[SOLVED] AVAssetResourceLoaderDelegate - Only requests first two bytes?
(https://tutel.me/c/programming/questions/24746453/avassetresourceloaderdelegate++only+requests+first+two+bytes)

By Alexander

2014-07-14 21:40:36 8 Comments

I'm implementing an `AVAssetResourceLoaderDelegate`, and I'm having a bit of trouble getting to to behave correctly. My goal is to intercept any requests made by the AVPlayer, make the request myself, write the data out to a file, then respond to the `AVPlayer` with the file data.

The issue I'm seeing: I can intercept the first request, which is only asking for two bytes, and respond to it. After that, I'm not getting any more requests hitting my `AVAssetResourceLoaderDelegate`.

When I intercept the very first `AVAssetResourceLoadingRequest` from the `AVPlayer` it looks like this:

```
<AVAssetResourceLoadingRequest: 0x17ff9e40,
URL request = <NSMutableURLRequest: 0x17f445a0> { URL: fakeHttp://blah.com/blah/blah.mp3 },
request ID = 1,
content information request = <AVAssetResourceLoadingContentInformationRequest: 0x17ff9f30,
content type = "(null)",
content length = 0,
byte range access supported = NO,
disk caching permitted = NO>,
data request = <AVAssetResourceLoadingDataRequest: 0x17e0d220,
requested offset = 0,
requested length = 2,
current offset = 0>>
```

As you can see, this is only a request for the first two bytes of data. I'm taking the `fakeHttp` protocol in the URL, replacing it with just `http`, and making the request myself.

Then, here's how I'm responding to the request once I have some data:

```
- (BOOL)resourceLoader:(AVAssetResourceLoader *)resourceLoader shouldWaitForLoadingOfRequestedResource:(AVAssetResou
rceLoadingRequest *)loadingRequest {

     //Make the remote URL request here if needed, omitted

    CFStringRef contentType = UTTypeCreatePreferredIdentifierForTag(kUTTagClassMIMEType, (__bridge CFStringRef)([sel
f.response MIMEType]), NULL);
    loadingRequest.contentInformationRequest.byteRangeAccessSupported = YES;
    loadingRequest.contentInformationRequest.contentType = CFBridgingRelease(contentType);
    loadingRequest.contentInformationRequest.contentLength = [self.response expectedContentLength];

    //Where responseData is the appropriate NSData to respond with
    [loadingRequest.dataRequest respondWithData:responseData];

    [loadingRequest finishLoading];
    return YES;
}
```

I've stepped through this and verified that everything in the `contentInformationRequest` is filled in correctly, and that the data I'm sending is NSData with the appropriate length (in this case, two bytes).

No more requests get sent to my delegate, and the player does not play (presumably because it only has two bytes of data, and hasn't requested any more).

Does anyone have experience with this to point me toward an area where I may be doing something wrong? I'm running iOS 7.

Edit: Here's what my completed request looks like, after I call `finishedLoading` :

```
<AVAssetResourceLoadingRequest: 0x16785680,
URL request = <NSMutableURLRequest: 0x166f4e90> { URL: fakeHttp://blah.com/blah/blah.mp3  },
request ID = 1,
content information request = <AVAssetResourceLoadingContentInformationRequest: 0x1788ee20,
content type = "public.mp3",
content length = 7695463,
byte range access supported = YES,
disk caching permitted = NO>,
data request = <AVAssetResourceLoadingDataRequest: 0x1788ee60,
requested offset = 0,
requested length = 2,
current offset = 2>>
```

2 comments

@Alexander 2014-07-16 23:40:24

Circling back to answer my own question in case anyone was curious.

The issue boiled down to threading. Though it's not explicitly documented anywhere, `AVAssetResourceLoaderDelegate` does some weird stuff with threads.

Essentially, my issue was that I was creating the `AVPlayerItem` and `AVAssetResourceLoaderDelegate` on the main thread, but responding to delegate calls on a background thread (since they were the result of network calls). Apparently, `AVAssetResourceLoader` just completely ignores responses coming in on a different thread than it was expecting.

I solved this by just doing everything, including `AVPlayerItem` creation, on the same thread.

@Alexander 2014-07-16 23:42:19

As an aside, the simulator is a bit more forgiving in these situations than an actual device. Some threading setups would work on the simulator then totally fail on a device. But from what I can tell, everything that works on an actual device works on the simulator.

@Chris Vasselli 2015-12-17 21:55:23

I've been banging my head against this all day. Thank you so much! =D

@Chris Vasselli 2015-12-17 22:08:24

One thing I also figured out: you must create the AVPlayer object on this thread as well. I was trying to use `replaceCurrentItemWithPlayerItem:` on an AVPlayer created on a different thread, and it didn't work.

@Serhii 2016-09-26 15:32:30

@Alexander, I have similar problem with strange lags when working with AVAssetResourceLoaderDelegate and AVPlayer. Is it required to create instances and respond to data requests in one thread or I also should call AVPlayer methods (play, pause, seekToTime...) in this thread?

@Alexander 2016-09-26 19:20:58

@Serhii AVPlayer should only be accessed by the thread it was created on, or it will start misbehaving. This includes calls like play/pause/seek/etc.

@Serhii 2016-09-27 16:46:53

@Alexander Thanks, it helped me!

@Michal Pietras 2014-07-14 22:13:00

```
- (BOOL)resourceLoader:(AVAssetResourceLoader *)resourceLoader shouldWaitForLoadingOfRequestedResource:(AVAssetResourc
eLoadingRequest *)loadingRequest
{
    loadingRequest.contentInformationRequest.contentType = @"public.aac-audio";
    loadingRequest.contentInformationRequest.contentLength = [self.fileData length];
    loadingRequest.contentInformationRequest.byteRangeAccessSupported = YES;

    NSData *requestedData = [self.fileData subdataWithRange:NSMakeRange((NSUInteger)loadingRequest.dataRequest.request
edOffset,
                                                                         (NSUInteger)loadingRequest.dataRequest.request
edLength)];
    [loadingRequest.dataRequest respondWithData:requestedData];
    [loadingRequest finishLoading];

    return YES;
}
```

This implementation works for me. It always asks for the first two bytes and then for the whole data. If you don't get another callback it means that there was something wrong with the first response you have made. I guess the problem is that you are using MIME content type instead of UTI.

@Alexander 2014-07-14 22:52:23

Thanks for the reply. This is exactly how I would expect this API to work, so that's encouraging that you can get it to work like that. My guess is that I'm doing something really stupid to mess it all up. I've added an edit to my original post to show a completed request. The contentType is actually in UTI format, so I don't think that's the issue.

@Michal Pietras 2014-07-15 09:26:05

It really looks fine and it should be working. Have you tried testing it on both the simulator and a device? AVFoundation's behavior differs slightly on iOS and OS X.

@Alexander 2014-07-15 16:40:48

I've tried it on a device only. The delegate doesn't get hit at all on the simulator, though I had read somewhere that this API only works on devices anyway.

@Michal Pietras 2014-07-15 22:40:47

It works on simulator as well, however, there are some differences. Your delegate implementation looks correct as long as you are responding with first two bytes of a valid mp3 file. Taking into account that the delegate is not getting called on the simulator at all I would suspect you have something messed up with the way you set up the delegate. It would be useful if you could share some more code.

@Alexander 2014-07-16 00:28:25

My mistake, it is indeed called in the simulator. I'll be making a sample app to demonstrate my issue. If I have any findings from it, I'll update here.

@Alexander 2014-07-16 20:38:04

I have it working great on a simulator now. But, on a device the delegate seems to act differently. After playing around a bit, I think the simulator and the device treat threads differently.

@Bruno Koga 2014-09-05 17:30:56

"I guess the problem is that you are using MIME content type instead of UTI." <- Thanks for that, @MichalPietras