Networking in a Short-Lived Extension

This thread has been locked by a moderator.

2.9k

There are many cases where you want to do long-running network operations in a short-lived app extension. For example, you might be creating a share extension that needs to upload a large image to an Internet server. Doing this correctly is quite tricky. The rest of this post describes some of these pitfalls and explains how to get around them.

Just by way of context:

- Most of the following was written during the iOS 8.1 timeframe, although AFAIK there's been no specific changes in this space since then.
- The specific focus here is a share extension, although the behaviour is likely to be similar for other short-lived extensions.
- I wasn't using `SLComposeServiceViewController`, although I have no reason to believe that makes a difference.
- I was testing on a device, not the simulator. In my experience the simulator is much less likely to terminate a share extension, which affects how things behave as I'll explain later.

My app and its share extension have an app group in common. I started by verifying that this app group was working as expected (using `UserDefaults`).

The `URLSession` must be in that app group; set this via the `sharedContainerIdentifier` property of the `URLSessionConfiguration` object you use to create the session.

The app and the share extension must use the same `URLSession` background session identifier (the value you pass in to `background(withIdentifier:)` when creating the configuration that you use to create the session).

When an `URLSession` background session is shared like this, it's critical to understand that the session only allows one process to 'connect' to it at a time. If a process is connected to the session and another tries to connect, the second process has its session immediately invalidated with `NSURLErrorBackgroundSessionInUseByAnotherProcess`.

The connected session is the one that receives the session's delegate callbacks.

IMPORTANT If callbacks are generated when no process is connected, the background session resumes (or relaunches) the app rather than the extension.

If a process is connected to a session and is then suspended or terminates, the session disconnects internally. If the process was terminated, the reconnection happens when your code creates its `URLSession` object on next launch. If the process was suspended, the reconnect happens when the app is resumed with the `application(_:handleEventsForBackgroundURLSession:completionHandler:)` delegate callback (and remember that this is always the app, not the extension, per the previous paragraph).

The only way to programmatically disconnect from a session is to invalidate it.

The expected behaviour here is that the extension will start an `URLSession` task and then immediately quit (by calling `completeRequest(returningItems:completionHandler:)`). The system will then resume (or relaunch) the container app to handle any delegate callbacks.

When the system resumes or relaunches the container app to handle background session events, it calls `application(_:handleEventsForBackgroundURLSession:completionHandler:)`. The container app is expected to:

1. Save away the completion handler.
2. Reconnect to the session, if necessary. This involves creating the `URLSession` object if it doesn't currently exist.
3. Handle delegate events from that session.
4. Invalidate the session when those events are all done. The app knows this because the session calls the `urlSessionDidFinishEvents(forBackgroundURLSession:)` delegate callback.
5. Call the completion handler that was saved in step 1.

This leaves the app disconnected from the session, so future invocations of the extension don't have to worry about the `NSURLErrorBackgroundSessionInUseByAnotherProcess` problem I mentioned earlier.

This design works best if each extension hosted by the app has its own shared session. If the app hosts multiple extensions, and they all used the same shared session, they could end up stomping on each other.

In my tests I've noticed that some annoying behaviour falls out of this design: If you start a task from an extension, it's non-deterministic as to whether the app or extension gets the 'did complete' callback. If the task runs super quickly, the extension typically gets the callback. If the task takes longer, the system has time to terminate the extension and the app is resumed to handle it.

There's really no way around this. The workaround is to put the code that handles request completion in both your app and your extension (possibly sharing the code via a framework).

It would be nice if the extension could disconnect from the session immediately upon starting its request. Alas, that's not currently possible (r. 18748008). The only way to programmatically disconnect from the session is to invalidate it, and that either cancels all the running tasks (`invalidateAndCancel()`) or waits for them to complete (`finishTasksAndInvalidate()`), neither of which is appropriate.

One interesting edge case occurs when the app is in the foreground while the share extension comes up. For example, the app might have a share button, from which the user can invoke the share extension. If the share extension starts an `URLSession` task in that case, it can result in the app's `application(_:handleEventsForBackgroundURLSession:completionHandler:)` callback being called while the app is in the foreground. The app doesn't need to behave differently in this case, but it's a little unusual.

Xcode's debugger prevents the system from suspending the process being debugged. So, if you run your extension from Xcode, or you attach to its process some time after launch, the process will continue to execute in situations where the system would otherwise have suspended it. This makes it tricky to investigate problems with the 'extension was suspended before the network request finished' case. The best way to investigate any issues you encounter is via logging.

Note For more information about debugging problems with background networking, see Testing Background Session Code.

Keep in mind that not all networking done by your extension has to use a background session. You should use a background session for long-running requests, but if you have short-running requests then it's best to run them in a standard session.

For example, imagine you have a share extension that needs to make two requests:

- The first request simply gets an upload authorisation token from the server. This request is expected to finish very quickly.
- The second request actually uploads the file (including the upload authorisation token from the previous request), and is expected to take a long time.

It makes sense to only use a background session for the second request. The first request, being short-running, can be done in a standard session, and that will definitely simplify your code.

When doing this you have to prevent your extension from suspending while the short-lived request is in flight. You can use `ProcessInfo.performExpiringActivity(withReason:using:)` for this.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Revision History

- **2023-08-30** Fixed the formatting. Adopted Swift terminology throughout. Made other minor editorial changes.
- **2017-04-26** Moved to the new DevForums. Made many editorial changes. Added the section on Xcode's debugger. Added a section on short-running network requests.
- **2014-12-05** First posted on the old DevForums.

CFNetwork Foundation Extensions

Reply

Posted 6 years ago by eskimo

Add a Comment