**AIDL的作用**

    由于每个应用程序都运行在自己的进程空间，并且可以从应用程序UI运行另一个服务进程，而且经常会在不同的进程间传递对象。在Android平台，一个进程通常不能访问另一个进程的内存空间，所以要想对话，需要将对象分解成操作系统可以理解的基本单元，并且有序的通过进程边界。  
    通过代码来实现这个数据传输过程是冗长乏味的，Android提供了AIDL工具来处理这项工作。

    AIDL (Android Interface Definition Language) 是一种IDL 语言，用于生成可以在Android设备上两个进程之间进行进程间通信(interprocess communication, IPC)的代码。如果在一个进程中（例如Activity）要调用另一个进程中（例如Service）对象的操作，就可以使用AIDL生成可序列化的参数。  
    AIDL IPC机制是面向接口的，像COM或Corba一样，但是更加轻量级。它是使用代理类在客户端和实现端传递数据。

**选择AIDL的使用场合**

    官方文档特别提醒我们何时使用AIDL是必要的：**只有你允许客户端从不同的应用程序为了进程间的通信而去访问你的service，以及想在你的service处理多线程。**

    如果不需要进行不同应用程序间的并发通信(IPC)，you should create your interface by implementing a Binder；或者你想进行IPC，但不需要处理多线程的，则implement your interface using a Messenger。无论如何，在使用AIDL前，必须要理解如何绑定service——bindService。

    在设计AIDL接口前，要提醒的是，调用AIDL接口是直接的方法调用的，不是我们所想象的调用是发生在线程里。而调用(call)来自local进程或者remote进程，有什么区别呢？尤其是以下情况(引用原文，不作翻译了，以免翻译有误)：

* Calls made from the local process are executed in the same thread that is making the call. If this is your main UI thread, that thread continues to execute in the AIDL interface. If it is another thread, that is the one that executes your code in the service. Thus, if only local threads are accessing the service, you can completely control which threads are executing in it (but if that is the case, then you shouldn't be using AIDL at all, but should instead create the interface by implementing a Binder).
* Calls from a remote process are dispatched from a thread pool the platform maintains inside of your own process. You must be prepared for incoming calls from unknown threads, with multiple calls happening at the same time. In other words, an implementation of an AIDL interface must be completely thread-safe.
* The oneway keyword modifies the behavior of remote calls. When used, a remote call does not block; it simply sends the transaction data and immediately returns. The implementation of the interface eventually receives this as a regular call from the Binder thread pool as a normal remote call. If oneway is used with a local call, there is no impact and the call is still synchronous.

**定义AIDL接口**

    AIDL接口文件，和普通的接口内容没有什么特别，只是它的扩展名为.aidl。保存在src目录下。如果其他应用程序需要IPC，则那些应用程序的src也要带有这个文件。Android SDK tools就会在gen目录自动生成一个IBinder接口文件。service必须适当地实现这个IBinder接口。那么客户端程序就能绑定这个service并在IPC时从IBinder调用方法。

    每个aidl文件只能定义一个接口，而且只能是接口的声明和方法的声明。

**1.创建.aidl文件**  
     AIDL使用简单的语法来声明接口，描述其方法以及方法的参数和返回值。这些参数和返回值可以是任何类型，甚至是其他AIDL生成的接口。

    其中对于Java编程语言的基本数据类型 (int, long, char, boolean等),String和CharSequence，集合接口类型List和Map，不需要import 语句。

    而如果需要在AIDL中使用其他AIDL接口类型，需要import，即使是在相同包结构下。AIDL允许传递实现Parcelable接口的类，需要import.  
    需要特别注意的是，对于非基本数据类型，也不是String和CharSequence类型的，需要有方向指示，包括in、out和inout，in表示由客户端设置，out表示由服务端设置，inout是两者均可设置。

    AIDL只支持接口方法，不能公开static变量。

例如 (IMyService.aidl)：

package com.demo;   
  
import com.demo.Person;   
  
interface IMyService {   
        void savePersonInfo(in Person person);   
        List<Person> getAllPerson();   
}

**2.实现接口**

    创建一个类实现刚才那个aidl的接口：

public class RemoteService extends Service {   
  
        private LinkedList<Person> personList = new LinkedList<Person>();   
           
        @Override   
        public IBinder onBind(Intent intent) {   
                return mBinder;   
        }   
  
        private final IMyService.Stub mBinder = new IMyService.Stub(){   
  
                @Override   
                public void savePersonInfo(Person person) throws RemoteException {   
                        if (person != null){   
                                personList.add(person);   
                        }   
                }   
  
                @Override   
                public List<Person> getAllPerson() throws RemoteException {   
                        return personList;   
                }   
        };   
}

    这里会看到有一个名为IMyService.Stub类，查看aidl文件生成的Java文件源代码就能发现有这么一段代码：

/\*\* Local-side IPC implementation stub class. \*/   
public static abstract class Stub extends android.os.Binder implements com.demo.IMyService

    原来Stub类就是继承于Binder类，也就是说RemoteService类和普通的Service类没什么不同，只是所返回的IBinder对象比较特别，是一个实现了AIDL接口的Binder。

    接下来就是关于所传递的数据Bean——Person类，是一个序列化的类，这里使用Parcelable 接口来序列化，是Android提供的一个比Serializable 效率更高的序列化类。

    Parcelable需要实现三个函数：  
    1) **void writeToParcel(Parcel dest, int flags)**将需要序列化存储的数据写入外部提供的Parcel对象dest。而看了网上的代码例子，个人猜测，读取Parcel数据的次序要和这里的write次序一致，否则可能会读错数据。具体情况我没试验过！  
    2) **describeContents()** 没搞懂有什么用，反正直接返回0也可以  
    3) **static final Parcelable.Creator对象CREATOR**  这个CREATOR命名是固定的，而它对应的接口有两个方法：  
    createFromParcel(Parcel source) 实现从source创建出JavaBean实例的功能  
  
    newArray(int size) 创建一个类型为T，长度为size的数组，仅一句话（return new T[size])即可。估计本方法是供外部类反序列化本类数组使用。

仔细观察Person类的代码和上面所说的内容：

public class Person implements Parcelable {   
  
        private String name;   
        private String telNumber;   
        private int age;   
  
        public Person() {}   
  
        public Person(Parcel pl){   
                name = pl.readString();   
                telNumber = pl.readString();   
                age = pl.readInt();   
        }   
  
        public String getName() {   
                return name;   
        }   
  
        public void setName(String name) {   
                this.name = name;   
        }   
  
        public String getTelNumber() {   
                return telNumber;   
        }   
  
        public void setTelNumber(String telNumber) {   
                this.telNumber = telNumber;   
        }   
  
        public int getAge() {   
                return age;   
        }   
  
        public void setAge(int age) {   
                this.age = age;   
        }   
  
        @Override   
        public int describeContents() {   
                return 0;   
        }   
  
        @Override   
        public void writeToParcel(Parcel dest, int flags) {   
                dest.writeString(name);   
                dest.writeString(telNumber);   
                dest.writeInt(age);   
        }   
  
        public static final Parcelable.Creator<Person> CREATOR = new Parcelable.Creator<Person>() {   
  
                @Override   
                public Person createFromParcel(Parcel source) {   
                        return new Person(source);   
                }   
  
                @Override   
                public Person[] newArray(int size) {   
                        return new Person[size];   
                }   
  
        };   
}

然后创建Person.aidl文件，注意这里的parcelable和原来实现的Parcelable 接口，开头的字母p一个小写一个大写：

package com.demo;   
  
parcelable Person;

     对于实现AIDL接口，官方还提醒我们：

    1. 调用者是不能保证在主线程执行的，所以从一调用的开始就需要考虑多线程处理，以及确保线程安全；

    2. IPC调用是同步的。如果你知道一个IPC服务需要超过几毫秒的时间才能完成地话，你应该避免在Activity的主线程中调用。也就是IPC调用会挂起应用程序导致界面失去响应，这种情况应该考虑单独开启一个线程来处理。  
    3. 抛出的异常是不能返回给调用者（跨进程抛异常处理是不可取的）。  
   
**3. 客户端获取接口**

    客户端如何获取AIDL接口呢？通过IMyService.Stub.asInterface(service)来得到IMyService对象：

private IMyService mRemoteService;   
  
private ServiceConnection mRemoteConnection = new ServiceConnection() {      
        public void onServiceConnected(ComponentName className, IBinder service) {      
                mRemoteService = IMyService.Stub.asInterface(service);      
        }      
  
        public void onServiceDisconnected(ComponentName className) {      
                mRemoteService = null;      
        }      
};

 在生成的IMyService.java里面会找到这样的代码：

/\*\*   
\* Cast an IBinder object into an com.demo.IMyService interface,   
\* generating a proxy if needed.   
\*/   
public static com.demo.IMyService asInterface(android.os.IBinder obj) {...}

而service的绑定没有什么不同：

if (mIsRemoteBound) {   
        unbindService(mRemoteConnection);   
}else{   
        bindService(new Intent("com.demo.IMyService"),   
                               mRemoteConnection, Context.BIND\_AUTO\_CREATE);   
}   
  
mIsRemoteBound = !mIsRemoteBound;

**通过IPC调用/传递数据**

    客户端绑定service后就能通过IPC来调用/传递数据了，直接调用service对象的接口方法：

addPersonButton.setOnClickListener(   
                new View.OnClickListener(){   
                        private int index = 0;   
  
                        @Override   
                        public void onClick(View view) {   
                                Person person = new Person();   
                                index = index + 1;   
                                person.setName("Person" + index);   
                                person.setAge(20);   
                                person.setTelNumber("123456");   
                                try {   
                                        mRemoteService.savePersonInfo(person);   
                                } catch (RemoteException e) {   
                                        e.printStackTrace();   
                                }   
                        }   
                });   
  
listPersonButton.setOnClickListener(   
                new View.OnClickListener(){   
  
                        @Override   
                        public void onClick(View view) {   
                                List<Person> list = null;   
  
                                try {   
                                        list = mRemoteService.getAllPerson();   
                                } catch (RemoteException e) {   
                                        e.printStackTrace();   
                                }   
  
                                if (list != null){   
                                        StringBuilder text = new StringBuilder();   
  
                                        for(Person person : list){   
                                                text.append("\nPerson name:");   
                                                text.append(person.getName());   
                                                text.append("\n             age :");   
                                                text.append(person.getAge());   
                                                text.append("\n tel number:");   
                                                text.append(person.getTelNumber());   
                                        }   
  
                                        inputPersonEdit.setText(text);   
                                }else {   
                                        Toast.makeText(ServiceActivity.this, "get data error",   
                                                        Toast.LENGTH\_SHORT).show();   
                                }   
                        }   
                });

 **Permission权限**  
    如果Service在AndroidManifest.xml中声明了全局的强制的访问权限，其他引用必须声明权限才能来start，stop或bind这个service.  
     另外，service可以通过权限来保护她的IPC方法调用，通过调用checkCallingPermission(String)方法来确保可以执行这个操作。

 **AndroidManifest.xml的Service元素**

<service android:name=".RemoteService" android:process=":remote">   
        <intent-filter>   
                <action android:name="com.demo.IMyService" />   
        </intent-filter>   
</service>

    这里的android:process=":remote"，一开始我没有添加的，在同一个程序里使用IPC，即同一个程序作为客户端/服务器端，结果运行mRemoteService = IMyService.Stub.asInterface(service);时提示空指针异常。观察了人家的在不同程序里进行IPC的代码，也是没有这个android:process=":remote"的。后来在官方文档<http://androidappdocs.appspot.com/guide/topics/manifest/service-element.html>里了解到（留意第二段文字）：

android:process  
The name of the process where the service is to run. Normally, all components of an application run in the default process created for the application. It has the same name as the application package. The <application> element's process attribute can set a different default for all components. But component can override the default with its own process attribute, allowing you to spread your application across multiple processes.

**If the name assigned to this attribute begins with a colon (':'), a new process, private to the application, is created when it's needed and the service runs in that process**. If the process name begins with a lowercase character, the service will run in a global process of that name, provided that it has permission to do so. This allows components in different applications to share a process, reducing resource usage.

 也就是说android:process=":remote"，代表在应用程序里，当需要该service时，会自动创建新的进程。而如果是android:process="remote"，没有“:”分号的，则创建全局进程，不同的应用程序共享该进程。