iOS学习笔记12—Runloop

一、Runloop简介:

Run loops 是线程相关的的基础框架的一部分。一个 run loop 就是一个事件处理 的循环,用来不停的调度工作以及处理输入事件。

使用 run loop的目的是让你的线程在有工作的时候忙于工作,而没工作的时候处于休眠状态。

Runloop还可以在loop在循环中的同时响应其他输入源，比如界面控件的按钮，手势等。

Run loop 接收输入事件来自两种不同的来源:

输入源(input source)和定时源 (timer source)。

输入源传递异步事件,通常消息来自于其他线程或程序。输入源的种类:基于端口的输入源和自定义输入源。

定时源则传递同步事件,发生在特定时间或者重复的时间间隔。

Run loop 模式是所有要监视的输入源和定时源以及要通知的 run loop 注册观察 者的集合。

可以将 Run loop 观察者和以下事件关联:

Run loop 入口

Run loop 何时处理一个定时器

Run loop 何时处理一个输入源

Run loop 何时进入睡眠状态

Run loop 何时被唤醒,但在唤醒之前要处理的事件

Run loop 终止

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。

Run loop 在你要和线程有更多的交互时才需要,比如以下情况:

使用端口或自定义输入源来和其他线程通信;

使用线程的定时器;

Cocoa 中使用任何performSelector...的方法;

使线程周期性工作。

二、举例说明Runloop的优点。

一般情况下，当我们使用NSRunLoop的时候，代码如下所示:

```
do {
    [[NSRunLoop currentRunLoop]
runMode:NSDefaultRunLoopModebeforeDate:[NSDate
distantFuture]];
} while (!done);
```

在上面的代码中，参数done为NO的时候，当前runloop会一直接收处理其他输入源，处理输入源之后会再回到runloop中等待其他的输入源；除非done为NO，否则当前流程一直再runloop中。

如下面的代码片段所示，有三个按钮，分别对应如下三个action消息，buttonNormalThreadTestPressed，buttonRunloopPressed，buttonTestPressed。

buttonNormalThreadTestPressed：启动一个线程，在while循环中等待线程执行完再接着往下运行。

buttonRunloopPressed：启动一个线程，使用runloop，等待线程执行完再接着往下运行。

buttonTestPressed：仅仅打印两条日志，用来测试UI是否能立即响应的。

在本测试中，待程序运行后，做如下操作对比：

1、点击buttonNormalThreadTestPressed，然后立刻点击buttonTestPressed，查看日志输出。

2、待1完成后，点击buttonRunloopPressed，然后立刻点击buttonTestPressed，查看日志输出，跟1的日志做对比，即可以发现步骤2即使线程没有完成，在runloop等待过程中，界面仍然能够响应。

```
BOOL threadProcess1Finished =NO;

-(void)threadProce1{

    NSLog(@"Enter threadProce1.");


    for (int i=0; i<5;i++) {

        NSLog(@"InthreadProce1 count = %d.", i);
```

```
        sleep(1);
    }
    threadProcess1Finished =YES;
    NSLog(@"Exit threadProce1.");
}

BOOL threadProcess2Finished =NO;
-(void)threadProce2{
    NSLog(@"Enter threadProce2.");

    for (int i=0; i<5;i++) {
        NSLog(@"InthreadProce2 count = %d.", i);
        sleep(1);
    }

    threadProcess2Finished =YES;
    NSLog(@"Exit threadProce2.");
}

- (IBAction)buttonNormalThreadTestPressed:(UIButton *)sender {
```

```
    NSLog(@"EnterbuttonNormalThreadTestPressed");

    threadProcess1Finished =NO;
    NSLog(@"Start a new thread.");
    [NSThreaddetachNewThreadSelector: @selector(threadProce1)
                            toTarget: self
                          withObject: nil];

    // 通常等待线程处理完后再继续操作的代码如下面的形式。
    // 在等待线程threadProce1结束之前，调用buttonTestPressed，界面没有响应，直到threadProce1运行完，才打印buttonTestPressed里面的日志。
    while (!threadProcess1Finished) {
        [NSThreadsleepForTimeInterval: 0.5];
    }

    NSLog(@"ExitbuttonNormalThreadTestPressed");
}

- (IBAction)buttonRunloopPressed:(id)sender {
```

```
    NSLog(@"Enter buttonRunloopPressed");
    threadProcess2Finished =NO;
    NSLog(@"Start a new thread.");
    [NSThreaddetachNewThreadSelector: @selector(threadProce2)
                            toTarget: self
                          withObject: nil];

    // 使用runloop，情况就不一样了。
    // 在等待线程threadProce2结束之前，调用buttonTestPressed，界面立马响应，并打印buttonTestPressed里面的日志。
    // 这就是runloop的神奇所在
    while (!threadProcess2Finished) {
        NSLog(@"Begin runloop");
        [[NSRunLoopcurrentRunLoop] runMode:NSDefaultRunLoopMode
                                beforeDate: [NSDate distantFuture]];
        NSLog(@"End runloop.");
    }
    NSLog(@"Exit buttonRunloopPressed");
}
```

```
- (IBAction)buttonTestPressed:(id)sender{
    NSLog(@"Enter buttonTestPressed");
    NSLog(@"Exit buttonTestPressed");
}
```

日志信息如下：

2013-04-07 14:25:22.829 Runloop[657:11303] EnterbuttonNormalThreadTestPressed

2013-04-07 14:25:22.830 Runloop[657:11303] Start a new thread.

2013-04-07 14:25:22.831 Runloop[657:1250f] Enter threadProce1.

2013-04-07 14:25:22.832 Runloop[657:1250f] In threadProce1 count = 0.

2013-04-07 14:25:23.833 Runloop[657:1250f] In threadProce1 count = 1.

2013-04-07 14:25:24.834 Runloop[657:1250f] In threadProce1 count = 2.

2013-04-07 14:25:25.835 Runloop[657:1250f] In threadProce1 count = 3.

2013-04-07 14:25:26.837 Runloop[657:1250f] In threadProce1 count = 4.

2013-04-07 14:25:27.839 Runloop[657:1250f] Exit threadProce1.

2013-04-07 14:25:27.840 Runloop[657:11303]Exit buttonNormalThreadTestPressed

2013-04-07 14:25:27.841 Runloop[657:11303]Enter buttonTestPressed

2013-04-07 14:25:27.842 Runloop[657:11303] Exit buttonTestPressed

2013-04-07 14:25:27.843 Runloop[657:11303] Enter buttonTestPressed

2013-04-07 14:25:27.844 Runloop[657:11303] Exit buttonTestPressed

2013-04-07 14:43:41.790 Runloop[657:11303] Enter buttonRunloopPressed

2013-04-07 14:43:41.790 Runloop[657:11303] Start a new thread.

2013-04-07 14:43:41.791 Runloop[657:11303] Begin runloop

2013-04-07 14:43:41.791 Runloop[657:14f0b] Enter threadProce2.

2013-04-07 14:43:41.792 Runloop[657:14f0b] In threadProce2 count = 0.

2013-04-07 14:43:42.542 Runloop[657:11303] End runloop.

2013-04-07 14:43:42.543 Runloop[657:11303] Begin runloop

2013-04-07 14:43:42.694 Runloop[657:11303]Enter

buttonTestPressed

2013-04-07 14:43:42.694 Runloop[657:11303]Exit buttonTestPressed

2013-04-07 14:43:42.695 Runloop[657:11303] End runloop.

2013-04-07 14:43:42.696 Runloop[657:11303] Begin runloop

2013-04-07 14:43:42.793 Runloop[657:14f0b] In threadProce2 count = 1.

2013-04-07 14:43:43.326 Runloop[657:11303] End runloop.

2013-04-07 14:43:43.327 Runloop[657:11303] Begin runloop

2013-04-07 14:43:43.438 Runloop[657:11303]Enter buttonTestPressed

2013-04-07 14:43:43.438 Runloop[657:11303]Exit buttonTestPressed

2013-04-07 14:43:43.439 Runloop[657:11303] End runloop.

2013-04-07 14:43:43.440 Runloop[657:11303] Begin runloop

2013-04-07 14:43:43.795 Runloop[657:14f0b] In threadProce2 count = 2.

2013-04-07 14:43:44.797 Runloop[657:14f0b] In threadProce2 count = 3.

2013-04-07 14:43:45.798 Runloop[657:14f0b] In threadProce2 count = 4.

2013-04-07 14:43:46.800 Runloop[657:14f0b] Exit threadProce2.

三、Runloop简单实例：

```
- (void)viewDidLoad

{

    [superviewDidLoad];

    // Doany additional setup after loading the view, typically from a
nib.


    [NSThreaddetachNewThreadSelector:
@selector(newThreadProcess)

                            toTarget: self

                          withObject: nil];

}



- (void)newThreadProcess

{

    @autoreleasepool {

        ////获得当前thread的Runloop

        NSRunLoop* myRunLoop = [NSRunLoop currentRunLoop];
```

```
    //设置Run loop observer的运行环境
    CFRunLoopObserverContext context = {0,self,NULL,NULL,NULL};

    //创建Run loop observer对象
    //第一个参数用于分配observer对象的内存
    //第二个参数用以设置observer所要关注的事件，详见回调函数myRunLoopObserver中注释
    //第三个参数用于标识该observer是在第一次进入runloop时执行还是每次进入run loop处理时均执行
    //第四个参数用于设置该observer的优先级
    //第五个参数用于设置该observer的回调函数
    //第六个参数用于设置该observer的运行环境
    CFRunLoopObserverRef observer =CFRunLoopObserverCreate(kCFAllocatorDefault,kCFRunLoopAllActivities, YES, 0, &myRunLoopObserver, &context);
    if(observer)
    {
        //将Cocoa的NSRunLoop类型转换成CoreFoundation的CFRunLoopRef类型
        CFRunLoopRef cfRunLoop = [myRunLoop getCFRunLoop];
```

```
        //将新建的observer加入到当前thread的runloop
        CFRunLoopAddObserver(cfRunLoop, observer,
kCFRunLoopDefaultMode);
    }

    //
    [NSTimerscheduledTimerWithTimeInterval: 1
                                    target: self
                                  selector:@selector(timerProcess)
                                  userInfo: nil
                                   repeats: YES];
    NSInteger loopCount = 2;
    do{
        //启动当前thread的loop直到所指定的时间到达，在loop运行时，runloop会处理所有来自与该run loop联系的inputsource的数据
        //对于本例与当前run loop联系的inputsource只有一个Timer类型的source。
        //该Timer每隔1秒发送触发事件给runloop，run loop检测到该事件时会调用相应的处理方法。
```

```
        //由于在run loop添加了observer且设置observer对所有的runloop行为都感兴趣。

        //当调用runUnitDate方法时，observer检测到runloop启动并进入循环，observer会调用其回调函数，第二个参数所传递的行为是kCFRunLoopEntry。

        //observer检测到runloop的其它行为并调用回调函数的操作与上面的描述相类似。

        [myRunLoop runUntilDate:[NSDate dateWithTimeIntervalSinceNow:5.0]];

        //当run loop的运行时间到达时，会退出当前的runloop。observer同样会检测到runloop的退出行为并调用其回调函数，第二个参数所传递的行为是kCFRunLoopExit。

        loopCount--;

    }while (loopCount);

  }

}


void myRunLoopObserver(CFRunLoopObserverRef observer,CFRunLoopActivityactivity,void *info)

{

  switch (activity) {

        //The entrance of the run loop, beforeentering the event processing loop.
```

```
        //This activity occurs once for each callto CFRunLoopRun
and CFRunLoopRunInMode
    case kCFRunLoopEntry:
        NSLog(@"run loop entry");
        break;
        //Inside the event processing loop beforeany timers are
processed
    case kCFRunLoopBeforeTimers:
        NSLog(@"run loop before timers");
        break;
        //Inside the event processing loop beforeany sources are
processed
    case kCFRunLoopBeforeSources:
        NSLog(@"run loop before sources");
        break;
        //Inside the event processing loop beforethe run loop sleeps,
waiting for a source or timer to fire.
        //This activity does not occur ifCFRunLoopRunInMode is
called with a timeout of 0 seconds.
        //It also does not occur in a particulariteration of the event
processing loop if a version 0 source fires
    case kCFRunLoopBeforeWaiting:
```

```
            NSLog(@"run loop before waiting");
            break;
            //Inside the event processing loop afterthe run loop wakes up, but before processing the event that woke it up.
            //This activity occurs only if the run loopdid in fact go to sleep during the current loop
        case kCFRunLoopAfterWaiting:
            NSLog(@"run loop after waiting");
            break;
            //The exit of the run loop, after exitingthe event processing loop.
            //This activity occurs once for each callto CFRunLoopRun and CFRunLoopRunInMode
        case kCFRunLoopExit:
            NSLog(@"run loop exit");
            break;
            /*
             A combination of all the precedingstages
             case kCFRunLoopAllActivities:
             break;
             */
```

```
        default:
            break;
    }
}


- (void)timerProcess{

    for (int i=0; i<5; i++) {
        NSLog(@"In timerProcess count = %d.", i);
        sleep(1);
    }
}
```

调试打印信息如下：

2012-12-18 09:51:14.174 Texta[645:14807] run loop entry

2012-12-18 09:51:14.175 Texta[645:14807] run loop before timers

2012-12-18 09:51:14.176 Texta[645:14807] run loop before sources

2012-12-18 09:51:14.177 Texta[645:14807] run loop before

waiting

2012-12-18 09:51:15.174 Texta[645:14807] run loop after waiting

2012-12-18 09:51:15.176 Texta[645:14807] In timerProcess count = 0.

2012-12-18 09:51:16.178 Texta[645:14807] In timerProcess count = 1.

2012-12-18 09:51:17.181 Texta[645:14807] In timerProcess count = 2.

2012-12-18 09:51:18.183 Texta[645:14807] In timerProcess count = 3.

2012-12-18 09:51:19.185 Texta[645:14807] In timerProcess count = 4.

2012-12-18 09:51:20.187 Texta[645:14807] run loop exit

2012-12-18 09:51:20.189 Texta[645:14807] run loop entry

2012-12-18 09:51:20.190 Texta[645:14807] run loop before timers

2012-12-18 09:51:20.191 Texta[645:14807] run loop before sources

2012-12-18 09:51:20.191 Texta[645:14807] run loop before waiting

2012-12-18 09:51:21.174 Texta[645:14807] run loop after waiting

2012-12-18 09:51:21.176 Texta[645:14807] In timerProcess count = 0.

2012-12-18 09:51:22.178 Texta[645:14807] In timerProcess count = 1.

2012-12-18 09:51:23.181 Texta[645:14807] In timerProcess count = 2.

2012-12-18 09:51:24.183 Texta[645:14807] In timerProcess count = 3.

2012-12-18 09:51:25.185 Texta[645:14807] In timerProcess count = 4.

2012-12-18 09:51:26.187 Texta[645:14807] run loop exit

四、Runloop可以阻塞线程，等待其他线程执行后再执行。

比如：

```
BOOL StopFlag =NO;

- (void)viewDidLoad

{

    [superviewDidLoad];

     // Doany additional setup after loading the view, typically from a nib.


    StopFlag =NO;

    NSLog(@"Start a new thread.");

    [NSThreaddetachNewThreadSelector:
```

```
@selector(newThreadProc)
                                toTarget:self
                              withObject: nil];
    while (!StopFlag) {
        NSLog(@"Beginrunloop");
        [[NSRunLoopcurrentRunLoop]
runMode:NSDefaultRunLoopMode
                                   beforeDate: [NSDate distantFuture]];
        NSLog(@"Endrunloop.");
    }

    NSLog(@"OK");
}


-(void)newThreadProc{
    NSLog(@"Enter newThreadProc.");

    for (int i=0; i<10; i++) {
        NSLog(@"InnewThreadProc count = %d.", i);
```

```
        sleep(1);
    }

    StopFlag =YES;

    NSLog(@"Exit newThreadProc.");
}
}
```

调试打印信息如下：

2012-12-18 08:50:34.220 Runloop[374:11303] Start a new thread.

2012-12-18 08:50:34.222 Runloop[374:11303] Begin runloop

2012-12-18 08:50:34.222 Runloop[374:14b03] Enter newThreadProc.

2012-12-18 08:50:34.223 Runloop[374:14b03] In newThreadProc count = 0.

2012-12-18 08:50:35.225 Runloop[374:14b03] In newThreadProc count = 1.

2012-12-18 08:50:36.228 Runloop[374:14b03] In newThreadProc count = 2.

2012-12-18 08:50:37.230 Runloop[374:14b03] In newThreadProc count = 3.

2012-12-18 08:50:38.233 Runloop[374:14b03] In newThreadProc count = 4.

2012-12-18 08:50:39.235 Runloop[374:14b03] In newThreadProc count = 5.

2012-12-18 08:50:40.237 Runloop[374:14b03] In newThreadProc count = 6.

2012-12-18 08:50:41.240 Runloop[374:14b03] In newThreadProc count = 7.

2012-12-18 08:50:42.242 Runloop[374:14b03] In newThreadProc count = 8.

2012-12-18 08:50:43.245 Runloop[374:14b03] In newThreadProc count = 9.

2012-12-18 08:50:44.247 Runloop[374:14b03] Exit newThreadProc.

2012-12-18 08:51:00.000 Runloop[374:11303] End runloop.

2012-12-18 08:51:00.001 Runloop[374:11303] OK

从调试打印信息可以看到，while循环后执行的语句会在很长时间后才被执行。因为，改变变量StopFlag的值，runloop对象根本不知道，runloop在这个时候未被唤醒。有其他事件在某个时点唤醒了主线程，这才结束了while循环，但延缓的时长总是不定的。。

将代码稍微修改一下：

[[NSRunLoopcurrentRunLoop] runMode:NSDefaultRunLoopMode

```
                                beforeDate:
[NSDatedateWithTimeIntervalSinceNow: 1]];
```

缩短runloop的休眠时间，看起来解决了上面出现的问题。

但这样会导致runloop被经常性的唤醒，违背了runloop的设计初衷。runloop的目的就死让你的线程在有工作的时候忙于工作，而没工作的时候处于休眠状态。

最后，看下下面正确的写法：

```
BOOL StopFlag =NO;

- (void)viewDidLoad

{

    [superviewDidLoad];

     // Doany additional setup after loading the view, typically from a
nib.


    StopFlag =NO;

    NSLog(@"Start a new thread.");

    [NSThreaddetachNewThreadSelector:
@selector(newThreadProc)

                            toTarget: self

                          withObject: nil];

    while (!StopFlag) {
```

```
        NSLog(@"Beginrunloop");
        [[NSRunLoopcurrentRunLoop]
runMode:NSDefaultRunLoopMode
                            beforeDate: [NSDatedistantFuture]];
        NSLog(@"Endrunloop.");
    }

    NSLog(@"OK");
}

-(void)newThreadProc{
    NSLog(@"Enter newThreadProc.");

    for (int i=0; i<10; i++) {
        NSLog(@"InnewThreadProc count = %d.", i);
        sleep(1);
    }
    [selfperformSelectorOnMainThread: @selector(setEnd)
                          withObject: nil
```

```
                    waitUntilDone: NO];


    NSLog(@"Exit newThreadProc.");
}
-(void)setEnd{
    StopFlag = YES;
}
```

调试打印信息如下：

2012-12-18 09:05:17.161 Runloop[410:11303] Start a new thread.

2012-12-18 09:05:17.163 Runloop[410:14a03] Enter newThreadProc.

2012-12-18 09:05:17.164 Runloop[410:14a03] In newThreadProc count = 0.

2012-12-18 09:05:17.165 Runloop[410:11303] Begin runloop

2012-12-18 09:05:18.166 Runloop[410:14a03] In newThreadProc count = 1.

2012-12-18 09:05:19.168 Runloop[410:14a03] In newThreadProc count = 2.

2012-12-18 09:05:20.171 Runloop[410:14a03] In newThreadProc count = 3.

2012-12-18 09:05:21.173 Runloop[410:14a03] In newThreadProc count = 4.

2012-12-18 09:05:22.175 Runloop[410:14a03] In newThreadProc count = 5.

2012-12-18 09:05:23.178 Runloop[410:14a03] In newThreadProc count = 6.

2012-12-18 09:05:24.180 Runloop[410:14a03] In newThreadProc count = 7.

2012-12-18 09:05:25.182 Runloop[410:14a03] In newThreadProc count = 8.

2012-12-18 09:05:26.185 Runloop[410:14a03] In newThreadProc count = 9.

2012-12-18 09:05:27.188 Runloop[410:14a03] Exit newThreadProc.

2012-12-18 09:05:27.188 Runloop[410:11303] End runloop.

2012-12-18 09:05:27.189 Runloop[410:11303] OK

把直接设置变量，改为向主线程发送消息，唤醒runloop，延时问题解决。