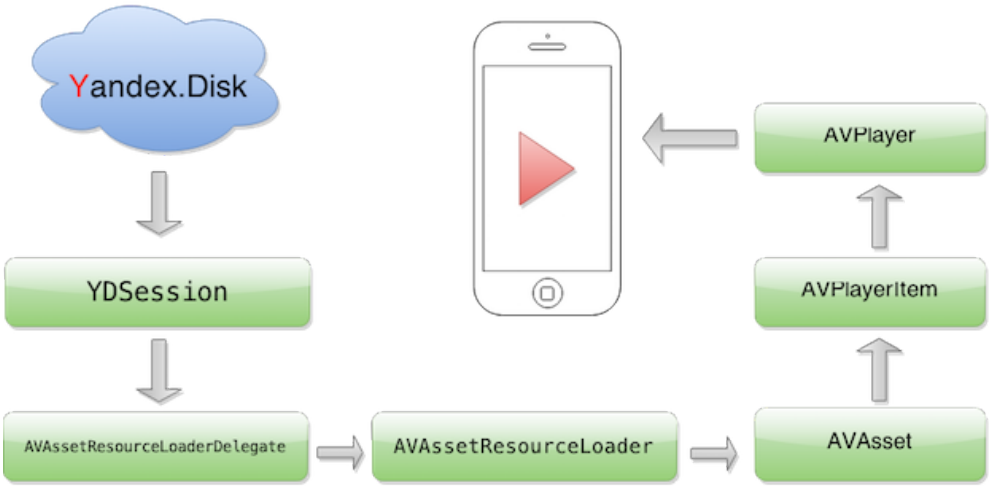
Audio streaming and caching in iOS using AVAssetResourceLoader and AVPlayer

leshkoapps, 11 Feb 2015

How you can use AVAssetResourceLoader and AVPlayer in your apps



Introduction

This article is about audio streaming and caching in iOS. If you want to know how we implemented audio streaming in our apps and how you can use AVAssetResourceLoader and AVPlayer in your apps this tutorial is for you.

We use AVPlayer for playing audio files from local storage and remote host. If you want to stream an audio file from some cloud service you should get a direct url of that file and initialize AVPlayer with received url. We use this approach for the most cloud services like DropBox, Box, Google Drive but it does not work when you play audio files from Yandex.Disk and Web Dav. It's all because of authorization header that you should set when you send GET request to the server. There is no public method in AVPlayer interface that allows you to do this.

So we started looking for a solution and came across the resourceLoader object in AVURLAsset. This is actually a great API and you can use it to provide controlled access to the remote file for AVPlayer. This works like a local HTTP proxy but without all the hassles.

The most important thing to remember is that AVPlayer uses resourceLoader when it doesn't know how to load a resource. The trick here is to change the protocol so AVPlayer is forced to defer the loading of the resource to our application. So we should change scheme of our resource and initialize AVPlayer with new url.

When you work with AVAssetResourceLoader you should implement two methods of AVAssetResourceLoaderDelegate:

Hide Copy Code

```
- (BOOL)resourceLoader:(AVAssetResourceLoader *)resourceLoader
shouldWaitForLoadingOfRequestedResource:(AVAssetResourceLoadingRequest *)loadingRequest;

- (void)resourceLoader:(AVAssetResourceLoader *)resourceLoader
didCancelLoadingRequest:(AVAssetResourceLoadingRequest *)loadingRequest;
```

Delegates receive resourceLoader: shouldWaitForLoadingOfRequestedResource: message when assistance is required of the application to load a resource. In this case we save AVAssetResourceLoadingRequest and start data loading operation. When data from the resource is no longer required or when a loading request is superseded by new requests for data from the same resource, delegates receive resourceLoader: didCancelLoadingRequest: and we cancel data loading operation.

Resource Loader

So let's create our AVPlayer with custom scheme:

Hide Copy Code

```
NSURL *url = [NSURL URLWithString:@"customscheme://host/audio.mp3"];
AVURLAsset *asset = [AVURLAsset URLAssetWithURL:url options:nil];
[asset.resourceLoader setDelegate:self queue:dispatch_get_main_queue()];

AVPlayerItem *item = [AVPlayerItem playerItemWithAsset:asset];
[self addObserversForPlayerItem:item];

self.player = [AVPlayer playerWithPlayerItem:playerItem];
[self addObserversForPlayer];
```

The first two lines create AVURLAsset with custom url and set AVAssetResourceLoaderDelegate with dispatch queue on which delegate methods will be invoked.

The next two lines create AVPlayerItem from AVURLAsset and AVPlayer from AVPlayerItem and add required observers.

The next thing we should do is create custom class that will load data of requested resource from the server and pass loaded data back to AVURLAsset. Let's name it LSFilePlayerResourceLoader and add two parameters in init method. The first parameter is requested file url and the second is YDSession object. This session object is responsible for getting file data from Yandex.Disk cloud server.

Our LSFilePlayerResourceLoader objects will be stored in dictionary and resource url will be the key.

Here's what our AVAssetResourceLoaderDelegate implementation will looks like:

Hide Shrink ▲ Copy Code

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

    NSURL *resourceURL = [loadingRequest.request URL];

    if([resourceURL.scheme isEqualToString:@"customscheme"]){
        LSFilePlayerResourceLoader *loader = [self resourceLoaderForRequest:loadingRequest];

        if(loader==nil){
            loader = [[LSFilePlayerResourceLoader alloc] initWithResourceURL:resourceURL session:self.session];
            loader.delegate = self;
            [self.resourceLoaders setObject:loader forKey:[self keyForResourceLoaderWithURL:resourceURL]];
        }

        [loader addRequest:loadingRequest];

        return YES;
    }
    return NO;
}

- (void)resourceLoader:(AVAssetResourceLoader *)resourceLoader
didCancelLoadingRequest:(AVAssetResourceLoadingRequest *)loadingRequest{

    LSFilePlayerResourceLoader *loader = [self resourceLoaderForRequest:loadingRequest];

    [loader removeRequest:loadingRequest];

}
```

First we should check resourceURL scheme and then get cached LSFilePlayerResourceLoader or create the new one. After this we add loadingRequest to our resource loader.

LSFilePlayerResourceLoader interface is below:

Hide Copy Code

```
@interface LSFilePlayerResourceLoader : NSObject

@property (nonatomic,readonly,strong)NSURL *resourceURL;
@property (nonatomic,readonly)NSArray *requests;
@property (nonatomic,readonly,strong)YDSession *session;
@property (nonatomic,readonly,assign)BOOL isCancelled;

@property (nonatomic,weak)id <LSFilePlayerResourceLoaderDelegate> delegate;

- (instancetype)initWithResourceURL:(NSURL *)url session:(YDSession *)session;
- (void)addRequest:(AVAssetResourceLoadingRequest *)loadingRequest;
- (void)removeRequest:(AVAssetResourceLoadingRequest *)loadingRequest;
- (void)cancel;

@protocol LSFilePlayerResourceLoaderDelegate <NSObject>

@optional

- (void)filePlayerResourceLoader:(LSFilePlayerResourceLoader *)resourceLoader
didFailWithError:(NSError *)error;

- (void)filePlayerResourceLoader:(LSFilePlayerResourceLoader *)resourceLoader
didLoadResource:(NSURL *)resourceURL;

@end
```

This interface has methods for managing requests in loader queue and LSFilePlayerResourceLoaderDelegate protocol which defines methods that allow your code to handle resource loading status.

When we add loadingRequest to the queue we save it in pendingRequests array and start data loading operation:

Hide Copy Code

```
- (void)addRequest:(AVAssetResourceLoadingRequest *)loadingRequest{

    if(self.isCancelled==NO){

        NSURL *interceptedURL = [loadingRequest.request URL];
        [self startOperationFromOffset:loadingRequest.dataRequest.requestedOffset
length:loadingRequest.dataRequest.requestedLength];

        [self.pendingRequests addObject:loadingRequest];
    }
    else{
        if(loadingRequest.isFinished==NO){
            [loadingRequest finishLoadingWithError:[self loaderCancelledError]];
        }
    }

}
```

First time we created a new data load operation for every upcoming request. That's why the file was loaded from multiple threads. But then we found out that when AVAssetResourceLoader began new loading request previously issued requests can be cancelled. So in our start operation method we cancel all previously started operations .

There are two data loading operations. A contentInfoOperation is operation to identify the content length, content type, and whether the resource supports byte range requests. With byte range requests, AVPlayer can get fancy and apply various optimizations. The second one is dataOperation which loads file data with offset. We get requested data offset from AVAssetResourceLoadingDataRequest.

Hide Shrink ▲ Copy Code

```
- (void)startOperationFromOffset:(unsigned long long)requestedOffset
                          length:(unsigned long long)requestedLength{

    [self cancelAllPendingRequests];
    [self cancelOperations];

    __weak typeof (self) weakSelf = self;

    void(^failureBlock)(NSError *error) = ^(NSError *error) {
        [weakSelf performBlockOnMainThreadSync:^{
            if(weakSelf && weakSelf.isCancelled==NO){
                [weakSelf completeWithError:error];
            }
        }];
    };

    void(^loadDataBlock)(unsigned long long off, unsigned long long len) = ^(unsigned long long offset,unsigned long long
length){

        [weakSelf performBlockOnMainThreadSync:^{

            NSString *bytesString = [NSString stringWithFormat:@"bytes=%lld-%lld",offset,(offset+length-1)];
            NSDictionary *params = @{@"Range":bytesString};

            id<ydsessionrequest> req =
            [weakSelf.session partialContentForFileAtPath:weakSelf.path withParams:params response:nil
                data:^(UInt64 recDataLength, UInt64 totDataLength, NSData *recData) {

                    [weakSelf performBlockOnMainThreadSync:^{

                        if(weakSelf && weakSelf.isCancelled==NO){

                            LSDataResonse *dataResponse =
                            [LSDataResonse responseWithRequestedOffset:offset
                                          requestedLength:length
                                          receivedDataLength:recDataLength
                                          data:recData];
                            [weakSelf didReceiveDataResponse:dataResponse];

                        }

                    }];
                }
                completion:^(NSError *err) {

                   if(err){
                       failureBlock(err);
                   }

                }];

           weakSelf.dataOperation = req;

        }];
    };

    if(self.contentInformation==nil){

        self.contentInfoOperation = [self.session fetchStatusForPath:self.path completion:^(NSError *err, YDItemStat
*item) {

            if(weakSelf && weakSelf.isCancelled==NO){

                if(err==nil){

                    NSString *mimeType = item.path.mimeTypeForPathExtension;
```

```
                    CFStringRef contentType = UTTypeCreatePreferredIdentifierForTag(kUTTagClassMIMEType,(__bridge
CFStringRef)(mimeType),NULL);
                    unsigned long long contentLength = item.size;

                    weakSelf.contentInformation = [[LSContentInformation alloc] init];
                    weakSelf.contentInformation.byteRangeAccessSupported = YES;
                    weakSelf.contentInformation.contentType = CFBridgingRelease(contentType);
                    weakSelf.contentInformation.contentLength = contentLength;

                    [weakSelf prepareDataCache];

                    loadDataBlock(requestedOffset,requestedLength);

                    weakSelf.contentInfoOperation = nil;
                }
                else{
                    failureBlock(err);
                }
            }
        }];
    }
    else{
        loadDataBlock(requestedOffset,requestedLength);
    }
}
</ydsessionrequest>
```

When the contentInformation request is received and no cached file exists, we init data cache and start fetching the audio file.

We use temporary file for caching received data from the web and read it when we need.

Hide Shrink ▲ Copy Code

```
- (void)prepareDataCache{

    self.cachedFilePath = [[self class] pathForTemporaryFile];

    NSError *error = nil;
    if ([[NSFileManager defaultManager] fileExistsAtPath:self.cachedFilePath] == YES){
        [[NSFileManager defaultManager] removeItemAtPath:self.cachedFilePath error:&error];
    }

    if (error == nil && [[NSFileManager defaultManager] fileExistsAtPath:self.cachedFilePath] == NO) {

        NSString *dirPath = [self.cachedFilePath stringByDeletingLastPathComponent];
        [[NSFileManager defaultManager] createDirectoryAtPath:dirPath
                                   withIntermediateDirectories:YES
                                                    attributes:nil
                                                         error:&error];

        if (error == nil) {
            [[NSFileManager defaultManager] createFileAtPath:self.cachedFilePath
                                                    contents:nil
                                                  attributes:nil];

            self.writingFileHandle = [NSFileHandle fileHandleForWritingAtPath:self.cachedFilePath];

            @try {
                [self.writingFileHandle truncateFileAtOffset:self.contentInformation.contentLength];
                [self.writingFileHandle synchronizeFile];
            }
            @catch (NSException *exception) {
                NSError *error = [[NSError alloc] initWithDomain: LSFilePlayerResourceLoaderErrorDomain
                                                            code: -1
                                                        userInfo: @{ NSLocalizedDescriptionKey : @"can not write to file"
}];
                [self completeWithError:error];
                return;
            }
            self.readingFileHandle = [NSFileHandle fileHandleForReadingAtPath:self.cachedFilePath];
        }
    }

    if (error != nil) {
        [self completeWithError:error];
    }
}
```

When new data is received we cache it on the disk, update receivedDataLength and then notify all pending requests.

Hide Copy Code

```
- (void)didReceiveDataResponse:(LSDataResonse *)dataResponse{

    [self cacheDataResponse:dataResponse];

    self.receivedDataLength=dataResponse.currentOffset;

    [self processPendingRequests];
}
```

Cache data response method is responsible for caching received data with requested offset.

Hide Copy Code

```
- (void)cacheDataResponse:(LSDataResonse *)dataResponse{

    unsigned long long offset = dataResponse.dataOffset;

    @try {
        [self.writingFileHandle seekToFileOffset:offset];
        [self.writingFileHandle writeData:dataResponse.data];
        [self.writingFileHandle synchronizeFile];
    }

    @catch (NSException *exception) {
        NSError *error = [[NSError alloc] initWithDomain: LSFilePlayerResourceLoaderErrorDomain
                                                    code: -1
                                                userInfo: @{ NSLocalizedDescriptionKey : @"can not write to file" }];
        [self completeWithError:error];
    }
}
```

Read data method is responsible for reading cached data from disk.

Hide Copy Code

```
- (NSData *)readCachedData:(unsigned long long)startOffset
length:(unsigned long long)numberOfBytesToRespondWith{

    @try {
        [self.readingFileHandle seekToFileOffset:startOffset];
        NSData *data = [self.readingFileHandle readDataOfLength:numberOfBytesToRespondWith];
        return data;
    }

    @catch (NSException *exception) {}

    return nil;

}
```

In processPendingRequests method we fill content information and write cached data. When all requested data is received we remove pending request from queue.

Hide Copy Code

```
- (void)processPendingRequests{

    NSMutableArray *requestsCompleted = [[NSMutableArray alloc] init];

    for (AVAssetResourceLoadingRequest *loadingRequest in self.pendingRequests){

        [self fillInContentInformation:loadingRequest.contentInformationRequest];

        BOOL didRespondCompletely = [self respondWithDataForRequest:loadingRequest.dataRequest];

        if (didRespondCompletely){

            [loadingRequest finishLoading];
            [requestsCompleted addObject:loadingRequest];

        }

    }
    [self.pendingRequests removeObjectsInArray:requestsCompleted];
}
```

Write information about content such as content length, content type, and whether the resource supports byte range requests.

Hide Copy Code

```
- (void)fillInContentInformation:(AVAssetResourceLoadingContentInformationRequest *)
contentInformationRequest{

    if (contentInformationRequest == nil || self.contentInformation == nil){
        return;
    }

    contentInformationRequest.byteRangeAccessSupported = self.contentInformation.byteRangeAccessSupported;
    contentInformationRequest.contentType = self.contentInformation.contentType;
    contentInformationRequest.contentLength = self.contentInformation.contentLength;

}
```

Read data from cache and pass it to pending requests.

Hide Shrink ▲ Copy Code

```
- (BOOL)respondWithDataForRequest:(AVAssetResourceLoadingDataRequest *)dataRequest{

    long long startOffset = dataRequest.requestedOffset;
    if (dataRequest.currentOffset != 0){
        startOffset = dataRequest.currentOffset;
    }

    // Don't have any data at all for this request
    if (self.receivedDataLength < startOffset){
        return NO;
    }
```

```
    // This is the total data we have from startOffset to whatever has been downloaded so far
    NSUInteger unreadBytes = self.receivedDataLength - startOffset;

    // Respond with whatever is available if we can't satisfy the request fully yet
    NSUInteger numberOfBytesToRespondWith = MIN(dataRequest.requestedLength, unreadBytes);

    BOOL didRespondFully = NO;

    NSData *data = [self readCachedData:startOffset length:numberOfBytesToRespondWith];

    if(data){
        [dataRequest respondWithData:data];
        long long endOffset = startOffset + dataRequest.requestedLength;
        didRespondFully = self.receivedDataLength >= endOffset;
    }

    return didRespondFully;
}
```

Cloud SDK

That's all with our loader. It's time to patch Yandex.Disk SDK and add method for getting partial content from server. In you case it might be different cloud service with it's own SDK but you should apply the same changes as described below. There are only 3 of them.

The first we should check if all requests in given SDK has cancel method. Unfortunately client code can not cancel any request in Yandex.Disk SDK so we should add this possibility. Just declare new protocol YDSessionRequest in YDSession.h and return it in all requests.

Hide Copy Code

```
@protocol YDSessionRequest <NSObject>

- (void)cancel;

@end


- (id<YDSessionRequest>)fetchDirectoryContentsAtPath:(NSString *)path
 completion:(YDFetchDirectoryHandler)block;

- (id<YDSessionRequest>)fetchStatusForPath:(NSString *)path
completion:(YDFetchStatusHandler)block;
```

The next we should implement method for getting partial data from requested file with given offset and length.

Hide Shrink Copy Code

```
- (id<YDSessionRequest>)partialContentForFileAtPath:(NSString *)srcRemotePath
                                         withParams:(NSDictionary *)params
                                           response:(YDDidReceiveResponseHandler)response
                                               data:(YDPartialDataHandler)data
                                         completion:(YDHandler)completion{

    return [self downloadFileFromPath:srcRemotePath
                               toFile:nil
                           withParams:params
                             response:response
                                 data:data
                             progress:nil
                           completion:completion];

}



- (id<YDSessionRequest>)downloadFileFromPath:(NSString *)path
                                      toFile:(NSString *)aFilePath
                                  withParams:(NSDictionary *)params
                                    response:(YDDidReceiveResponseHandler)responseBlock
                                        data:(YDPartialDataHandler)dataBlock
                                    progress:(YDProgressHandler)progressBlock
                                  completion:(YDHandler)completionBlock{

    NSURL *url = [YDSession urlForDiskPath:path];
    if (!url) {
        completionBlock([NSError errorWithDomain: kYDSessionBadArgumentErrorDomain
                                            code: 0
                                        userInfo: @{ @"getPath": path }]);
        return nil;
    }

    BOOL skipReceivedData = NO;

    if(aFilePath==nil){
        aFilePath = [[self class] pathForTemporaryFile];
        skipReceivedData = YES;
    }
```

```
    NSURL *filePath = [YDSession urlForLocalPath:aFilePath];
    if (!filePath) {
        completionBlock([NSError errorWithDomain: kYDSessionBadArgumentErrorDomain
                                            code: 1
                                        userInfo: @{ @"toFile": aFilePath }]);
        return nil;
    }

    YDDiskRequest *request = [[YDDiskRequest alloc] initWithURL:url];
    request.fileURL = filePath;
    request.params = params;
    request.skipReceivedData = skipReceivedData;
    [self prepareRequest:request];

    NSURL *requestURL = [request.URL copy];

    request.callbackQueue = _callBackQueue;

    request.didReceiveResponseBlock = ^(NSURLResponse *response, BOOL *accept) {
        if(responseBlock){
            responseBlock(response);
        }
    };

    request.didGetPartialDataBlock = ^(UInt64 receivedDataLength, UInt64 expectedDataLength, NSData *data){
        if(progressBlock){
            progressBlock(receivedDataLength,expectedDataLength);
        }
        if(dataBlock){
            dataBlock(receivedDataLength,expectedDataLength,data);
        }
    };

    request.didFinishLoadingBlock = ^(NSData *receivedData) {

        if(skipReceivedData){
            [[self class] removeTemporaryFileAtPath:aFilePath];
        }

        NSDictionary *userInfo = @{@"URL": requestURL,
                                   @"receivedDataLength": @(receivedData.length)};
        [[NSNotificationCenter defaultCenter] postNotificationInMainQueueWithName: kYDSessionDidDownloadFileNotification
                                                                            object: self
                                                                          userInfo: userInfo];
        completionBlock(nil);
    };

    request.didFailBlock = ^(NSError *error) {

        if(skipReceivedData){
            [[self class] removeTemporaryFileAtPath:aFilePath];
        }

        NSDictionary *userInfo = @{@"URL": requestURL};
        [[NSNotificationCenter defaultCenter] postNotificationInMainQueueWithName:
kYDSessionDidFailToDownloadFileNotification
                                                                            object: self
                                                                          userInfo: userInfo];

        completionBlock([NSError errorWithDomain:error.domain code:error.code userInfo:userInfo]);
    };

    [request start];

    NSDictionary *userInfo = @{@"URL": request.URL};
    [[NSNotificationCenter defaultCenter] postNotificationInMainQueueWithName: kYDSessionDidStartDownloadFileNotification
                                                                        object: self
                                                                      userInfo: userInfo];
    return (id<ydsessionrequest>)request;
}

</ydsessionrequest>
```

And the last but not least we should check if our Yandex.Disk SDK has serial callback queue. It's very important to use serial queue because playback issues may occur if parallel queue is used. In our case Yandex.Disk SDK has parallel callback queue by default so let's fix this.

Hide Copy Code

```
- (instancetype)initWithDelegate:(id<YDSessionDelegate>)delegate
callBackQueue:(dispatch_queue_t)queue{

    self = [super init];

    if (self) {
        _delegate = delegate;
        _callBackQueue = queue;
    }

    return self;
}

 YDDiskRequest *request = [[YDDiskRequest alloc] initWithURL:url];
 request.fileURL = filePath;
```

```
request.params = params;
[self prepareRequest:request];
request.callbackQueue = _callBackQueue;
```

Source Code

The source code given in this tutorial is available on **GitHub**.

License

Share

About the Author

leshkoapps
Software Developer (Senior) Leshko Apps
United States
No Biography provided

You may also be interested in...

SAPrefs - Netscape-like Preferences Dialog
OLE DB - First steps
Window Tabs (WndTabs) Add-In for DevStudio
To Heap or not to Heap; That's the Large Object Question?
Introduction to D3DImage
Creating alternate GUI using Vector Art

Comments and Discussions