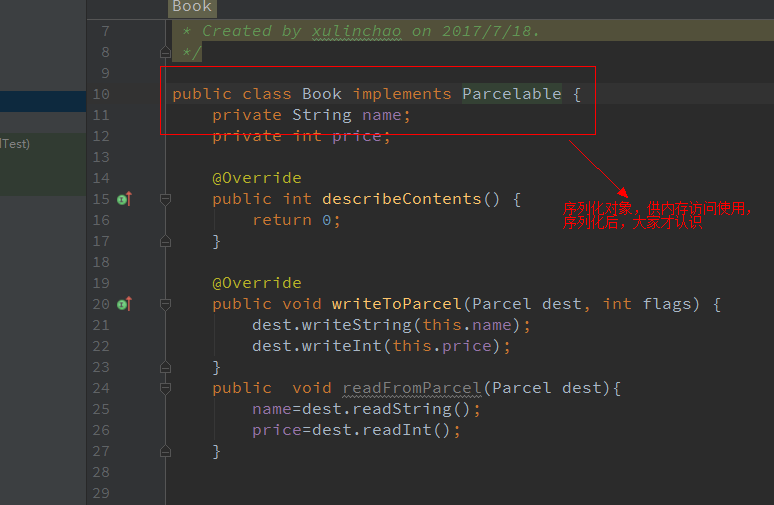
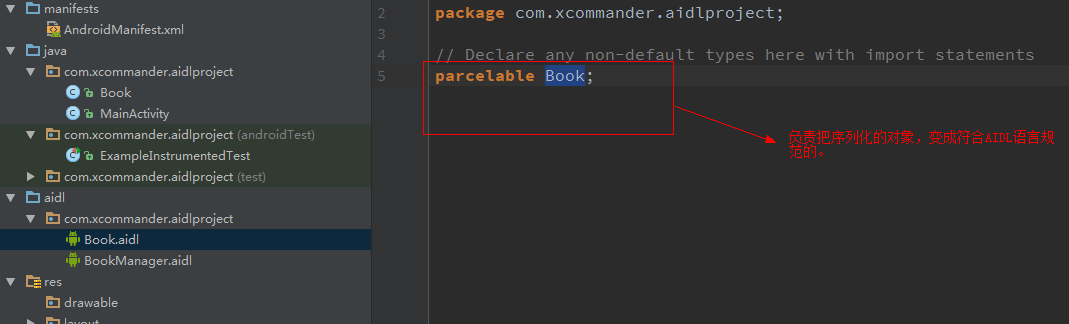
AIDL适用的范围是跨进程访问。

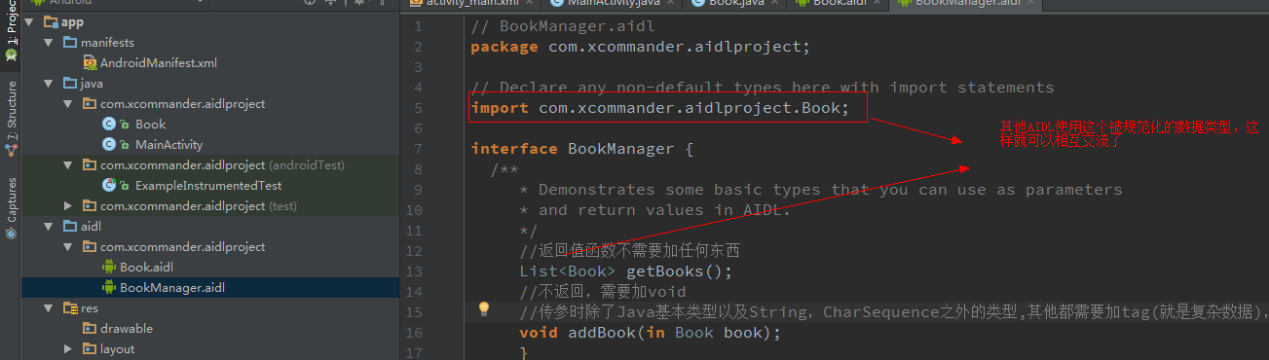
什么是跨进程访问：在android程序中，不同的应用程序（进程）是不能共享内存的。通过某种方式访问一个进程的内存，这样的方式叫做跨进程访问。

一般来说AIDL用来跨进程访问的。对于复杂数据来说，比如一个对象，我们一般是通过序列化，然后通过一个AIDL语法包装成一个可识别的对象。这样其他AIDL才能识别。

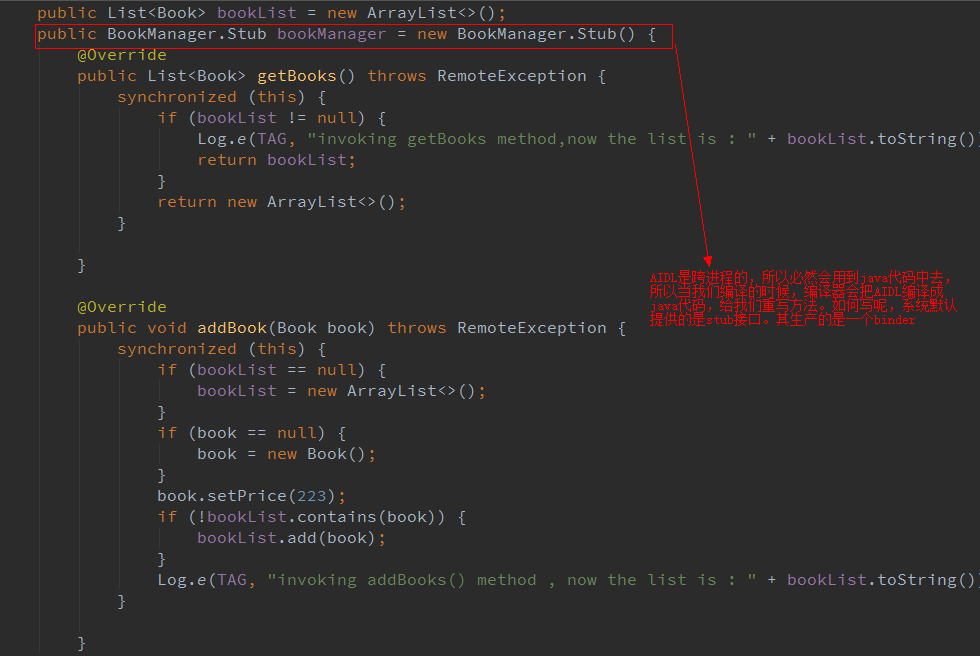
也就是对于AIDL跨进程通信来说，对于引入一个复杂数据，必须先给这个类序列化，然后通过一个AIDL包装，这样才可以给别的AIDL使用。之所以序列化的对象要用AIDL包装，是为了把这个序列化对象改成AIDL语言类型。这样就符合AIDL语言规范了。（记住AIDL是一种简化的语言）。如下：

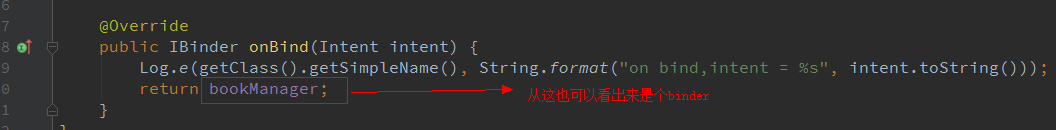




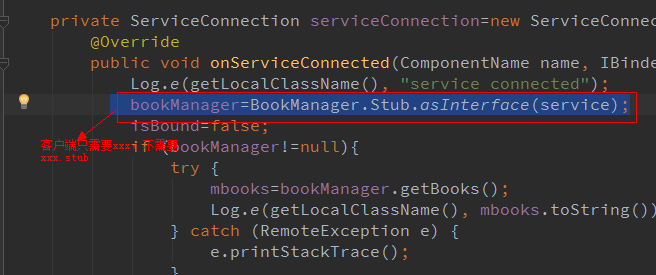


本质上说AIDL编译后，生成的java文件，其实就是binder，其接口就是sub。所以说，我们在java代码中使用AIDL 这里指的是service端，一般使用的是重写xx.Stub即可。如下图





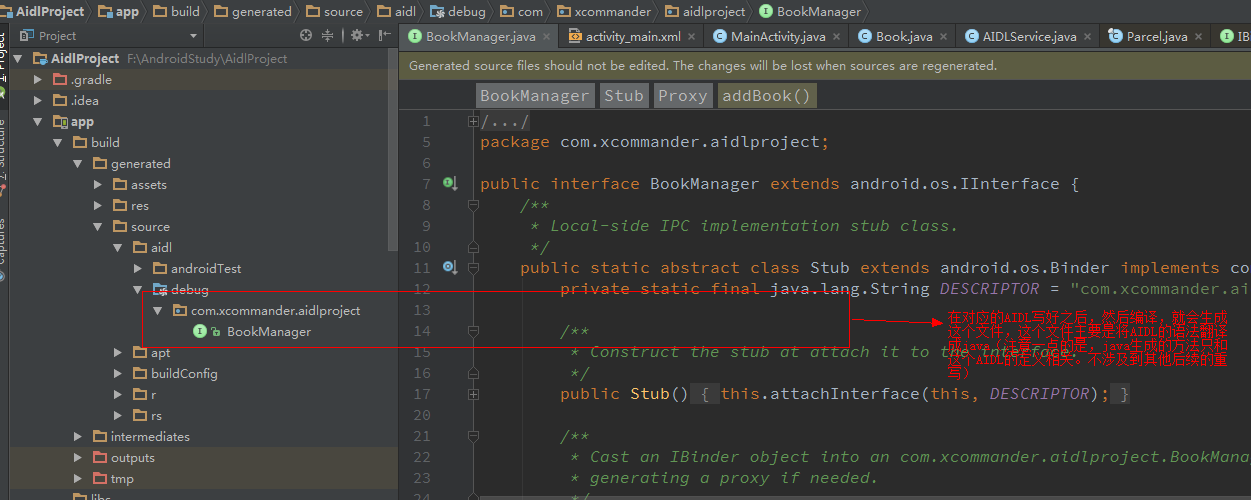
在客户端，也就是其他进程，直接使用xxx，就可以了，不需要xxx.stub。



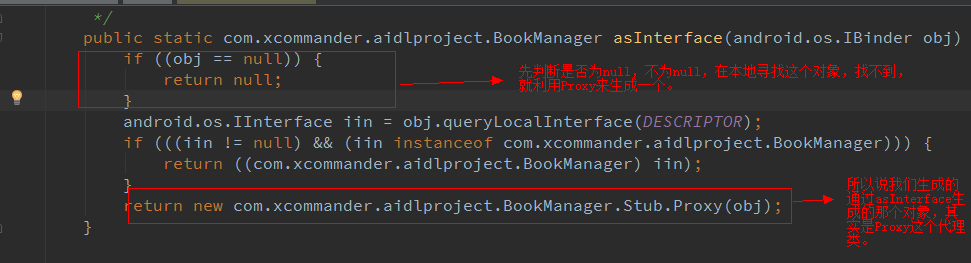
简单来说，用AIDL方式，被访问的进程，需要重写xxx.Stub，然后作为binder返回。对于访问的进程来说，就直接可以使用xxx来申明对象。

总结来说，**所以说AIDL语言只是在简化我们写这个 .java 文件的工作而已，而要研究AIDL是如何帮助我们进行跨进程通信的，其实就是研究这个生成的 .java 文件是如何工作的。**

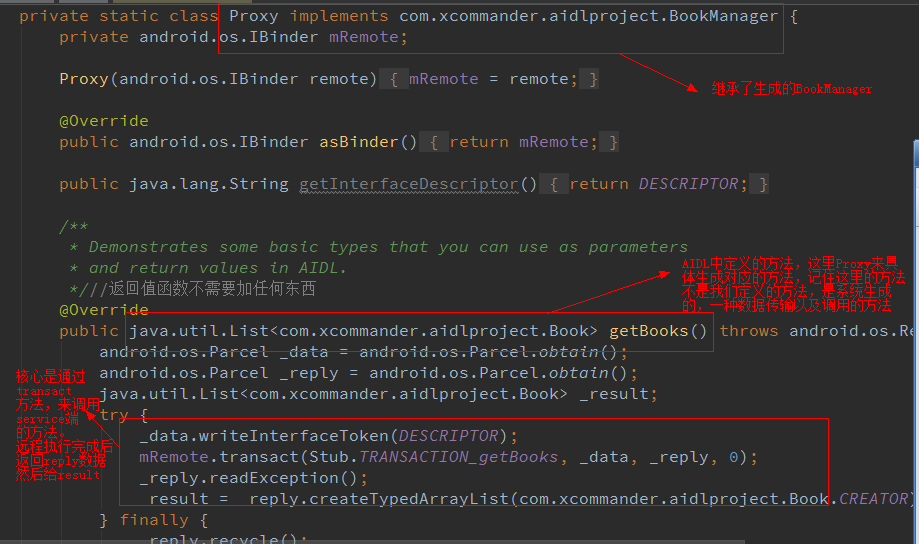
现在我们来看下，这个java文件是如何工作的，首先我们客户端，只需要定义xxx，然后通过xxx. Stub.*asInterface*(service),来生成对象。都是在这个java类中做的



我们来看下xxx. Stub.*asInterface*(service)生成的源码。



我们接下来，继续深入，Proxy这个class。



从上面这个图我们看的出这个Proxy代理类，做了几个步骤。

* 1，生成 \_data 和 \_reply 数据流，并向 \_data 中存入客户端的数据。
* 2，通过 transact() 方法将它们传递给服务端，并请求服务端调用指定方法。
* 3，接收 \_reply 数据流，并从中取出服务端传回来的数据。

简单来说这个就是java的代理模式。

Client端获得代理对象Proxy（代理对象肯定继承xxx），然后调用相关方法（代理类的），这时候，代理类就会在这个方法中，包装数据\_data，然后通过transact（）方法，将数据传入到service的代理中（stub类中）也就是调用Stub的onTransact方法，然后onTransact中根据传来的参数具体来分析，然后处理完成后，将数据放回。

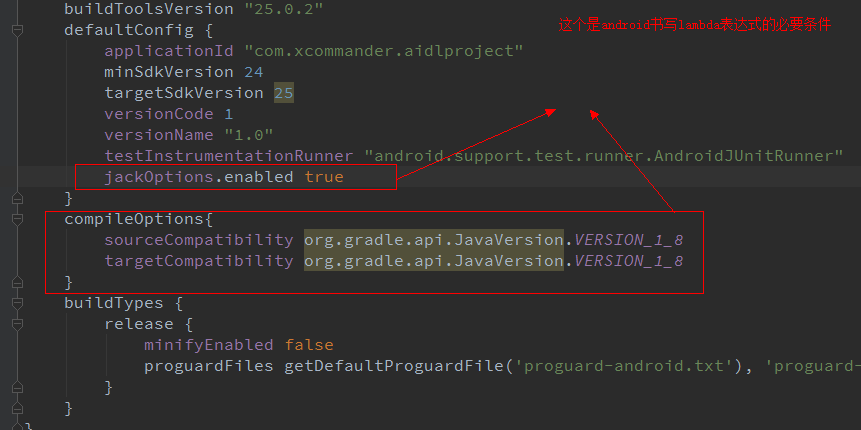
也就是说Client的代理类是Proxy，service的代理类是Stub。

从getBooks()方法来分析上面过程。

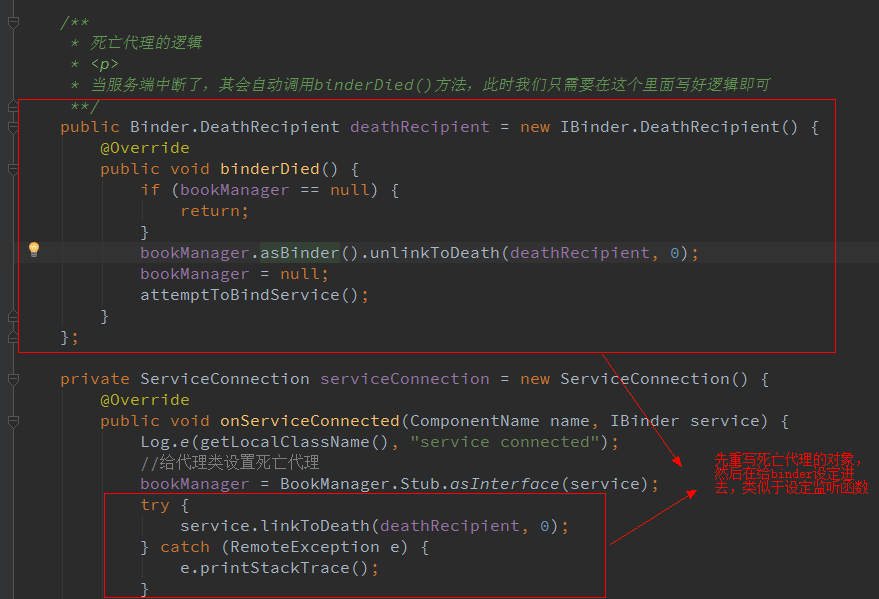
1. bookManager=BookManager.Stub.*asInterface*(service);//proxy类
2. bookManager.getBooks();//进入proxy中的getBooks()
3. mRemote.transact(Stub.*TRANSACTION\_getBooks*, \_data, \_reply, 0);//然后调用Stub中的Transact方法，接着转换到Stub中的onTransact方法。
4. public boolean onTransact(int code, android.os.Parcel data, android.os.Parcel reply, int flags) //开始具体处理,处理完成后，再返回。



简单来说AIDL是简化binder跨进程写法的语法。

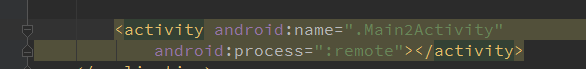


Binder中的死亡代理设置。（当服务断开时，我们就要重新连接，这时候死亡代理就是当服务因为未知原因断开，我们要继续开始）



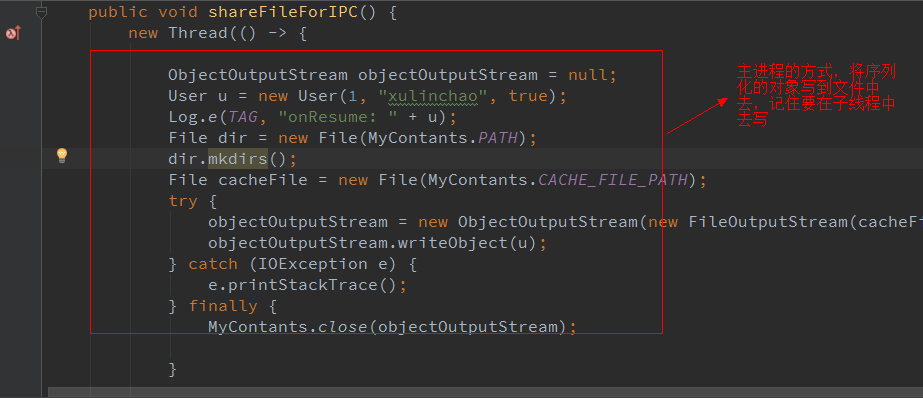
其他IPC的方式。

Android中提供多线程的方式，只有一种规范。

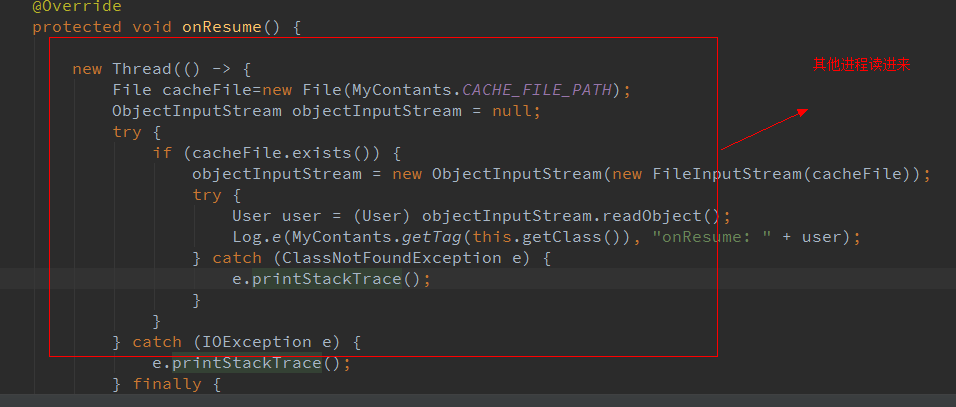


1. 文件共享方式

主进程：



其他进程：

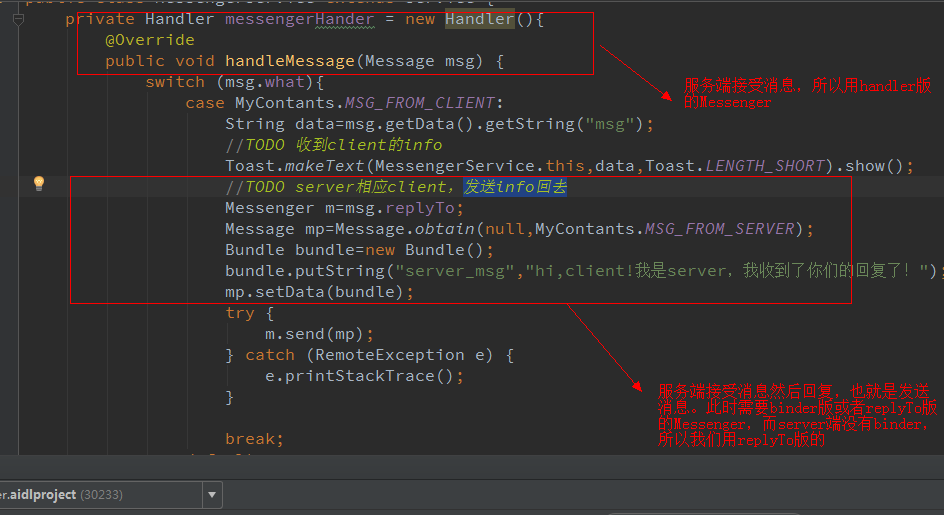


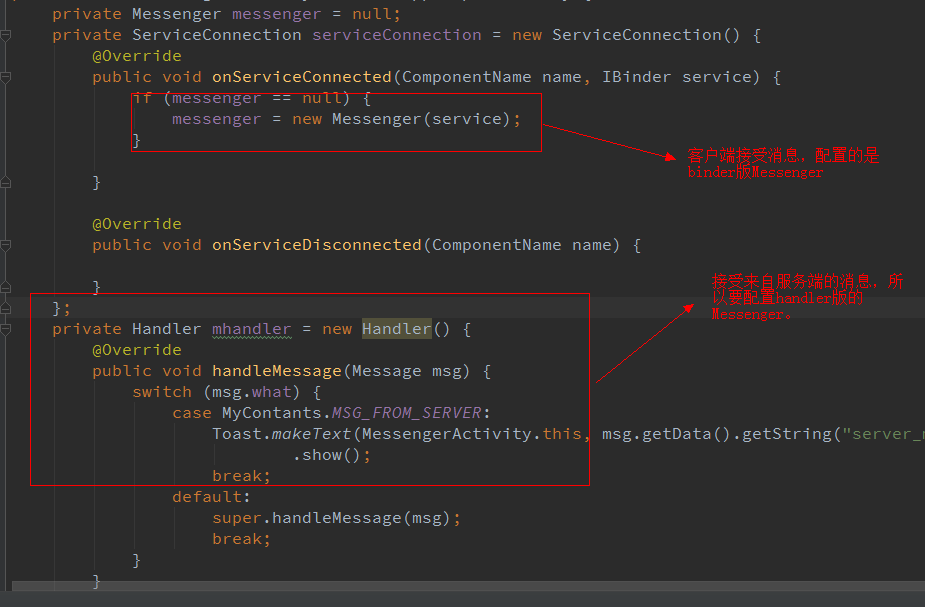
1. 使用Messenger方式。

Messenger有两钟构造方式，一个是handler，一个是binder。其实都是对AIDL的封装。

就结果来看，服务端我们一般使用handler构造，然后返回，这样服务端就能处理数据，通过handler来处理。在客户端我们用binder构造，然后通过send来发送信息。客户端选择binder，是因为服务端会返回binder，正好可以用现成的，然后发送消息。服务端能够接受消息，那么就是handler来处理了。

一个规律，那边接受处理消息，那么这端必须提供handler以及handler版的Messenger。而发送消息的这边，如果有binder，则有binder提供生成对应的Messenger，如果没有，则利用Messager的载体Message.replyTo来生成。服务端和客户端都必须有Messenger。





Message的获得，我们用线程池方法

