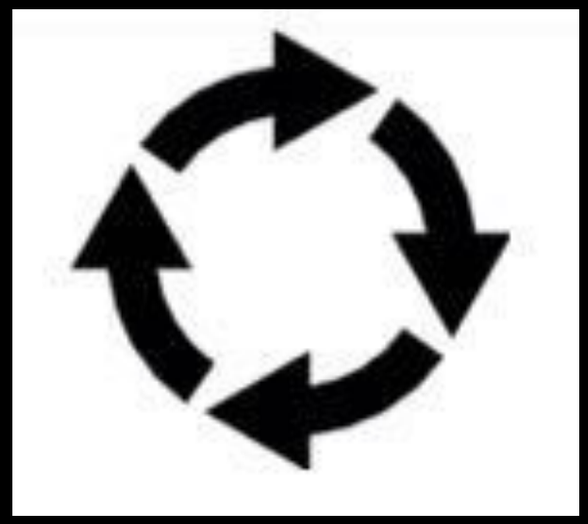
RunLoop

- 什么是RunLoop
- 基本作用、存在价值
- RunLoop对象、线程、类
- RunLoop处理逻辑
- RunLoop应用

什么是RunLoop?

- iOS 和 OSX 开发中的一个核心概念
- 运行循环
- 跑圈



OS X/iOS 的系统架构

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

存在价值

- 如果没有RunLoop

```
int main(int argc, char * argv[]) {
    NSLog(@"execute main function");      // 程序开始

    return 0;                             // 程序结束
}
```

- 如果有了RunLoop

```
int main(int argc, char * argv[]) {
    BOOL running = YES;                    // 程序开始
    do {
        // 执行各种任务，处理各种事件
        // ......
    } while (running);

    return 0;
}
```

由于RunLoop启动了一个RunLoop，程序并不会马上退出，保持持续运行状态。

```objc
int main(int argc, char * argv[]) {
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate
            class]));
    }
}
```

- 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

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封装

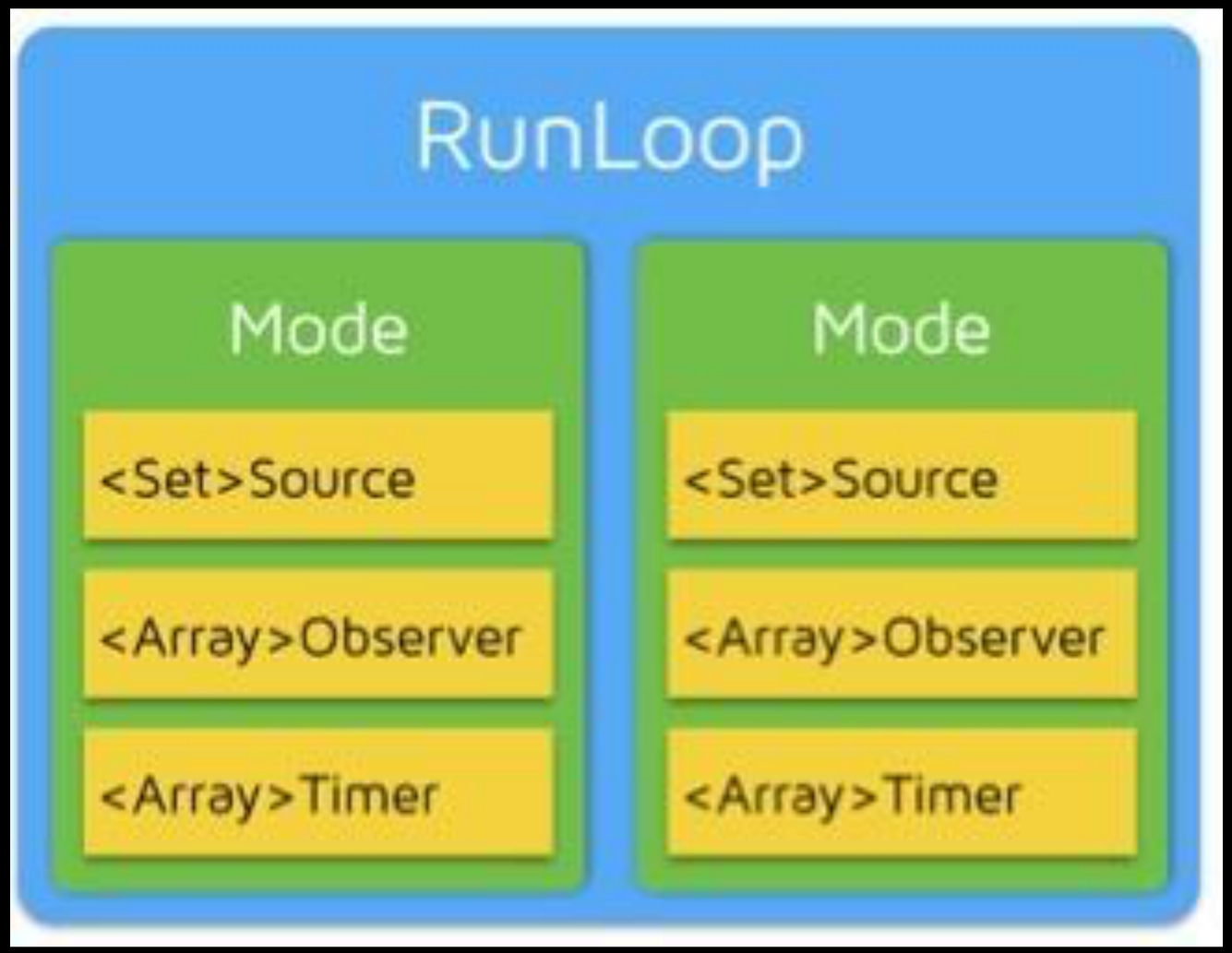
RunLoop相关类

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
    - 这样的目的是分隔开不同组的Source/Timer/Observer，让其互不影响


RunLoop
Mode
<Set>Source
<Array>Observer
<Array>Timer
Mode
<Set>Source
<Array>Observer
<Array>Timer

系统默认注册了5个Mode:

- **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

- **kCFRunLoopCommonModes**: 这是一个占位用的Mode，不是一种真正的Mode

- 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

CFRunLoopTimerRef

- CFRunLoopTimerRef是基于时间的触发器
- iOS中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

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

CFRunLoopSourceRef

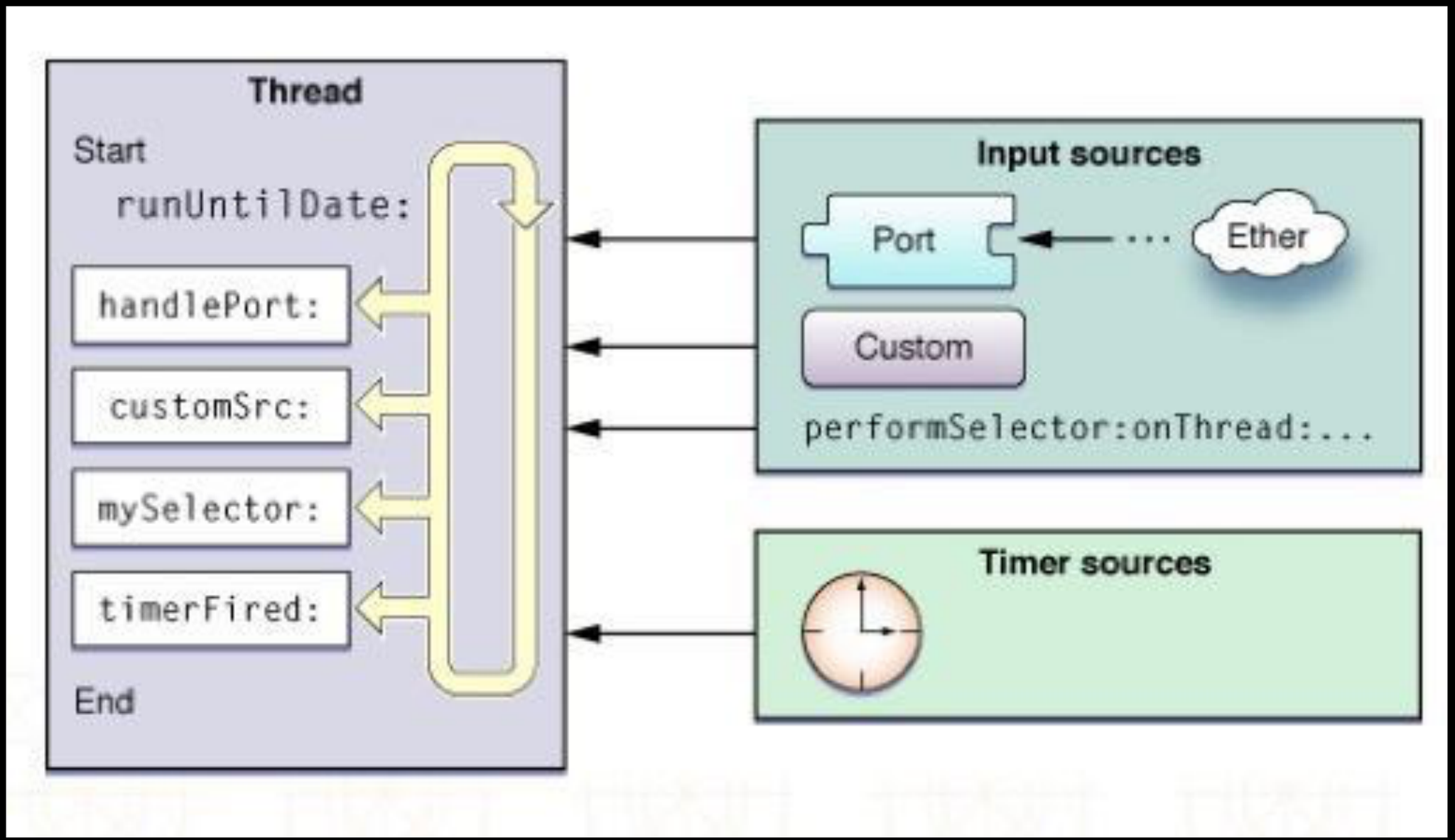
- CFRunLoopSourceRef是事件源（输入源）
- 按照官方文档的分类
  - Port-Based Sources (基于端口,跟其他线程交互,通过内核发布的消息)
  - Custom Input Sources (自定义)
  - Cocoa Perform Selector Sources (performSelector...方法事件源)
- 按照函数调用栈的分类
  - Source0：非基于Port的（用户主动触发的事件）
  - Source1：基于Port的（通过内核和其他线程发送的消息）

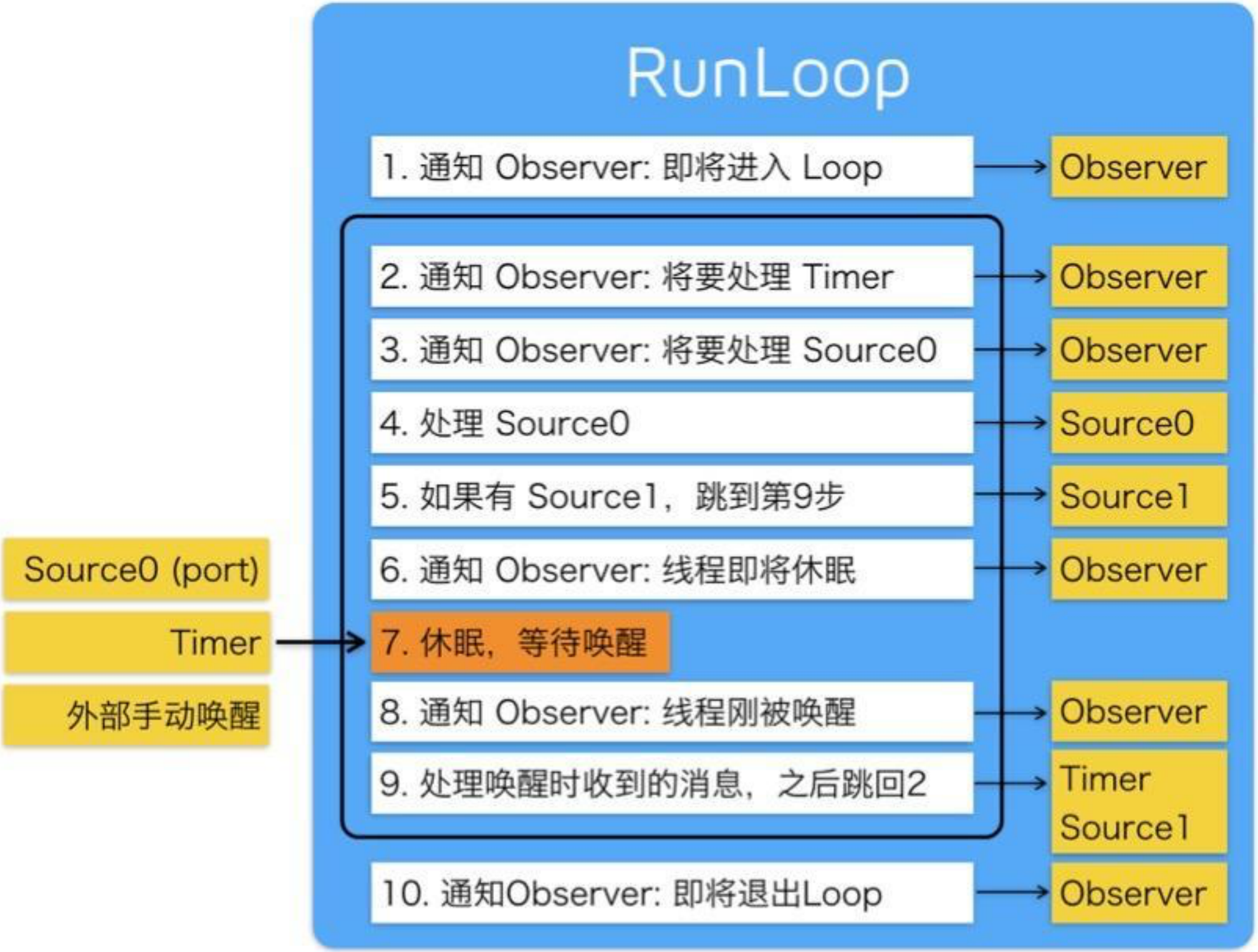
CFRunLoopObserverRef

- 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

```
/* Run Loop Observer Activities */
typedef CF_OPTIONS(CFOptionFlags, CFRunLoopActivity) {
    kCFRunLoopEntry = (1UL << 0),
    kCFRunLoopBeforeTimers = (1UL << 1),
    kCFRunLoopBeforeSources = (1UL << 2),
    kCFRunLoopBeforeWaiting = (1UL << 5),
    kCFRunLoopAfterWaiting = (1UL << 6),
    kCFRunLoopExit = (1UL << 7),
    kCFRunLoopAllActivities = 0x0FFFFFFFU
};
```

RunLoop处理逻辑




RunLoop
1. 通知 Observer: 即将进入 Loop
Observer
2. 通知 Observer: 将要处理 Timer
Observer
3. 通知 Observer: 将要处理 Source0
Observer
4. 处理 Source0
Source0
5. 如果有 Source1，跳到第9步
Source1
6. 通知 Observer: 线程即将休眠
Observer
Source0 (port)
Timer
外部手动唤醒
7. 休眠，等待唤醒
8. 通知 Observer: 线程刚被唤醒
Observer
9. 处理唤醒时收到的消息，之后跳回2
Timer
Source1
10. 通知Observer: 即将退出Loop
Observer

RunLoop应用

- NSTimer
- 监听RunLoop状态
- ImageView显示
- PerformSelector
- 常驻线程

Demo