**BnMediaPlayer BpMediaPlayer**

Code路径：frameworks/av/media/libmediaplayerservice/MediaPlayerService.cpp

frameworks/av/include/media/IMediaPlayer.h

frameworks/av/media/libmedia/ IMediaPlayer.cpp

**class MediaPlayerService :: Client : public BnMediaPlayer**

MediaPlayerService的内部类Client，继承BnMediaPlayer，与MediaPlayerClient相对，各自保存对方的binder代理对象BpMediaPlayerClient、BpMediaPlayer，通过binder机制实现全双工通信，本篇分析Bn Bp MediaPlayer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。

**1、BnMediaPlayer**

MediaPlayerService::create中sp<Client> c = new Client(this, pid, connId, client, audioSessionId, IPCThreadState::self()->getCallingUid());

构造函数：MediaPlayerService::Client::Client(

const sp<MediaPlayerService>& service, pid\_t pid,

int32\_t connId, const sp<IMediaPlayerClient>& client,

int audioSessionId, uid\_t uid)

{

mPid = pid;

mConnId = connId;

mService = service;

mClient = client;

mLoop = false;

mStatus = NO\_INIT;

mAudioSessionId = audioSessionId;

mUID = uid;

mRetransmitEndpointValid = false;

mAudioAttributes = NULL;

}

新建时，获得进程号、线程号、类MediaPlayerService指针、BpMediaPlayerClient指针

**1.1、class BnMediaPlayer: public BnInterface<IMediaPlayer>**

先分析基类

**1.1.1、BnInterface**

模板类，《Bn与Bp MediaPlayerService》中分析过了，多继承**IMediaPlayer和BBinder，同时拥有IInterface和IBinder能力，可向上转换到不同的基类，**

**1.1.2、BBinder**

分析过了，主要功能：transact、linkToDeath、localBinder等，

**1.1.3、IMediaPlayer**

和所有继承IInterface的类一样，引用下面的宏：

DECLARE\_META\_INTERFACE和IMPLEMENT\_META\_INTERFACE，实现asInterface，描述符是"android.media.IMediaPlayer"

函数接口定义，Bn和Bp继承后各自实现，Bp是Bn的代理，通过binder机制转发消息给Bn。

功能接口：disconnect、setDataSource、setVideoSurfaceTexture、prepareAsync、start、stop、pause、seekTo、getCurrentPosition、getDuration、reset、getMetadata等等，是MediaPlayer播放器的播放流程接口，参考《MediaPlayer java与jni机制》，与java和jni中MediaPlayer类的功能接口一一对应。

**1.1.4、BnMediaPlayer**

只覆盖了BBinder中的onTransact方法，同BnMediaPlayerService，IMediaPlayer中的player功能接口由派生类MediaPlayerService :: Client实现，

onTransact中根据Bp侧的cmd调用派生类MediaPlayerService :: Client中的实现。比如MediaPlayerService::Client::setDataSource。这些功能接口的实现在多媒体播放器mediaplayer中介绍，参考《MediaPlayer Native机制》。

**2、BpMediaPlayer**

在BpMediaPlayerService create中创建

**2.1、class BpMediaPlayer: public BpInterface<IMediaPlayer>**

BpMediaPlayer本身的构造函数需要一个BpBinder的智能指针参数，去赋值BpRefBase(remote)。其他函数，比如setDataSource等都是通过binder机制把cmd发送给BnMediaPlayer处理，cmd的命名，BpMediaPlayer和BnMediaPlayer要统一。比如SET\_DATA\_SOURCE\_FD，一般是IMediaPlayer.cpp中定义一个枚举结构，BnMediaPlayer和BpMediaPlayer的实现都在这个文件中，并且需要调用IMPLEMENT\_META\_INTERFACE实现asInterface。

**2.2、BpInterface**

《Bn与Bp MediaPlayerService》介绍过了，主要是构造函数赋值BpRefBase(remote)，onAsBinder返回remote()。

**2.3、IMediaPlayer**

上面介绍过了，接口类，提取要实现的功能方法，和两个宏DECLARE\_META\_INTERFACE和IMPLEMENT\_META\_INTERFACE

**2.4、BpRefBase**

保存一个BpBinder。

**3、如何自己写一个service，通过binder机制，让client和service运行在不同的进程中?**

**3.1、头文件IXXX.h**

**3.1.1、接口类定义，必须I开头，IXXX**

class IXXX: public IInterface

{

DECLARE\_META\_INTERFACE(XXX);

}，需要实现两点：

1、继承IInterface。

2、引用宏DECLARE\_META\_INTERFACE，参数是接口类名字去除I，即，XXX。

**3.1.2、Bn类定义**

class BnXXX: public BnInterface<IXXX>

1、必须继承BnInterface这个模板类，模板是接口类IXXX。

2、定义onTransact，处理所有Bp端发送过来的指令。

**3.2、实现IXXX.cpp**

**3.2.1、定义enum结构，自定义Bp与Bn通信时的指令，比如FUNC\_FD**

**3.2.2、实现class BpXXX: public BpInterface<IXXX>**

1、必须继承BpInterface，模板是IXXX

2、构造函数：BpXXX(const sp<IBinder>& impl) : BpInterface<IXXX>(impl){ } ，需要给基类传入参数const sp<IBinder>&

3、新增的方法实现：

void func()

{

Parcel data, reply; //定义两个Parcel，data写入数据，reply用来接收数据

data.writeInterfaceToken(IXXX::getInterfaceDescriptor()); //data中第一个元素一定要是IXXX的描述符，因为Bn侧要校验这个值。

data.xxxxxx(); //加上其他数据

remote()->transact(枚举变量FUNC\_FD, data, &reply); //调用remote()->transact进行binder通信，发数据和指令到Bn测

}

**3.2.3、IMPLEMENT\_META\_INTERFACE(XXX, " YYY.YYY.IXXX");**

调用这个宏，第一个参数是name，即，XXX，第二个参数是描述符，一般声明为"YYY.YYY.IXXX"。

**3.2.4、status\_t BnXXX::onTransact(){}**

接收并处理Bp发过来的cmd FUNC\_FD。

switch (code) {

case FUNC\_FD: {

CHECK\_INTERFACE(IXXX, data, reply); //每一个cmd执行之前，先检测数据的目的地是否是IXXX。

调用FUNC\_FD的真正实现，处理数据;

reply->xxxx(); //把上一步的结果写进reply。Binder机制保证这个数据会返回到Bp侧

default: //默认case走BBinder onTransact

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

}

**3.2.5、实现XXX类，XXX.h、XXX.cpp**

class XXX : public BnXXX, public virtual IBinder::DeathRecipient

{

实现接口类IXXX中的所有方法。

实现DeathRecipient的binderDied方法，接收该service所依赖的service，如果有的话，监听它，如果依赖的service挂掉了，执行相应的清理。

}