学习过程中，将Threading PG中的Run Loops翻译了下，权当是做为笔记。原文见 [Run Loops。](http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/Multithreading/RunLoopManagement/RunLoopManagement.html)

转载请注明，谢谢。

http://www.cnblogs.com/scorpiozj/

Run loops是线程的基础架构部分。一个run loop就是一个事件处理循环，用来不停的调配工作以及处理输入事件。使用run loop的目的是使你的线程在有工作的时候工作，没有的时候休眠。

Run loop的管理并不完全是自动的。你仍必须设计你的线程代码以在适当的时候启动run loop并正确响应输入事件。Cocoa和CoreFundation都提供了run loop对象方便配置和管理线程的run loop。你创建的程序不需要显示的创建run loop；每个线程，包括程序的主线程（main thread）都有与之相应的run loop对象。但是，自己创建的次线程是需要手动运行run loop的。在carbon和cocoa程序中，程序启动时，主线程会自行创建并运行run loop。

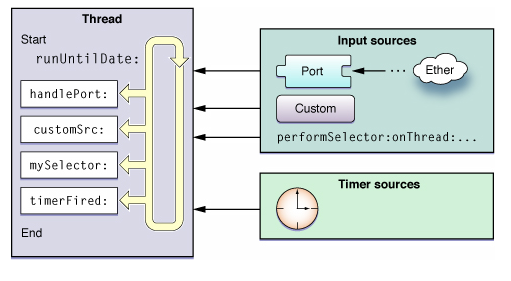
接下来的部分将会详细介绍run loop以及如何为你的程序管理run loop。关于run loop对象可以参阅sdk文档。

解析Run Loop

run loop，顾名思义，就是一个循环，你的线程在这里开始，并运行事件处理程序来响应输入事件。你的代码要有实现循环部分的控制语句，换言之就是要有while或for语句。在run loop中，使用run loop对象来运行事件处理代码：响应接收到的事件，启动已经安装的处理程序。

Run loop处理的输入事件有两种不同的来源：输入源（input source）和定时源（timer source）。输入源传递异步消息，通常来自于其他线程或者程序。定时源则传递同步消息，在特定时间或者一定的时间间隔发生。两种源的处理都使用程序的某一特定处理路径。

图1－1显示了run loop的结构以及各种输入源。输入源传递异步消息给相应的处理程序，并调用runUntilDate：方法退出。定时源则直接传递消息给处理程序，但并不会退出run loop。

除了处理输入源，run loop也会生成关于run loop行为的notification。注册的run－loop 观察者可以收到这些notification，并做相应的处理。可以使用Core Foundation在你的线程注册run－loop观察者。

下面介绍run loop的组成，以及其运行的模式。同时也提及在处理程序中不同时间发送不同的notification。

Run Loop Modes

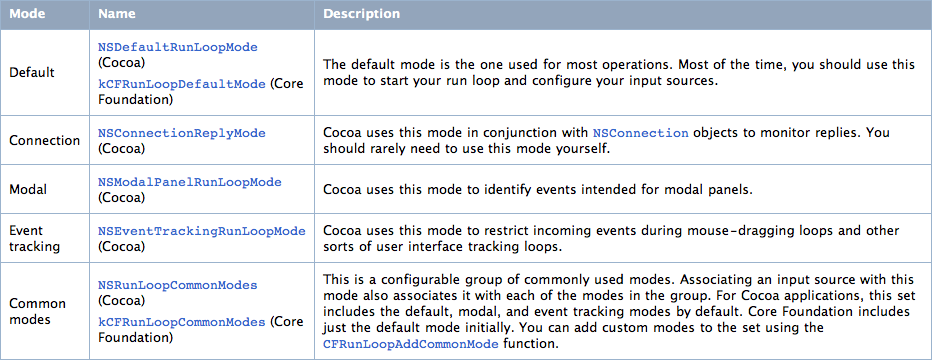
Run loop模式是所有要监视的输入源和定时源以及要通知的注册观察者的集合。每次运行run loop都会指定其运行在哪个模式下。以后，只有相应的源会被监视并允许接收他们传递的消息。（类似的，只有相应的观察者会收到通知）。其他模式关联的源只有在run loop运行在其模式下才会运行，否则处于暂停状态。

通常代码中通过指定名字来确定模式。Cocoa和core foundation定义了默认的以及一系列常用的模式，都是用字符串来标识。当然你也可以指定字符串来自定义模式。虽然你可以给模式指定任何名字，但是所有的模式内容都是相同的。你必须添加输入源，定时器或者run loop观察者到你定义的模式中。

通过指定模式可以使得run loop在某一阶段只关注感兴趣的源。大多数时候，run loop都是运行在系统定义的默认模式。但是模态面板（modal panel）可以运行在 “模态”模式下。在这种模式下，只有和模态面板相关的源可以传递消息给线程。对于次线程，可以使用自定义模式处理时间优先的操作，即屏蔽优先级低的源传递消息。

Note：模式区分基于事件的源而非事件的种类。例如，你不可以使用模式只选择处理鼠标按下或者键盘事件。你可以使用模式监听端口， 暂停定时器或者其他对源或者run loop观察者的处理，只要他们在当前模式下处于监听状态。

表1－1列出了cocoa和Core Foundation预先定义的模式。



输入源

输入源向线程发送异步消息。消息来源取决于输入源的种类：基于端口的输入源和自定义输入源。基于端口的源监听程序相应的端口，而自定义输入源则关注自定义的消息。至于run loop，它不关心输入源的种类。系统会去实现两种源供你使用。两类输入源的区别在于如何显示的：基于端口的源由内核自动发送，而自定义的则需要人工从其他线程发送。

当你创建输入源，你需要将其分配给run loop中的一个或多个模式。模式只会在特定事件影响监听的源。大多数情况下，run loop运行在默认模式下，但是你也可以使其运行在自定义模式。若某一源在当前模式下不被监听，那么任何其生成的消息只有当run loop运行在其关联的模式下才会被传递。

下面讨论这几种输入源。

http://www.cnblogs.com/scorpiozj/

基于端口的源：

cocoa和core foundation为使用端口相关的对象和函数创建的基于端口的源提供了内在支持。Cocoa中你从不需要直接创建输入源。你只需要简单的创建端口对象，并使用NSPort的方法将端口对象加入到run loop。端口对象会处理创建以及配置输入源。

在core foundation，你必须手动的创建端口和源，你都可以使用端口类型（CFMachPortRef，CFMessagePortRef，CFSocketRef）来创建。

更多例子可以看 配置基于端口的源。

自定义输入源：

在Core Foundation程序中，必须使用CFRunLoopSourceRef类型相关的函数来创建自定义输入源，接着使用回调函数来配置输入源。Core Fundation会在恰当的时候调用回调函数，处理输入事件以及清理源。

除了定义如何处理消息，你也必须定义源的消息传递机制——它运行在单独的进程，并负责传递数据给源和通知源处理数据。消息传递机制的定义取决于你，但最好不要过于复杂。

关于创建自定义输入源的例子，见 定义自定义输入源。关于自定义输入源的信息参见CFRunLoopSource。

Cocoa Perform Selector Sources：

除了基于端口的源，Cocoa提供了可以在任一线程执行函数（perform selector）的输入源。和基于端口的源一样，perform selector请求会在目标线程上序列化，减缓许多在单个线程上容易引起的同步问题。而和基于端口的源不同的是，perform selector执行完后会自动清除出run loop。

当perform selector在其它线程中执行时，目标线程须有一活动中的run loop。对于你创建的线程而言，这意味着线程直到你显示的开始run loop否则处于等待状态。然而，由于主线程自己启动run loop，在程序调用applicationDidFinishlaunching：的时候你会遇到线程调用的问题。因为Run loop通过每次循环来处理所有排列的perform selector调用，而不时通过每次的循环迭代来处理perform selector。

定时源

定时源在预设的时间点同步地传递消息。定时器时线程通知自己做某事的一种方法。例如，搜索控件可以使用定时器，当用户连续输入的时间超过一定时间时，就开始一次搜索。这样，用户就可以有足够的时间来输入想要搜索的关键字。

尽管定时器和时间有关，但它并不是实时的。和输入源一样，定时器也是和run loop的运行模式相关联的。如果定时器所在的模式未被run loop监视，那么定时器将不会开始直到run loop运行在相应的模式下。类似的，如果定时器在run loop处理某一事件时开始，定时器会一直等待直到下次run loop开始相应的处理程序。如果run loop不再运行，那定时器也将永远不开始。

你可以选择定时器工作一次还是定时工作。如果定时工作，定时器会基于安排好的时间而非实际时间，自动的开始。举个例子，定时器在某一特定时间开始并设置5 秒重复，那么定时器会在那个特定时间后5秒启动，即使在那个特定时间定时器延时启动了。如果定时器延迟到接下来设定的一个会多个5秒，定时器在这些时间段 中也只会启动一次，在此之后，正常运行。（假设定时器在时间1，5，9。。。运行，如果最初延迟到7才启动，那还是从9，13，。。。开始）。

Run Loop观察者

源是同步或异步的传递消息，而run loop观察者则是在运行run loop的时候在特定的时候开始。你可以使用run loop观察者来为某一特定事件或是进入休眠的线程做准备。你可以将观察者将以下事件关联：

* Run loop入口
* Run loop将要开始定时
* Run loop将要处理输入源
* Run loop将要休眠
* Run loop被唤醒但又在执行唤醒事件前
* Run loop终止

你可以给cocoa和carbon程序随意添加观察者，但是如果你要定义观察者的话就只能使用core fundation。使用CFRunLoopObserverRed类型来创建观察者实例，它会追踪你自定义的回调函数以及其它你感兴趣的地方。

和定时器类似，观察者可以只用一次或循环使用。若只用一次，那在结束的时候会移除run loop，而循环的观察者则不会。你需要制定观察者是一次／多次使用。

消息的run loop顺序

每次启动，run loop会自动处理之前未处理的消息，并通知观察者。具体的顺序，如下：

1. 通知观察者，run loop启动
2. 通知观察者任何即将要开始的定时器
3. 通知观察者任何非基于端口的源即将启动
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. 将线程之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
* 设置了run loop的终止时间
* run loop唤醒

1. 通知观察者线程将被唤醒。
2. 处理未处理的事件

* 如果用户定义的定时器启动，处理定时事件并重启run loop。进入步骤2
* 如果输入源启动，传递相应的消息
* run loop唤醒但未终止，重启。进入步骤2

1. 通知观察者run loop结束。

（标号应该连续，不知道怎么改）

因为观察者的消息传递是在相应的事件发生之前，所以两者之间可能存在误差。如果需要精确时间控制，你可以使用休眠和唤醒通知以此来校对实际发生的事件。

因为定时器和其它周期性事件那是在run loop运行后才启动，撤销run loop也会终止消息传递。典型的例子就是鼠标路径追踪。因为你的代码直接获取到消息而不是经由程序传递，从而不会在实际的时间开始而须使得鼠标追踪结束并将控制权交给程序后才行。

使用run loop对象可以唤醒Run loop。其它消息也可以唤醒run loop。例如，添加新的非基于端口的源到run loop从而可以立即执行输入源而不是等待其他事件发生后再执行。

何时使用Run Loop

http://www.cnblogs.com/scorpiozj/archive/2011/05/26/2058167.html

只有在为你的程序创建次线程的时候，才需要运行run loop。对于程序的主线程而言，run loop是关键部分。Cocoa和carbon程序提供了运行主线程run loop的代码同时也会自动运行run loop。IOS程序UIApplication中的run方法在程序正常启动的时候就会启动run loop。同样的这部分工作在carbon程序中由RunApplicationEventLoop负责。如果你使用xcode提供的模板创建的程序，那你永远不需要自己去启动run loop。

而对于次线程，你需要判断是否需要run loop。如果需要run loop，那么你要负责配置run loop并启动。你不需要在任何情况下都去启动run loop。比如，你使用线程去处理一个预先定义好的耗时极长的任务时，你就可以毋需启动run loop。Run loop只在你要和线程有交互时才需要，比如以下情况：

* 使用端口或自定义输入源和其他线程通信
* 使用定时器
* cocoa中使用任何performSelector
* 使线程履行周期性任务

如果决定在程序中使用run loop，那么配置和启动都需要自己完成。和所有线程编程一样，你需要计划好何时退出线程。在退出前结束线程往往是比被强制关闭好的选择。详细的配置和推出run loop的信息见 使用run loop对象。

使用Run loop对象

run loop对象提供了添加输入源，定时器和观察者以及启动run loop的接口。每个线程都有唯一的与之关联的run loop对象。在cocoa中，是NSRunLoop对象；而在carbon或BSD程序中则是指向CFRunLoopRef类型的指针。

获得run loop对象

获得当前线程的run loop，可以采用：

* cocoa：使用NSRunLoop的currentRunLoop类方法
* 使用CFRunLoopGetCurrent函数

虽然CFRunLoopRef类型和NSRunLoop对象并不完全等价，你还是可以从NSRunLoop对象中获取CFRunLoopRef类型。你可以使用NSRunLoop的getCFRunLoop方法，返回CFRunLoopRef类型到Core Fundation中。因为两者都指向同一个run loop，你可以任一替换使用。

配置run loop

在次线程启动run loop前，你必须至少添加一类源。因为如果run loop没有任何源需要监视的话，它会在你启动之际立马退出。

此外，你也可以添加run loop观察者来监视run loop的不同执行阶段。首先你可以创建CFRunLoopObserverRef类型并使用CFRunLoopAddObserver将它添加金run loop。注意即使是cocoa程序，run loop观察者也需要由core foundation函数创建。

如果线程运行事件长，最好添加一个输入源到run loop以接收消息。虽然你可以使用定时器，但是定时器一旦启动后当它失效时也会使得run loop退出。虽然定时器可以循环使得run loop运行相对较长的时间，但是也会导致周期性的唤醒线程。与之相反，输入源会等待某事件发生，于是线程只有当事件发生后才会从休眠状态唤醒。

启动run loop

run loop只对程序的次线程有意义，并且必须添加了一类源。如果没有，在启动后就会退出。有几种启动的方法，如：

* 无条件的
* 预设的时间
* 特定的模式

无条件的进入run loop是最简单的选择，但也最不提倡。因为这样会使你的线程处在一个永久的run loop中，这样的话你对run loop本身的控制就会很小。你可以添加或移除源，定时器，但是只能通过杀死进程的办法来退出run loop。并且这样的run loop也没有办法运行在自定义模式下。

用预设时间来运行run loop是一个比较好的选择，这样run loop在某一事件发生或预设的事件过期时启动。如果是事件发生，消息会被传递给相应的处理程序然后run loop退出。你可以重新启动run loop以处理下一个事件。如果是时间过期，你只需重启run loop或使用定时器做任何的其他工作。＊＊

此外，使run loop运行在特定模式也是一个比较好的选择。模式和预设时间不是互斥的，他们可以同时存在。模式对源的限制在run loop模式部分有详细说明。

因为run loop有可能迭代启动，也就是说你可以使用CFRunLoopRun，CFRunLoopRunInMode或者任一NSRunLoop的方法来启动run loop。这样做的时候，你可以使用任何模式启动迭代的run loop，包括被外层run loop使用的模式。

退出run loop

在run loop处理事件前，有两种方法使其退出：

* 设置超时限定
* 通知run loop停止

如果可以配置的话，使用第一种方法是较好的选择。这样，可以使run loop完成所有正常操作，包括发送消息给run loop观察者，最后再退出。

使用CFRunLoopStop来停止run loop也有类似的效果。Run loop也会把所有未发送的消息发送完后再退出。与设置时间的区别在于你可以在任何情况下停止run loop。

尽管移除run loop的输入源和定时器也可以使run loop退出，但这并不是可靠的退出run loop的办法。一些系统程序会添加输入源来处理必须的事件。而你的代码未必会考虑到这些，这样就没有办法从系统程序中移除，从而就无法退出run loop。

线程安全和run loop对象

线程是否安全取决于你使用哪种API操纵run loop。Core Foundation中的函数通常是线程安全的可以被任意线程调用。但是，如果你改变了run loop的配置然后需要进行某些操作，你最好还是在run loop所在线程去处理。如果可能的话，这样是个好习惯。

至于Cocoa的NSRunLoop则不像Core Foundation具有与生俱来的线程安全性。你应该只在run loop所在线程改变run loop。如果添加yuan或定时器到属于另一个线程的run loop，程序会崩溃或发生意想不到的错误。

Run loop 源的配置

下面的例子说明了如果使用cocoa和core foundation来建立不同类型的输入源。

定义自定义输入源

遵循下列步骤来创建自定义的输入源：

* 输入源要处理的信息
* 使感兴趣的客户知道如何和输入源交互的调度程序
* 处理客户发送请求的程序
* 使输入源失效的取消程序

由于你自己创建源来处理消息，实际配置设计得足够灵活。调度，处理和取消程序是你创建你得自定义输入源时总会需要用到得关键程序。但是，输入源其他的大部分行为都是由其他程序来处理。例如，由你决定数据传输到输入源的机制，还有输入源和其他线程的通信机制。

图3－2列举了自定义输入源的配置。在这个例子中，程序的主线程保持了输入源，输入源所需的命令缓冲区和输入源所在的run loop的引用。当主线程有任务，需要分发给目标线程，主线程会给命令缓冲区发送命令和必须的信息，这样活动线程就可以开始执行任务。（因为主线程和输入 源所在线程都须访问命令缓冲区，所以他们的操作要注意同步。）一旦命令传送了，主线程会通知输入源并且唤醒活动线程的run loop。而一收到唤醒命令，run loop会调用输入源的处理部分，由它来执行命令缓冲区中相应的命令。

