解决Android AIDL通信中DeadObjectException问题的示例方法

解决Android AIDL通信中DeadObjectException问题的示例方法

目录 +

崩溃来源

使用过AIDL进行跨进程通信的同学，肯定遇到过**DeadObjectException**这个崩溃，那么这个崩溃是怎么来的，我们又该如何解决它呢？今天这篇文章就来聊一聊。
首先，这个崩溃的意思是，多进程在进行跨进程**Binder**通信的时候，发现通信的**Binder**对端已经死亡了。
抛出异常的**Java**堆栈最后一行是**BinderProxy**.transactNative，所以我们从这个方法入手，看看崩溃是在哪里产生的。
很显现，transactNative对应的是一个**native**方法，我们找到对应的**native**方法，在android_util_Binder.cpp中。

```
static jboolean android_os_BinderProxy_transact(JNIEnv* env, jobject obj,
        jint code, jobject dataObj, jobject replyObj, jint flags) // throws
RemoteException
{
    // 如果data数据为空，直接抛出空指针异常
    if (dataObj == NULL) {
        jniThrowNullPointerException(env, NULL);
        return JNI_FALSE;
    }
    // 将Java层传入的对象转换为C++层的指针，如果转换出错，中断执行，返回JNI_FALSE
    Parcel* data = parcelForJavaObject(env, dataObj);
    if (data == NULL) {
        return JNI_FALSE;
    }
    Parcel* reply = parcelForJavaObject(env, replyObj);
    if (reply == NULL && replyObj != NULL) {
        return JNI_FALSE;
    }
    // 获取C++层的Binder代理对象指针
    // 如果获取失败，会抛出IllegalStateException
    IBinder* target = getBPNativeData(env, obj)->mObject.get();
```

```
    if (target == NULL) {
        jniThrowException(env, "java/lang/IllegalStateException", "Binder has
been finalized!");
        return JNI_FALSE;
    }
    // 调用BpBinder对象的transact方法
    status_t err = target->transact(code, *data, reply, flags);
    // 如果成功，返回JNI_TRUE，如果失败，返回JNI_FALSE
    if (err == NO_ERROR) {
        return JNI_TRUE;
    } else if (err == UNKNOWN_TRANSACTION) {
        return JNI_FALSE;
    }
    // 处理异常情况的抛出
    signalExceptionForError(env, obj, err, true /*canThrowRemoteException*/,
data->dataSize());
    return JNI_FALSE;
}
```

可以看到，这个方法主要做的事情是：
将Java层传入的data，转换成C++层的指针
获取C++层的Binder代理对象
调用BpBinder对象的transact方法
处理transact的结果，抛出异常
接下来我们看看，BpBinder的transact方法。

```
status_t BpBinder::transact(uint32_t code, const Parcel& data, Parcel* reply,
uint32_t flags)
{
    // 首先判断Binder对象是否还存活，如果不存活，直接返回DEAD_OBJECT
    if (mAlive) {
            ...
            status = IPCThreadState::self()->transact(binderHandle(), code,
data, reply, flags);
            return status;
    }
    return DEAD_OBJECT;
}
```

transact的具体方法，我们这里先不讨论。我们可以看到，在这里会判断当前的Binder对象是否alive，如果不alive，会直接返回DEAD_OBJECT的状态。
返回的结果，在android_util_Binder的signalExceptionForError中处理。

```
void signalExceptionForError(JNIEnv* env, jobject obj, status_t err,
        bool canThrowRemoteException, int parcelSize)
{
       // 省略其他异常处理的代码
        ....
        case DEAD_OBJECT:
            // DeadObjectException is a checked exception, only throw from
certain methods.
            jniThrowException(env, canThrowRemoteException
                    ? "android/os/DeadObjectException"
                            : "java/lang/RuntimeException", NULL);
            break;
}
```

这个方法，其实包含非常多异常情况的处理。为了看起来更清晰，这里我们省略了其他异常的处理逻辑，只保留了DEAD_OBJECT的处理。可以很明显的看到，在这里我们抛出了**DeadObjectException**异常。

解决方法

通过前面的源码分析，我们知道**DeadObjectException**是发生在，当我们调用transact接口发现**Binder**对象不再存活的情况。
解决方案也很简单，就是当这个**Binder**对象死亡之后，不再调用transact接口。

方法1 调用跨进程接口之前，先判断Binder是否存活

这个方案比较简单粗暴，就是在多有调用跨进程接口的地方，都加一个Binder是否存活的判断。

```
        if (mService != null &amp;amp;&amp;amp; mService.asBinder().isBinderAlive())
{
            mService.test();
        }
```

我们来看下isBinderAlive的源码，就是判断mAlive标志位是否为0。

```
bool BpBinder::isBinderAlive() const
{
    return mAlive != 0;
}
```

方法2 监听Binder死亡通知

先初始化一个**DeathRecipient**，用来监听死亡通知。

```
    private IBinder.DeathRecipient mDeathRecipient = new
IBinder.DeathRecipient() {
        @Override
        public void binderDied() {
            // 解绑当前监听，重新启动服务
            mService.asBinder().unlinkToDeath(mDeathRecipient, 0);
            if (mService != null)
                bindService(new Intent("com.service.bind"), mService,
BIND_AUTO_CREATE);
        }
    };
```

在这个死亡监听里，我们可以选择几种处理方式：
什么都不做，直接将mService设置为空
再次尝试启动和绑定服务
在onServiceConnected方法中，注册死亡监听：

```
public void onServiceConnected(ComponentName name, IBinder service) {
    mService = IServiceInterface.Stub.asInterface(service);
    //获取服务端提供的接口
    try {
        // 注册死亡代理
        if(mService != null){
            service.linkToDeath(mDeathRecipient, 0);
        }
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
```

总结

跨进程通信时，无法避免出现**Binder**对端挂掉的情况，所以在调用相关通信接口时，一定要判断连接是否可用，否则就会出现**DeadObjectException**的崩溃。
以上就是Android AIDL通信DeadObjectException解决方法示例的详细内容，更多关于Android AIDL通信的资料请关注其它相关文章！