# Android视频硬解稳定性问题探讨和处理

罗升阳

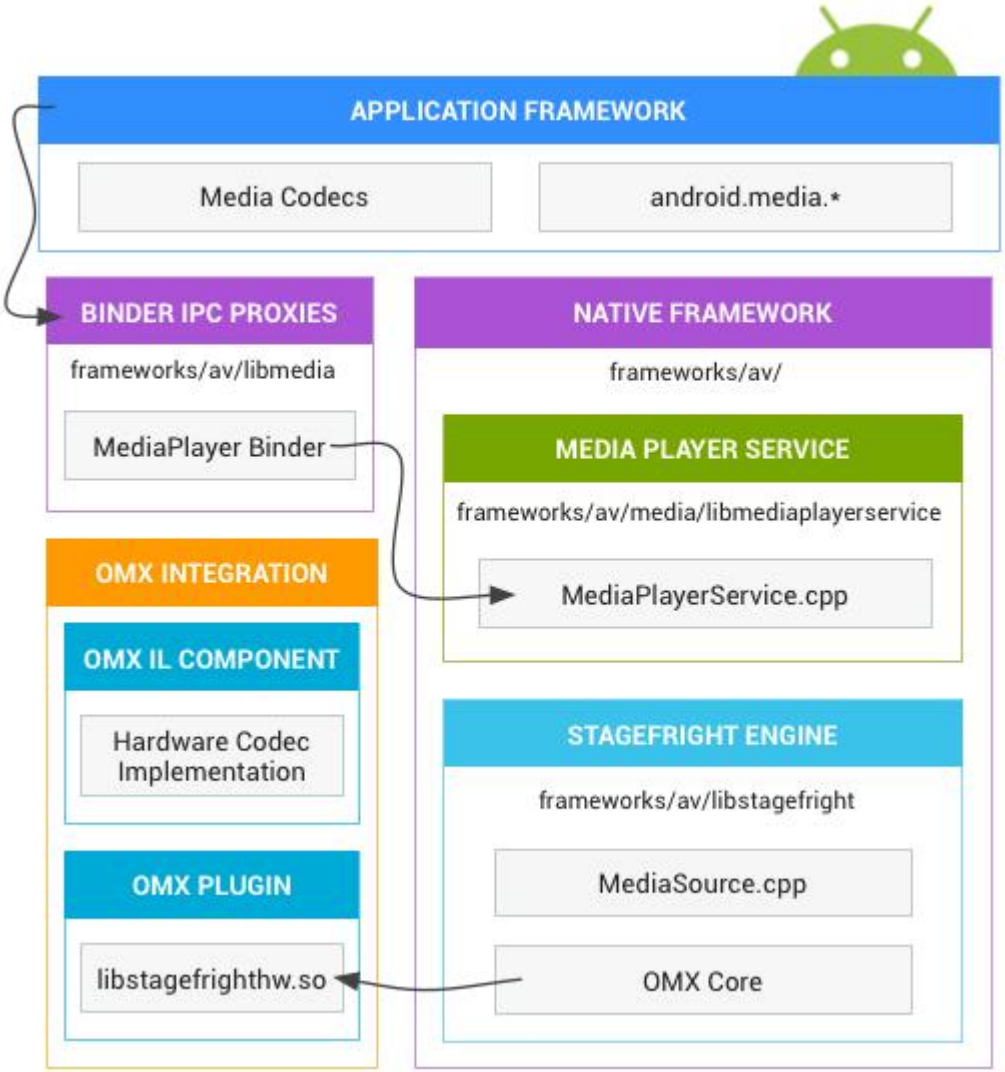
# 议题

- 背景
- 硬解框架
- 硬解崩溃特点
- 硬解使用方法
- 硬解崩溃原因
- 常见解决方案
- 终极解决方案

# 背景

- 某款日活500万的视频类App硬解码崩溃率

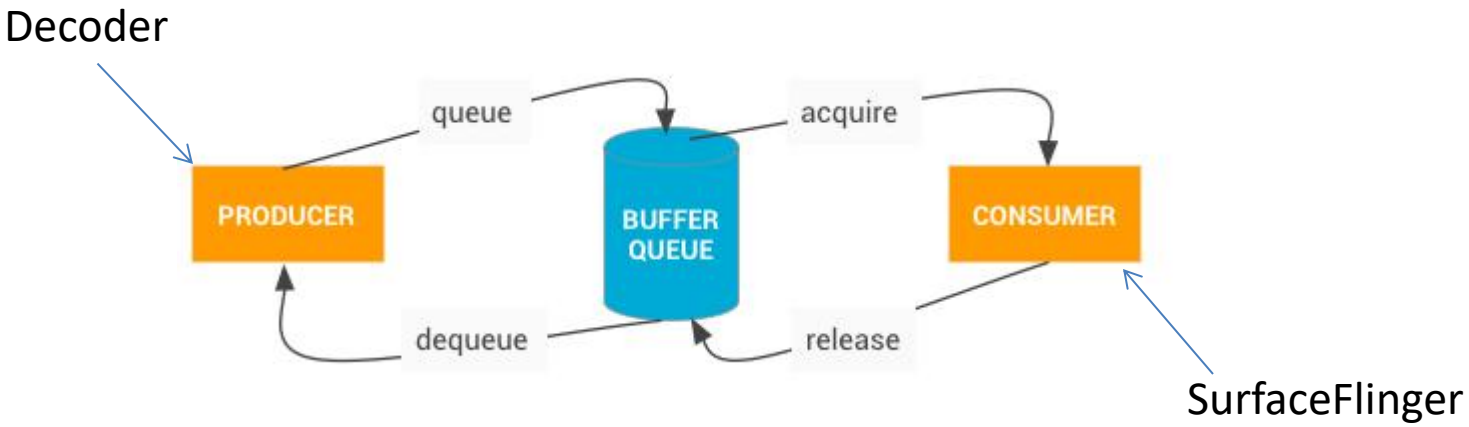
# 硬解框架


APPLICATION FRAMEWORK
Media Codecs
android.media.*
BINDER IPC PROXIES
frameworks/av/libmedia
MediaPlayer Binder
NATIVE FRAMEWORK
frameworks/av/
MEDIA PLAYER SERVICE
frameworks/av/media/libmediaplayerservice
MediaPlayerService.cpp
OMX INTEGRATION
OMX IL COMPONENT
Hardware Codec Implementation
OMX PLUGIN
libstagefrighthw.so
STAGEFRIGHT ENGINE
frameworks/av/libstagefright
MediaSource.cpp
OMX Core

# 硬解使用用法

- 初始化
  - codec = MediaCodec.createDecoderByType(decoderType);
  - MediaFormat format = MediaFormat.createVideoFormat(decoderType, width, height);
  - codec.configure(format, surface, null, 0);
  - codec.start();
- 解码
  - int index = codec.dequeueInputBuffer(TIMEOUT_USEC);
  - codec.queueInputBuffer(index, 0, buffer.size, buffer.pts, 0);
  - int status = codec.dequeueOutputBuffer(bufferInfo, TIMEOUT_USEC);
  - codec.releaseOutputBuffer(status, true);

# Surface（BufferQueue）



# 硬解崩溃特点

- Native Crash占大部分
  - libstagefright.so
- 偶发性
  - 测试时难以发现
  - 上线后会放大问题
- 临时性
  - 只是暂时不能用
- 不可控
  - Crash发生在系统库，应用难以处理
- 影响大
  - 应用闪退

# 硬解崩溃原因

- Surface相关
  - Surface销毁后，解码器仍然在使用
  - SurfaceHolder.Callback. surfaceDestroyed(SurfaceHolder holder) {
    *What happens while stopping MediaCodec asynchronously Here?*
    }
- 硬件相关
  - 频繁创建导致临时不可用
  - 申请不到连续物理内存导致不可用
- StageFright相关
  - 发现硬解器状态不对时自动Crash
  - ssize_t index = dequeuePortBuffer(kPortIndexOutput);
    if (index < 0) {
    CHECK_EQ(index, -EAGAIN);
    return false;
    }

# 常见解决方案

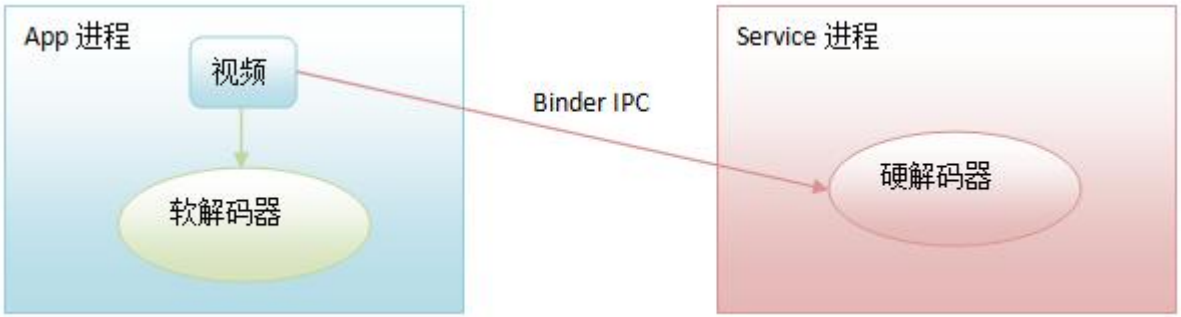
- 白名单
  - 只在严格测试过的手机上使用
  - 过于保守
- 黑名单
  - 无法获取所有的黑名单
- 无论白名单还是黑名单，都无法解决临时不可用的问题

# 终极解决方案

- 目标
  - 最大化使用硬解
  - 崩溃时App不闪退
  - 崩溃时无缝切换到软解
- 思考
  - IE浏览器曾经碰到类似问题，渲染某些页面的偶发性崩溃导致其它正常网页不可用
  - Chrome浏览器保证渲染一个网页出现的崩溃不会影响到其它的网页

# 多进程方案

- 将硬解码器放在一个独立的进程




- 使用流程
  - 播放器一开始都是使用硬解码器
  - 检测到硬解所在的Service进程崩溃后，就切换至App进程中的软解器

# 问题#1

- App进程和MediaCodec所在的Service进程如何建立连接？

```
public class MediaServiceConnection
  implements ServiceConnection {

  private IMediaService service = null;

  public boolean bind() {
    final Intent intent = createServiceBindIntent();
    return context.bindService(intent,
              this, Context.BIND_AUTO_CREATE);
  }

  @Override
  public void onServiceConnected(
    ComponentName className, IBinder binder) {
    service = MediaServiceProxy.asInterface(binder);
  }
}
```

```
public class MediaService extends Service {
  ......

  private final MediaServiceStub binder = new MediaServiceStub() {
    @Override
    public IVideoDecoder createH264HardwareDecoder()
      throws RemoteException {
      return new HardwareDecoderWrapper("video/avc");
  }

  @Override
  public IBinder onBind(Intent intent) {
    return binder;
  }
}
```

# 问题#2

- MediaCodec使用到的Surface来自于App进程中的SurfaceView，如何将该Surface从一个进程传递至别一个进程？

```
decoder = mediaService.createH264HardwareDecoder();

Bundle params = new Bundle();
params.putParcelable(VideoDecoder.KEY_SURFACE, surface);

decoder.config(params);

public class VideoDecoderProxy implements IVideoDecoder {
    ......

    public boolean config(Bundle params)
      throws RemoteException {

      data.writeInterfaceToken(DESCRIPTION);
      params.writeToParcel(data, 0);

      remote.transact(CONFIG_TRANSACTION, data, reply, 0);

      reply.readException();
      return reply.readInt();

    }

    ......
}
```

```
public abstract class VideoDecoderStub
  extends Binder implements IVideoDecoder {

  @Override
  public boolean onTransact(int code, Parcel data,
    Parcel reply, int flags) throws RemoteException {
    switch (code) {
    case CONFIG_TRANSACTION: {
      data.enforceInterface(DESCRIPTION);

      Bundle params = data.readBundle();
      Surface surface
           = (Surface)params.getParcelable(KEY_SURFACE);

      return true;
    }
    }

    return super.onTransact(code, data, reply, flags);
  }
}
```

# 问题#3

- App进程如何检测MediaCodec崩溃？

```
public class MediaServiceConnection implements ServiceConnection {
  .....

  public boolean bind() {
    final Intent intent = createServiceBindIntent();
    return context.bindService(intent, this, Context.BIND_AUTO_CREATE);
  }

  @Override
  public void onServiceDisconnected(ComponentName className) {
    // MediaCodec is crashed
  }
}
```

# 问题#4

- 数据传输效率问题
  - 假设视频码流为2M，那么一秒钟要从App进程传输250000字节到MediaCodec所在的进程。
- 使用匿名共享内存
  - android.os.MemoryFile
- 方法
  - MediaCodec所在进程创建一个MemoryFile
  - 通过反射获得MemoryFile内部的文件描述符
  - 将该文件描述符通过Binder IPC传递给App进程
  - App进程将该文件描述符映射到自己的地址空间

# 问题#4

- 创建一个MemoryFile，并且获得其内部的文件描述符

```
inputMemoryFile = new MemoryFile("hardware_decoder_input_memory_file", inputMemoryFileSize);
inputMemoryFileDiscriptor = getMemoryFileDiscriptor(inputMemoryFile);

private FileDescriptor getMemoryFileDiscriptor(MemoryFile file) {
  Class<?> clazz = file.getClass();
  FileDescriptor fd = null;

  try {
     Method method = clazz.getMethod("getFileDescriptor");
     fd = (FileDescriptor) method.invoke(file);
  } catch (NoSuchMethodException ex) {
     Log.e(LOG_TAG, "Cann't find getFileDescriptor method from class MemoryFile");
  } catch (IllegalArgumentException e) {
     Log.e(LOG_TAG, "Illegal Argument while calling getFileDescriptor method of MemoryFile");
  } catch (IllegalAccessException e) {
     Log.e(LOG_TAG, "Illegal Access while calling getFileDescriptor method of MemoryFile");
  } catch (InvocationTargetException e) {
     Log.e(LOG_TAG, "Invocation Error while calling getFileDescriptor method of MemoryFile");
  }

  return fd;
 }
```

# 问题#4

- 通过Binder IPC传递文件描述符

```
public class VideoDecoderProxy implements IVideoDecoder {
  public ParcelFileDescriptor getInputMemoryFile()
    throws RemoteException {
    Parcel data = Parcel.obtain();
    Parcel reply = Parcel.obtain();

    ParcelFileDescriptor pfd = null;

    try {
      data.writeInterfaceToken(DESCRIPTION);
      remote.transact(GET_INPUT_MEMORY_FILE_TRANSACTION,
                data, reply, 0);
      pfd = ParcelFileDescriptor.CREATOR.createFromParcel(reply);
    } finally {
      reply.recycle();
      data.recycle();
    }

    return pfd;
  }
}
```

```
public abstract class VideoDecoderStub
  extends Binder implements IVideoDecoder {
  @Override
  public boolean onTransact(int code, Parcel data,
    Parcel reply, int flags) throws RemoteException {
    switch (code) {
    case GET_INPUT_MEMORY_FILE_TRANSACTION: {
     data.enforceInterface(DESCRIPTION);
     ParcelFileDescriptor pfd = getInputMemoryFile();
     pfd.writeToParcel(reply, 0);
     return true;
    }
    }
  }
}

public class HardwareDecoderWrapper
  extends VideoDecoderStub
  implements VideoDecoder.VideoDecoderClient {

  private FileDescriptor inputMemoryFileDiscriptor;

  public ParcelFileDescriptor getInputMemoryFile()
          throws RemoteException {
    return ParcelFileDescriptor.dup(inputMemoryFileDiscriptor);
  }
}
```

# 问题#4

- App进程将文件描述符映射到自己的地址空间

```
FileDescriptor fileDescriptor = pfd.getFileDescriptor();

-----------------------------------------------------------------

int fd = jniGetFDFromFileDescriptor(env, fileDescriptor);
void* addr = mmap(NULL, length, prot, MAP_SHARED, fd, 0);

int jniGetFDFromFileDescriptor(C_JNIEnv* env, jobject fileDescriptor) {
    JNIEnv* e = reinterpret_cast<JNIEnv*>(env);
    static jfieldID fid = e->GetFieldID(JniConstants::fileDescriptorClass, "descriptor", "I");
    return (*env)->GetIntField(e, fileDescriptor, fid);
}
```

# 思考

- 对于App中的不稳定模块，都可以考虑将其放在独立的进程中运行，例如Android WebView。
- 如何将Android WebView运行在独立的进程中？

# Q&A