NSDefaultRunLoopMode vs NSRunLoopCommonModes

Dear good people of stackoverflow,

Just like the last time, I hereby bring up a question I recently tumble upon. I hope someone out there could shed some light on me.

Whenever I try to download a big file behind scrollview, mkmapview or something, the downloading process gets halted as soon as I touch iPhone screen. Thankfully, an awesome blog post by Jörn suggests an alternative option, using NSRunLoopCommonModes for connection.

That gets me look into detail of the two modes, NSDefaultRunLoopMode and NSRunLoopCommonModes, but the apple document does not kindly explain, other than saying

NSDefaultRunLoopMode

> The mode to deal with input sources other than NSConnection objects. This is the most commonly used run-loop mode.

NSRunLoopCommonModes

> Objects added to a run loop using this value as the mode are monitored by all run loop modes that have been declared as a member of the set of “common" modes; see the description of CFRunLoopAddCommonMode for details.

CFRunLoopAddCommonMode

> Sources, timers, and observers get registered to one or more run loop modes and only run when the run loop is running in one of those modes. Common modes are a set of run loop modes for which you can define a set of sources, timers, and observers that are shared by these modes. Instead of registering a source, for example, to each specific run loop mode, you can register it once to the run loop’s common pseudo-mode and it will be automatically registered in each run loop mode in the common mode set. Likewise, when a mode is added to the set of common modes, any sources, timers, or observers already registered to the common pseudo-mode are added to the newly added common mode.

Can anyone please explain the two in human language?

iphone ios multithreading nsrunloop

edited May 28 '12 at 10:09
Lorenzo Boaro
22.9k 17 92 163

asked Aug 28 '11 at 17:05
Stkim1
460 1 5 10

1 Answer

A run loop is a mechanism that allows the system to wake up sleeping threads so that they may manage asynchronous events. Normally when you run a thread (with the exception of the main thread) there is an option to start the thread in a run loop or not. If the thread runs some sort or long-running operation without interaction with external events and without timers, you do not need a run loop, but if your thread needs to respond to incoming events, it should be attached to a run loop in order to wake up the thread when new events arrive. This is the case of `NSURLConnection` generated threads, as they only wake on incoming events (from the network).

Each thread can be associated to multiple run loops, or can be associated to a specific run loop that can be set to work in different modes. A "run loop mode" is a convention used by the OS to establish some rules for when to deliver certain events or collect them to be delivered later.

Usually all run loops are set to the "default mode" which establishes a default way to manage input events. For example: as soon as a mouse-dragging (Mac OS) or touch (on iOS) event happens then the mode for this run loop is set to event tracking; this means that the thread will not be woken up on new network events but these events will be delivered later when the user

input event terminates and the run loop set to default mode again; obviously this is a choice made by the OS architects to give priority to user events instead of background events.

If you decide to change the run loop mode for your `NSURLConnection` thread, by using `scheduleInRunLoop:forModes:`, then you can assign the thread to a special run loop *mode*, rather than the specific default run loop. The special pseudo-mode called `NSRunLoopCommonModes` is used by many input sources including event tracking. For example assigning `NSURLConnection` 's instance to the common mode means associates its events to "tracking mode" in addition to the "default mode". One advantage/disadvantage of associating threads with `NSRunLoopCommonModes` is that the thread will not be blocked by touch events.

New modes can be added to the common modes, but this is quite a low-level operation.

I would like to close by adding a few notes:

- Typically we need to use a set of images or thumbnails downloaded from the network with a table view. We may think that downloading these images from the network while the table view is scrolling could improve the user experience (since we could see the images while scrolling), but this is not advantageous since the fluidity of scrolling can suffer greatly. In this example with `NSURLConnection` a run loop should not be used; it would be better to use the `UIScrollView` delegate methods to detect when scrolling is terminated and then update the table and download new items from the network;
- You may consider using GCD which will help you to "shield" your code from run loop management issues. In the example above, you may consider adding your network requests to a custom serial queue.

edited Jul 6 '15 at 16:59

Warpling
601 2 7 22

answered Aug 28 '11 at 20:38

viggio24
10.6k 5 32 33

6 Dear Viggio24, thank you so much for this clean, precise explanation. I would ask Apple to include your comment to their API guide. ;) – Stkim1 Aug 29 '11 at 1:09

I upvoted your answer because is absolutely clear!! Thanks for sharing your knowledge. – Lorenzo Boaro May 28 '12 at 9:54

3 **viggio24**'s answer is perfect. For those interested in, I would point out that **Session 208 (Network Apps for iPhone OS, Part 2)** from WWDC 2010 contains an intro on run loops. If you are interested take a look. Hope it helps. – Lorenzo Boaro May 28 '12 at 10:00

4 Just a note for myself: `NSRunLoopCommonModes` allows timer event while scrolling in `UIScrollView` . `NSDefaultRunLoopMode` prevent timer while scrolling. – Eonil Sep 26 '13 at 3:39

1 I found very interesting the the comment about the scroll view update, as it mentions a very challenging topic. Just to add more details on this: When you set a mode for a NSURLConnection, this only affects the execution of the delegate call-backs. I understand that updating the scrollView here could result in a performance issue, but why is this happening? If the answer is that the picture has to be loaded into memory, you could do that by writing into a graphic context on the background, and update your view main thread layer after doing this. does this sound reasonable? – nebillo Jul 3 '14 at 11:18