Moving to Fewer, Larger Transfers

This thread has been locked by a moderator.

4.3k

Note Much of this content has been rolled into URL Loading System documentation, but I'm leaving this doc here for my own reference.

`NSURLSession` background sessions are optimised for transferring a small number of large resources. Moreover, for download transfers it's best if the transfer is resumable. This design makes the best use of client device resources and the available network bandwidth. If your app runs a lot of tasks in a background session, you should rethink its design. Below you'll find a number of options you might consider.

Most of these options require server-side support. If your server does not have this support, and you can't add it — perhaps you're writing a client app for some fourth-party server — you won't be able to implement these options directly. In that case consider creating your own server that sits between your app and the final server and implements the necessary smarts required to optimise your app's network usage.

If that's not possible, a final option is to not use a background session but instead take advantage of the BackgroundTasks framework. See *BackgroundTasks Framework*, below.

Basics

The basic strategy here is to have the sender (the server for a download, your app for an upload) pack the data into some sort of archive, transfer that archive over the network, and then have the receiver unpack it. There are, however, a number of complications, as described in the subsequent sections.

Archive Format

The obvious choices for the archive format are zip and tar. macOS has lots of options for handling these formats but none of that support is present on iOS (r. 22151959). OTOH, it's easy to find fourth-party libraries to fill in this gap.

Incremental Transfers

It's common to have a corpus of data at one end of the connection that you need to replicate at the other. If the data is large, you don't want to transfer the whole thing every time there's an update. Consider using the following strategies to deal with this:

- Catalogue diff — In this approach the receiver first downloads a catalogue from the sender, then diffs its current state against that catalogue, then requests all the things that are missing. Alternatively, the receiver passes a catalogue of what it has to the sender, at which point the sender does the diff and returns the things that are missing. The critical part is that, once the diff has been done, all of the missing resources are transferred in a single archive.

  The biggest drawback here is resume. If the sender is working with lots of different receivers, each of which has their own unique needs, the sender must keep a lot of unique archives around so it can resume a failed transfer. This can be a serious headache.
- Versions — In this approach you manage changes to the data as separate versions. The receiver passes the version number it has to the sender, at which point the sender knows exactly what data the receiver needs.

  This approach requires a bit more structure but it does avoid the above-mentioned problem with resume. The sender only needs to maintain a limited number of version diffs. In fact, you can balance the number of diffs against your desire to reduce network usage: Maintaining a lot of diffs means that you only have to transfer exactly what the receiver needs, while maintaining fewer diffs makes for a simpler server at the cost of a less efficient use of the network.

Download versus Upload

The discussion so far has applied equally to both downloads and uploads. There is, however, one key difference: `NSURLSession` does not support resumable uploads. When doing an upload you have to balance the number of tasks you submit to the session against the negative effects of a transfer failing. For example, if you do a single large upload then it's annoying if the transfer fails when it's 99% complete. On the other hand, if you do lots of tiny uploads, you're working against the `NSURLSession` background session design.

It *is* possible to support resumable uploads with sufficient server-side support. For example, you could implement an algorithm like this:

1. Run an initial request to allocate an upload ID.
2. Start the upload with that upload ID.
3. If it completes successfully, you're done.
4. If it fails, make a request with the upload ID to find out how much the server received.
5. Start a new upload for the remaining data.

(r. 22323347)

BackgroundTasks Framework

If you're unable to use an `NSURLSession` background session effectively, you do have an alternative, namely, combining a standard session with the BackgroundTasks framework, and specifically the `BGProcessingTaskRequest`. This allows you to request extended processing time from the system. Once you've been granted that time, use it to run your many small network requests in a standard session.

The main drawback to this approach is latency: The system may not grant your request for many hours. Indeed, it's common for these requests to run overnight, once the user has connected their device to a power source.

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

Change history:

- 18 Aug 2015 — First written.
- 24 Mar 2018 — Added the *BackgroundTasks Framework* section. Other editorial changes.
- 31 Jan 2022 — Fixed the formatting and tags. Added a link to the official docs.

CFNetwork Background Tasks

Reply

Posted 7 years ago by eskimo

Add a Comment