**具体实现方式简述：**

**一、主要包含三部分**

1. aidl文件

* 新建 aidl 文件夹，在其中创建接口 aidl 文件以及实体类的映射 aidl 文件
* 如果需要访问自定义对象，创建要操作的实体类，实现 Parcelable 接口
* 创建接口aidl文件，定义方法
* 客户端和服务端相同的aidl文件

1. 服务端

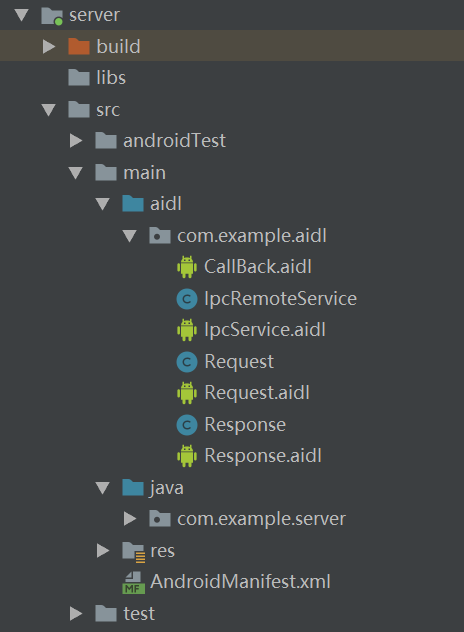
* 创建 Service，在service中创建上面生成的 Binder 对象实例，实现aidl接口定义的方法，处理客户端的请求；在onbind方法将binder返回回去
* 在AndroidManifest配置service，将service暴露出去

1. 客户端

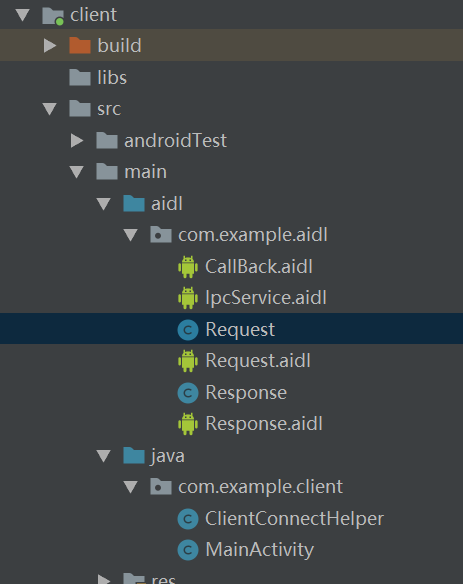
* 实现 ServiceConnection 接口，并在其中拿到aidl类
* 绑定服务
* 通过aidl类调用其中的方法

1. **目录结构图**:

服务端：



客户端:



1. **实现方法:**

**服务端:**

**1、**服务端创建aidl folde,与java文件夹同级目录,创建Request.aidl文件

package com.example.aidl;  
  
  
parcelable Request;

创建Response.aidl文件

package com.example.aidl;  
  
  
  
parcelable Response;

1. 创建aidl接口文件IpcService.aidl文件定义一个请求接口。注意这里要引入自定义类(例如Request),自定义类的包名一定要和aidl文件的包名相同。

package com.example.aidl;  
  
// Declare any non-default types here with import statements  
import com.example.aidl.Request;  
import com.example.aidl.Response;  
import com.example.aidl.CallBack;  
  
interface IpcService{  
 Response send(in Request request);  
 void register(CallBack callback);  
 void unregister(CallBack callback);  
}

1. 回调机制。主要是客户端向服务端发送数据的同时需要服务端返回数据给客户端。首先在服务端创建CallBack.aidl文件。并同时在IpcService.aidl文件中添加注册回调方法和解除回调方法（如上图）。

package com.example.aidl;  
  
// Declare any non-default types here with import statements  
  
interface CallBack{  
 void onSuccess(String result);  
  
 void onFailed(String errorMsg);  
}

4、创建service来响应客户端的请求。创建IpcRemoteService.java文件。创建Binder对象，并在onBind()方法返回回去。实现aidl接口文件里的请求方法，根据request.msgType实现不同的业务逻辑。

class RemoteBinder extends IpcService.Stub {  
  
 @Override  
 public Response send(Request request) throws RemoteException {  
 String msgContent = request.msgContent;  
 Response response = new Response();  
 //这里根据msgType处理不同的业务逻辑  
 switch (request.msgType) {  
 case Request.*MSG\_TYPE\_0*:  
 Log.*d*(*TAG*, msgContent);  
 response.setResult("return success");  
 //有返回的可以不用这个回调  
 dispatchResult(true, msgContent);  
 break;  
 case Request.*MSG\_TYPE\_1*:  
  
 break;  
 case Request.*MSG\_TYPE\_2*:  
  
 break;  
 case Request.*MSG\_TYPE\_3*:  
  
 break;  
 case Request.*MSG\_TYPE\_4*:  
  
 break;  
 default:  
 break;  
 }  
  
  
 return response;  
 }

这里需要注意的是如果你的java文件放在src/main/java目录下,则你的java目录下的包名必须和aidl目录下的包名一致,如果你的java文件放在src/main/aidl目录下,则必须在app的build.gradle文件里面加入如下代码：

sourceSets {  
 *main* {  
 java.srcDirs = ["src/main/java", "src/main/aidl"]  
 }  
}

1. 权限验证。这里通过包名来做权限验证，主要是在服务里Binder里重写onTransact()方法,如果包名不一致返回false,客户端请求失败。

@Override  
public boolean onTransact(int code, Parcel data, Parcel reply, int flags) throws RemoteException {  
 String packageName = null;  
 String[] packages = IpcRemoteService.this.getPackageManager().  
 getPackagesForUid(*getCallingUid*());  
 if (packages != null && packages.length > 0) {  
 packageName = packages[0];  
 }  
 Log.*d*(*TAG*, "onTransact: " + packageName);  
 if (!*PACKAGE\_SAYHI*.equals(packageName)) {  
 return false;  
 }  
  
 return super.onTransact(code, data, reply, flags);  
}

1. 注册服务，在AndroidManifest.xml里面公开服务，添加action，以供客户端调用。

**客户端：**

1. 首先将服务端aidl文件夹下的aidl文件复制一份放到客户端目录下。Response.java和Reponse.java也可以复制过去。
2. 通过bindService启动远程服务。

public void bindService() {  
 Intent serviceIntent = new Intent();  
 serviceIntent.setAction(*BIND\_SERVICE\_ACTION*);  
 final Intent mIntent = new Intent(achieveExplicitFromImplicitIntent(mContext, serviceIntent));  
 isBind = mContext.bindService(mIntent, mConnection, Context.*BIND\_AUTO\_CREATE*);  
 Log.*d*(*TAG*, "bindService");  
}  
  
public Intent achieveExplicitFromImplicitIntent(Context context, Intent implicitIntent) {  
 PackageManager pm = context.getPackageManager();  
 List<ResolveInfo> resolveInfo = pm.queryIntentServices(implicitIntent, 0);  
  
 if (resolveInfo == null || resolveInfo.size() != 1) {  
 return null;  
 }  
  
 ResolveInfo serviceInfo = resolveInfo.get(0);  
 String packageName = serviceInfo.serviceInfo.packageName;  
 String className = serviceInfo.serviceInfo.name;  
 Log.*d*(*TAG*, "packageName = " + packageName);  
 Log.*d*(*TAG*, "className = " + className);  
 ComponentName component = new ComponentName(packageName, className);  
  
 Intent explicitIntent = new Intent(implicitIntent);  
  
 explicitIntent.setComponent(component);  
 return explicitIntent;  
}

这里主要是通过ServiceConnection来监控远程服务的状态的。服务连接成功会触发onServiceConnected方法，服务断开连接或者崩溃会触发onServiceDisconnected方法，我们在这里重新绑定服务。

private ServiceConnection mConnection = new ServiceConnection() {  
 @Override  
 public void onServiceConnected(ComponentName name, IBinder service) {  
 ipcService = IpcService.Stub.*asInterface*(service);  
 Log.*d*(*TAG*, "onServiceConnected");  
 if (ipcService == null) {  
 Log.*i*(*TAG*, "mStudentService == null");  
 return;  
 }  
 try {  
 //设置死亡代理  
 ipcService.asBinder().linkToDeath(mDeathRecipient, 0);  
  
 if (mCallback != null) {  
 Log.*i*(*TAG*, "mCallback != null");  
 ipcService.register(mCallback);  
 } else {  
 Log.*i*(*TAG*, "mCallback == null");  
 }  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 public void onServiceDisconnected(ComponentName name) {  
 Log.*i*(*TAG*, "client onServiceDisconnected");  
 ipcService = null;  
 // 重新绑定服务  
 bindService();  
 }  
  
};

1. 在服务连接的时候为Binder设置死亡代理。当Binder死亡的时候会触发binderDied方法，在这里同样需要重新绑定服务。保证服务端和客户端一直保持连接。

private IBinder.DeathRecipient mDeathRecipient = new IBinder.DeathRecipient() {  
  
 @Override  
 public void binderDied() {  
 if (ipcService == null) {  
 return;  
 }  
 //解除死亡代理  
 ipcService.asBinder().unlinkToDeath(mDeathRecipient, 0);  
 ipcService = null;  
 //重新绑定服务  
 bindService();  
 Log.*i*(*TAG*, "binderDied, bindService again");  
 }  
};

1. 实现aidl接口文件方法。绑定服务后拿到service调用请求方法发送请求。

public Response sendRequest(Request request) {  
 Response response = null;  
 if (ipcService != null) {  
 try {  
 response = ipcService.send(request);  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 }  
 }  
 return response;  
}

1. 解绑服务。需要的时候调用。

public void unBindSevice() {  
 if (isBind) {  
 mContext.unbindService(mConnection);  
 isBind = false;  
 ipcService = null;  
 Log.*d*(*TAG*, "unbindService");  
 }  
}