**MediaPlayer Native机制**

接《mediaplayer java与jni机制》，本文分析MediaPlayer的native部分。

frameworks/av/media/libmedia/MediaPlayer.cpp

frameworks/av/include/media/MediaPlayer.h

**1、MediaPlayer简述**

class MediaPlayer : public BnMediaPlayerClient, public virtual IMediaDeathNotifier

**1.1、基类IMediaDeathNotifier**

两个作用：

1、getMediaPlayerService

sm->getService，一直循环执行，直到获取到BpMediaPlayerService，静态智能指针sDeathNotifier，通过linkToDeath(sDeathNotifier)绑定死亡通知DeathNotifier对象到MediaPlayerService的BpBinder，BpBinder把自己注册到驱动中的binder引用里，如果BnMediaPlayerService所在的进程挂掉，binder驱动会主动调用在binder引用中注册过死亡通知的BpBinder，走到sDeathNotifier里的binderDied方法，详细流程参考《通过MediaPlayerService分析binder机制-getService》。

2、内部类DeathNotifier: public IBinder::DeathRecipient

监听binder对应service是否活着，并覆盖（派生）方法binderDied，由于DeathNotifier是IMediaDeathNotifier的内部类，所以在IMediaDeathNotifier构建的时候需要保存自己到sObitRecipients，这样，在DeathNotifier::binderDied中才能调用到IMediaDeathNotifier::died方法，died是纯虚函数，被IMediaDeathNotifier的派生类MediaPlayer实现 -- died：{ notify(MEDIA\_ERROR, MEDIA\_ERROR\_SERVER\_DIED, 0); }，确保在MediaPlayerService消失后，app能够收到server的死亡通知MEDIA\_ERROR\_SERVER\_DIED。

**1.2、MediaPlayer**

主要封装BpMediaPlayer、BpMediaPlayerService的方法，与java和jni中MediaPlayer类的方法对接。实现BpMediaPlayerClient中的notify。接收死亡通知消息，实现died方法。

1、实现IMediaPlayerClient中的方法notify，接收BpMediaPlayerClient（BnMediaPlayer）发送过来的消息通知。

2、实现IMediaDeathNotifier中的纯虚方法died，通知app：server已经不存在。

3、player，封装BpMediaPlayer中的方法，参考《MediaPlayer java与jni机制》，与java和jni中MediaPlayer类的方法接口对应。

4、封装BpMediaPlayerService中的方法decode。

**2、MediaPlayer类具体实现**

此类功能和方法，与java和jni中MediaPlayer类相对应。构造函数太简单了，就是一些变量的初始化，不介绍了。

**2.1、析构函数**

disconnect(); //释放智能指针mPlayer（BpMediaPlayer的强引用）。执行完BpMediaPlayer disconnect，此函数返回后BpMediaPlayer的强引用应该减为零。

看看BpMediaPlayer中的disconnect干了什么。

1、释放mClient（BpMediaPlayerClient的强引用指针）、mPlayer（MediaPlayerBase的强引用指针，这个player是，比如Nuplayer\MstPlayer等）

2、调用mPlayer的setNotifyCallback，把通知函数清空，reset：复位MediaPlayerBase状态；disconnectNativeWindow清除Display Adapter，具体再分析。

3、IPCThreadState::self()->flushCommands() //这里走到talkWithDriver(false)，如果mOut中有数据，则会写入binder驱动，处理后mOut清零；如果mOut中没有数据，会在if ((bwr.write\_size == 0) && (bwr.read\_size == 0)) return NO\_ERROR处直接返回。这个flushCommands的作用就是清除当前线程中与binder驱动有关系的指令，没有则直接返回，有则处理掉。

**2.2、setListener**

jni设置一个listener（MediaPlayerListener，callback）下来，用来返回底层的通知信息。

**2.3、attachNewPlayer**

1、player状态机判断，只有idle和error状态下才可以绑定新的player（BpMediaPlayer）。

2、取出之前的mPlayer（BpMediaPlayer），释放掉。把新建的player赋值给mPlayer，mCurrentState = MEDIA\_PLAYER\_INITIALIZED初始化完成。

**2.4、setDataSource**

url分http、本地播放、StreamSource三种，方法重载，功能是一样的，完成player的初始化:

1、getMediaPlayerService，分析过了，新建并获取BpServiceManager、BpMediaPlayerService

2、service->create，分析过了，新建并获取BpMediaPlayer

3、player->setDataSource，初始化底层播放，调用到BnMediaPlayer（MediaPlayerService::Client）的setDataSource

4、attachNewPlayer，上面有讲，把新建的BpMediaPlayer保存在该对象的私有数据mPlayer中。

setDataSource执行之后，state变成MEDIA\_PLAYER\_INITIALIZED。

**2.5、其它方法**

1、invoke、setMetadataFilter、getMetadata、setVideoSurfaceTexture、prepareAsync\_l、pause、isPlaying、start、stop、getCurrentPosition、getDuration、reset\_l等，都是和BpMediaPlayer的方法一一对应，封装BpMediaPlayer中的方法，注意每个方法中对player状态机的判断。

2、reset的状态机，任何状态都可以执行reset，执行后状态是idle，如果本身就是idle，会直接返回。

3、prepare与prepareAsync的不同之处：通过mSignal.wait(mLock)等待Bn的返回，所以prepare是同步的；而prepareAsync是异步的，不会等待另一个线程，监听MEDIA\_PLAYER\_PREPARED消息，其实prepare也能监听到MEDIA\_PLAYER\_PREPARED。

4、seekTo、seekTo\_l：

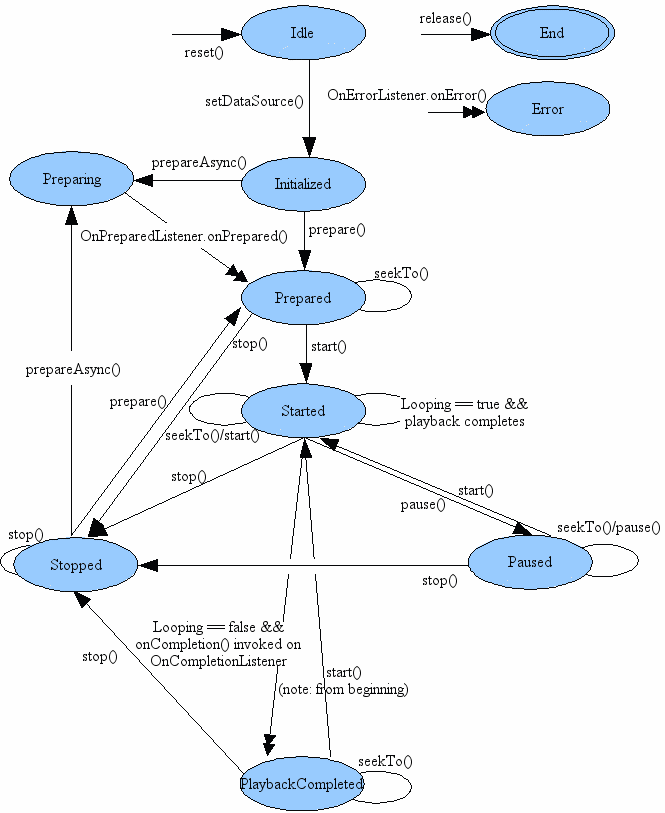
状态机判断，只有(STARTED | PREPARED | PAUSED | PLAYBACK\_COMPLETE)状态可以执行seek。Seek时间和duration比较。mSeekPosition为-1说明当前不在seeking状态，可以seek，seek完成会把mSeekPosition赋为-1。

5、decode，jni中没有调用？

6、notify，前面介绍过，覆盖IMediaPlayerClient中的notify方法，接收BpMediaPlayerClient发过来的通知。然后通过mListener将收到的通知发送给jni。

7、died，也介绍过了，接收service的死亡通知的。发送MEDIA\_ERROR\_SERVER\_DIED到jni

8、MediaPlayer状态机：



**3、总结**

MediaPlayer类，其实是一个封装接口类，只是一个搬砖的，实现app进程与MediaPlayerService进程的交互。

疑点：

1. IPCThreadState::self()->flushCommands()，目的是清除mOut数据，把它刷成零。感觉mOut中不应该有未处理的数据，这个调用是不是多余的？

2、decode方法什么时候被调用？