               NSLog(@"开始");
                main入口
                               int re = UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
                               NSLog(@"结束");
                               return re;
                                         struct __CFRunLoop {
                                                                  /* locked for accessing mode list */
                                            pthread_mutex_t _lock;
                                            pthread_t _pthread;
                                            CFMutableSetRef _commonModes;
                                            CFMutableSetRef _commonModeItems;
                                            CFRunLoopModeRef _currentMode;
                                            CFMutableSetRef _modes;
                                            CFAbsoluteTime _runTime;
                                            CFAbsoluteTime _sleepTime;
                                            CFTypeRef _counterpart; 等等
                                             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Source0/Source1/Timer/Observer,而RunLoop启动时只
                                 运行模式
                                             能选择其中一个Mode作为currentMode
                                             CFRunLoopModeRef 其实是指向__CFRunLoopMc
                                                                                  le结构体的指针
                RunLoop结构体
                                                                     CFMutableSetRef _sources0;
                                                                                                触摸事件, PerformSelectors
                                                                                                基于Port的线程间通信
                                                                     CFMutableSetRef _sources1;
                                          struct __CFRunLoopMode
                                                                     CFMutableArrayRef _observers;
                                                                                                  监听器,用于监听RunLoop的状态
                                          成员变量
                                                                                               定时器(默认的,在tableView滚
                                                                     CFMutableArrayRef _timers;
                                                                                               动时会暂停计时,需设置mode)
                                                        只能退出RunLoop,再重新指定一个Mode进入,这样做主要是为了分隔开不
                                 Mode
                                                        同组的Source、Timer、Observer,让其互不影响
                                          切换Mode
                                                        后面通过举例Timer, performSelector等等来解释
                                                     如果Mode里没有任何Source0/Source1/Timer/Observer,
                                          不能空
                                                     RunLoop会立马退出, 所以4个属性必须要有一个属性有值
                                          添加source
                                                        可以添加到不同的线程中的mode去
                                                                observer和timer只能被添加到一个RunLoop中的一
                                          添加Observer和Timer
                                                                个或者多个mode中
                                 下面源码显示: 判断除了stop和finish外会一直循环
                                 // 用DefaultMode启动
                                 void CFRunLoopRun(void) { /* DOES CALLOUT */
                                  int32_t result;
                RunLoop的源码
                                  do {
                                    result = CFRunLoopRunSpecific(CFRunLoopGetCurrent(), kCFRunLoopDefaultMode, 1.0e10, false);
                                    CHECK_FOR_FORK();
                                   while (kCFRunLoopRunStopped! = result && kCFRunLoopRunFinished! = result);
                               Fundation框架 (基于CFRunLoopRef的OC封装)
                                                                                         // 拿到当前Runloop 调用_CFRunLoopGet0
                                                                                         CFRunLoopRef CFRunLoopGetCurrent(void) {
                                                                                           CHECK_FOR_FORK();
                               Foundation
                                                                                           CFRunLoopRef rl =
                               [NSRunLoop currentRunLoop]; // 获得当前线程的RunLoop对象
                                                                                         (CFRunLoopRef)_CFGetTSD(__CFTSDKeyRunLoop);
                               [NSRunLoop mainRunLoop]; // 获得主线程的RunLoop对象
               RunLoop对象
                                                                                           if (rl) return rl;
                                                                                           return _CFRunLoopGet0(pthread_self());
                               Core Foundation
                               CFRunLoopGetCurrent(); // 获得当前线程的RunLoop对象
                               CFRunLoopGetMain(); // 获得主线程的RunLoop对象
                                       (1)每条线程都有唯一的一个与之对应的RunLoop对象
                                       (2)RunLoop保存在一个全局的Dictionary里,线程作为key,RunLoop作为v
                                       (3)主线程的RunLoop已经自动创建好了,子线程的RunLoop需要主动创建
                                       (4)RunLoop在第一次获取时创建,在线程结束时轻
                                                   //全局字典
                                                   CFMutableDictionaryRef dict = CFDictionaryCreateMutable(kCFAllocatorSystemDefault,....
                RunLoop和线程间的关系
                                                   // 根据传入的主线程获取主线程对应的RunLoop
                                                               mainLoop = __CFRunLoopCreate(pthread_main_thread_np());
                                       存取源码
                                                   // 保存主线程 将主线程-key和RunLoop-Value保存到字典中
                                                   CFDictionarySetValue(dict, pthreadPointer(pthread_main_thread_np()), mainLoop);
                                                   // 从字典里面拿,将线程作为key从字典里获取一个loop
                                                               loop = (CFRunLoopRef)CFDictionaryGetValue(__CFRunLoops, pthreadPointer(t));
                                       1. kCFRunLoopDefaultMode:
                                       App的默认Mode,通常主线程是在这个Mode下运行
                                       2. 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(比如NSTimer计时)
                                       3. UllnitializationRunLoopMode:
                系统默认注册的5个Mode
                                       在刚启动 App 时第进入的第一个 Mode,启动完成后就不再使用,会切换到kCFRunLoopDefaultMode,
                                       4. GSEventReceiveRunLoopMode:
                                        接受系统事件的内部 Mode, 通常用不到
                                       5. kCFRunLoopCommonModes:
                                       这是一个占位用的Mode,作为标记kCFRunLoopDefaultMode和UlTrackingRunLoopMode用,并不是一种真正的Mode
                               主线程销毁RunLoop退出
                               Mode中(Timer、Source、Observer)为空的时候,RunLoop会立刻退出
                               不停止 [NSRunLoop currentRunLoop]run];
                RunLoop退出
                               设置什么时候停止
                               [NSRunLoop currentRunLoop]runUntilDate:<#(nonnull NSDate *)#>
                               [NSRunLoop currentRunLoop]runMode:<#(nonnull NSString *)#> beforeDate:<#(nonnull NSDate *)#>
                                                 NSTimer *timer = [NSTimer timerWithTimeInterval:2.0 target:self selector:@selector(show) userInfo:nil repeats:YES];
                                                                                         加入到RunLoop中才可以运行
                                                 (方式1): [[NSRunLoop mainRunLoop]
                                                 addTimer:timer
                                                                                         当UIScrollView滚动时,NSTimer就会暂停,当我们停
                                                 forMode:NSDefaultRunLoopMode];
                                                                                         止滑动以后,NSTimer又会重新恢复
                                                 当我们滑动ScrollView时,RunLoop的Mode会切换到UlTrackingRunLoopMode,因此在主
                                                 线程的定时器就不在管用了,调用的方法也就不再执行了,当我们停止滑动时,RunLoop的
                                                 Mode切换回kCFRunLoopDefaultMode,所以NSTimer就又管用了
                                     NSTimer
                                                 (方式2) [[NSRunLoop mainRunLoop]
                                                                                   只有我们在滑动textField/
                                                 addTimer:timer
                                                                                   UIScrollView时timer才会运行
                                                 forMode:UITrackingRunLoopMode];
                                                                                       可以在UITrackingRunLoopMode,
                                                 (方式3) [[NSRunLoop mainRunLoop]
                                                                                       kCFRunLoopDefaultMode两种模式
RunLoop
                                                 addTimer:timer
                                                                                       下运行
(v1.nxb)
                                                 forMode:NSRunLoopCommonModes];
                                                                                       方式3 是比较不受影响的方式
                                                             performSelector默认是在default模式下运行,因此在滑动ScrollView时,图片不会加载
                                                             [self.imageView performSelector:@selector(setImage:) withObject:[UIImage
                                                             imageNamed:@"abc"] afterDelay:2.0 ];
                示例 (Mode间的切换)
                                     UllmageView
                                                             传入Mode数组(两种模式下均可加载图片)
                                                             [self.imageView performSelector:@selector(setImage:) withObject:[UIImage
                                                             imageNamed:@"abc"] afterDelay:2.0
                                                             inModes:@[NSDefaultRunLoopMode,UITrackingRunLoopMode]];
                                                          1 创建队列
                                                          dispatch_queue_t queue = dispatch_get_global_queue(0, 0);
                                                          创建一个GCD定时器
                                                          dispatch_source_t timer = dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER, 0, 0, queue);
                                                          2 需要对timer进行强引用,保证其不会被释放掉,才会按时调用block块
                                                          self.timer = timer;
                                                          3设置定时器的开始时间,间隔时间,精准度
                                     结合GCD创建计时器
                                                          dispatch_source_set_timer(timer, DISPATCH_TIME_NOW, 2.0 *
                                                         NSEC_PER_SEC, 0 * NSEC_PER_SEC);
                                                          4 设置定时器要执行的事情
                                                          dispatch_source_set_event_handler(timer, ^{
                                                            //do something
                                                         5 启动
                                                          dispatch_resume(timer);
                                               @property (nonatomic, strong) NSThread *thread;
                                               2 创建线程
                                               self.thread = [[NSThread alloc]initWithTarget:self selector:@selector(addRun)
                                               object:nil];
                                               [self.thread start];
                                               3 线程常驻方法实现(添加一个source)
                                               – (void)addRun {
                                                 //1.获取当前线程对应的 RunLoop
                                                 NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
                                                 //2.CFRunloopSourceRef 方式实现,添加了一个source0,保证RunLoop不退出
                                                 [runLoop addPort:[NSPort port] forMode:NSDefaultRunLoopMode];
                                                 //3.开启(默认是没有开启的)
                                                 [runLoop run];
                                               4 线程常驻方法实现(添加一个Timer来实现)
                                               – (void)addRun {
                                                 NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
                                                 NSTimer *timer = [NSTimer timerWithTimeInterval:2.0 repeats:YES block:^(NSTimer * _Nonnull timer) {
                                                 [runLoop addTimer:timer forMode:NSDefaultRunLoopMode];
                                                 [runLoop run];
                                   常驻线程
                                               5 线程常驻方法实现(死循环)
                                               while (1) {
                                                [[NSRunLoop currentRunLoop] run];
                                                                     //让task1方法在线程thread上实现
                                                                     [self performSelector:@selector(task1) onThread:self.thread
                                                                     withObject:nil waitUntilDone:NO];
                                                                                                                          添加 autoreleasepool:线程中
                                                                                                                          子任务结束后要及时销毁对象
                                                                                                                          //其他回到主线程方法 1.NSThread
                                                                     //task1方法
                                                                     - (void)task1{
                                                                                                                          [self performSelectorOnMainThread:@selector(updateUI)
                                                                      @autoreleasepool{
                                                                                                                         withObject:nil waitUntilDone:NO];
                                                                        //耗时操作
                                               6. 常驻线程中添加任务
                                                                                                                         - (void)updateUI {
                                                                                                                           // UI更新代码
                                                                        dispatch_async(dispatch_get_main_queue(), ^{
                                                                         //在主线程更新UI(GCD方法)
                                                                         self.nameLab.text = @"hello";
                                                                                                                          //其他回到主线程方法 2.NSOperationQueue
                                                                                                                         [[NSOperationQueue mainQueue]
                                                                                                                         addOperationWithBlock:^{
                                                                                                                            // UI更新代码
                RunLoop其他应用
                                                 1 利用@autoreleasepool优化循环,下面方法内部已实现
                                                 [array enumerateObjectsUsingBlock:^(id items, NSUInteger idx, BOOL *stop) { }];
                                                 2 如果你的应用程序或者线程是要长期运行的并且有可能产生大量autoreleased对象,
                                                 你应该使用autorelease pool blocks
                                   自动释放池
                                                3 长期在后台中运行的线程任务, 方法
                                                                        RunLoop的三种状态相关联:
                                                                        kCFRunLoopEntry; // 进入runloop之前,创建一个自动释放池
                                                 自动释放池的创建与销毁
                                                                        kCFRunLoopBeforeWaiting; // 体眠之前,销毁自动释放池,然后创建一个新的自动释放池
                                                                                    it; // 退出runloop之前,销
                                                                                                         自动释放池
                                                       CADisplayLink是一个执行频率(fps)和屏幕刷新相同(可以修改preferredFramesPerSecond
                                                       改变刷新频率)的定时器,它也需要加入到RunLoop才能执行
                                   CADisplayLink定时器
                                                       底层使用mk_timer,和NSTimer相比它精度更高,如果遇到大任务它仍然存在丢帧现
                                                        象。通常情况下CADisaplayLink用于构建帧动画,看起来相对更加流畅
                                                             苹果注册了一个 Source1 (基于 mach port 的) 用来接收系统事件,其回调函数为
                                                              _IOHIDEventSystemClientQueueCallback()
                                                             当一个硬件事件(触摸/锁屏/摇晃等)发生后,首先由 IOKit.framework 生成一个 IOHIDEvent 事件并由
                                                             SpringBoard 接收。SpringBoard 只接收按键(锁屏/静音等),触摸,加速,接近传感器等几种 Event,随后
                                                             用 mach port 转发给需要的App进程。随后苹果注册的那个 Source1 就会触发回调,并调用
                                   事件响应UIEvent -> Source1
                                                             _UIApplicationHandleEventQueue() 进行应用内部的分发
                                                             _UIApplicationHandleEventQueue() 会把 IOHIDEvent 处理并包装成 UIEvent 进行处理或分发,其中包括识别
                                                             UIGesture/处理屏幕旋转/发送给 UIWindow 等。通常事件比如 UIButton 点击、touchesBegin/Move/End/
                                                             Cancel 事件都是在这个回调中完成的
                                                      当事件响应的 _UIApplicationHandleEventQueue() 识别了一个手势时,其首先会调用 Cancel 将当前的
                                                      touchesBegin/Move/End 系列回调打断。随后系统将对应的 UIGestureRecognizer 标记为待处理
                                                      苹果注册了一个 Observer 监测 BeforeWaiting (Loop即将进入休眠) 事件,这个Observer的回调函数是
                                   手势识别-Observer
                                                      _UIGestureRecognizerUpdateObserver(),其内部会获取所有刚被标记为待处理的 GestureRecognizer,并执行
                                                      GestureRecognizer的回调
                                                      当有 UIGestureRecognizer 的变化(创建/销毁/状态改变)时,这个回调都会进行相应处理
                                             在当前正在执行大量的逻辑运算可能UI的更新就会比较卡
                                                               AsyncDisplayKit其实是将UI排版和绘制运算尽可能放到后台,将UI的最终更新操作放到主线程(这一步也必
                                   UI更新
                                             facebook推出了
                                                               须在主线程完成),同时提供一套类UIView或CALayer的相关属性,尽可能保证开发者的开发习惯。这个过
                                             AsyncDisplayKit
                                                               程中AsyncDisplayKit在主线程RunLoop中增加了一个Observer监听即将进入休眠和退出RunLoop
                                                               收到回调时遍历队列中的待处理任务——执行
                                                                                  CFRunLoopRef runLoop = CFRunLoopGetCurrent();
                                                                                  CFStringRef runLoopMode = kCFRunLoopDefaultMode;
                                                                                  CFRunLoopObserverRef observer = CFRunLoopObserverCreateWithHandler
                                                                                  (kCFAllocatorDefault, kCFRunLoopBeforeWaiting, true, 0, ^(CFRunLoopObserverRef observer,
                                                        在RunLoop 处于"空闲"状态
                                   计算cell的预缓存高度
                                                                                  CFRunLoopActivity _) {
                                                        Mode,并且马上要休眠时
                                                                                     // TODO here
```

CFRunLoopAddObserver(runLoop, observer, runLoopMode);

在其中的 TODO 位置,就可以开始任务的收集和分发了,当然,不能忘记适时的移除这个 observer