码农人生

ChengYin's coding life

Where there is a will, there is a way. -- Thomas Edison

May 24th, 2016
Audio, iOS, iOS Audio

iOS音频播放 (九)：边播边缓存

Audio Playback in iOS (Part 9) : Cache While Streaming

好久没写过博客了，在这期间有很多同学通过博客评论、微博私信、邮件和我交流iOS音频方面的相关问题，其中被问到最多的是如何实现“边播边缓存”，这篇就来说一说这个话题。顺便一提，本文的题目虽然为“iOS音频播放”，但其中所涉及的部分技术方案在OSX平台或者在流播放视频下同样适用。

我能想到的方案

这类的技术方案其实有不少（其实在第一篇的末尾也略微有所涉及）：

思路1. 最直接的方式，自行实现音频数据的请求在请求的过程中把数据缓存到磁盘，然后基于磁盘的数据自己实现解码、播放等功能；这个方法作为直接也最为复杂，开发者需要对音频播放的原理、操作系统等知识有一定程度的理解。如果能够实现这种方式所达到的效果也将会是最好的，整个过程都由开发者掌控，出现问题也可以对症下药。开源播放器FreeStreamer就是一个很好的例子，使用带有cache功能开源播放器或在其基础上进行二次开发也是不错的选择；

思路2. 请求拦截的方式，首先你需要一个能够进行流播放的播放器（如Apple提供的AVPlayer），通过拦截播放器发送的请求可以知道需要下载哪一段数据，于是就可以根据本地缓存文件的情况分段为播放器提供数据，如遇到已缓存的数据段直接从缓存中获取数据塞回给播放器，如遇到未缓存的数据段就发送请求获取数据，得到response和数据后保存到磁盘同时塞回给播放器。这种思路下有三个分支：

思路2.1 流播放器 + LocalServer，首先在搭建一个LocalServer（例如使用GCDWebServer），然后将URL组织成类似这种形式：

> http://localhost:port?url=urlEncode(audioUrl)

把组织好的URL交给播放器播放，播放器把请求发送到LocalServer上，LocalServer解析到实际的音频地址后发送请求或者读取已缓存的数据。

思路2.2 流播放器 + NSURLProtocol，大家都知道NSURLProtocol可以拦截Cocoa平台下URL Loading System中的请求，如果播放器的请求是运行在`URL Loading System`下的话使用这个方法可以轻松的拦截到播放器所发送的请求然后自己再进行请求或者读取缓存数据。这里需要注意如果使用AVPlayer作为播放器的话这种方法只在模拟器上才work，真机上并不能拦截到任何请求。这也证明AVPlayer在真机上并没有运行在`URL Loading System`下，但模拟器上却是（不知道在OSX下是否能work，有兴趣的同学可以尝试一下）。

注：如果播放器使用的是CFNetwork，也可以尝试拦截，例如使用FB的fishhook，这hook方法应该会遇上不少坑，请做好心理准备。。

思路2.3 AVPlayer + AVAssetResourceLoader，AVAssetResourceLoader是iOS 6之后添加的类其主要作用是让开发者能够掌控`AVURLAsset`加载数据的整个过程。这正好符合我们的需求，`AVAssetResourceLoader`会通过delegate把`AVAssetResourceLoadingRequest`对象传递给开发者，开发者可以根据其中的一些属性得知需要加载的数据段。在得到数据后也可以通过`AVAssetResourceLoadingRequest`向AVPlayer传递response和数据。

思路3. 取巧的方式，自行实现音频数据的请求在请求的过程中把数据缓存到磁盘，然后使用系统提供的播放器（如AVAudioPlayer、AVPlayer进行播放）。这种实现方式中需要注意的是要播放的音频文件需要预先缓存一定量之后才能够播放，具体缓存多少完全频个人感觉，并且有可能会产生播放失败或者播放错误。这种方式的另一个缺点是无法进行任意的seek；

方案的选择

上面提到了3种思路共5个方案，那么在实际开发过程中开发者应该可以根据各个方案的优劣结合自己的实际情况选择最适合自己的方案。

思路1：优点在于整个播放过程可控，出现问题可调试，但开发复杂度较高，故选择有对应功能的开源播放器是一个比较好途径。在使用开源播放器之前最好能阅读其代码，掌握整个播放流程，出了问题才能迅速定位。推荐以播放为核心功能的app使用此方案；

思路2：优点在于开发者不必关心播放的整个过程，对音频播放的相关知识也不必有太多的了解，整个开发过程只要关心请求的解析、缓存数据的读取和保存以及数据的回填即可；至于缺点，首先你的有一个靠谱的流播放器，如果使用AVPlayer那么请做踩坑准备；

思路2.1：各类流播放器通吃，如果方案2.2和2.3不管用2.1是最好的选择；

思路2.2：需要播放器有指定的请求方式，如运行在`URL Loading System`下；

思路2.3：如果你用的就是AVPlayer那么可以尝试使用这个思路，但对于播放列表形式(M3U8)的音频这种方式是无效的；

思路3：如果你选择这条路，那说明你真的懒得不行。。。

思路2缓存和数据读取细节

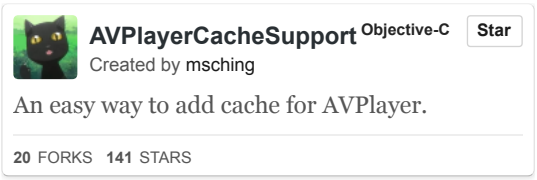
一般音频流或者视频流都会支持HTTP协议中的Range request header，所以大多数的流播放器都会对`Range header`进行支持，在数据源支持Range的情况下拦截到请求时有必要注意播放器所请求的数据段并根据当前数据缓存的状态进行分段处理。

举个例子，播放器请求bytes=0-100，其中10-20、50-60已经被缓存，那么这个请求就应该被分为下面几段来处理：

1. 0-10，网络请求
2. 10-20，本地缓存
3. 20-50，网络请求
4. 50-60，本地缓存
5. 60-100，网络请求

以上几段数据请求按顺序执行并进行数据回填，其中通过网络请求的数据在收到之后加入缓存以便下一次请求再次使用。另外要注意的是由于播放器本身只发送了一个请求所以response还是只有一个并且Content-Range还是应该为`0-100/FileLength`。

AVAssetResourceLoader踩坑



AVPlayerCacheSupport是我使用`AVAssetResourceLoader`进行实践后实现的一个开源项目，在开发的过程中踩到的坑也在这里分享给大家。

shceme必须自定义

非自定义的URL Scheme不会触发AVAssetResourceLoader的delegate方法。这一点并不难发现，Stackoverflow上和github上都有提到这一点。所以在构造AVPlayItem时必须使用自定义Scheme的URL才行，这里我是在原有的Scheme后加上了`-streaming`，在收到AVAssetResourceLoader的回调之后实际发送请求时再把`-streaming`后缀去掉。

AVURLAsset.resourceLoader的delegate必须在AVPlayerItem生成前赋值

看代码感受一下吧，这样写能接到回调：

```
1 AVURLAsset *asset = [AVURLAsset URLAssetWithURL:url] options:options];
2 [asset.resourceLoader setDelegate:self queue:dispatch_get_main_queue()];
3 AVPlayerItem *item = [self playerItemWithAsset:asset];
```

下面这种写法是无法接到回调的：

```
1 AVURLAsset *asset = [AVURLAsset URLAssetWithURL:url] options:options];
2 AVPlayerItem *item = [self playerItemWithAsset:asset];
3 [[(AVURLAsset *)item.asset resourceLoader] setDelegate:self queue:dispatch_get_main_queue()];
```

不支持Playlist类型的播放

AVAssetResourceLoader不支持类似M3U和M3U8这类播放列表类型的流，这个问题的回答来自SO链接，官方文档中关于HTTP Live Streaming的一段话也印证了这一点。

在搜索相关问题之前，我尝试了使用AVAssetResourceLoader去加载M3U8播放列表，其中M3U8文件可以获取到，但并非获取了之后直接存储就完事了，还需要进行一些处理：

M3U8中一般有两种类型的URL：相对地址的URL和绝对地址的URL，其中相对地址的URL不需要处理AVPlayer会根据原先的host（也就是带了-streaming后缀的host）进行请求，这样的请求还是会被AVAssetResourceLoader拦截到。而绝对地址的URL则需要对其中的scheme进行处理使其能

够被AVAssetResourceLoader拦截。

处理完所有的URL以后才能把M3U8文件进行保存。

M3U8处理完成之后，就尝试处理其中的一些媒体文件地址，例如ts格式的视频，但经过尝试后发现这类ts的链接并不能被AVAssetResourceLoader拦截到，这才去搜索相关内容后找到了上述的SO链接和官方文档。

AVAssetResourceLoadingContentInformationRequest的contentLength和contentType

`AVAssetResourceLoadingContentInformationRequest`是`AVAssetResourceLoadingRequest`的一个属性

```
1 /*!
2  @property       contentInformationRequest
3  @abstract       An instance of AVAssetResourceLoadingContentInformationRequest that you should populate with information about the
4 */
5 @property (nonatomic, readonly, nullable) AVAssetResourceLoadingContentInformationRequest *contentInformationRequest NS_AVAILABLE(
```

其作用是告诉AVPlayer当前加载的资源类型、文件大小等信息。

`AVAssetResourceLoadingContentInformationRequest`有这样一个属性：

```
1 /*!
2  @property       contentLength
3  @abstract       Indicates the length of the requested resource, in bytes.
4  @discussion Before you finish loading an AVAssetResourceLoadingRequest, if its contentInformationRequest is not nil, you should se
5 */
6 @property (nonatomic) long long contentLength;
```

乍看上去可以把当前所请求数据的`Content-Length`直接赋给这个属性，例如请求range=0-100的那么其`Content-Length`就是100。如果当前数据无缓存的话，就直接把`NSURLResponse`的`expectedContentLength`属性值赋值给了contentLength。

但经过实践发现上面的做法并不正确。对于支持`Range`的请求，如range=0-100，`NSURLResponse`的`expectedContentLength`属性值为100，但这里需要填入的是文件的总长。所以对于response header中包含`Content-Range`的请求，需要解析出其中的文件总长再赋值给`AVAssetResourceLoadingContentInformationRequest`的`contentLength`属性。

接下来是`contentType`：

```
1 /*!
2  @property       contentType
3  @abstract       A UTI that indicates the type of data contained by the requested resource.
4  @discussion Before you finish loading an AVAssetResourceLoadingRequest, if its contentInformationRequest is not nil, you should se
5 */
6 @property (nonatomic, copy, nullable) NSString *contentType;
```

这里的`contentType`是UTI，和`NSURLResponse`的`MIMEType`并不相同。需要进行转换：

```
1 NSString *mimeType = [response MIMEType];
2 CFStringRef contentType = UTTypeCreatePreferredIdentifierForTag(kUTTagClassMIMEType, (__bridge CFStringRef)(mimeType), NULL);
3 loadingRequest.contentInformationRequest.contentType = CFBridgingRelease(contentType);
```

总结

要说的就这么多，希望能帮到大家 =)。

7

Comments