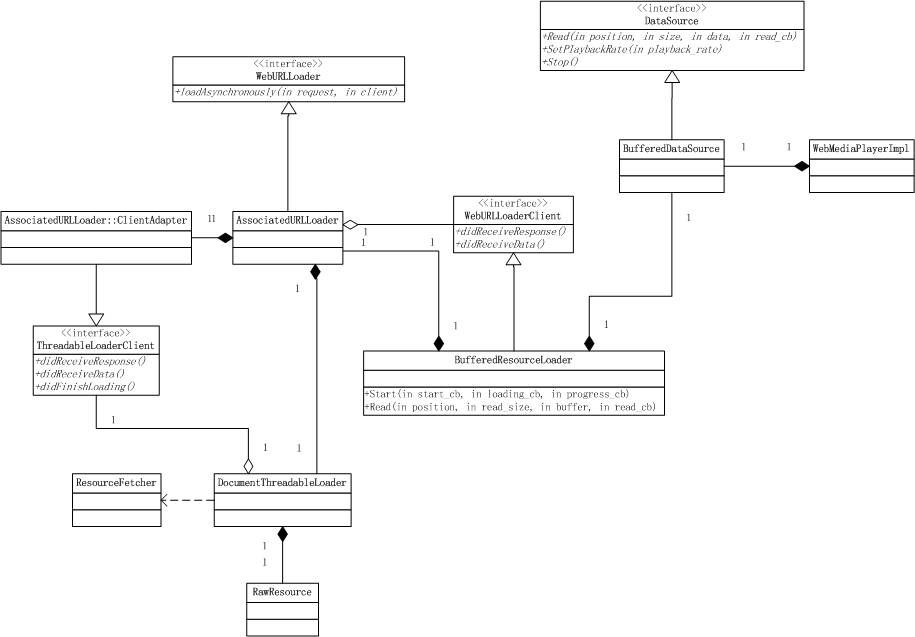
转载请注明出处:http://www.cnblogs.com/fangkm/p/3797278.html

承接上一篇文章。媒体播放,需要指定一个源文件,html5用URL格式来指定视频源文件地址,可以是http链接,也可以使本地源文件(不能直接指定,需要借助blob二进制类型)。播放网络文件比播放本地文件多了个下载流程, 所以下面直接分析网络文件的播放流程,本地文件的播放流程也就清楚了。首先分析下网络视频资源的加载流程,相关结构图如下:



WebMediaPlayerImpl类有一成员BufferedDataSource来负责管理URL网络资源的加载逻辑。

BufferedDataSource资源加载逻辑主要由BufferedResourceLoader完成,

BufferedResourceLoader类维护一个WebURLLoader接口的派生类AssociatedURLLoader对象, AssociatedURLLoader类也并没有真正的和webkit\_glue层的WebURLLoaderImpl一样实现WebURLLoader的接口,而是通过DocumentThreadableLoader类最终依赖WebURLLoaderImpl的实现给主进程发送URL请求WebURLLoaderImpl(WebURLLoaderImpl的流程请参见:http://www.cnblogs.com/fangkm/p/3784660.html)。

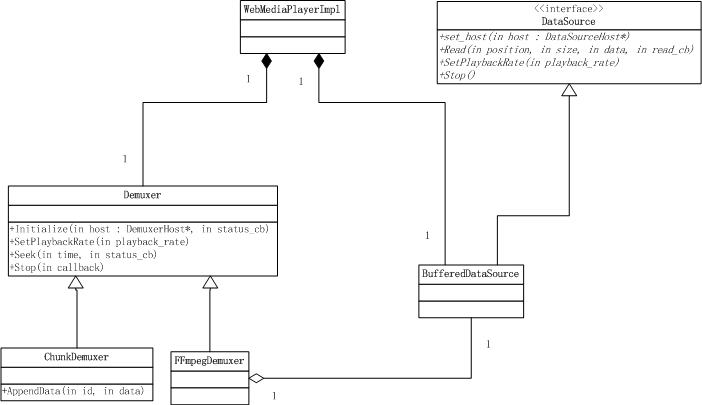
AssociatedURLLoader对象与frame对象关联,当调用WebFrame的stopLoading方法时,该请求也会取消。

BufferedResourceLoader内部维护一个可增长的内存缓冲区来保存请求到的视频数据。

分析到这里我始终没发现暂停缓冲机制,也没有找到缓冲到磁盘文件的地方,如果视频文件过大的话,全部积放在内存,资源消耗过大肯定造成极不好的程序体验。当然我这里的Chromium代码也有点老了，可能新版的已经改进了。

视频数据已经准备完毕,接下来的工作就是解析音视频数据了。在分析这部分之前首先简单普及下音视频的相关概念。一般的视频文件都有视频流和音频流两部分组成，不同的视频格式音视频的封装格式肯定不一样。将音频流和视频流合成文件的过程称为muxer,反之从媒体文件中分离音频流和视频流的过程称为demuxer. 播放视频文件就需要从文件流中分离出音频流和视频流,分别对其进行解码，解码后的视频帧可以直接渲染,音频帧可以送到音频输出设备的缓冲区进行播放,当然,视频渲染和音频播放的时间戳一定要控制同步。

WebMediaPlayerImpl中有关demuxer的逻辑结构如下:

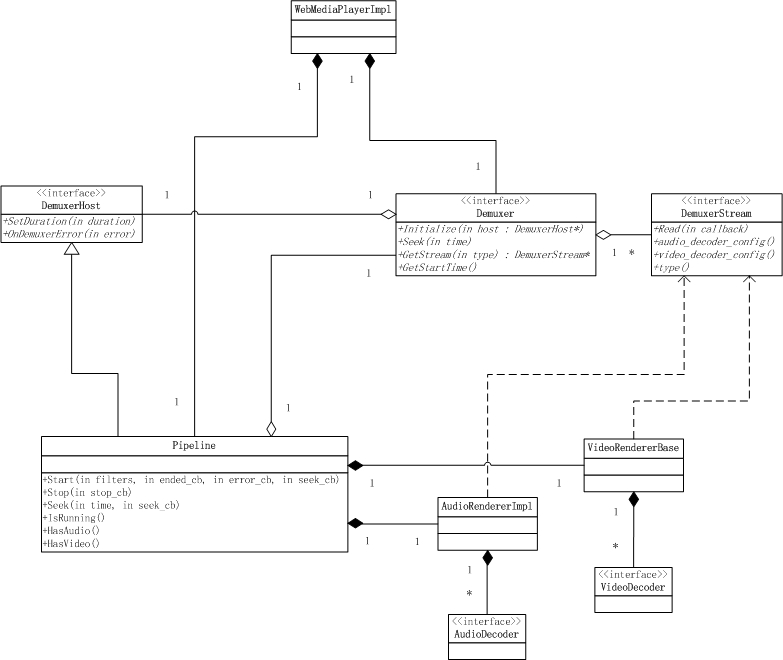


WebMediaPlayerImpl根据资源的不同创建不同的demuxer对象。

如果视频源是通过JavaScript传送过来的二进制数据,则创建ChunkDemuxer对象来分离音频流和视频流;

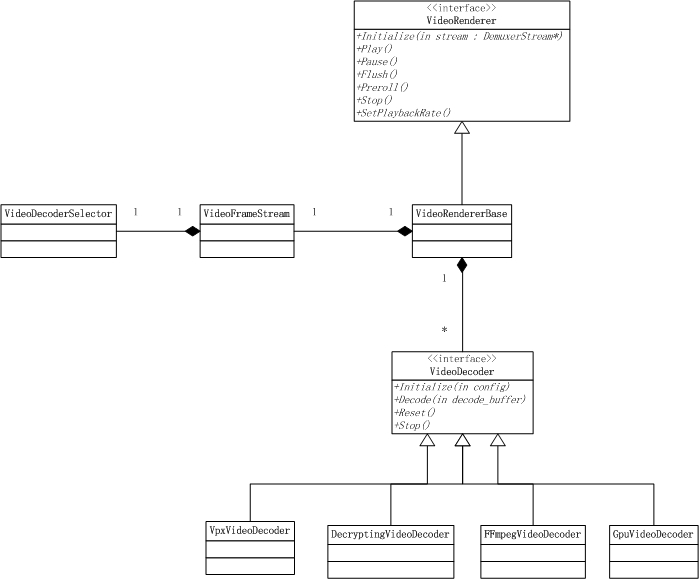
如果视频源是通过URL指定的网络源,则创建FFmpegDemuxer对象,依赖BufferedDataSource对象来访问通过网络加载的媒体流数据。

ChunkDemuxer和FFmpegDemuxer的具体实现暂且不表,先只需要了解他们的作用是将媒体流分离出视频流和音频流。先分析整个播放流程。



WebMediaPlayerImpl类有一个Pipeline对象来负责视频的播放流程, Pipeline本身就是流水线的意思,正适合视频播放的一系列流程。Pipeline内部利用状态机维护播放中的各种阶段的逻辑。Pipeline调用AudioRendererImpl初始化时,会调用Demuxer的GetStream方法,指定获取音频流数据传入AudioRendererImpl对象;同理调用VideoRendererBase初始化时,会取到视频流数据来传入。音视频流的读取操作由DemuxerStream接口来抽象。

下面分析一下VideoRendererBase的流程, VideoRendererBase这名字起的有点奇怪,带个Render单词,确做的是视频流的解码逻辑,真正的绘制操作还是抛到WebMediaPlayerImpl类,具体请参见WebMediaPlayerImpl的paint方法。先看结构:



VideoRendererBase维护了一个VideoDecoder列表,内部主要逻辑都交给VideoFrameStream处理, VideoFrameStream的主要功能包括解码器的选取、从DemuxerStream读取视频流进行解码,解码后的结果为一视频帧结构VideoFrame,这个结构封装的是YUV数据,可以直接或转换成RGB进行渲染操作。

简单介绍下这里的视频解码器创建和选取逻辑：

在WebMediaPlayerImpl类中就已经创建好视频解码器列表,按顺序依次为:

1. 如果gpu支持视频解码,则创建GpuVideoDecoder对象

2. 创建VpxVideoDecoder对象

3. 创建FFmpegVideoDecoder对象

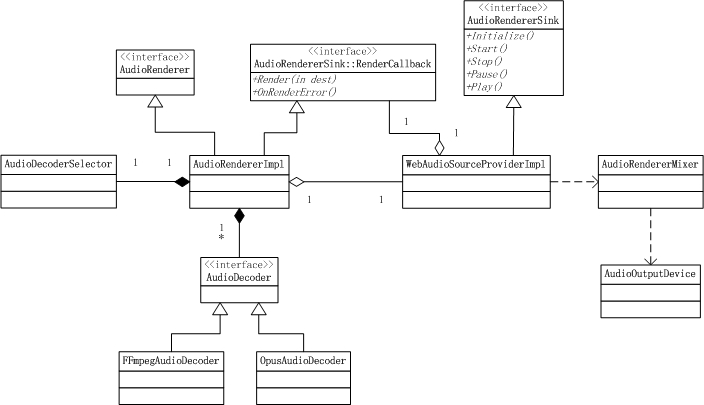
创建好解码器列表后传入VideoRendererBase对象,最终由VideoFrameStream来管理选取逻辑：

1. 如果视频配置信息里有加密选项,则创建DecryptingVideoDecoder做为解码器

2. 如果无加密选项,则从传入的解码器列表中选择第一个做为解码器。

3. 如果调用选择的解码器的Initialize无效(解码器不支持该格式的解码),则按顺序选择列表中的下一个解码器。

AudioRendererImpl的结构与VideoRendererBase类似,在音频渲染方面比视频渲染稍微复杂一点,需要将音频数据输出到声卡设备进行播放。相关结构图:



到此为止,网页播放器整个流程差不多已经清晰了,当然还有很多细节没扒,比如说Media Source流对应的分离器ChunkDemuxer的实现、FFmpegDemuxer内部怎么对ffmpeg的使用、各种解码器的实现等,没有开发经验,要研究透这些细节真的很耗时,有兴趣的童鞋可以自己研究。