Android系统的Binder机制之二——服务代理对象（1）

2011 一月 01

作者: *Simon_fu* 目录: Android, 嵌入式, ... 评论: 2 条评论

上文《Android系统的Binder机制之一——Service Manager》我们学习了Service Manager在Android Binder中的作用——服务（Service）注册，服务（Service）查询的功能。本文我们一起学习服务（Service）在客户端中的代理机制。重点介绍其核心对象BpBinder。

1、服务代理的原理

如下是客户端请求service服务的场景：

> 1、首先客户端向Service manager查找相应的Service。上文《Android系统的Binder机制之一——Service Manager》有比较详细的介绍。注意客户端和Service可能在两个不同的进程中。
> 2、Android系统将会为客户端进程中创建一个Service代理。下文将详细介绍该创建过程。
> 3、客户端视角只有Service代理，他所有对Service的请求都发往Service代理，然后有Service代理把用户请求转发给Service本身。Service处理完成之后，把结果返回给Service代理，Service代理负责把处理结果返回给客户端。注意客户端对Service代理的调用都是同步调用（调用挂住，直到调用返回为止），这样客户端视角来看调用远端Service的服务和调用本地的函数没有任何区别。这也是Binder机制的一个特点。

2、Android进程环境——ProcessState类型和对象

在Android系统中任进程何，要想使用Binder机制，必须要创建一个ProcessState对象和IPCThreadState对象。当然如果Android进程不使用Binder机制，那么这两个对象是不用创建的。这种情况很少见，因为Binder机制是整个Android框架的基础，可以说影响到Android方方面面。所以说了解这两个对象的作用非常重要。

台湾的高焕堂先生一片文章《认识ProcessState类型和对象》，可以在我的博文《（转）高焕堂——Android框架底层结构知多少？》中找到。可以先通过这篇文章对ProcessState进行一个大概了解。

ProcessState是一个singleton类型，一个进程只能创建一个他的对象。他的作用是**维护当前进程中所有Service代理（BpBinder对象）**。一个客户端进程可能需要多个Service的服务，这样可能会创建多个Service代理（BpBinder对象），客户端进程中的ProcessState对象将会负责维护这些Service代理。

我们研究一下创建一个Service代理的代码：

```
1: sp<IBinder> ProcessState::getStrongProxyForHandle(int32_t handle)
2: {
3:     sp<IBinder> result;
4:
5:     AutoMutex _l(mLock);
6:
7:     handle_entry* e = lookupHandleLocked(handle);
8:
9:     if (e != NULL) {
10:         // We need to create a new BpBinder if there isn't currently one, OR we
11:         // are unable to acquire a weak reference on this current one.  See comment
12:         // in getWeakProxyForHandle() for more info about this.
```

getWeakProxyForHandle函数的作用是根据一个binder句柄（上文《Android系统的Binder机制之一——Service Manager》提到Binder驱动为每个Service维护一个Binder句柄，客户端可以通过句柄来和Service通讯）创建对应的Service代理对象。

当前进程首先调用lookupHandleLocked函数去查看当前进程维护的Service代理对象的列

表，该待创建Service代理对象是否已经在当前进程中创建，如果已经创建过了，则直接返回其引用就可以了。否则将会在Service代理对象的列表增加相应的位置（注意系统为了减少分配开销，可能会多分配一些空间，策略是“以空间换时间”），保存将要创建的代理对象。具体代码请参考lookupHandleLocked的源码。

后面代码就好理解了，如果Service代理对象已经创建过了，直接增加引用计数就行了。若没有创建过，则需要创建一个新的Service代理对象。

3、Android进程环境——IPCThreadState类型和对象

Android进程中可以创建一个ProcessState对象，该对象创建过程中会打开/dev/binder设备，并保存其句柄。并初始化该设备。代码如下：

```
ProcessState::ProcessState()
    : mDriverFD(open_driver())
    , mVMStart(MAP_FAILED)
    , mManagesContexts(false)
    , mBinderContextCheckFunc(NULL)
    , mBinderContextUserData(NULL)
    , mThreadPoolStarted(false)
    , mThreadPoolSeq(1)
{
    if (mDriverFD >= 0) {
        // XXX Ideally, there should be a specific define for whether we
        // have mmap (or whether we could possibly have the kernel module
```

mDriverFD保存了/dev/binder设备的句柄，如果仔细查看ProcessState的源码会发现这个句柄不会被ProcessState对象使用。那么保存这个句柄做什么用呢？被谁使用呢？非常奇怪。经过查看ProcessState的头文件，发现如下代码：

```
friend class IPCThreadState;
```

发现IPCThreadState是ProcessState的友元类，那么就可以怀疑这个句柄是被IPCThreadState的对象使用的，然后查看代码发现确实如此。

IPCThreadState也是一个singleton的类型，一个进程中也只能有一个这样的对象。我们查看一下它的talkWithDriver函数：

```
.....
if (ioctl(mProcess->mDriverFD, BINDER_WRITE_READ, &bwr) >= 0)
    err = NO_ERROR;
else
    err = -errno;
.....
```

IPCThreadState通过ioctl系统调用对ProcessState打开的句柄进行读写。这样我们也可以看出IPCThreadState对象的作用：

> 1、维护当前进程中所有对/dev/binder的读写。换句话说当前进程通过binder机制进行跨进程调用都是通过IPCThreadState对象来完成的。
>
> 2、IPCThreadState也可以理解成/dev/binder设备的封装，用户可以不直接通过ioctl来操作binder设备，都通过IPCThreadState对象来代理即可。

我们这里可以再深入地聊一下，不管是客户端进程和Service进程都是需要用IPCThreadState来和binder设备通讯的。如果是客户端进程则通过服务代理BpBinder对象，调用transact函数，该函数作用就是把客户端的请求写入binder设备另一端的Service进程，具体请参阅IPCThreadState类的transact方法。作为Service进程，当他完成初始化工作之后，他需要他们需要进入循环状态等待客户端的请求，Service进程调用它的IPCThreadState对象的joinThreadPool方法，开始轮询binder设备，等待客户端请求的到来，后面我们讨论Service时候会进一步讨论joinThreadPool方法。有兴趣的朋友可以先通过查看代码来了解joinThreadPool方法。

4、Service代理对象BpBinder

上文关于ProcessState的介绍提到了，客户端进程创建的Service代理对象其实就是BpBinder对象。

我们首先了解怎样创建BpBinder对象。

```
BpBinder::BpBinder(int32_t handle)
    : mHandle(handle)
    , mAlive(1)
    , mObitsSent(0)
    , mObituaries(NULL)
{
    LOGV("Creating BpBinder %p handle %d\n", this, mHandle);

    extendObjectLifetime(OBJECT_LIFETIME_WEAK);
    IPCThreadState::self()->incWeakHandle(handle);
}
```

我们可以看出首先是通过IPCThreadState读写binder设备增加中相应binder句柄上的Service的引用计数。然后本地保存代理Service的binder句柄mHandle。

客户进程对Service的请求都通过调用BpBinder的transact方法来完成：

```
status_t BpBinder::transact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    // Once a binder has died, it will never come back to life.
    if (mAlive) {
        status_t status = IPCThreadState::self()->transact(
            mHandle, code, data, reply, flags);
        if (status == DEAD_OBJECT) mAlive = 0;
        return status;
    }

    return DEAD_OBJECT;
}
```

在transact方法中，还是调用客户进程的IPCThreadState对象来完成对相应Service请求。注意transact方法是同步方法，将会挂住客户进程的当前线程，直到service把请求处理完成，并返回结果。这时客户进程当前线程的transact方法返回。

5、Android系统对Binder机制的抽象——IBinder

上面我们讲解了Binder机制比较底层的机制，这些机制直接用还是比较麻烦的，比如使用binder设备的ioctl，需要记住很多ioctl的代码。

Android为了是Binder机制容易使用，对Binder机制进行了抽象，定义了IBinder接口，该接口在C/C++和Java层都有定义。IBinder定义了一套使用Binder机制使用和实现客户程序和服务器的通讯协议。可以理解如下定义：

> 1、向Android注册的Service也必须是IBinder（继承扩展IBinder接口）对象。后续文章中我们讨论Service的时候我们会介绍到这方面的内容。
>
> 2、客户端得到Service代理对象也必须定义成IBinder（继承扩展IBinder接口）对象。这也是为什么BpBinder就是继承自IBinder。
>
> 3、客户端发送请求给客户端，调用接口的Service代理对象IBinder接口的transact方法。
>
> 4、Android系统Binder机制将负责把用户的请求，调用Service对象IBinder接口的onTransact方法。具体实现我们将在以后介绍Service的时候讨论。

6、Service Manager代理对象

我们知道Service Manager是Android Binder机制的大管家。所有需要通过Binder通讯的进程都需要先获得Service Manager的代理对象才能进行Binder通讯。Service Manager即在C/C++层面提供服务代理，又在Java层面提供服务代理，本文先介绍一下C/C++层面的服务代理，Java层面的服务代理将在后续文章中介绍。

进程在C/C++层面上面，Android在Android命名空间中定义了一个全局的函数defaultServiceManager（定义在framework/base/libs/binder），通过这个函数可以使进程在C/C++层面获得Service Manager的代理。我们先看一下该函数的定义：

```
sp<IServiceManager> defaultServiceManager()
{
    if (gDefaultServiceManager != NULL) return gDefaultServiceManager;

    {
        AutoMutex _l(gDefaultServiceManagerLock);
        if (gDefaultServiceManager == NULL) {
            gDefaultServiceManager = interface_cast<IServiceManager>(
                ProcessState::self()->getContextObject(NULL));
        }
    }

    return gDefaultServiceManager;
}
```

我们可以看到defaultServiceManager是调用ProcessState对象的getContextObject方法获得Service Manager的getContextObject方法获得Service Manager代理对象。我们再看一下getContextObject函数的定义：

```
sp<IBinder> ProcessState::getContextObject(const sp<IBinder>& caller)
{
    if (supportsProcesses()) {
        return getStrongProxyForHandle(0);
    } else {
        return getContextObject(String16("default"), caller);
    }
}
```

我们可以看出其实是调用我们上面描述过的getStrongProxyForHandle方法，并以句柄0为参数来获得Service Manager的代理对象。

ProcessState::self()->getContextObject(NULL)返回一个IBinder对象，怎样把它转化成一个IServiceManager的对象呢？这就是模板函数interface_cast<IServiceManager>的作用了。调用的是IServiceManager.asInterface方法。IServiceManager的asInterface方法通过DECLARE_META_INTERFACE和IMPLEMENT_META_INTERFACE宏来定义，详细情况请查看IServiceManager类的定义。IMPLEMENT_META_INTERFACE宏关于asInterface的定义如下：

```
android::sp<I##INTERFACE> I##INTERFACE::asInterface(            \
        const android::sp<android::IBinder>& obj)                \
{                                                                \
    android::sp<I##INTERFACE> intr;                              \
    if (obj != NULL) {                                           \
        intr = static_cast<I##INTERFACE*>(                       \
            obj->queryLocalInterface(                            \
                    I##INTERFACE::descriptor).get());            \
        if (intr == NULL) {                                      \
            intr = new Bp##INTERFACE(obj);                       \
        }                                                        \
    }                                                            \
```

最终asInterface将会用一个IBinder对象创建一个BpServiceManager对象，并且BpServiceManager继承自IServiceManager，这样我们就把IBinder对象转换成了IServiceManager对象。如果你仔细查看BpServiceManager的定义，你会发现查询Service，增加Service等方法其实都是调用底层的IBinder对象来完成的。

当我们在C/C++层面编写程序使用Binder机制的时候将会调用defaultServiceManager函数来获得Service Manager，比如：很多Android系统Service都是在C/C++层面实现的，他们就需要向Service Manager注册其服务，那么这些服务将调用defaultServiceManager获得Service Manager代理对象。我们在后续介绍Android系统Service的时候将会详细介绍。

7、总结

本文中我们介绍了C++层面的Service代理，后续文章我们将介绍Java层面的Service代理。

标签：android, binder, framework, program

相关文章

Android系统的Binder机制之一——Service Manager
Android系统的Binder机制之四——系统Service
Android系统的Binder机制之三——服务代理对象（2）
Android系统集成第三方pre-build库和程序
（转）Android核心分析
(翻译)Android属性系统
Android应用程序获得root权限
（转）高焕堂——Android框架底层结构知多少？
Android JNI编程提高篇之二
Android JNI编程提高篇之一
Android JNI开发入门之二
Android JNI开发入门之一
Android应用开发之Java基础篇——内部类
观察者（Observer）模式在Android应用
模板方法（Template method）模式在Android应用

2 条评论

galoisy
2011-03-03 12:03:50

Hi Simon
请问你在哪里工作，我们公司在上海，希望有机会合作。
我的联系方式是galoisy@hotmai.com(msn/email)，能否告诉我你的联系方式

谢谢

[回复]

Simon_fu 回复:
三月 3rd, 2011 at 12:48

我的EMail: simon.fu@live.cn，欢迎和我交流。
目前也在上海，你那里高就阿？

[回复]

评论

作者

E-Mail

个人网址

评论（必填）

发表评论

有人回复时邮件通知我