ExoPlayer 解码器扩展

SimpleDecoderVideoRenderer的子类实现

实现类SimpleDecoderVideoRenderer的子类, 例如FFmpegDecoderVideoRenderer, 然后实现构造函数, 对于父类的构造函数, drmSessionManager和playClearSamplesWithoutKeys可以根据实际情况填写成null和false

然后实现方法:supportsFormatInternal(), createDecoder(), renderOutputBufferToSurface()和setDecoderOutputMode(), 其中:

- protected int supportsFormatInternal(@Nullable DrmSessionManager<ExoMediaCrypto> drmSessionManager, Format format) {}

  调用RendererCapabilities.create()方法返回是否支持要求的类型, 其中:

  - 如果不支持, 则return RendererCapabilities.create(FORMAT_UNSUPPORTED_TYPE);
  - 如果支持, 则需要根据情况, 调用RendererCapabilities 的
    static int create(@FormatSupport int formatSupport, @AdaptiveSupport int adaptiveSupport, @TunnelingSupport int tunnelingSupport)
    方法, 表示对要求的类型的支持情况

- protected SimpleDecoder<VideoDecoderInputBuffer, ? extends VideoDecoderOutputBuffer, ? extends VideoDecoderException> createDecoder(Format format, @Nullable ExoMediaCrypto mediaCrypto) throws VideoDecoderException {}

  需要返回一个SimpleDecoder的子类, 例如:FFmpegDecoder, 其基本写法:
  public class FFmpegDecoder extends SimpleDecoder<VideoDecoderInputBuffer, VideoDecoderOutputBuffer, FFmpegDecoderException>{}
  其中: FFmpegDecoderException 是VideoDecoderException的子类, 需要实现其两种形式的构造函数:

  - public FFmpegDecoderException(String message)
  - public FFmpegDecoderException(String message, Throwable cause)
    推荐在FFmpegDecoder类中加载需要的so库文件, 例如:

    ```
    public class FFmpegDecoder extends ... {
        static {
            System.loadLibrary("avutil");
            System.loadLibrary("swresample");
            System.loadLibrary("avformat");
            System.loadLibrary("avcodec");
            System.loadLibrary("swscale");
            System.loadLibrary("avfilter");
            System.loadLibrary("avresample");
            System.loadLibrary("avdevice");
        }
        ...
    }
    ```

    然后实现其构造函数
    public FFmpegDecoder(VideoDecoderInputBuffer[] inputBuffers, VideoDecoderOutputBuffer[] outputBuffers, long ffmpeg_ctx)
    分别重写以下方法:

    - protected VideoDecoderInputBuffer createInputBuffer()
      此方法可以直接返回return new VideoDecoderInputBuffer();, 例如:

      ```
      @Override
      protected VideoDecoderInputBuffer createInputBuffer() {
          return new VideoDecoderInputBuffer();
      }
      ```

    - protected VideoDecoderOutputBuffer createOutputBuffer()
      同上, 例如:

      ```
      @Override
      protected VideoDecoderOutputBuffer createOutputBuffer() {
          return new VideoDecoderOutputBuffer(this::releaseOutputBuffer);
      }
      ```

      注意return new VideoDecoderOutputBuffer(this::releaseOutputBuffer);时有一个参数, 这个参数可以是重写后的this::releaseOutputBuffer, 负责释放输出的Buffer.
    - protected FFmpegDecoderException createUnexpectedDecodeException(Throwable error) 用于创建VideoDecoderException的子类FFmpegDecoderException, 例如:

      ```
      @Override
      protected VpxDecoderException createUnexpectedDecodeException(Throwable error) {
          return new VpxDecoderException("Unexpected decode error", error);
      }
      ```

    - protected FFmpegDecoderException decode(VideoDecoderInputBuffer inputBuffer, VideoDecoderOutputBuffer outputBuffer, boolean reset) 实现实际的解码功能, 此处返回的是FFmpegDecoderException, 对于没有异常的情况, 返回NULL,
    - public String getName()
      返回解码器的名称
    - protected void releaseOutputBuffer(VideoDecoderOutputBuffer outputBuffer)
      释放解码器的输出Buffer

- protected void renderOutputBufferToSurface(VideoDecoderOutputBuffer outputBuffer, Surface surface) throws VideoDecoderException {}

  完成渲染数据到Surface的工作

- protected void setDecoderOutputMode(int outputMode)

  设置输出模式, 输出模式有三种:

```
/** Video decoder output mode is not set. */
public static final int VIDEO_OUTPUT_MODE_NONE = -1;
/** Video decoder output mode that outputs raw 4:2:0 YUV planes. */
public static final int VIDEO_OUTPUT_MODE_YUV = 0;
/** Video decoder output mode that renders 4:2:0 YUV planes directly to a surface. */
public static final int VIDEO_OUTPUT_MODE_SURFACE_YUV = 1;
```

VideoDecoderInputBuffer.data 到 Native

VideoDecoderInputBuffer中的数据是放在其父类DecoderInputBuffer中的:

```
/** The buffer's data, or {@code null} if no data has been set. */
  @Nullable public ByteBuffer data;
```

所以一般情况下到达Native层的ByteBuffer会表现为: jobject, 此时: 使用JNIEnv的GetDirectBufferAddress()方法可以得到ByteBuffer的数据指针, 类型为:void*, 然后用reinterpret_cast<const uint8_t*>(<指针>)对void*进行转换, 比如可得到const uint8_t* const buffer

然后调用ffmpeg进行解码后, 得到的数据指针.

VideoDecoderOutputBuffer 到 Native

对于VideoDecoderOutputBuffer直接到Native的情况:

1. 通过JNIEnv的jclass FindClass(const char* name)方法获得VideoDecoderOutputBuffer对应的jclass, 例如:

```
// Populate JNI References.
const jclass outputBufferClass = env->FindClass(
    "com/google/android/exoplayer2/video/VideoDecoderOutputBuffer");
```

2. 然后通过JNIEnv的jfieldID GetFieldID(jclass clazz, "data", "I")方法得到data成员的jfieldID, 例如:

```
jfieldID dataField = env->GetFieldID(outputBufferClass, "data",
                           "Ljava/nio/ByteBuffer;");
```

3. 最后通过JNIEnv的GetObjectField()方法, 通过传入data成员的jfieldID, 这样可以得到对应的jobject, 最后通过JNIEnv的Get方法得到byte *指针, 例如:

```
const jobject dataObject = env->GetObjectField(jOutputBuffer, dataField);
jbyte* const data =
    reinterpret_cast<jbyte*>(env->GetDirectBufferAddress(dataObject));
```

4. 使用memcpy()向data拷贝数据, 但是请注意拷贝的数据长度