IBinder & Binder

参见：google-doc-IBinder

Android 中 ipc 的本质是 ibinder 和 binder

核心机制：

Ibinder 只提供了一个核心的方法：transact

service 端有一个 ibinder

当 client 和 service 处于同一个进程中时，client 可以找到 service 的 ibinder，并直接拿它来用

当 client 和 service 不在同一个进程中时，client 通过 bindservice 得到一个 ibinder，通过 ibinder.transact(int code, Parcel data, Parcel reply, int flags)来与 service 通信，即是 service 的 ibinder.transact 会被调用。但对于 IPC 来说，ibinder.transact 要做很多工作，但在各式各样的 service 端实现逻辑之上，却额外有一些通用的东西，将这些东西抽取出来放入 binder.transact 以复用整体流程（binder 是 ibinder 的子类），而提供 binder.ontransact 以让 service 端简洁的实现各自的核心逻辑（一个钩子，IOC），那么服务端的 ibinder 一般都继承 binder

---

Android 的 Binder 更多考虑了数据交换的便捷，并且只是解决本机的进程间的通讯

1. IBinder.transact(int code, Parcel data, Parcel reply, int flags) matched by Binder.onTransact(int code, Parcel data, Parcel reply, int flags). These methods allow you to send a call to an IBinder object and receive a call coming in to a Binder object, respectively. This transaction API is synchronous, such that a call to transact() does not return until the target has returned from Binder.onTransact();
2. The data sent through transact() is a Parcel, a generic buffer of data that also maintains some meta-data about its contents. The meta data is used to manage IBinder object references in the buffer, so that those references can be maintained as the buffer moves across processes.

   你可以用 parcel.obatin 获得一个新的对象

   在一个函数中，你先往 parcel 中写东西，然后，读这个 parcel，是读不出东西的；但是通过 parcel 传给另外一个 comp 时，却可以读出内容

   当你往 parcel 中按 int,string 的顺序写，但是却以 string,int 的顺序读，会出错的
3. The system maintains a pool of transaction threads in each process that it runs in. These threads are used to dispatch all IPCs coming in from other processes.

For example, when an IPC is made from process A to process B,

the calling thread in A blocks in IBinder.transact() as it sends the transaction to process B.

The next available pool thread in B receives the incoming transaction, calls Binder.onTransact() on the target object, and replies with the result Parcel.

Upon receiving its result, the thread in process A returns to allow its execution to continue. In effect, other processes appear to use as additional threads that you did not create executing in your own process.

Stub & proxy

开发者不应该看到 ibinder 和 binder，因为那是 java 端太底层的东西（在 c++端，还存在着 bpbinder 和 bbinder 等着你），开发者在乎的是 service 端的核心逻辑要如何实现。他会说：定义一个接口，我完成了它的实现。

但是对于 ipc 来说，开发者的核心逻辑的调用必须要经过 ontransact

client 更不应该看到 ibinder 和 binder，它们只知道：嘿，我知道我在进行一个远程调用，而且我知道开发者给我提供了一个接口和那个接口的一个对象，那么，我开始调用了

但是对于 ipc 来说，client 必须要通过 ibinder.transact 来开始调用

那么，所有的一切都明了了，client 和服务器端都需要代理：stub.proxy 和 stub

stub.proxy

client 要调用函数了，我必须将它转化为 ibinder.transact，刚好，client 已经将 ibinder 给我了

好的，给我了实参，我将把它们放入到 parcel 中

我知道 client 调用的方法的名字，将它放入到 code 中

OK，完成了，调用 ibinder.transact(int code, Parcel data, Parcel reply, int flags)

等等，为了让 client 看起来在使用开发者给他提供的那个接口，我必须要假装实现了那个接口

stub

我知道某个家伙在调用 transact，所以我将调用 ontransact

我知道他提供了哪些参数（parcel data），我也知道他想调用开发者提供的哪个方法（code）

我将调用那个方法，并将方法的返回值写入到 parcel reply 中

OK，完成了

等等，我必须要在 service.onbind 中返回，所以我必须继承 binder（不原始实现 ibinder 了）；

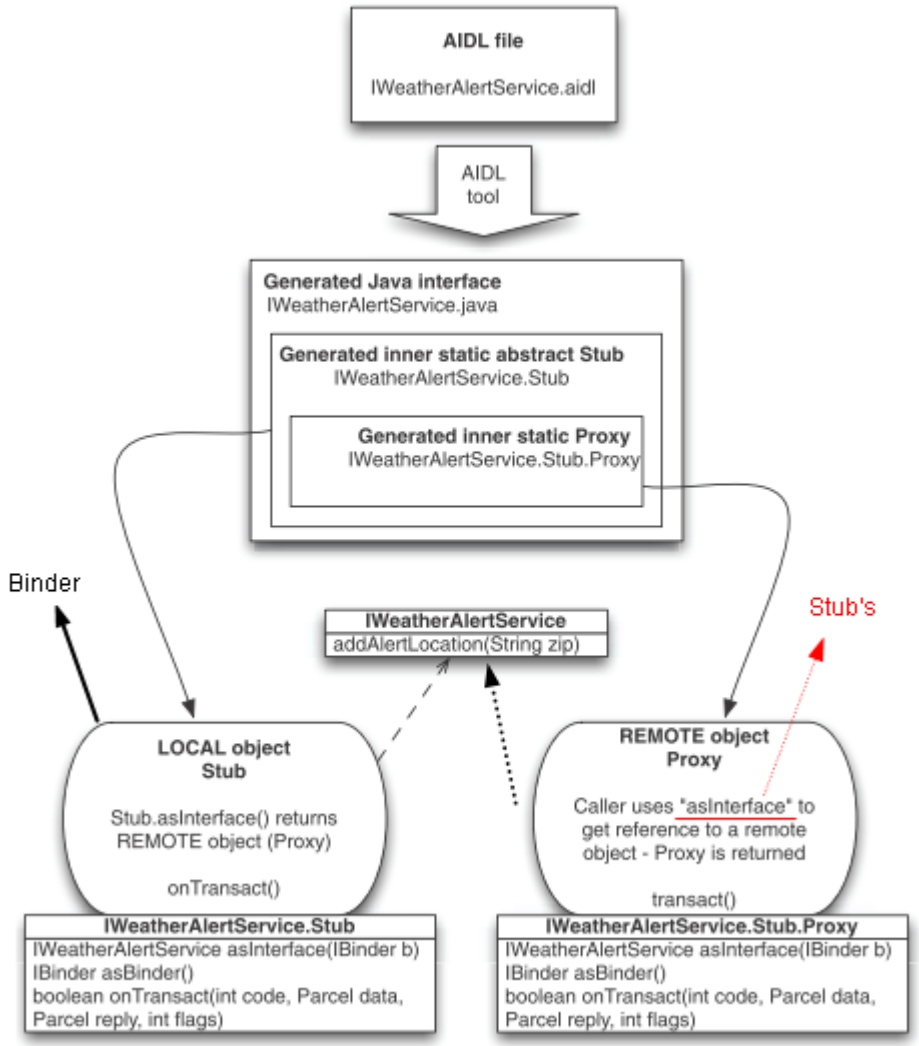
而我需要调用开发者的接口，那么我需要开发者接口的一个对象（采用组合的形式），或者，

我声称实现了开发者的接口，让开发者来继承我以填充他的接口实现（继承的方式，这也就是当前 android 采用的方式）

AIDL

AIDL (Android Interface Definition Language) is an IDL(interface definition language) language used to generate code that enables two processes on an Android-powered device to talk using IPC

通过 AIDL，android 帮助我们生成了 stub 和 proxy，而留给我们做的，只是完成 service 端的核心逻辑



Follow these steps to implement an IPC service using AIDL.

参见：ipc.rar，google-doc-developing-tools-aidl

1. **Create your .aidl file** - This file defines an interface (***YourInterface***.aidl) that defines the methods and fields available to a client.

2. **Add the .aidl file to your makefile** - (the ADT Plugin for Eclipse manages this for you).

The AIDL compiler(in the tools/ directory) creates an interface in the Java programming language from your AIDL file

这一个 interface 本身继承 iinterface

如果你想让 YourInterface.class 的所在包为 com.ql.rs，那么可以在 src 下建立相应的目录，并在该目录下编写 YourInterface.aidl

Stub class represents the local side of your remotable interface. Stub.asInterface(IBinder binder) method returns a remote version of your interface type - Proxy.

Proxy is used to wire up the plumbing. Client can use it to invoke remove methods

3. **Implement your interface methods** - You must create a class that extends ***YourInterface***.Stub and implements the methods you declared in your .aidl file.
4. **Expose your interface to clients** - If you're writing a service, you should extend Service and override Service.onBind(Intent) to return an instance of the class mentioned in step3.
5. **bind to the service**

过程

- 服务器端

YourService exntends Service

Onbind中，return一个YourInterface.Stub的实现类，猜想：作为local object

- Client 端

Context.bindService (Intent service, ServiceConnection conn, int flags)

ServiceConnection.onServiceConnected(ComponentName className, IBinder service)

该方法会在“main thread”中被调用中，而 service 中的 binder 的方法是在“bind 线程池的线程中”被调用

YourInterface interface = YourInterface.Stub.asInterface(service);

猜想：返回的这个对象是 YourInterface.Stub.Proxy，作为 remote object

至此，你能根据 interface 这个对象来操纵 local object 了（尽管它们在不同的 process 中）(pass objects between processes)，隐约的，你操控了这个 service

ServiceConnection.onServiceDisconnected(ComponentName className)中

This is called when the connection with the service has been unexpectedly disconnected -- that is, its process crashed.

说明：

ipc 的 method 中可以包含输入和返回参数，参数类型参见 google doc-developing-tools-aidl-create an .aidl file

service call back client

参见：google samples-*.app.RemoteServiceBinding*

client 和 service 建立了一条连接，client 通过 ipc 控制了 service，那么 service 有可能需要通知 client 某些信息，如：每一百 ms，service 将修改某些关键参数，然后通知 client 该参数。那么此刻，必须通过一种机制来完成这种更新：

方法 1：

//注册 client 提供的 callback

//当 client bindservice 后，client 通过 ipc 间接调用该函数，该函数最终发生在 service 端

//该 RemoteCallbackList 对象存在于 service 端，而 E 是一个 IInterface 的对象，该对象存在于 client 端，通过 ipc 的方式传递给 service，则该对象相对于 service 来说，是一个 remote

RemoteCallbackList.register(E callback)

RemoteCallbackList.unregister(E callback)

//取出 client 提供的 callback，调用 E 的方法，以完成对 client 的通知

RemoteCallbackList.beginBroadcast()

E RemoteCallbackList.getBroadcastItem(int index)

RemoteCallbackList.finishBroadcast()

//disable this callback list, and remove all the registered callbacks

RemoteCallbackList.kill()

方法 2：通过单纯的 binder 机制

在 transact 中，parcel.writestrongbinder，此后原 service 得到一个 ibinder，其可以作为 new client 来和原 client 进行交互

方法 3：service.aidl 中的方法包含了 client 提供的 iinterface 类型的参数

参见：service-client give a ipc

Local service binding

如果 service 与 client 在同一个 process 中，而 client 又想获得 service 的引用，那么 client 可以采用 Context.bindService (Intent service, ServiceConnection conn, int flags)，而 service 的 onBind 中应该返回一个 IBinder，那么 service 提供 Binder 的子类，且在该子类中提供诸如 getService 来返回 service 实例，而后 client 在 ServiceConnection.onserviceconnected 中通过 ibinder.getservice 来获取该实例