**BpMediaPlayerService create**

create方法走完以后，最终创建了BpMediaPlayer与MediaPlayerService::Client(BnMediaPlayer)、BpMediaPlayerClient与BnMediaPlayerClient。

**1、BpMediaPlayerService::create**

BpMediaPlayerService有很多代理功能，以create为代表，分析BpMediaPlayerService怎样实现对BnMediaPlayerService的代理。

mediaplayer中setDataSource，先通过getMediaPlayerService得到service：BpMediaPlayerService，参考《4-通过MediaPlayerService分析binder机制-getService》，getMediaPlayerService中做了很多工作。调用service->create(this, mAudioSessionId)创建player，这个player是MediaPlayerService::Client的代理BpMediaPlayer，Client本身是一个BnMediaPlayer。注意传递的参数有一个this，后面会用到，它是MediaPlayer对象本身，即，BnMediaPlayerClient。

**2、看看create封了什么数据到parcel**

data.writeInterfaceToken(IMediaPlayerService::getInterfaceDescriptor());//IMediaPlayerService的描述符，校验用，说明发送的目标进程是MediaPlayerService。

data.writeStrongBinder(client->asBinder());这里是BnMediaPlayerClient，是一个BBinder，所以里面填充obj.type = BINDER\_TYPE\_BINDER

data.writeInt32(audioSessionId);

remote()->transact(CREATE, data, &reply);

在addService和getService时，remote()->transact的流程已经分析过了，最终会走到IPCThreadState::talkWithDriver的系统调用ioctl，然后当前进程陷入内核态，只是这一次的目标进程和cmd不一样，目标进程是MediaPlayerService。传送的数据经过3次封装，每次的cmd分别是：BINDER\_TYPE\_BINDER，BC\_TRANSACTION，BINDER\_WRITE\_READ。

**3、binder驱动的工作**

**3.1、BINDER\_WRITE\_READ**

根据cmd，binder驱动binder\_ioctl进入case BINDER\_WRITE\_READ，调用binder\_ioctl\_write\_read，先进入binder\_thread\_write，根据cmd BC\_TRANSACTION又进入binder\_transaction。

tr->target.handle非零，binder\_get\_ref得到该进程中handle对应的binder引用，即，MediaPlayerService在驱动中的binder引用。target\_node = ref->node，拿到binder node实体。target\_proc = target\_node->proc;拿到目标进程。target\_list、target\_wait拿到目标进程的todo列表和wait队列。创建事务t和tcomplete。填充t，比如t->from = thread;、t->to\_proc = target\_proc;、t->to\_thread = target\_thread;、 t->code = tr->code;，把用户空间数据拷贝到t->buffer->data和offp，for循环取出fp数据。根据对用户空间的操作分析这里应该只有一个fp。

**3.2、BINDER\_TYPE\_BINDER**

1、binder\_transaction经过一系列的数据读取和层层解封装，得到对象flat\_binder\_object \*fp，它的type是BINDER\_TYPE\_BINDER。

binder\_get\_node(proc, fp->binder)，在当前进程（MediaPlayer进程）中寻找BnMediaPlayerClient对象在内核中的binder实体，根据proc->nodes红黑树。第一次进来肯定找不到，新建一个binder\_node，里面保存BnMediaPlayerClient对象指针和MediaPlayer进程信息，把用户空间的BBinder与驱动中的binder node实体关联起来。

2、binder\_get\_ref\_for\_node(target\_proc, node)，从目标进程中的proc->refs\_by\_node引用红黑树中查找第1步新建的BnMediaPlayerClient binder\_node实体的binder引用，第一次进来，这里会新建一个binder引用，关联BnMediaPlayerClient的binder\_node实体和目标进程MediaPlayerService进程。同时把这个binder引用添加到MediaPlayerService的binder引用红黑树refs\_by\_node和refs\_by\_desc中，下次再进来就可以直接查询到了。接下来把fp->type = BINDER\_TYPE\_HANDLE，并把新建binder引用的描述符赋给fp->handle = ref->desc，这两步很重要，因为最终会把这个fp数据传到MediaPlayerService进程的用户空间。

3、唤醒目标进程，当前进程处理BINDER\_WORK\_TRANSACTION\_COMPLETE事务

t->work.type = BINDER\_WORK\_TRANSACTION;

list\_add\_tail(&t->work.entry, target\_list);

tcomplete->type = BINDER\_WORK\_TRANSACTION\_COMPLETE;

if (target\_wait)

wake\_up\_interruptible(target\_wait);

当前进程对BINDER\_WORK\_TRANSACTION\_COMPLETE的处理不再介绍了，可以参考前面几篇文章，当前进程处理完这个事务后最终进入中断等待状态，等待被MediaPlayerService进程的reply唤醒。

wake\_up\_interruptible(target\_wait)唤醒目标进程，事务类型：BINDER\_WORK\_TRANSACTION，目标进程从binder\_thread\_read中的wait\_event\_freezable\_exclusive处被唤醒，进入while循环，把BINDER\_WORK\_TRANSACTION事务读取出来，数据读取，target\_node非空，所以：

struct binder\_node \*target\_node = t->buffer->target\_node;

tr.target.ptr = target\_node->ptr;

tr.cookie = target\_node->cookie;得到目标node实体中保存的用户空间的BBinder指针

cmd = BR\_TRANSACTION;

把数据拷贝到用户空间，删除事务t后，再次copy数据到用户空间，函数调用层层返回后给MediaPlayerService返回的是BR\_NOOP、BR\_TRANSACTION指令。

4、对BR\_NOOP的处理是空，MediaPlayerService进程继续读出BR\_TRANSACTION指令，进入IPCThreadState::executeCommand的case BR\_TRANSACTION处理驱动发过来的数据，先把数据填充在Parcel buffer中。

tr.target.ptr不为空，看到曙光了：reinterpret\_cast<BBinder\*>(tr.cookie)->transact(tr.code, buffer, &reply, tr.flags)，取出binder驱动传过来的用户空间的BnMediaPlayerService对象指针cookie，前面有介绍，调用他的transact方法，还记得BBinder的transact实现吗？它会调用派生类，即，BnMediaPlayerService的onTransact。

5、onTransact

{

case CREATE: {

CHECK\_INTERFACE(IMediaPlayerService, data, reply); // IMediaPlayerService描述符校验，确认是发给自己的。

sp<IMediaPlayerClient> client =

interface\_cast<IMediaPlayerClient>(data.readStrongBinder());

int audioSessionId = data.readInt32();

sp<IMediaPlayer> player = create(client, audioSessionId); // MediaPlayerService::create，新建Client : public BnMediaPlayer，MediaPlayerService的内部类。Player中保存了client 指针，即，BpMediaPlayerClient对象指针。

reply->writeStrongBinder(player->asBinder());//把新建的player (BnMediaPlayer)写进reply，最终通过IPCThreadState::sendReply发给Binder驱动。这里同样填充了obj.type = BINDER\_TYPE\_BINDER; obj.binder和obj.cookie = reinterpret\_cast<uintptr\_t>(local)。

return NO\_ERROR;

} break;

}

来看看sp<IMediaPlayerClient> client = interface\_cast<IMediaPlayerClient>(data.readStrongBinder())，这里的client到底怎样变成BpMediaPlayerClient的？

注意：还记得上面提到的“fp->type = BINDER\_TYPE\_HANDLE，并把新建binder ref的描述符赋给fp->handle = ref->desc”吗？readStrongBinder调用unflatten\_binder：

unflatten\_binder {

if (flat) {

switch (flat->type) {

case BINDER\_TYPE\_BINDER:

\*out = reinterpret\_cast<IBinder\*>(flat->cookie);

return finish\_unflatten\_binder(NULL, \*flat, in);

case BINDER\_TYPE\_HANDLE:

\*out = proc->getStrongProxyForHandle(flat->handle);

return finish\_unflatten\_binder(

static\_cast<BpBinder\*>(out->get()), \*flat, in);

}

}

}，又看到getStrongProxyForHandle，从mHandleToObject查找BnMediaPlayerClient binder引用描述符对应的BpBinder，第一次查找不到，走b = new BpBinder(handle)，并把它保存进mHandleToObject，下次再进来直接查找获取。这样就有了下面的表达式：

sp<IMediaPlayerClient> client = interface\_cast<IMediaPlayerClient>(BpBinder);

通过interface\_cast模板函数，新建一个BpMediaPlayerClient，里面有一个BpBinder，mHandle等于BnMediaPlayerClient binder ref的描述符。

6、onTransact返回后，到IPCThreadState::executeCommand，进入sendReply(reply, 0)

status\_t IPCThreadState::sendReply(const Parcel& reply, uint32\_t flags)

{

status\_t err;

status\_t statusBuffer;

err = writeTransactionData(BC\_REPLY, flags, -1, 0, reply, &statusBuffer);

if (err < NO\_ERROR) return err;

return waitForResponse(NULL, NULL);

}

进驱动binder\_thread\_write中看看怎样处理BC\_REPLY，又进入binder\_transaction了，此时是reply，得到目标进程、线程、以及todo、wait等信息。注意没有给target node赋值。新建事务t和tcomplete，读取出reply中的数据fp，type是BINDER\_TYPE\_BINDER，这里和前面讲BnMediaPlayerClient时的流程一样，给BnMediaPlayer新建binder实体和引用，binder实体属于MediaPlayerService进程，并保存在进程的nodes红黑树中；引用属于目标进程MediaPlayer，并保存在目标进程的两个引用红黑树中。

处理完之后，同样的操作fp->type = BINDER\_TYPE\_HANDLE; fp->handle = ref->desc;，然后处理事务t和tcomplete。

当前进程处理tcomplete，这只是一个通知事务，当前进程读出该事务后基本没有什么实际的操作，然后进入wait\_event\_freezable\_exclusive继续回到中断等待状态，等待被唤醒。

唤醒目标进程MediaPlayer，从wait\_event\_freezable\_exclusive处唤醒，执行while循环，读取事务t。由于target node是空的，所以：

tr.target.ptr = 0;

tr.cookie = 0;

cmd = BR\_REPLY;

tr.sender\_pid = 0

删除事务list\_del(&t->work.entry);

\*consumed = ptr - buffer;

数据拷贝到用户空间，函数调用层层返回，回到waitForResponse，读出数据存在mIn中，走到case BR\_REPLY：ipcSetDataReference，把数据填充到Parcel reply中。

函数调用层层返回，回到BpMediaPlayerService create。

**4、回到BpMediaPlayerService::create**

interface\_cast<IMediaPlayer>(reply.readStrongBinder())，又来了，和onTransact case：CREATE中得到BpMediaPlayerClient的流程一样。简述下：

**4.1、reply.readStrongBinder()**

第一次新建BpBinder，b = new BpBinder(handle)，并保存到mHandleToObject。

**4.2、interface\_cast<IMediaPlayer>(BpBinder(handle))**

interface\_cast模板函数，根据BpBinder新建一个BpMediaPlayer对象。整个流程结束了。

getMediaPlayerService获取BpMediaPlayerService，之后MediaPlayerService进入中断等待，再调用service->create(this, mAudioSessionId)，获取BpMediaPlayer之后BpMediaPlayerService再次进入中断等待。

好了，现在有三对儿Bn与Bp：

BpMediaPlayer与MediaPlayerService::Client(BnMediaPlayer)、BpMediaPlayerClient与BnMediaPlayerClient、BpMediaPlayerService与BnMediaPlayerService。

BnMediaPlayer、BpMediaPlayerClient、BnMediaPlayerService属于进程MediaPlayerService。

BpMediaPlayer、BnMediaPlayerClient、BpMediaPlayerService属于进程MediaPlayer。

这几对儿Bp与Bn的Binder通信流程和上面讲述的create的流程（BpMediaPlayerService与BnMediaPlayerService通信）基本一致，只是目标进程和cmd指令有变化。

**5、总结**

整理一下逻辑：

1、在MediaPlayerService进程，通过addService，把自己注册到ServiceManager，同时ServiceManager会绑定一个死亡通知对象在驱动中，监测MediaPlayerService是否还活着。这一步会在驱动中新建MediaPlayerService的binde实体和引用，实体属于MediaPlayerService，引用属于ServiceManager，并把引用描述符和service name存在ServiceManager的svclist中。

2、在MediaPlayer进程，通过getService获取BpMediaPlayerService，与BnMediaPlayerService通信，新建一个binder引用关联到BnMediaPlayerService，保存在当前进程，属于进程MediaPlayer。

3、MediaPlayer类是BnMediaPlayerClient，即MediaPlayerClient，通过BpMediaPlayerService create方法把自己传入binder驱动，新建MediaPlayerClient的binder实体和引用，实体属于进程MediaPlayer，引用属于进程MediaPlayerService，并把引用描述符传到BnMediaPlayerService onTransact生成BpMediaPlayerClient和BnMediaPlayer（包含BpMediaPlayerClient），reply，BnMediaPlayerService再次回到驱动新建BnMediaPlayer的binder实体和引用，实体属于进程MediaPlayerService，引用属于进程MediaPlayer，把引用描述符返给BpMediaPlayerService，BpMediaPlayerService新建BpMediaPlayer返给BnMediaPlayerClient，即，MediaPlayer类。

BpMediaPlayerService create方法调用后：

MediaPlayer进程拥有BpMediaPlayerService、BnMediaPlayerClient、BnMediaPlayerClient又包含BpMediaPlayer

MediaPlayerService进程拥有BnMediaPlayerService、BnMediaPlayer、BnMediaPlayer又拥有BpMediaPlayerClient.。

此时MediaPlayer(BnMediaPlayerClient)就可以和MediaPlayerService::client(BnMediaPlayer)双向binder通信了，类似于全双工通信。