Android Background Processing – Services and IPC

Android AIDL

Overview

AIDL- Android Interface Definition Language

- Works like client & server application.
- Client app initiate some action, pass some data to server app for process, Server app process the data passed by client and return the appropriate result.
- Possible to pass custom object across apps.

Simple data passing

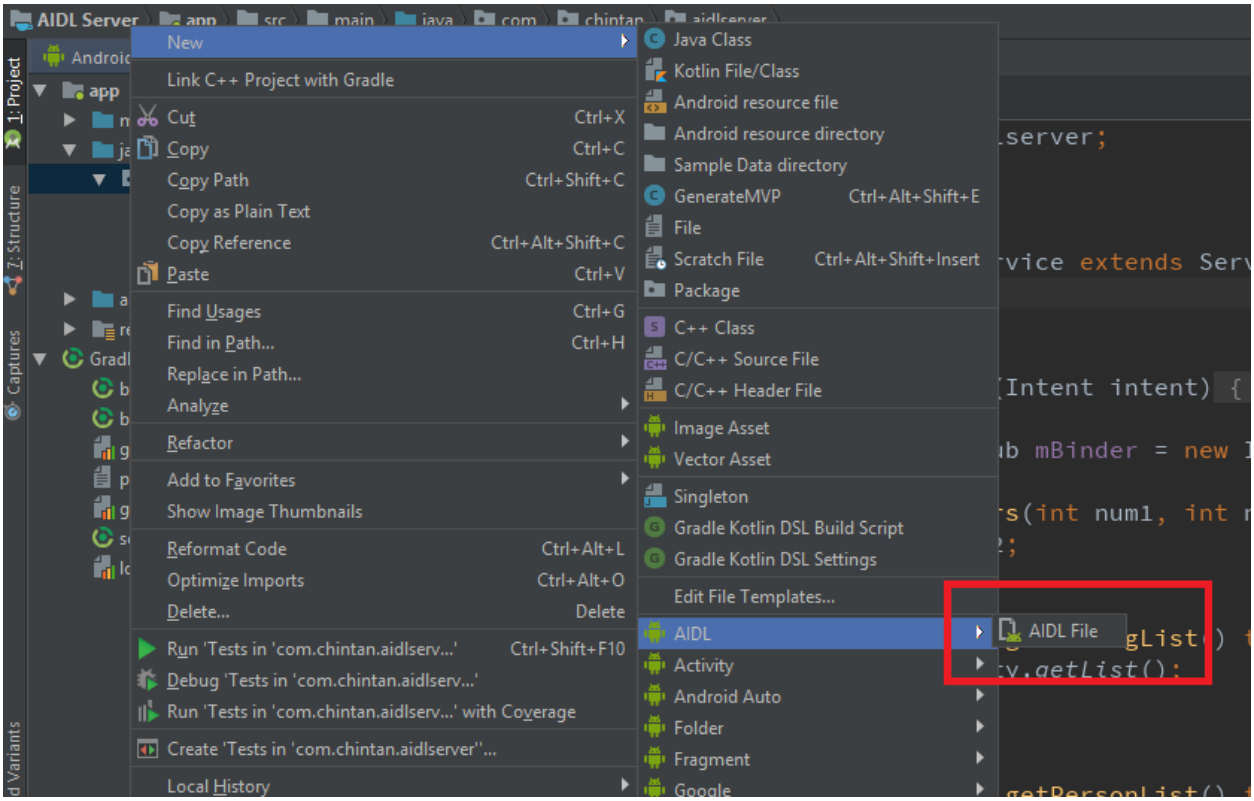
- You can send any primitive data across client and server app without any other hassle.
- If you want to send any user defined type across apps than you have to do extra work

To make AIDL possible we have to make 2 apps from which one will act as server and another as client which will access remote methods on server app.

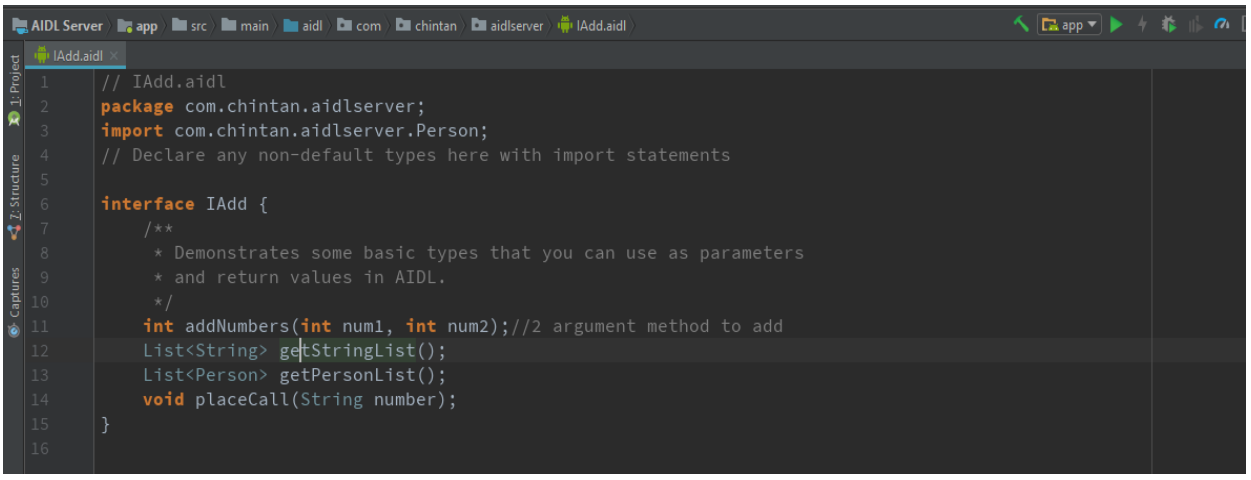
Getting Started with server app

- Create one server project [Normal project which acts as serving to another app].
- Add empty activity which act as launching activity.

- Add one aidl file

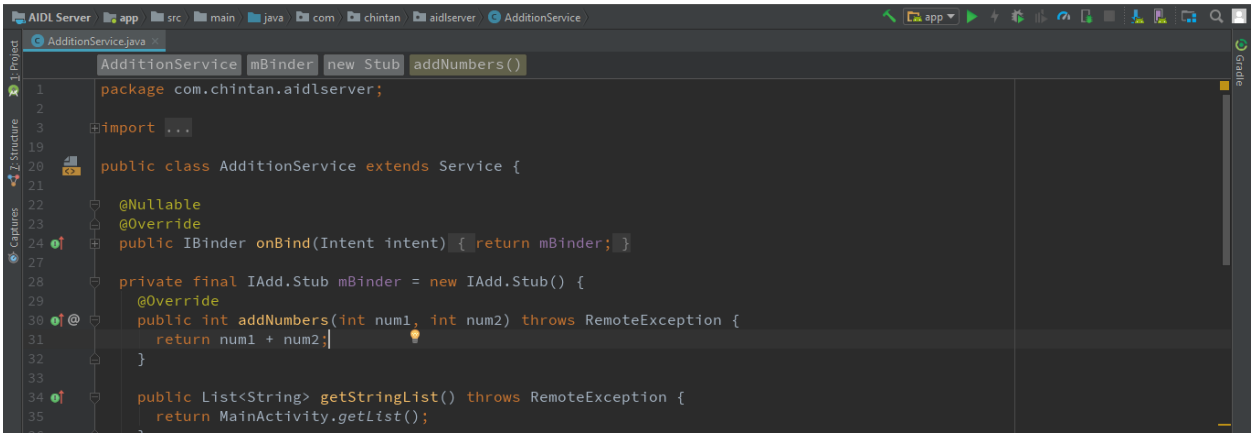


- AIDL file defines programming interface with method signature. AIDL interface should be defined in a ".aidl" file



Here we have 2 methods one is **addNumbers** with 2 arguments which will return int data and List with string type with no argument. Both methods will be implemented in class **AdditionService.java** class.

- Make one java file in this case it is **AdditionService.java** which extends Service class and implement your interface with your method implementation. Here Stub is an automatically generated class by android for you and you might need to rebuild project several times if needed.

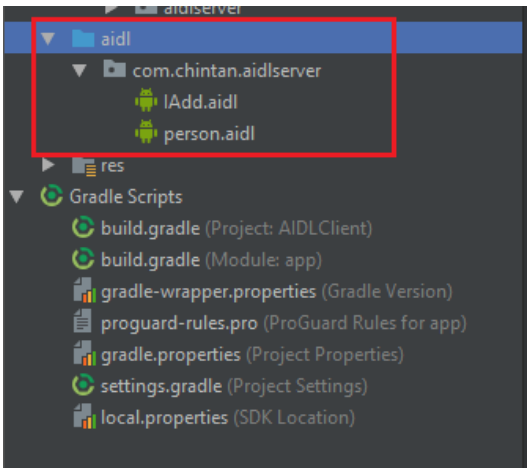


- **AndroidManifest.xml** File

You must declare your service in manifest file and make process to **.remote** and give unique name to action in intent-filter through which your client application will call your service.

Creating Client app

- Create your basic UI with number input for addition and displaying result.
- Make one package with same name as your server app's package name in which your aidl file resides and in that put same ".aidl" files of server.



- Make object of your interface in this case IAdd and ServiceConnection class.

```
private EditText num1, num2;
private Button btnAdd, btnNonPremitive, btnCall;
private TextView total;
protected IAdd addService;
private String Tag = "Client Application";
private String serverAppUri = "com.chintan.aidlserver";
```

```
private ServiceConnection serviceConnection = new ServiceConnection() {
  @Override
  public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
    Log.d(Tag, msg: "Service Connected");
    addService = IAdd.Stub.asInterface((IBinder) iBinder);
  }

  @Override
  public void onServiceDisconnected(ComponentName componentName) {
    Log.d(Tag, msg: "Service Disconnected");
    addService = null;
  }
};
```

- Make one function in MainActivity, initConnection()

```
private void initConnection() {
  if (addService == null) {
    Intent intent = new Intent(IAdd.class.getName());

          /*this is service name*/
    intent.setAction("service.calc");

          /*From 5.0 annonymous intent calls are suspended so replacing with server app's package name*/
    intent.setPackage("com.chintan.aidlserver");

    // binding to remote service
    bindService(intent, serviceConnection, Service.BIND_AUTO_CREATE);
  }
}
```

- That's it you just have to perform calling function like any other function.

```
if (num1.length() > 0 && num2.length() > 0 && addService != null) {
  try {
    total.setText("");
    total.setText("Result: " + addService
        .addNumbers(Integer.parseInt(num1.getText().toString()),
            Integer.parseInt(num2.getText().toString())));
```

- Here we have call addNumbers function of our ".aidl" file.

This procedure is for simple and primitive types.

User defined object passing

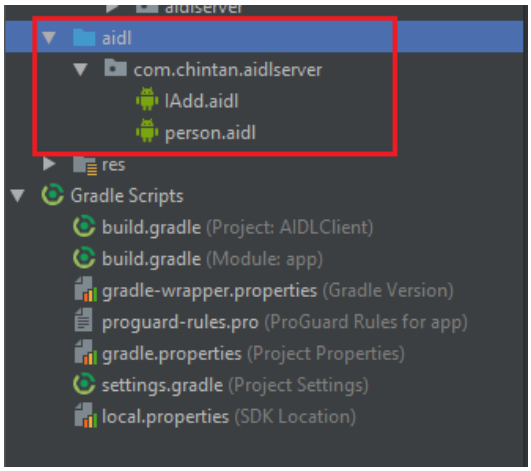
- Android AIDL also supports user defined object passing.
- For passing user defined objects you have to make some changes in both client and server app.

Server App changes

- Make your model class and implements it with **Parcelable**.
- The **Parcelable** interface have one method Creator<T> this method you need to override. Maybe it is possible Android studio will do the job for you.

```
public static final Creator<Person> CREATOR = new Creator<Person>() {
    @Override
    public Person createFromParcel(Parcel in) { return new Person(in); }

    @Override
    public Person[] newArray(int size) { return new Person[size]; }
};

@Override
public int describeContents() { return 0; }

@Override
public void writeToParcel(Parcel parcel, int i) {
    parcel.writeString(name);
    parcel.writeInt(age);
}
```

- Make another ".aidl" file with same name as class name and in the same package where other ".aidl" files resides.



- **DO NOT WRITE ANYTHING EXCEPT FOLLOWING LINE in custom class ".aidl"**
  parcelable "Your class name";

person.aidl

```
// personAIDL.aidl
package com.chintan.aidlserver;
parcelable Person;
```

- Now come to your main ".aidl" file and do the following change.
  - Write import statement for your custom class
    Import "package name.classname"
- At last insert your function name which connects to your custom class in interface.

IAdd.aidl

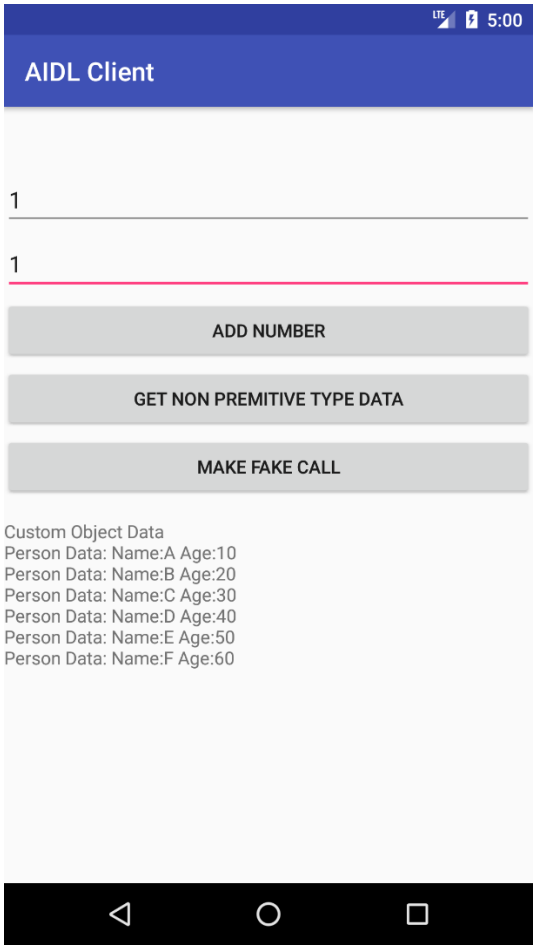
```
// IAdd.aidl
package com.chintan.aidlserver;
import com.chintan.aidlserver.Person;
// Declare any non-default types here with import statements

interface IAdd {
    /**
     * Demonstrates some basic types that you can use as parameters
     * and return values in AIDL.
     */
    int addNumbers(int num1, int num2);//2 argument method to add
    List<String> getStringList();
    List<Person> getPersonList();
    void placeCall(String number);
}
```

Client App changes

- Make exact same copy of the model class.
- Copy same custom class's ".aidl" file to client's aidl folder without any change.

- Do the same import statement for your client's main ".aidl" as server.
- Make the appropriate call to get your data.

- Method to run
  - First Install server application in your phone or emulator.
  - Install client application in your phone or emulator.
  - Output :



- **References**
  - Official Documentation
  - Simple Data passing (primitive data type)
  - Complex Data Passing (User Defined type)