跨进程通信

# 1. 引言

本文主要总结了三种跨进程通信的方法，（本文所指的跨进程通信主要指两个App之间的通信交互，同一个App下的不同进程通信较前者稍有区别）：全局广播、跨进程启动Service和基于AIDL的远程Service。

# 2. 全局广播

## 2.1 说明

在Android 8.0的平台上，App不能对**大部分**的广播进行静态注册，也就是说，不能在AndroidManifest文件对有些广播进行静态注册，（开机广播等还可以静态注册）故本文主要使用动态注册来发送全局广播。

该方式仅仅适用于A App向B App传参数的场景，使用结束后要对广播进行解除注册。

## 2.2 具体步骤

**1. 在A app中定义权限，并且使用自定义权限。**

<uses-permission android:name="com.cn.customview.permissions.MY\_BROADCAST" />

<permission

android:name="com.cn.customview.permissions.MY\_BROADCAST"

android:protectionLevel="signature" >

</permission>

注意：android:protectionLevel=”signature” 如果是同一个应用可以这么声明，如果跨app发送广播（比如A app向B app发送广播），则需要去掉android:protectionLevel=”signature”。

**2. 在B app中定义注册广播。**

private static final String BROADCAST\_PERMISSION\_DISC = "com.cn.customview.permissions.MY\_BROADCAST";

private static final String BROADCAST\_ACTION\_DISC = "com.cn.customview.permissions.my\_broadcast";

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.broadcast\_permissions\_activity);

ButterKnife.bind(this);

// 注册广播接收

BroadcastReceiver receiveBroadCast = new ReceiveBroadCast();

IntentFilter filter = new IntentFilter();

filter.addAction(BROADCAST\_ACTION\_DISC); // 只有持有相同的action的接受者才能接收此广播

registerReceiver(receiveBroadCast, filter,BROADCAST\_PERMISSION\_DISC,null);

}

public class ReceiveBroadCast extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

Toast.makeText(BroadcastPermissionsActivity.this,

"receive broadcast", 0).show();

}

}

注册一个广播，并且申明，这个广播需要BROADCAST\_PERMISSION\_DISC权限才能收到消息。但是我们应用程序已经注册了这个权限。所以是有这个权限的。

同时在B app的AndroidManifest中声明权限：

<uses-permission android:name="com.cn.customview.permissions.MY\_BROADCAST" />

**3. 在A app中发送广播。**

public void sendBroadcastWithPermissions() {

Intent intent = new Intent(); //Itent就是我们要发送的内容

intent.putExtra("data", "this is data from broadcast ");

intent.setAction(BROADCAST\_ACTION\_DISC); //设置你这个广播的action，只有和这个action一样的接受者才能接受者才能接收广播

sendBroadcast(intent,BROADCAST\_PERMISSION\_DISC); //发送广播

}

# 3. 跨进程启动Service

## 3.1 说明

跨进程启动Service同前述的全局广播类似，其也适用于A App向B App传参数的场景。

跨应用启动Service需要注意的地方有2点:  
1. 目标service需要声明exported=true 的属性，表示允许其他应用访问该服务。  
2. Android5.0之前是可以通过设置隐式意图来跨应用打开Service的，5.0之后就必须要通过显示意图来开启Service。

## 3.2 具体步骤

**1. 创建APPService，并在AndroidManifest.xml中进行声明。**

public class AppService extends Service {

    public AppService() {

    }

    @Override

    public IBinder onBind(Intent intent) {

        return null;

    }

    @Override

    public void onCreate() {

        super.onCreate();

        Log.i("AppService", "onCreate");

    }

    @Override

    public int onStartCommand(Intent intent, int flags, int startId) {

        if (null != intent) {

            Log.i("AppService", "onStartCommand接收到的数据是:" + intent.getStringExtra("data"));

        }

        return super.onStartCommand(intent, flags, startId);

    }

    @Override

    public void onDestroy() {

        super.onDestroy();

        Log.i("AppService", "onDestroy");

    }

}

目标AppService 需要声明exported=true 的属性,表示允许其他应用访问该服务。

<service

            android:name=".AppService"

            android:enabled="true"

            android:exported="true" />

**2. 在App2Activity中启动AppService。**

通过Intent的setComponent方法，可以传递一个ComponentName对象，该对象有一个接受2个参数的构造方法，第一参数传递目标Service所在的包名，第二个参数传递目标Service的完整类名。

public class App2Activity extends AppCompatActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_app2);

        //注意:android 5.0之前是可以通过隐式意图打开其他app的服务的,5.0之后只能通过显式意图来打开.

        final Intent intent = new Intent();

        //ComponentName的参数1:目标app的包名,参数2:目标app的Service完整类名

        intent.setComponent(new ComponentName("com.example.mchenys.aidlservicedemo", "com.example.mchenys.aidlservicedemo.AppService"));

        //打开目标AppService

        findViewById(R.id.id\_btn\_start).setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                //设置要传送的数据

                intent.putExtra("data", "Hello AppService,I am App2Activity");

                startService(intent);

            }

        });

        //关闭目标AppService

        findViewById(R.id.id\_btn\_stop).setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                stopService(intent);

            }

        });

    }

}

## 3.3 注意

1. 该种方式需要保证被调用的Service已经被启动，否则无法进行调用。

2. 当被调用方启动Service时，若调用方调用startService ()，此时Service实例不会重复创建，而且当调用方调用stopService()后Service随即停止。

# 4. 基于AIDL的远程Service

## 4.1 说明

前述介绍的两种方法都只适用于A App向B App传参数的场景，很多时候需要跨进程调用服务端的方法，在这种业务场景下，前述的两种方法就不能很好的满足需求。本节介绍的基于AIDL的远程Service可以很好的满足这种需求，其可以使服务端和客户端进行跨进程交互。

## 4.2 Binder基础

在开始介绍基于AIDL的远程Service前首先介绍AIDL的基础Binder机制。Binder是Android中的一个类，其实现IBinder接口。从IPC的角度理解，Binder是Android中的一种跨进程通信的媒介，通过Binder对象，服务端和客户端可以完成通信交互。Binder在Android开发中主要应用于Service中，包括：AIDL和Messenger，而Messenger的底层实现就是ADIL，故本节通过分析AIDL来介绍Binder的工作机制。

首先新建一个AIDL示例，通过编译，SDK会自动生成对应的Binder类，由于代码过多，本节仅贴出部分重要代码。

public interface IBookManager extends android.os.IInterface

{

public static abstract class Stub extends android.os.Binder implements com.example.stan.aidlandobserver.IBookManager

{

private static final java.lang.String DESCRIPTOR = "com.example.stan.aidlandobserver.IBookManager";

public Stub()

{

this.attachInterface(this, DESCRIPTOR);

}

public static com.example.stan.aidlandobserver.IBookManager asInterface(android.os.IBinder obj)

{

android.os.IInterface iin = obj.queryLocalInterface(DESCRIPTOR);

if (((iin!=null)&&(iin instanceof com.example.stan.aidlandobserver.IBookManager))) {

return ((com.example.stan.aidlandobserver.IBookManager)iin);

}

return new com.example.stan.aidlandobserver.IBookManager.Stub.Proxy(obj);

}

@Override public android.os.IBinder asBinder()

{

return this;

}

@Override public boolean onTransact(int code, android.os.Parcel data, android.os.Parcel reply, int flags) throws android.os.RemoteException

{

switch (code)

{

case INTERFACE\_TRANSACTION:

{

reply.writeString(DESCRIPTOR);

return true;

}

case TRANSACTION\_getBookList:

{

data.enforceInterface(DESCRIPTOR);

java.util.List<com.example.stan.aidlandobserver.Book> \_result = this.getBookList();

reply.writeNoException();

reply.writeTypedList(\_result);

return true;

}

case TRANSACTION\_addBook:

{

data.enforceInterface(DESCRIPTOR);

com.example.stan.aidlandobserver.Book \_arg0;

if ((0!=data.readInt())) {

\_arg0 = com.example.stan.aidlandobserver.Book.CREATOR.createFromParcel(data);

}

else {

\_arg0 = null;

}

this.addBook(\_arg0);

reply.writeNoException();

return true;

}

}

return super.onTransact(code, data, reply, flags);

}

private static class Proxy implements com.example.stan.aidlandobserver.IBookManager

{

private android.os.IBinder mRemote;

Proxy(android.os.IBinder remote)

{

mRemote = remote;

}

@Override public android.os.IBinder asBinder()

{

return mRemote;

}

@Override public java.util.List<com.example.stan.aidlandobserver.Book> getBookList() throws android.os.RemoteException

{

android.os.Parcel \_data = android.os.Parcel.obtain();

android.os.Parcel \_reply = android.os.Parcel.obtain();

java.util.List<com.example.stan.aidlandobserver.Book> \_result;

try {

\_data.writeInterfaceToken(DESCRIPTOR);

mRemote.transact(Stub.TRANSACTION\_getBookList, \_data, \_reply, 0);

\_reply.readException();

\_result = \_reply.createTypedArrayList(com.example.stan.aidlandobserver.Book.CREATOR);

}

finally {

\_reply.recycle();

\_data.recycle();

}

return \_result;

}

@Override public void addBook(com.example.stan.aidlandobserver.Book book) throws android.os.RemoteException

{

android.os.Parcel \_data = android.os.Parcel.obtain();

android.os.Parcel \_reply = android.os.Parcel.obtain();

try {

\_data.writeInterfaceToken(DESCRIPTOR);

if ((book!=null)) {

\_data.writeInt(1);

book.writeToParcel(\_data, 0);

}

else {

\_data.writeInt(0);

}

mRemote.transact(Stub.TRANSACTION\_addBook, \_data, \_reply, 0);

\_reply.readException();

}

finally {

\_reply.recycle();

\_data.recycle();

}

}

}

}

从代码中可以看出，其声明了一个内部类Stub，其继承了Binder类，在Stub类的内部又声明了一个内部静态私有类Proxy。其中Stub类为Binder的本地对象，Proxy为Binder的代理对象（句柄）。当客户端和服务端位于同一个进程时，客户端获取到的就是Stub对象，而当客户端和服务端位于不同进程时，客户端获取到的就是Proxy对象。

下面介绍Stub类和Proxy类。

1. DESCRIPTOR

Binder的唯一标识，用于查询当前是本地调用还是跨进程调用。

2. asInterface()

用于将服务端的Binder对象转换成客户端所需要的ADIL接口类型的对象。

上述代码中android.os.IInterface iin = obj.queryLocalInterface(DESCRIPTOR);即为查询过程，若查询到的iin不为空，则说明客户端和服务端处于同一进程，则直接返回本地对象，否则返回远程代理对象。

3. asBinder()

该方法用于返回当前的Binder对象

4. onTransact()

该方法主要运行在服务端中的Binder线程池中，当客户端发起跨进程请求时，远程请求会通过系统底层封装后交由此方法来处理。服务端通过code参数来判断客户端请求的目标方法是什么。

5. Proxy#getBookList()

该方法运行在客户端，客户端就是通过该方法获取到返回值。

6. Proxy#addBook()

该方式同样运行在客户端，客户端通过该方法可以向服务端添加信息数据。

Binder的工作机制如下图所示：



## 4.3 具体步骤

**1. 服务端：**

首先在服务项目中建一个自定义BookManagerService类并继承Android的Service类，实现onBind()方法。然后在AndroidManifest文件处声明Service，并指定一个Action用于客户项目绑定这个Service。

新建AIDL文件(AIDL目录与java、res目录同级，文件在包名的最里面，例如com.example.stan.aidlandobserver.\*\*\*.aidl)，接口定义大致和class相同，配置好AIDL文件后编译一下让编译器自动生成对应的接口类 。

在MyService中实例化一个Stub接口类，并在onBind()方法中返回它。

@Override

public IBinder onBind(Intent intent) {

return mBinder;

}

**2. 客户端：**

客户端的配置较服务端简单许多，拷贝服务端的AIDL文件到当前项目中，包名目录也必须和服务端的包名目录一致。

Intent intent = new Intent(this, BookManagerService.class);

bindService(intent, mConnection, Context.BIND\_AUTO\_CREATE);

private ServiceConnection mConnection = new ServiceConnection() {

@Override

public void onServiceConnected(ComponentName name, IBinder service) {

IBookManager bookManager = IBookManager.Stub.asInterface(service);

//通过bookManager调用服务端的方法

}

@Override

public void onServiceDisconnected(ComponentName name) {

}

};

# 5. 待补充