RunLoop

@M了个J

https://github.com/CoderMJLee

实力IT教育 www.520it.com

小码哥教育 SEEMYGO

面试题

- 讲讲 RunLoop，项目中有用到吗？
- runloop内部实现逻辑？
- runloop和线程的关系？
- timer 与 runloop 的关系？
- 程序中添加每3秒响应一次的NSTimer，当拖动tableview时timer可能无法响应要怎么解决？
- runloop 是怎么响应用户操作的， 具体流程是什么样的？
- 说说runLoop的几种状态
- runloop的mode作用是什么？

什么是RunLoop

- 顾名思义
  - 运行循环
  - 在程序运行过程中循环做一些事情

- 应用范畴
  - 定时器（Timer）、PerformSelector
  - GCD Async Main Queue
  - 事件响应、手势识别、界面刷新
  - 网络请求
  - AutoreleasePool

小码哥教育 SEEMYGO

如果没有RunLoop

```
11 int main(int argc, const char * argv[]) {
12     @autoreleasepool {
13         NSLog(@"Hello, World!");
14     }
15     return 0;
16 }
```

- 执行完第13行代码后，会即将退出程序

如果有了RunLoop

```
12 int main(int argc, char * argv[]) {
13     @autoreleasepool {
14         return UIApplicationMain(argc, argv, nil,
15                                  NSStringFromClass([AppDelegate class]));
16     }
17 }
```

```
22 int main(int argc, char * argv[]) {
23     @autoreleasepool {
24         int retVal = 0;
25         do {
26             // 睡眠中等待消息
27             int message = sleep_and_wait();
28             // 处理消息
29             retVal = process_message(message);
30         } while (0 == retVal);
31         return 0;
32     }
33 }
```

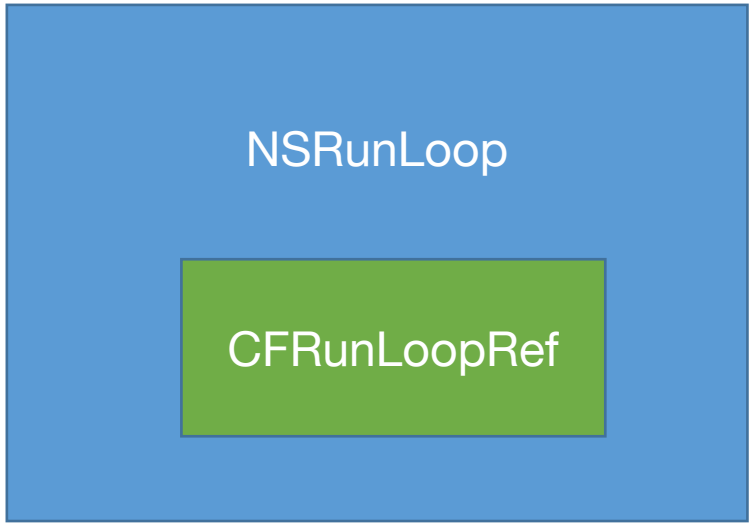
- 程序并不会马上退出，而是保持运行状态

- RunLoop的基本作用
  - 保持程序的持续运行
  - 处理App中的各种事件（比如触摸事件、定时器事件等）
  - 节省CPU资源，提高程序性能：该做事时做事，该休息时休息
  - ……

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
  - CFRunLoopRef是开源的
  - ✓ https://opensource.apple.com/tarballs/CF/



小码哥教育 SEEMYGO

RunLoop与线程

- 每条线程都有唯一的一个与之对应的RunLoop对象
- RunLoop保存在一个全局的Dictionary里，线程作为key，RunLoop作为value
- 线程刚创建时并没有RunLoop对象，RunLoop会在第一次获取它时创建
- RunLoop会在线程结束时销毁
- 主线程的RunLoop已经自动获取（创建），子线程默认没有开启RunLoop

小码哥教育 SEEMYGO

获取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`

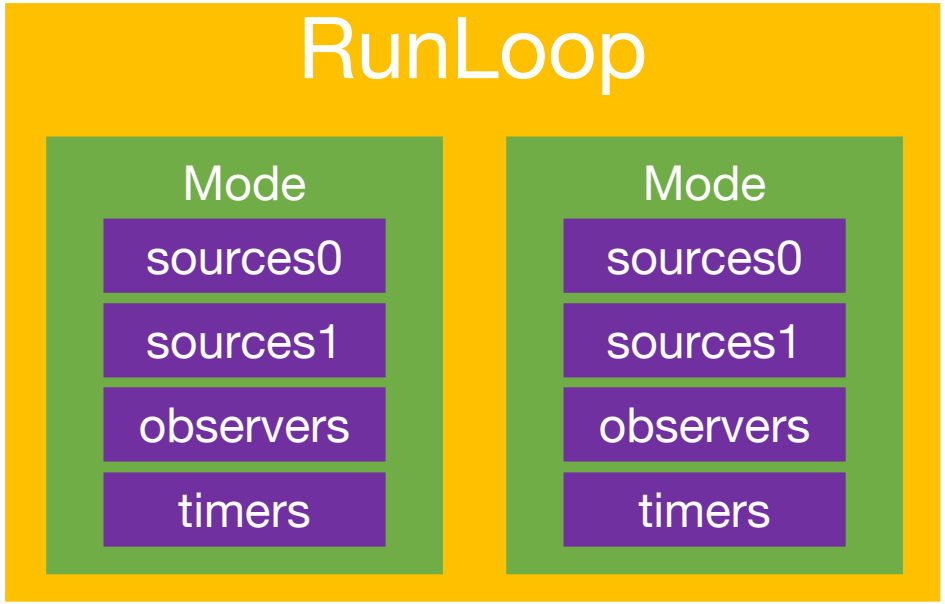
- Core Foundation
  - `CFRunLoopGetCurrent(); // 获得当前线程的RunLoop对象`
  - `CFRunLoopGetMain(); // 获得主线程的RunLoop对象`

RunLoop相关的类

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

```
typedef struct __CFRunLoop * CFRunLoopRef;
struct __CFRunLoop {
    pthread_t _pthread;
    CFMutableSetRef _commonModes;
    CFMutableSetRef _commonModeItems;
    CFRunLoopModeRef _currentMode;
    CFMutableSetRef _modes;
};
```

```
typedef struct __CFRunLoopMode *CFRunLoopModeRef;
struct __CFRunLoopMode {
    CFStringRef _name;
    CFMutableSetRef _sources0;
    CFMutableSetRef _sources1;
    CFMutableArrayRef _observers;
    CFMutableArrayRef _timers;
};
```



小码哥教育 SEEMYGO

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Source0/Source1/Timer/Observer
- RunLoop启动时只能选择其中一个Mode，作为currentMode
- 如果需要切换Mode，只能退出当前Loop，再重新选择一个Mode进入
  - 不同组的Source0/Source1/Timer/Observer能分隔开来，互不影响
- 如果Mode里没有任何Source0/Source1/Timer/Observer，RunLoop会立马退出

小码哥教育 SEEMYGO

CFRunLoopModeRef

- 常见的2种Mode
  - kCFRunLoopDefaultMode（NSDefaultRunLoopMode）：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

小码哥教育 SEEMYGO

CFRunLoopObserverRef

```
/* Run Loop Observer Activities */
typedef CF_OPTIONS(CFOptionFlags, CFRunLoopActivity) {
    kCFRunLoopEntry = (1UL << 0),            // 即将进入Loop
    kCFRunLoopBeforeTimers = (1UL << 1),     // 即将处理Timer
    kCFRunLoopBeforeSources = (1UL << 2),    // 即将处理Source
    kCFRunLoopBeforeWaiting = (1UL << 5),    // 即将进入休眠
    kCFRunLoopAfterWaiting = (1UL << 6),     // 刚从休眠中唤醒
    kCFRunLoopExit = (1UL << 7),             // 即将退出Loop
    kCFRunLoopAllActivities = 0x0FFFFFFFU
};
```

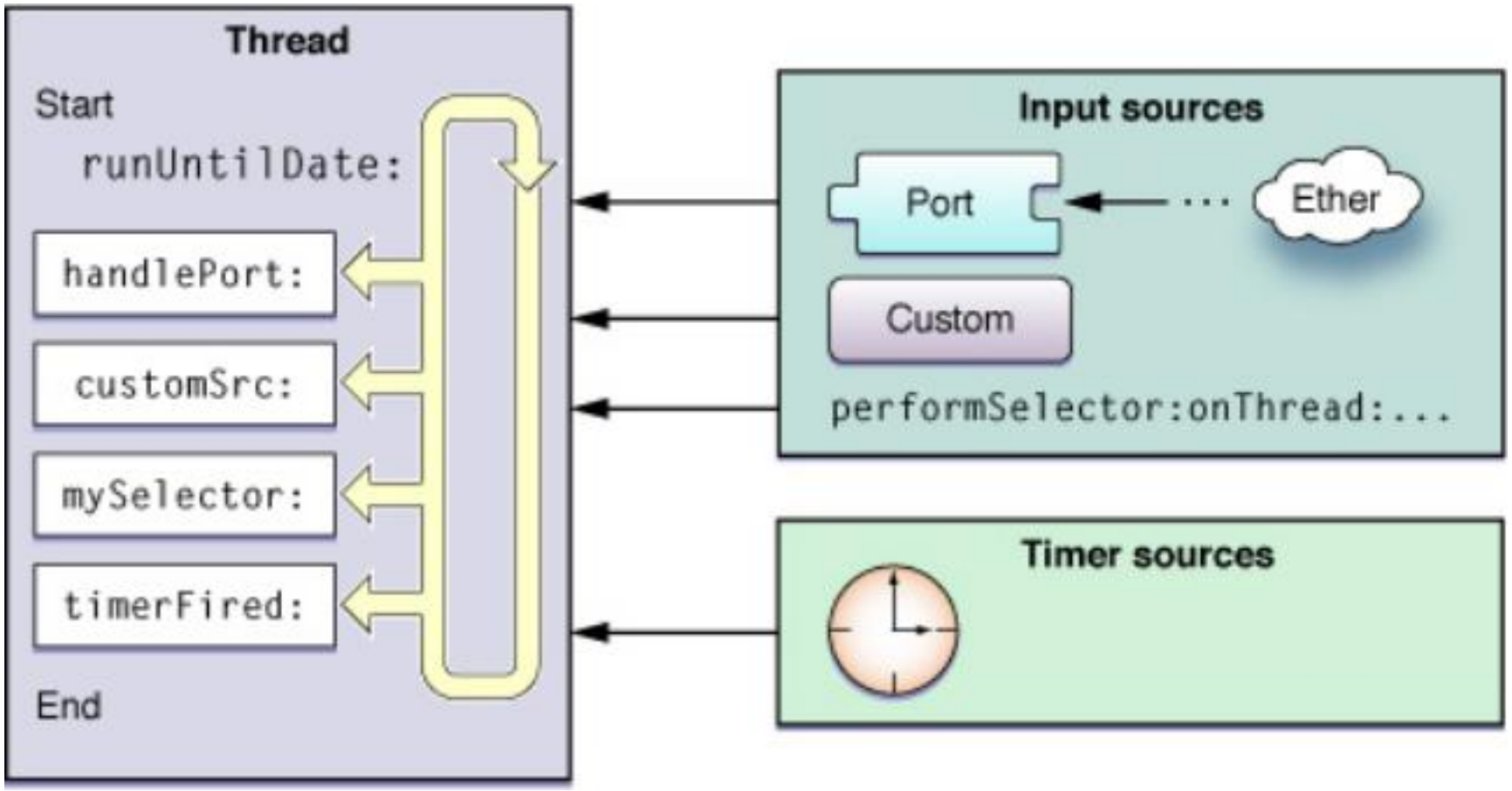
小码哥教育 SEEMYGO

添加Observer监听RunLoop的所有状态

```
CFRunLoopObserverRef observer = CFRunLoopObserverCreateWithHandler(kCFAllocatorDefault,
    kCFRunLoopAllActivities, YES, 0, ^void(CFRunLoopObserverRef observer, CFRunLoopActivity activity) {
    switch (activity) {
        case kCFRunLoopEntry:
            NSLog(@"kCFRunLoopEntry");
            break;
        case kCFRunLoopBeforeTimers:
            NSLog(@"kCFRunLoopBeforeTimers");
            break;
        case kCFRunLoopBeforeSources:
            NSLog(@"kCFRunLoopBeforeSources");
            break;
        case kCFRunLoopBeforeWaiting:
            NSLog(@"kCFRunLoopBeforeWaiting");
            break;
        case kCFRunLoopAfterWaiting:
            NSLog(@"kCFRunLoopAfterWaiting");
            break;
        case kCFRunLoopExit:
            NSLog(@"kCFRunLoopExit");
            break;
        default:
            break;
    }
});
CFRunLoopAddObserver(CFRunLoopGetCurrent(), observer, kCFRunLoopCommonModes);
CFRelease(observer);
```

小码哥教育 SEEMYGO

RunLoop的运行逻辑

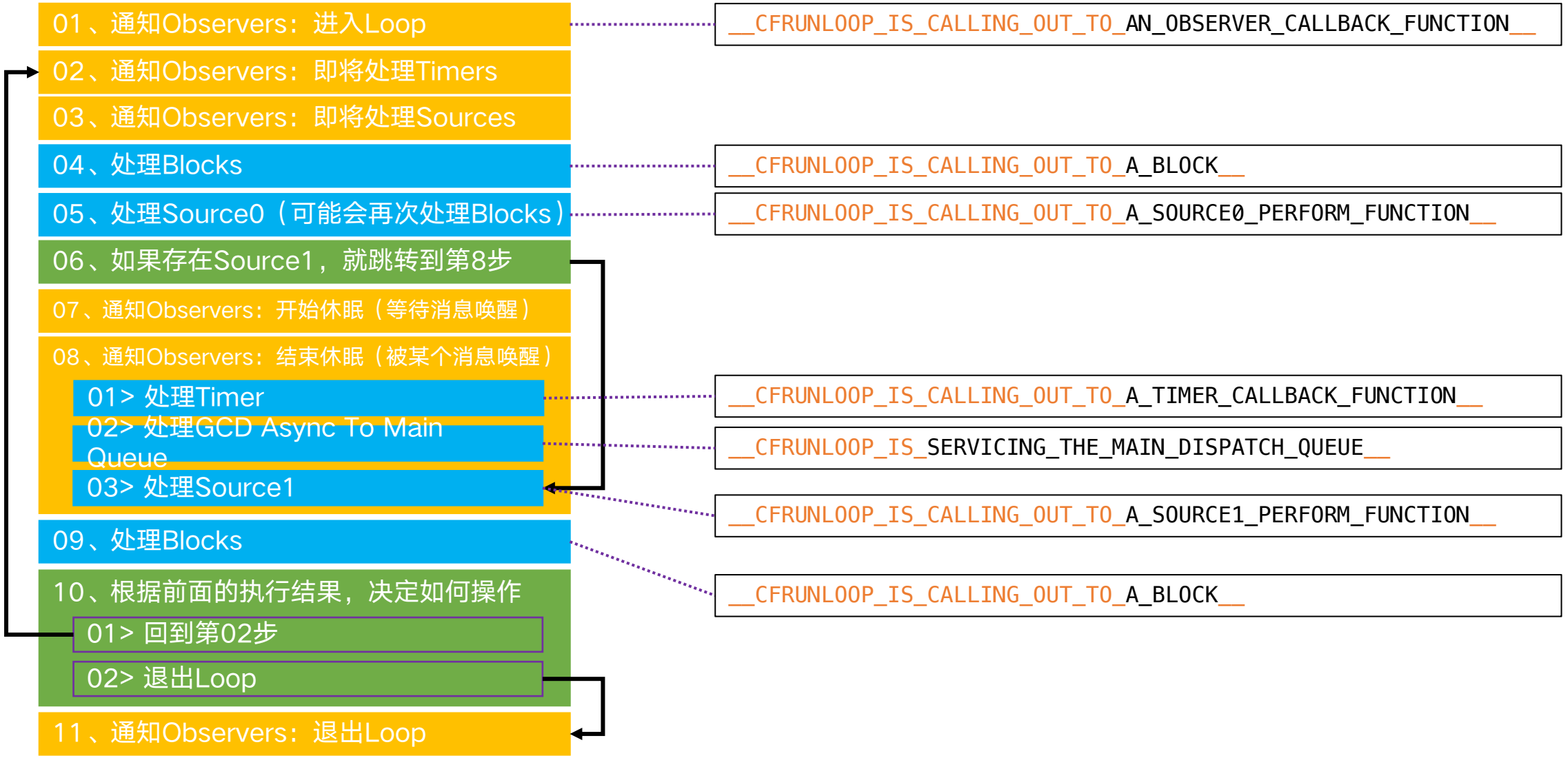


RunLoop的运行逻辑

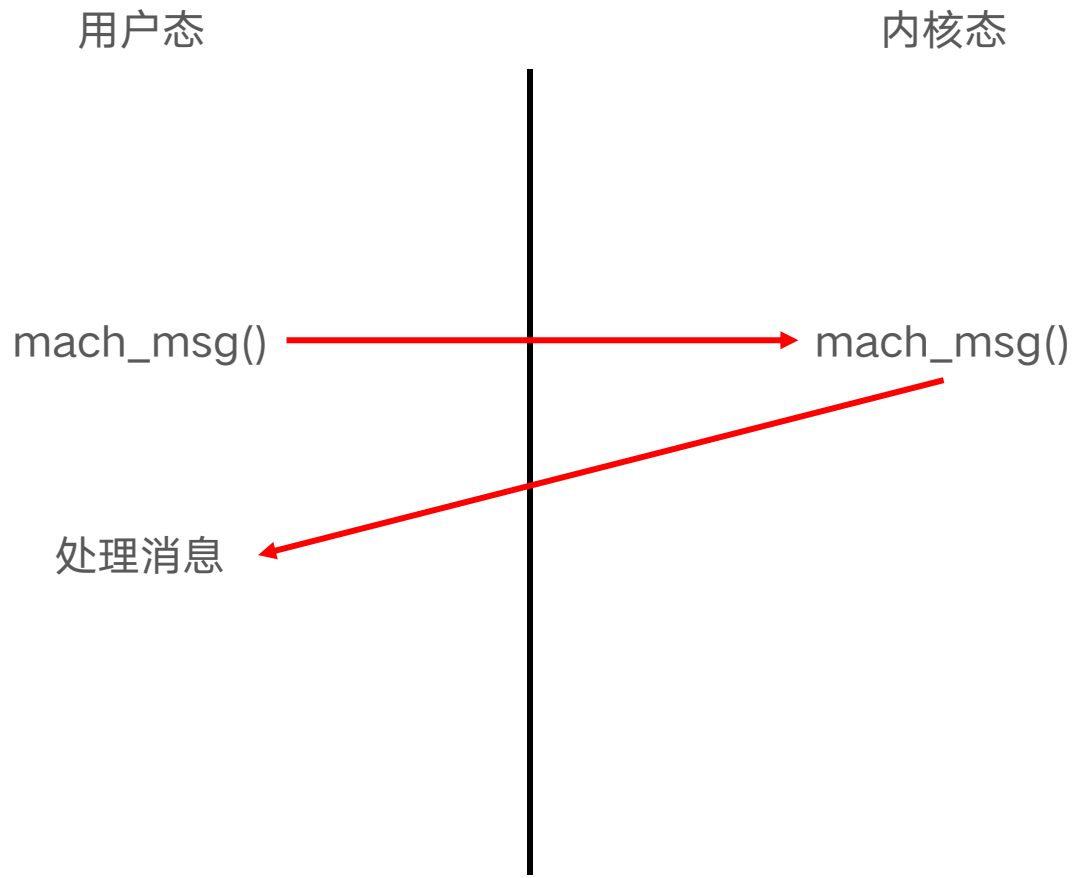
- Source0
  - 触摸事件处理
  - `performSelector:onThread:`
- Source1
  - 基于Port的线程间通信
  - 系统事件捕捉
- Timers
  - NSTimer
  - `performSelector:withObject:afterDelay:`
- Observers
  - 用于监听RunLoop的状态
  - UI刷新（`BeforeWaiting`）
  - Autorelease pool（`BeforeWaiting`）

- 01、通知Observers：进入Loop
- 02、通知Observers：即将处理Timers
- 03、通知Observers：即将处理Sources
- 04、处理Blocks
- 05、处理Source0（可能会再次处理Blocks）
- 06、如果存在Source1，就跳转到第8步
- 07、通知Observers：开始休眠（等待消息唤醒）
- 08、通知Observers：结束休眠（被某个消息唤醒）
  - 01> 处理Timer
  - 02> 处理GCD Async To Main Queue
  - 03> 处理Source1
- 09、处理Blocks
- 10、根据前面的执行结果，决定如何操作
  - 01> 回到第02步
  - 02> 退出Loop
- 11、通知Observers：退出Loop

RunLoop的运行逻辑



RunLoop休眠的实现原理



- 等待消息
- 没有消息就让线程休眠
- 有消息就唤醒线程

小码哥教育 SEEMYGO

RunLoop在实际开中的应用

- 控制线程生命周期（线程保活）
- 解决NSTimer在滑动时停止工作的问题
- 监控应用卡顿
- 性能优化