**BnMediaPlayerClient BpMediaPlayerClient**

Code路径：

frameworks/av/include/media/IMediaPlayerClient.h

frameworks/av/media/libmedia/IMediaPlayerClient.cpp

frameworks/av/include/media/mediaplayer.h

frameworks/av/media/libmedia/mediaplayer.cpp

frameworks/av/include/media/IMediaDeathNotifier.h

frameworks/av/media/libmedia/IMediaDeathNotifier.cpp

**class MediaPlayer : public BnMediaPlayerClient, public virtual IMediaDeathNotifier**

本篇分析Bn Bp MediaPlayerClient类，参考《Bn与Bp MediaPlayerService》。和MediaPlayerService的Bn Bp类基本一样，主要讲述差异部分，即，派生的新增方法功能。

**1、BnMediaPlayerClient**

**1.1、class BnMediaPlayerClient: public BnInterface<IMediaPlayerClient>**

BnInterface参考《Bn与Bp MediaPlayerService》与《Bn与Bp MediaPlayer》。

**1.1.1、IMediaPlayerClient**

讲一下IMediaPlayerClient，只包含一个纯虚函数：

virtual void notify(int msg, int ext1, int ext2, const Parcel \*obj) = 0;

**1.2、BnMediaPlayerClient**

一样的套路，只有一个方法onTransact：

status\_t BnMediaPlayerClient::onTransact(

uint32\_t code, const Parcel& data, Parcel\* reply, uint32\_t flags)

{

switch (code) {

case NOTIFY: {

CHECK\_INTERFACE(IMediaPlayerClient, data, reply);

int msg = data.readInt32();

int ext1 = data.readInt32();

int ext2 = data.readInt32();

Parcel obj;

if (data.dataAvail() > 0) {

obj.appendFrom(const\_cast<Parcel \*>(&data), data.dataPosition(), data.dataAvail());

}

notify(msg, ext1, ext2, &obj);

return NO\_ERROR;

} break;

default:

return BBinder::onTransact(code, data, reply, flags);

}

}

调用派生类MediaPlayer中的notify，reply中没有写入任何数据。

**2、BpMediaPlayerClient**

enum {

NOTIFY = IBinder::FIRST\_CALL\_TRANSACTION,

};

class BpMediaPlayerClient: public BpInterface<IMediaPlayerClient>

{

public:

BpMediaPlayerClient(const sp<IBinder>& impl)

: BpInterface<IMediaPlayerClient>(impl)

{

}

virtual void notify(int msg, int ext1, int ext2, const Parcel \*obj)

{

Parcel data, reply;

data.writeInterfaceToken(IMediaPlayerClient::getInterfaceDescriptor());

data.writeInt32(msg);

data.writeInt32(ext1);

data.writeInt32(ext2);

if (obj && obj->dataSize() > 0) {

data.appendFrom(const\_cast<Parcel \*>(obj), 0, obj->dataSize());

}

remote()->transact(NOTIFY, data, &reply, IBinder::FLAG\_ONEWAY);

}

};

IMPLEMENT\_META\_INTERFACE(MediaPlayerClient, "android.media.IMediaPlayerClient");

和BpMediaPlayer一致，只不过只有一个notify方法。

**3、MediaPlayer**

class MediaPlayer : public BnMediaPlayerClient, public virtual IMediaDeathNotifier

**3.1、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。

**3.2、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已经死亡。

3、player，封装BpMediaPlayer中的方法，参考《MediaPlayer java与jni机制》，与java和jni中MediaPlayer类的方法接口一一对应。

4、封装BpMediaPlayerService中的方法decode，对接java和jni中MediaPlayer类的decode方法。