**addService-通过MediaPlayerService分析binder机制**

1. **Main\_mediaserver.cpp，创建BnMediaPlayerService**

Code路径：frameworks/av/media/mediaserver/ main\_mediaserver.cpp

init.rc中的定义：

service media /system/bin/mediaserver

class main

user media

group audio camera inet net\_bt net\_bt\_admin net\_bw\_acct drmrpc mediadrm sdcard\_r

ioprio rt 4

kernel启动完成，加载android的init rc时启动mediaserver这个守护进程，main函数如下：

sp<ProcessState> proc(ProcessState::self());

sp<IServiceManager> sm = defaultServiceManager();

ALOGI("ServiceManager: %p", sm.get());

AudioFlinger::instantiate();

MediaPlayerService::instantiate();

CameraService::instantiate();

AudioPolicyService::instantiate();

SoundTriggerHwService::instantiate();

registerExtensions();

ProcessState::self()->startThreadPool();

IPCThreadState::self()->joinThreadPool();

1. **ProcessState**

Code路径：frameworks/native/include/binder/ ProcessState.h

frameworks/native/libs/binder/ ProcessState.cpp

此类为单例模式，ProcessState::self()新建一个全局ProcessState变量gProcess，在《2-Bp-通过ServiceManager分析binder机制-defaultServiceManager》已经分析过。

完成的工作：

1、打开binder设备节点，该进程保存此处打开binder节点所返回的file指针（fd句柄），同时内核binder驱动新建一个binder\_proc对象，保存进程上下文信息，binder\_proc放在file指针的private\_data中，还有一个binder\_threads对象，保存线程信息。

2、完成内存映射，把用户空间和内核空间映射到同一个物理页面上，这里映射的内存大小：(1\*1024\*1024) - (4096 \*2)单位是小b。

总结：其实这里是给此mediaserver进程在内核binder驱动中映射一个proc实例。

1. **defaultServiceManager**

frameworks/native/libs/binder/ IServiceManager.cpp

defaultServiceManager()得到BpServiceManager（new BpBinder（0））。在《2-Bp-通过ServiceManager分析binder机制-defaultServiceManager》已经分析过。

1. **MediaPlayerService**

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

frameworks/av/media/libmediaplayerservice/ MediaPlayerService.cpp

**4.1、MediaPlayerService::instantiate();**

Code路径：frameworks/native/libs/binder/ IServiceManager.cpp

frameworks/native/libs/binder/ Parcel.cpp

frameworks/native/libs/binder/ BpBinder.cpp

frameworks/native/libs/binder/ IPCThreadState.cpp

defaultServiceManager()->addService(String16("media.player"), new MediaPlayerService());

这里调用BpServiceManager的addService方法，要把MediaPlayerService对象以及他的描述符"media.player"注册到ServiceManager中，注意new MediaPlayerService()得到的是一个BnMediaPlayerService，而ServiceManager中只保存Binder实体的引用。

看code实现：

virtual status\_t addService(const String16& name, const sp<IBinder>& service, bool allowIsolated)

{

Parcel data, reply;

data.writeInterfaceToken(IServiceManager::getInterfaceDescriptor());//这里是"android.os.IServiceManager"

data.writeString16(name);//这里是"media.player"

data.writeStrongBinder(service);// BnMediaPlayerService

data.writeInt32(allowIsolated ? 1 : 0);//默认是0

status\_t err = remote()->transact(ADD\_SERVICE\_TRANSACTION, data, &reply);

return err == NO\_ERROR ? reply.readExceptionCode() : err;

}

**4.1.1、data.writeStrongBinder(service)**

flatten\_binder(ProcessState::self(), val, this)

IBinder \*local = binder->localBinder();//BBinder指针，其实是BnMediaPlayerService自己

flat\_binder\_object obj;

obj.type = BINDER\_TYPE\_BINDER;

obj.binder = reinterpret\_cast<uintptr\_t>(local->getWeakRefs());

obj.cookie = reinterpret\_cast<uintptr\_t>(local);

发送给驱动的数据是MediaPlayerService的本地Binder对象，即BBinder实例。准确的来说，该数据是保存在flat\_binder\_object obj.cookie中的，该数据的类型是BINDER\_TYPE\_BINDER

调用finish\_flatten\_binder()将flat\_binder\_object数据写入到Parcel中

**4.1.2、接下来的Code流程**

走到IPCThreadState::self()->transact(0, code, data, reply, 0)，第一个0说明是BpBinder(0)在调用。Code是ADD\_SERVICE\_TRANSACTION

writeTransactionData(BC\_TRANSACTION, flags, handle, code, data, NULL) flags是0，handle是0，code是ADD\_SERVICE\_TRANSACTION，把以上参数全部封在结构体binder\_transaction\_data tr中，然后把tr和BC\_TRANSACTION写进mOut。

waitForResponse(reply)，talkWithDriver：把mOut/mIn填进结构binder\_write\_read bwr中，用ioctl(mProcess->mDriverFD, BINDER\_WRITE\_READ, &bwr)与内核binder驱动通信。

数据经过了三次封装后："BINDER\_WRITE\_READ"+"binder\_write\_read结构体对象"。在binder\_write\_read的write\_buffer中包含了事务数据；而在数据的data中又包含了flat\_binder\_object等数据。在flat\_binder\_object中就包含了需要传输的MediaPlayerService对象。

数据经过好几次封装，cmd依次是：BINDER\_WRITE\_READ、BC\_TRANSACTION、BINDER\_TYPE\_BINDER、ADD\_SERVICE\_TRANSACTION。

**4.2、Binder驱动**

[device/hisilicon/bigfish/sdk/source/kernel/linux-3.18.y/drivers/staging/android/](http://androidxref.moretv.cn/source/xref/hisi-l-551/device/hisilicon/bigfish/sdk/source/kernel/linux-3.18.y/drivers/staging/android/)binder.c

解析出IPCThreadState的请求数据（bwr），把结果回写在mIn中。

static long binder\_ioctl(struct file \*filp, unsigned int cmd, unsigned long arg)

thread = binder\_get\_thread(proc); 在proc进程（ProcessState::self()时创建的内核binder proc对象）中查找该线程对应的binder\_thread；若查找失败，则新建一个binder\_thread，并添加到proc->threads红黑树中，之前讲过了。

case BINDER\_WRITE\_READ: ret = binder\_ioctl\_write\_read(filp, cmd, arg, thread);

copy\_from\_user(&bwr, ubuf, sizeof(bwr))从用户空间获取binder\_write\_read bwr数据，拿到BC\_TRANSACTION。

binder\_thread\_write case：BC\_TRANSACTION。copy\_from\_user(&tr, ptr, sizeof(tr))从用户空间获取binder\_transaction\_data tr数据。已经解封装两次数据了。进入binder\_transaction。

**4.2.1、binder\_transaction：这个方法比较复杂**

1、由于目标handle是0，所以直接获取到目标server的node实体：target\_node = binder\_context\_mgr\_node，这个全局变量就是ServiceManager，ServiceManager在binder驱动中根本没有binder引用，只有一个实体，根据0可以直接找到。直接获取目标进程信息target\_proc = target\_node->proc;target\_list = &target\_proc->todo;target\_wait = &target\_proc->wait;下一步就是唤醒它了。

新建一个待处理事务t = kzalloc(sizeof(\*t), GFP\_KERNEL)，并且表明t->from = thread这个事务来自当前线程（MediaPlayerService），接着对该事务进行赋值，比如t->to\_proc = target\_proc; t->to\_thread = target\_thread; t->code = tr->code; t->flags = tr->flags; t->buffer->target\_node = target\_node。把用户空间传过来的数据直接拷贝到事务数据中：copy\_from\_user(t->buffer->data, (const void \_\_user \*)(uintptr\_t) tr->data.ptr.buffer, tr->data\_size))。获取用户空间保存的flat\_binder\_object对象的偏移地址和个数copy\_from\_user(offp, (const void \_\_user \*)(uintptr\_t) tr->data.ptr.offsets, tr->offsets\_size)，for (; offp < off\_end; offp++)中挨个取出这些flat\_binder\_object对象。switch (fp->type)：

2、case BINDER\_TYPE\_BINDER: 这里binder\_get\_node(proc, fp->binder)先从当前进程中查找BnMediaPlayerService在驱动中相对应的binder实体，第一次肯定找不到，会binder\_new\_node(proc, fp->binder, fp->cookie)，新建一个node，fp->cookie 和fp->binder就是BnMediaPlayerService的强弱引用，data.writeStrongBinder(service)时写在flat\_binder\_object obj中的，node中保存当前进程信息proc和BnMediaPlayerService的强弱引用，最后把node保存到proc的proc->nodes红黑树中，下次就可以直接查找使用了。

3、binder\_get\_ref\_for\_node(target\_proc, node)，在target\_proc(即，ServiceManager的进程上下文proc)中查找是否包含"该Binder实体的引用"，根据红黑树refs\_by\_node，依据node实体查找。如果没有找到的话，第一次新建并将"该binder实体的引用"添加到target\_proc->refs\_by\_node和refs\_by\_desc红黑树中；new\_ref->desc引用描述符从1开始递增。这个binder引用保存了BnMediaPlayerService的binde实体和ServiceManager进程信息proc，其实此引用属于ServiceManager进程。这样，ServiceManager的进程上下文proc（内核态、驱动）中就存在一个MediaPlayerService的Binder引用，然后，修改fp->type=BINDER\_TYPE\_HANDLE，并使fp->handle = ref->desc，这个fp是一个局部变量指针，指向的内容是上面copy过的t->buffer->data加上flat\_binder\_object对象的偏移量。当事务t传给ServiceManager进程时fp->type和fp->handle都会被用到。

4、fp->handle = ref->desc，这里将handle设为引用的描述，后面会保存到ServiceManager的svcinfo链表中。根据该handle可以找到"该binder实体在target\_proc中的binder引用"；即，可以根据该handle从ServiceManager找到对应的Binder实体的引用，从而获取Binder实体。

5、t->work.type = BINDER\_WORK\_TRANSACTION; // 设置事务的类型为BINDER\_WORK\_TRANSACTION

list\_add\_tail(&t->work.entry, target\_list); // 将事务添加到target\_list队列中，前面分析过target\_list = &target\_proc->todo，所以ServiceManager现在有一个待处理事务了，类型是BINDER\_WORK\_TRANSACTION。

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

list\_add\_tail(&tcomplete->entry, &thread->todo); // 设置待完成工作的类型为BINDER\_WORK\_TRANSACTION\_COMPLETE，并将其添加到当前线程的待完成工作中。此时，Binder驱动已经收到并处理了MediaPlayerService的请求，这个所谓的待完成工作，就是用来让Binder驱动告诉MediaPlayerService，它的请求已经被处理了，但是并没有完成，等待ServiceManager处理。

wake\_up\_interruptible(target\_wait); // 唤醒目标进程ServiceManager

注意：这里一直都是MediaPlayerService进程。此时，MediaPlayerService进程还会继续运行，而且它又通过wake\_up\_interruptible()唤醒了ServiceManager进程。ServiceManager被唤醒后，所做的工作就是将MediaPlayerService注册到它的服务队列中进行管理，具体流程参考《1-Bn-通过ServiceManager分析binder机制-ServiceManager服务启动》。现在继续分析完MediaPlayerService进程。

总结：在binder\_transaction()中，主要进行了以下工作：

(01) 解析出来自MediaPlayerService的请求数据。

(02) 新建MediaPlayerService（BBinder）对应的Binder实体和Binder引用，并将Binder引用保存到ServiceManager的进程上下文proc中。

(03) 新建了待处理事务，并将该事务添加到了ServiceManager的待处理事务队列中。然后，唤醒ServiceManager来处理该事务。

(04) 新建了待完成工作，并将待完成工作添加到了当前线程的待完成工作队列中。

**4.2.2、返回到binder\_thread\_write**

1、binder\_transaction结束后返回到binder\_thread\_write函数，更新\*consumed的值，\*consumed = ptr – buffer。即bwr.write\_consumed的值，说明已经处理了用户空间写进来的数据。binder\_thread\_write进一步返回到binder\_ioctl\_write\_read，接着执行binder\_thread\_read：

A、由于\*consumed == 0，所以会写入一个cmd：BR\_NOOP。走到wait\_event\_freezable(thread->wait, binder\_has\_thread\_work(thread))，binder\_has\_thread\_work返回true，所以继续往下执行。

B、取出上一步的待完成工作switch (w->type) case BINDER\_WORK\_TRANSACTION\_COMPLETE: 写入cmd：BR\_TRANSACTION\_COMPLETE，然后list\_del(&w->entry)，从驱动中清除这个工作。此时t是空了，跳回while开始处，根据逻辑最终break跳出while循环，更新\*consumed = ptr – buffer，之后返回到binder\_ioctl\_write\_read。将bwr数据拷贝到用户空间后返回到进程的用户空间。

到这里驱动一共返回了两个指令给用户空间：BR\_NOOP和BR\_TRANSACTION\_COMPLETE，然后bwr.read\_consumed值变为8。

2、回到IPCThreadState::talkWithDriver()，写进去的数据已经被处理，进行mOut.setDataSize(0)清除mOut中数据，所以mOut中已经是空了，后面的逻辑会用到。读出内核驱动返回的数据bwr中的read buffer到mIn。

回到waitForResponse，读取mIn中的数据。而mIn中的数据就是Binder驱动返回的"BR\_NOOP和BR\_TRANSACTION\_COMPLETE两个指令。先读出的指令是BR\_NOOP，因此这里执行executeCommand(cmd)。BR\_NOOP没有进行任何操作，直接返回。继续回到waitForResponse()中，重新开始while循环，执行talkWithDriver()，此次在if ((bwr.write\_size == 0) && (bwr.read\_size == 0)) return NO\_ERROR这里直接返回了，没有走到ioctl。读取下一个数据BR\_TRANSACTION\_COMPLETE后，reply非空，所以while循环再次开始：talkWithDriver，needRead为true，mOut.dataSize()为0，此时bwr.write\_size=0，bwr.read\_size>0，只读，传到底层binder驱动，执行读取动作，因为此时没有事件待处理，所以wait\_for\_proc\_work为true。默认non\_block为false，执行wait\_event\_freezable\_exclusive(proc->wait, binder\_has\_proc\_work(proc, thread))，binder\_has\_proc\_work(proc, thread)返回false。所以会进入中断等待，当前进程休眠。当ServiceManager处理完MediaPlayerService的addService请求之后，就会将其唤醒。

**4.3、唤醒ServiceManager**

frameworks/native/cmds/servicemanager/ service\_manager.cpp

frameworks/native/cmds/servicemanager/ binder.c

从上面的分析，MediaPlayerService新建了待处理事务，并将该事务添加到了ServiceManager的待处理事务队列中，参考**《1-Bn-通过ServiceManager分析binder机制-ServiceManager服务启动》**，本来ServiceManager线程sleep在binder\_thread\_read中的wait\_event\_interruptible\_exclusive，被MediaPlayerService唤醒后，继续执行binder\_thread\_read。

这次进入binder\_thread\_read会用到用户空间与内核空间共同映射到同一个物理页这个点，在binder\_transaction\_data数据填充时，data中的数据是指针，指针指向的内容不需要执行copy\_to\_user到用户空间，因为共同映射的原因，直接把内核指针加上偏移量转成用户空间指针即可，指针偏移量在mmap时已经保存。注意：内核空间的指针在用户空间是失效的，因为32位系统内核空间是0-1G，用户空间是1-3G。

最后把最终得到的binder\_transaction\_data数据copy\_to\_user(ptr, &tr, sizeof(tr)拷贝到用户空间。函数调用层层返回，回到binder\_loop中的ioctl，接着执行到binder\_parse把数据解析并处理。

**回到《1-Bn-通过ServiceManager分析binder机制-ServiceManager服务启动》分析ServiceManager被唤醒后的操作吧。**

**4.4、ServiceManager唤醒MediaPlayerService**

wait\_event\_freezable\_exclusive，从这里被唤醒。进入while循环提取ServiceManager填入的事务t。处理完事务t，把t从todo中删除。

看看case BINDER\_WORK\_TRANSACTION: 这里什么都没做，直接跳出switch，t非空，if (!t) continue;这里没进入，接着向下：

tr.target.ptr = 0;

tr.cookie = 0;

cmd = BR\_REPLY;

list\_del(&t->work.entry);删除t

\*consumed = ptr - buffer;更新consumed

到这里共反馈了两个指令到MediaPlayerService的用户空间：BR\_NOOP和BR\_REPLY（ServiceManager给驱动的是BC\_REPLY）。

从talkWithDriver（因为在这个函数中的ioctl进入内核binder\_thread\_read方法休眠的）开始，系统调用返回，到waitForResponse中的case BR\_NOOP，什么操作都没有，**MediaPlayerService进程**会再次进入talkWithDriver，没有走到ioctl，在调用ioctl之前直接返回了，waitForResponse中读取到指令BR\_REPLY。

waitForResponse中的case BR\_REPLY，reply->ipcSetDataReference，ipcSetDataReference()是根据参数的值重新初始化Parcel的数据和对象。addService时ServiceManager填的数据都是空，这个指令就是告诉addService已经成功处理完毕，因为没有真实的数据，所以层层向上返回。这样，MediaPlayerService::instantiate()就执行完了。

总结：addService到底干了啥?

1. 在当前binder驱动中新建binder node实体，添加到当前进程的proc->nodes红黑树中，这个binder node中保存有用户空间新建的MediaPlayerService实例指针和当前进程信息结构体对象proc。
2. 新建1中的binder引用，这个引用保存1中的node实体和目标进程ServiceManager的proc。并添加到目标进程ServiceManager的proc->refs\_by\_node、refs\_by\_desc红黑树中
3. 唤醒ServiceManager，ServiceManager在svcinfo链表中新建一个节点，保存addService时service的name和2中binder引用的描述符。
4. 被ServiceManager唤醒，回到MediaPlayerService进程的用户空间。

那么，MediaPlayerService是何时进入消息循环的呢？回到MediaPlayerService进程的main()函数入口中，它后面会通过**startThreadPool** ()进入消息循环。

1. **ProcessState::self()->startThreadPool()**

创建主线程进入消息循环，等待Client的请求，会发送BC\_ENTER\_LOOPER指令到Binder驱动，告诉Binder驱动，MediaPlayerService进入了消息循环状态。

**5.1、startThreadPool**

mThreadPoolStarted = true;

进入spawnPooledThread，然后makeBinderThreadName给线程取一个名字Binder\_XXX，然后new PoolThread (isMain=true)，设为主线程。

class PoolThread : public Thread

{

protected:

virtual bool threadLoop()

{

IPCThreadState::self()->joinThreadPool(mIsMain);

return false;

}

};

PoolThread继承Thread，运行run(name.string())后执行threadLoop，走到IPCThreadState::self()->joinThreadPool(mIsMain)。

**5.2、IPCThreadState::joinThreadPool**

isMain=true，因此会先将BC\_ENTER\_LOOPER指令写入到mOut中。

接着调用getAndExecuteCommand()。该函数会调用talkWithDriver()和Binder驱动进行交互，binder驱动处理BC\_ENTER\_LOOPER消息，之前有讲过BC\_ENTER\_LOOPER，只是把驱动中的thread->looper添加BINDER\_LOOPER\_STATE\_ENTERED属性。驱动返回，继续joinThreadPool中的while，再次调用getAndExecuteCommand，进入talkWithDriver，这次是只读模式，待处理事务列表todo为空，因此MediaPlayerService线程会像ServiceManager一样走到wait\_event\_freezable\_exclusive，进入中断等待状态。当有Client向MediaPlayerService发送请求时，MediaPlayerService就会被唤醒。

1. **IPCThreadState::self()->joinThreadPool()**

这个好像是多余的，因为上一步ProcessState::self()->startThreadPool()中已经调用到了IPCThreadState::self()->joinThreadPool(true)，这里只是会创建一个子线程，IPCThreadState会一直loop在while中。传到binder驱动的cmd是BC\_REGISTER\_LOOPER。Binder驱动会另起一个线程结构thread，状态thread->looper是BINDER\_LOOPER\_STATE\_REGISTERED，不知具体作用是啥?

1. **综述：**

MediaServer进程打开binder设备驱动，里面通过addService创建并添加很多服务到ServiceManager，比如MediaPlayerService，新建MediaPlayerService对应的Binder实体和Binder引用，此binder引用加入ServiceManager进程proc中，并将ServiceManager进程proc存在此Binder引用中；Binder实体中保存MediaPlayerService的对象指针和MediaPlayerService进程proc对象。创建MediaServer进程的主线程进入消息循环，并通知底层的binder驱动，把底层thread looper状态置为BINDER\_LOOPER\_STATE\_EXITED，进入中断等待，等待被唤醒。

1. **延伸**

本章只关注binder，关于BpMediaPlayerService和BnMediaPlayerService类本身的分析参考《Bn与Bp MediaPlayerService》。