

2021

第7章 后台服务

复旦大学 陈辰



学习目标

AIMS

- 01 了解Service的原理和用途
- 02 掌握本地服务的管理方法
- 03 掌握服务的显式启动方法
- 04 了解线程的启动、挂起和停止方法
- 05 了解跨线程的界面更新方法
- 06 掌握远程服务的绑定和调用方法
- 07 了解AIDL语言的用途和语法

7.1 Service简介

•Service

- Android系统的服务组件，适用于开发没有用户界面且长时间在后台运行的应用功能
- 因为手机硬件性能和屏幕尺寸的限制，通常Android系统仅允许一个应用程序处于激活状态并显示在手机屏幕上，而暂停其他处于未激活状态的程序
- Android系统需要一种后台服务机制
 - 没有用户界面
 - 能够长时间在后台运行
 - 实现应用程序的后台服务功能
- 例子：MP3播放器
 - 使用Service组件中实现无界面音乐回放功能

7.1 Service简介

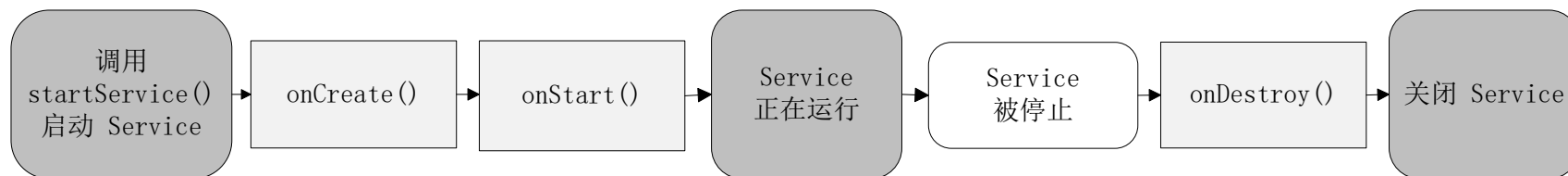
•Service的优势

- 没有用户界面，更加有利于降低系统资源的消耗
- Service比Activity具有更高的优先级，因此在系统资源紧张时，Service不会被Android系统优先终止
- 即使Service被系统终止，在系统资源恢复后Service也将自动恢复运行状态，可以认为Service是在系统中永久运行的组件
- Service除了可以实现后台服务功能，还可以用于进程间通信（Inter Process Communication，IPC），解决不同Android应用程序进程之间的调用和通讯问题

7.1 Service简介

•Service生命周期

- onCreate()函数
 - Service的生命周期开始，完成Service的初始化工作
- onStart() 函数
 - 启动线程
- onDestroy() 函数
 - Service的生命周期结束，释放Service所有占用的资源



7.1

Service简介

•Service生命周期

- Service生命周期包括
 - 完整生命周期从onCreate()开始到onDestroy()结束，在onCreate()中完成Service的初始化工作，在onDestroy()中释放所有占用的资源
 - 活动生命周期从onStart()开始，但没有与之对应的“停止”函数，因此可以粗略的认为活动生命周期是以onDestroy()标志结束
- Service的使用方式一般有两种
 - 启动方式
 - 绑定方式

7.1

Service简介

• 启动方式

- 通过调用Context.startService()启动Service，通过调用Context.stopService()或Service.stopSelf()停止Service。因此，Service一定是由其它的组件启动的，但停止过程可以通过其它组件或自身完成
- 在启动方式中，启动Service的组件不能够获取到Service的对象实例，因此无法调用Service中的任何函数，也不能够获取到Service中的任何状态和数据信息
- 能够以启动方式使用的Service，需具备自我管理的能力，而且不需要从通过函数调用获取Service的功能和数据

7.1

Service简介

• 绑定方式

- Service的使用是通过服务链接（Connection）实现的，服务链接能够获取Service的对象实例，因此绑定Service的组件可以调用Service中实现的函数，或直接获取Service中的状态和数据信息
- 使用Service的组件通过Context.bindService()建立服务链接，通过Context.unbindService()停止服务链接
- 如果在绑定过程中Service没有启动，Context.bindService()会自动启动Service，而且同一个Service可以绑定多个服务链接，这样可以同时为多个不同的组件提供服务

7.1 Service简介

• 启动方式和绑定方式的结合

- 这两种使用方法并不是完全独立的，在某些情况下可以混合使用
 - 以MP3播放器为例，在后台工作的Service通过Context.startService()启动某个音乐播放，但在播放过程中如果用户需要暂停音乐播放，则需要通过Context.bindService()获取服务链接和Service对象实例，进而通过调用Service对象实例中的函数暂停音乐播放过程，并保存相关信息
 - 在这种情况下，如果调用Context.stopService()并不能够停止Service，需要在所有的服务链接关闭后，Service才能够真正的停止

7.2 本地服务

- 本地服务的调用者和服务都在同一个程序中，是不需要跨进程就可以实现服务的调用
- 本地服务涉及服务的建立、启动和停止，服务的绑定和取消绑定，以及如何在线程中实现服务
- **7.2.1 服务管理**
 - 服务管理主要指服务的启动和停止
 - 首先说明如何在代码中实现Service。Service是一段在后台运行、没有用户界面的代码，其最小代码集如下：

7.2 本地服务

•7.2.1 服务管理

```
7  import android.app.Service;
8  import android.content.Intent;
9  import android.os.IBinder;
10
11  public class RandomService extends Service{
12      @Override
13      public IBinder onBind(Intent intent) {
14          return null;
15      }
16  }
```

7.2 本地服务

• 7.2.1 服务管理

- 除了在第1行到第3行引入必要包外，仅在第5行声明了RandomService继承了android.app.Service类，在第7行到第9行重载了onBind()函数
- onBind()函数是在Service被绑定后调用的函数，能够返回Service的对象实例

7.2 本地服务

• 7.2.1 服务管理

- 这个Service最小代码集并没有任何实际的功能，为了使Service具有实际意义，一般需要重载onCreate()、onStart()和onDestroy()。Android系统在创建Service时，会自动调用onCreate()，用户一般在onCreate()完成必要的初始化工作，例如创建线程、建立数据库链接等
- 在Service关闭前，系统会自动调用onDestroy()函数释放所有占用的资源。通过Context.startService(Intent)启动Service，onStart()则会被调用，重要的参数通过参数Intent传递给Service
- 当然，不是所有的Service都需要重载这三个函数，可以根据实际情况选择需要重载的函数

7.2 本地服务

•7.2.1 服务管理

```
1  public class RandomService extends Service{
2      @Override
3      public void onCreate() {
4          super.onCreate();
5      }
6      @Override
7      public void onStart(Intent intent, int startId) {
8          super.onStart(intent, startId);
9      }
10     @Override
11     public void onDestroy() {
12         super.onDestroy();
13     }
14 }
```

7.2 本地服务

•7.2.1 服务管理

- 重载onCreate()、onStart()和onDestroy()三个函数时，务必要在代码中调用父函数，如代码的第4行、第8行和第12行
- 完成Service类后，需要在AndroidManifest.xml文件中注册这个Service
- 注册Service非常重要，如果开发人员不对Service进行注册，则Service根本无法启动
- AndroidManifest.xml文件中注册Service的代码如下：

1 <service android:name=".RandomService"/>

- 使用<service>标签声明服务，其中的android:name表示Service类的名称，一定要与建立的Service类名称一致

7.2 本地服务

•7.2.1 服务管理

- 在完成Service代码和在AndroidManifest.xml文件中注册后，下面来说明如何启动和停止Service。Android5.0版本之前，有两种方法启动Service，显式启动和隐式启动
- 显式启动需要在Intent中指明Service所在的类，并调用startService(Intent)启动Service，示例代码如下：

```
1 final Intent serviceIntent = new Intent(this, RandomService.class);
```

```
2 startService(serviceIntent);
```

- 在上面的代码中，Intent指明了启动的Service所在类为RandomSerevice

7.2 本地服务

•7.2.1 服务管理

- 为了确保应用安全性，从Android5.0（API级别21）之后，在启动Service时，必须使用显式启动，否则系统可能会抛出异常。所以在这里不再介绍隐式启动。

7.2 本地服务

•7.2.1 服务管理

- 无论是显式启动还是隐式启动，停止Service的方法都是相同的，将启动Service的Intent传递给stopService(Intent)函数即可，示例代码如下：

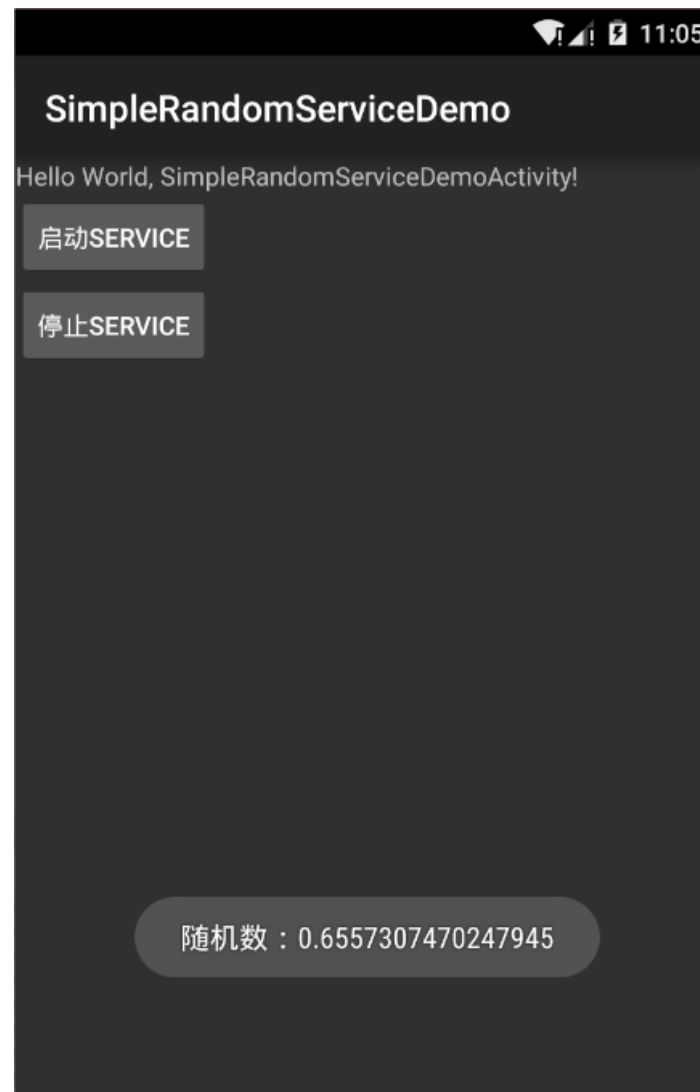
1 stopService(serviceIntent);

- 在首次调用startService(Intent)函数启动Service后，系统会先后调用onCreate()和onStart()
- 如果是第二次调用startService(Intent)函数，系统则仅调用onStart()，而不再调用onCreate()
- 在调用stopService(Intent)函数停止Service时，系统会调用onDestroy()
- 无论调用过多少次startService(Intent)，在调用stopService(Intent)函数时，系统仅调用一次onDestroy()

7.2 本地服务

• 7.2.1 服务管理

- SimpleRandomServiceDemo是在应用程序中使用Service的示例，这个示例使用显式启动的方式启动Service
- 在工程中创建了RandomService服务，该服务启动后会产生一个随机数，并使用Toast显示在屏幕上，如右图所示：



7.2 本地服务

• 7.2.1 服务管理

- 示例

- 通过界面上的“启动Service”按钮调用startService(Intent)函数，启动RandomService服务
- “停止Service”按钮调用stopService(Intent)函数，停止RandomService服务
- 为了能够清晰的观察Service中onCreate()、onStart()和onDestroy()三个函数的调用顺序，在每个函数中都使用Toast在界面上产生提示信息
- RandomService.java文件的代码如下：

7.2

本地服务

•7.2.1 服务管理

- RandomService.java文件的代码

```
1  package edu.hrbeu.SimpleRandomServiceDemo;
2
3  import android.app.Service;
4  import android.content.Intent;
5  import android.os.IBinder;
6  import android.widget.Toast;
7
8  public class RandomService extends Service{
9
10     @Override
11     public void onCreate() {
12         super.onCreate();
13         Toast.makeText(this, "(1) 调用onCreate()",
```

7.2 本地服务

• 7.2.1 服务管理

- RandomService.java文件的代码

```
14         Toast.LENGTH_LONG).show();
15     }
16
17     @Override
18     public void onStart(Intent intent, int startId) {
19         super.onStart(intent, startId);
20         Toast.makeText(this, "(2) 调用onStart()",
21             Toast.LENGTH_SHORT).show();
22
23         double randomDouble = Math.random();
24         String msg = "随机数: " + String.valueOf(randomDouble);
25         Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
26     }
```

7.2

本地服务

•7.2.1 服务管理

- RandomService.java文件的代码

```
27
28     @Override
29     public void onDestroy() {
30         super.onDestroy();
31         Toast.makeText(this, "(3) 调用onDestroy()",
32             Toast.LENGTH_SHORT).show();
33     }
34
35     @Override
36     public IBinder onBind(Intent intent) {
37         return null;
38     }
39 }
```


7.2

本地服务

• 7.2.1 服务管理

- 示例

- 在onStart()函数中添加生产随机数的代码，第23行生产一个介于0和1之间的随机数，并在第24行构造供Toast显示的消息
- AndroidManifest.xml文件的代码如下：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="edu.hrbeu.SimpleRandomServiceDemo"
4      android:versionCode="1"
5      android:versionName="1.0">
6      <application android:icon="@drawable/icon"
          android:label="@string/app_name">
```

7.2 本地服务

• 7.2.1 服务管理

- AndroidManifest.xml文件的代码

```
7      <activity android:name=".SimpleRandomServiceDemo"  
8          android:label="@string/app_name">  
9          <intent-filter>  
10             <action android:name="android.intent.action.MAIN" />  
11             <category android:name="android.intent.category.LAUNCHER" />  
12          </intent-filter>  
13        </activity>  
14        <service android:name=".RandomService"/>  
15    </application>  
16    <uses-sdk android:minSdkVersion="14" />  
17 </manifest>
```

7.2 本地服务

• 7.2.1 服务管理

- 示例

- 在调用AndroidManifest.xml文件中，在<application>标签下，包含一个<activity>标签和一个<service>标签，在<service>标签中，声明了RandomService所在的类
- SimpleRandomServiceDemoActivity.java文件的代码如下：

```
1  package edu.hrbeu.SimpleRandomServiceDemo;  
2  
3  import android.app.Activity;  
4  import android.content.Intent;  
5  import android.os.Bundle;  
6  import android.view.View;  
7  import android.widget.Button;
```

7.2

本地服务

•7.2.1 服务管理

- SimpleRandomServiceDemoActivity.java文件的代码

```
8
9  public class SimpleRandomServiceDemoActivity extends Activity {
10     @Override
11     public void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.main);
14
15         Button startButton = (Button)findViewById(R.id.start);
16         Button stopButton = (Button)findViewById(R.id.stop);
17         final Intent serviceIntent = new Intent(this, RandomService.class);
18         startButton.setOnClickListener(new Button.OnClickListener(){
19             public void onClick(View view){
20                 startService(serviceIntent);
```

7.2 本地服务

• 7.2.1 服务管理

- SimpleRandomServiceDemoActivity.java文件的代码

```
21         }  
22     });  
23     stopButton.setOnClickListener(new Button.OnClickListener(){  
24         public void onClick(View view){  
25             stopService(serviceIntent);  
26         }  
27     });  
28 }  
29 }
```


7.2 本地服务

• 7.2.1 服务管理

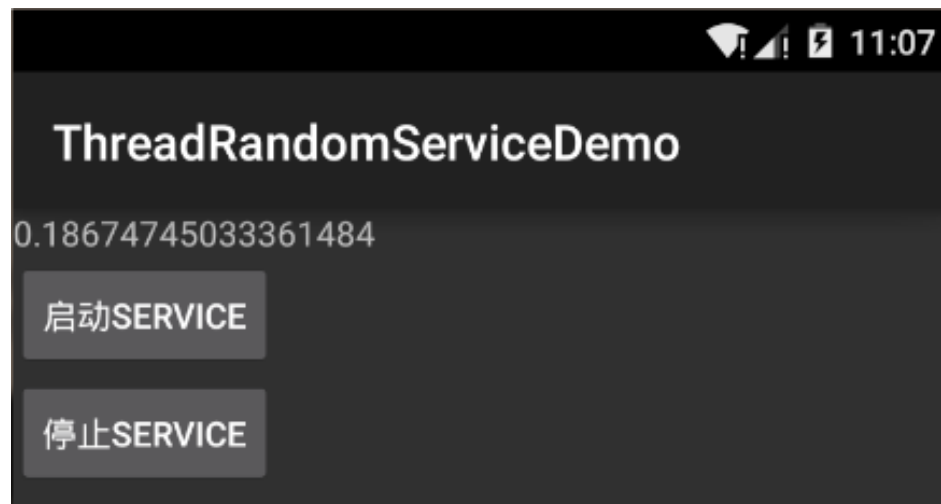
- 示例

- SimpleRandomServiceDemoActivity.java文件是应用程序中的Activity代码，第20行和第25行分别是启动和停止Service的代码

7.2 本地服务

• 7.2.2 使用线程

- 在Android系统中，Activity、Service和BroadcastReceiver都是工作在主线程上，因此任何耗时的处理过程都会降低用户界面的响应速度，甚至导致用户界面失去响应
- 当用户界面失去响应超过5秒后，Android系统会允许用户强行关闭应用程序，提示如下图所示：



7.2 本地服务

• 7.2.2 使用线程

- 因此，较好的解决方法是将耗时的处理过程转移到子线程上，这样可以缩短主线程的事件处理时间，从而避免用户界面长时间失去响应
- “耗时的处理过程”一般指复杂运算过程、大量的文件操作、存在延时的网络通讯和数据库操作等
- 线程是独立的程序单元，多个线程可以并行工作。在多处理器系统中，每个中央处理器（CPU）单独运行一个线程，因此线程是并行工作的
- 在单处理器系统中，处理器会给每个线程一小段时间，在这个时间内线程是被执行的，然后处理器执行下一个线程，这样就产生了线程并行运行的假象

7.2 本地服务

•7.2.2 使用线程

- 无论线程是否真的并行工作，在宏观上可以认为子线程是独立于主线程的，且能与主线程并行工作的程序单元
- 在Java语言中，建立和使用线程比较简单，首先需要实现Java的Runnable接口，并重载run()函数，在run()中放置代码的主体部分

```
1  private Runnable backgroudWork = new Runnable(){  
2      @Override  
3      public void run() {  
4          //过程代码  
5      }  
6  };
```

7.2 本地服务

• 7.2.2 使用线程

- 然后创建Thread对象，并将Runnable对象作为参数传递给Thread对象
- 在Thread的构造函数中，第1个参数用来表示线程组，第2个参数是需要执行的Runnable对象，第3个参数是线程的名称

```
1 private Thread workThread;
```

```
2 workThread = new Thread(null, backgroundWork, "WorkThread");
```

- 最后，调用start()方法启动线程

```
1 workThread.start();
```

7.2 本地服务

•7.2.2 使用线程

- 当线程在run()方法返回后，线程就自动终止了
- 当然，也可以调用stop()在外部终止线程，但这种方法并不推荐使用，因为这种方法并不安全，有一定可能性会产生异常
- 最好的方法是通知线程自行终止，一般调用interrupt()方法通告线程准备终止，线程会释放它正在使用的资源，在完成所有的清理工作后自行关闭

1 workThread.interrupt();

- 其实interrupt()方法并不能直接终止线程，仅是改变了线程内部的一个布尔值，run()方法能够检测到这个布尔值的改变，从而在适当的时候释放资源和终止线程

7.2 本地服务

• 7.2.2 使用线程

- 在run()中的代码一般通过Thread.interrupted()方法查询线程是否被中断
- 一般情况下，子线程需要无限运行，除非外部调用interrupt()方法中断线程，所以通常会将程序主体放置在while()函数内，并调用Thread.interrupted()方法判断线程是否应被中断
- 下面的代码中以1秒为间隔循环检测线程是否应被中断

```
1  public void run() {  
2      while(!Thread.interrupted()){  
3          //过程代码  
4          Thread.sleep(1000);  
5      }  
6  }
```

7.2 本地服务

•7.2.2 使用线程

- 第5行代码使线程休眠1000毫秒
- 当线程在休眠过程中线程被中断，则会产生InterruptedException异常
- 因此代码中需要捕获InterruptedException异常，保证安全终止线程

```
1  public void run() {  
2      try {  
3          while(true){  
4              //过程代码  
5              Thread.sleep(1000);  
6          }  
7      } catch (InterruptedException e) {  
8          e.printStackTrace();  
9      }  
10 }
```

7.2 本地服务

• 7.2.2 使用线程

- 使用Handler更新用户界面
 - Handler允许将Runnable对象发送到线程的消息队列中，每个Handler实例绑定到一个单独的线程和消息队列上
 - 当用户建立一个新的Handler实例，通过post()方法将Runnable对象从后台线程发送给GUI线程的消息队列，当Runnable对象通过消息队列后，这个Runnable对象将被运行

7.2 本地服务

•7.2.2 使用线程

```
1  private static Handler handler = new Handler();
2
3  public static void UpdateGUI(double refreshDouble){
4      handler.post(RefreshLable);
5  }
6  private static Runnable RefreshLable = new Runnable(){
7      @Override
8      public void run() {
9          //过程代码
10     }
11 };
```

7.2 本地服务

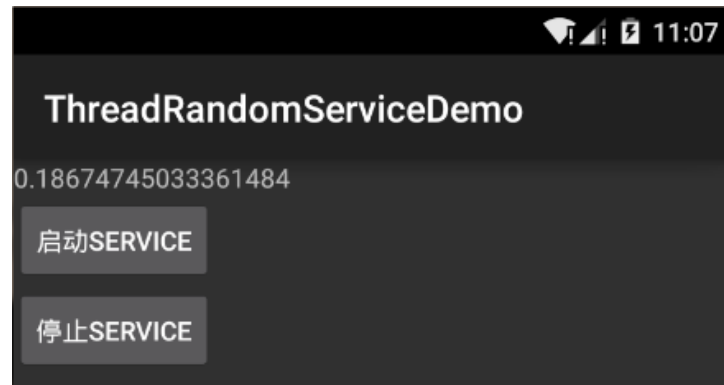
• 7.2.2 使用线程

- 第1行建立了一个静态的Handler实例，但这个实例是私有的，因此外部代码并不能直接调用这个Handler实例
- 第3行UpdateGUI()是公有的界面更新函数，后台线程通过调用该函数，将后台产生的数据refreshDouble传递到UpdateGUI()函数内部，然后直接调用post()方法，将第6行创建的Runnable对象传递给界面线程（主线程）的消息队列中
- 第8行到第10行代码是Runnable对象中需要重载的run()函数，界面更新代码就在这里

7.2 本地服务

• 7.2.2 使用线程

- ThreadRandomServiceDemo是使用线程持续产生随机数的示例
- 点击“启动Service”后将启动后台线程
- 点击“停止Service”将关闭后台线程
- 后台线程每1秒钟产生一个0到1之间的随机数，并通过Handler将产生的随机数显示在用户界面上。ThreadRandomServiceDemo的用户界面如下图所示：



7.2 本地服务

• 7.2.2 使用线程

- 在ThreadRandomServiceDemo示例中，RandomService.java文件是定义Service的文件，用来创建线程、产生随机数和调用界面更新函数
- ThreadRandomServiceDemoActivity.java文件是用户界面的Activity文件，封装Handler界面更新的函数就在这个文件中
- 下面是RandomService.java和ThreadRandomServiceDemoActivity.java文件的完整代码

7.2 本地服务

• 7.2.2 使用线程

- RandomService.java 文件代码

```
1  package edu.hrbeu.ThreadRandomServiceDemo;  
2  
3  import android.app.Service;  
4  import android.content.Intent;  
5  import android.os.IBinder;  
6  import android.widget.Toast;  
7  
8  public class RandomService extends Service{  
9  
10     private Thread workThread;  
11  
12     @Override
```

7.2 本地服务

• 7.2.2 使用线程

- RandomService.java 文件代码

```
13     public void onCreate() {  
14         super.onCreate();  
15         Toast.makeText(this, "(1) 调用onCreate()",  
16             Toast.LENGTH_LONG).show();  
17         workThread = new Thread(null,backgroudWork,"WorkThread");  
18     }  
19  
20     @Override  
21     public void onStart(Intent intent, int startId) {  
22         super.onStart(intent, startId);  
23         Toast.makeText(this, "(2) 调用onStart()",  
24             Toast.LENGTH_SHORT).show();
```

7.2

本地服务

• 7.2.2 使用线程

- RandomService.java 文件代码

```
25         if (!workThread.isAlive()){
26             workThread.start();
27         }
28     }
29
30     @Override
31     public void onDestroy() {
32         super.onDestroy();
33         Toast.makeText(this, "(3) 调用onDestroy()",
34             Toast.LENGTH_SHORT).show();
35         workThread.interrupt();
36     }
```

7.2

本地服务

• 7.2.2 使用线程

- RandomService.java 文件代码

```
37
38     @Override
39     public IBinder onBind(Intent intent) {
40         return null;
41     }
42
43     private Runnable backgroudWork = new Runnable(){
44         @Override
45         public void run() {
46             try {
47                 while(!Thread.interrupted()){
48                     double randomDouble = Math.random();
```

7.2 本地服务

• 7.2.2 使用线程

- RandomService.java 文件代码

```
49         ThreadRandomServiceDemoActivity.UpdateGUI(randomDouble);
50         Thread.sleep(1000);
51     }
52     } catch (InterruptedException e) {
53         e.printStackTrace();
54     }
55 }
56 };
57 }
```


7.2

本地服务

• 7.2.2 使用线程

- ThreadRandomServiceDemoActivity.java 文件代码

```
1 package edu.hrbeu.ThreadRandomServiceDemo;  
2  
3 import android.app.Activity;  
4 import android.content.Intent;  
5 import android.os.Bundle;  
6 import android.os.Handler;  
7 import android.view.View;  
8 import android.widget.Button;  
9 import android.widget.TextView;  
10  
11 public class ThreadRandomServiceDemoActivity extends Activity {  
12
```

7.2

本地服务

•7.2.2 使用线程

- ThreadRandomServiceDemoActivity.java 文件代码

```
13  private static Handler handler = new Handler();
14  private static TextView labelView = null;
15  private static double randomDouble ;
16
17  public static void UpdateGUI(double refreshDouble){
18      randomDouble = refreshDouble;
19      handler.post(RefreshLable);
20  }
21
22  private static Runnable RefreshLable = new Runnable(){
23      @Override
24      public void run() {
```

7.2

本地服务

• 7.2.2 使用线程

- ThreadRandomServiceDemoActivity.java 文件代码

```
25         labelView.setText(String.valueOf(randomDouble));
26     }
27 };
28
29 @Override
30 public void onCreate(Bundle savedInstanceState) {
31     super.onCreate(savedInstanceState);
32     setContentView(R.layout.main);
33     labelView = (TextView)findViewById(R.id.label);
34     Button startButton = (Button)findViewById(R.id.start);
35     Button stopButton = (Button)findViewById(R.id.stop);
36     final Intent serviceIntent = new Intent(this, RandomService.class);
```

7.2

本地服务

• 7.2.2 使用线程

- ThreadRandomServiceDemoActivity.java 文件代码

```
37
38     startButton.setOnClickListener(new Button.OnClickListener(){
39         public void onClick(View view){
40             startService(serviceIntent);
41         }
42     });
44     stopButton.setOnClickListener(new Button.OnClickListener(){
45         public void onClick(View view){
46             stopService(serviceIntent);
47         }
48     });
49 }
50 }
```

7.2 本地服务

•7.2.3 服务绑定

- 以绑定方式使用Service，能够获取到Service实例，不仅能够正常启动Service，还能够调用Service中的公有方法和属性
- 为了使Service支持绑定，需要在Service类中重载onBind()方法，并在onBind()方法中返回Service实例，示例代码如下：

7.2 本地服务

•7.2.3 服务绑定

```
1  public class MathService extends Service{
2      private final IBinder mBinder = new LocalBinder();
3
4      public class LocalBinder extends Binder{
5          MathService getService() {
6              return MathService.this;
7          }
8      }
9
10     @Override
11     public IBinder onBind(Intent intent) {
12         return mBinder;
13     }
14 }
```


7.2 本地服务

• 7.2.3 服务绑定

- 当Service被绑定时，系统会调用onBind()函数，通过onBind()函数的返回值，将Service实例返回给调用者
- 从第11行代码中可以看出，onBind()函数的返回值必须符合IBinder接口，因此在代码第2行声明一个接口变量mBinder，mBinder符合onBind()函数返回值的要求，因此可将mBinder传递给调用者
- IBinder是用于进程内部和进程间过程调用的轻量级接口，定义了与远程对象交互的抽象协议，使用时通过继承Binder的方法来实现
- 继承Binder的代码在第4行，LocalBinder是继承Binder的一个内部类，并在代码第5行实现了getService()函数，当调用者获取到mBinder后，通过调用getService()即可获取到Service实例

7.2 本地服务

• 7.2.3 服务绑定

- 调用者通过bindService()函数绑定服务
 - 调用者通过bindService()函数绑定服务，并在第1个参数中将Intent传递给bindService()函数，声明需要启动的Service
 - 第3个参数Context.BIND_AUTO_CREATE表明只要绑定存在，就自动建立Service
 - 同时也告知Android系统，这个Service的重要程度与调用者相同，除非考虑终止调用者，否则不要关闭这个Service

```
1 final Intent serviceIntent = new Intent(this,MathService.class);  
2 bindService(serviceIntent,mConnection,Context.BIND_AUTO_CREATE);
```

7.2 本地服务

• 7.2.3 服务绑定

- `bindService()`函数的第2个参数是`ServiceConnection`
 - 当绑定成功后，系统将调用`ServiceConnection`的`onServiceConnected()`方法
 - 当绑定意外断开后，系统将调用`ServiceConnection`中的`onServiceDisconnected`方法
 - 因此，以绑定方式使用`Service`，调用者需要声明一个`ServiceConnection`，并重载内部的`onServiceConnected()`方法和`onServiceDisconnected`方法，两个方法的重载代码如下：

7.2 本地服务

• 7.2.3 服务绑定

```
1  private ServiceConnection mConnection = new ServiceConnection() {  
2      @Override  
3      public void onServiceConnected(ComponentName name, IBinder service) {  
4          mathService = ((MathService.LocalBinder)service).getService();  
5      }  
6      @Override  
7      public void onServiceDisconnected(ComponentName name) {  
8          mathService = null;  
9      }  
10 };
```

7.2 本地服务

• 7.2.3 服务绑定

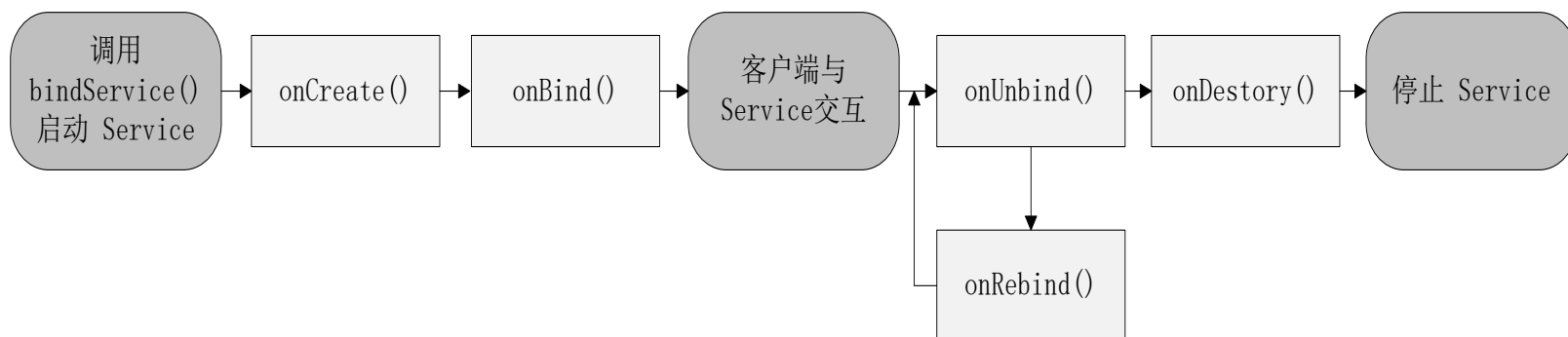
- 在代码的第4行中，绑定成功后通过getService()获取Service实例，这样便可以调用Service中的方法和属性
- 代码第8行将Service实例为null，表示绑定意外失效时，Service实例不再可用
- 取消绑定仅需要使用unbindService()方法，并将ServiceConnection传递给unbindService()方法
- 但需要注意的是，unbindService()方法成功后，系统并不会调用onServiceConnected()，因为onServiceConnected()仅在意外断开绑定时才被调用

1 unbindService(mConnection);

7.2 本地服务

• 7.2.3 服务绑定

- 绑定方式中，当调用者通过bindService()函数绑定Service时， onCreate()函数和 onBind()函数将被先后调用
- 当调用者通过unbindService()函数取消绑定Service时， onUnbind()函数将被调用。若onUnbind()函数返回true，则表示重新绑定服务时， onRebind()函数将被调用。绑定方式的函数调用顺序如下图所示：



7.2 本地服务

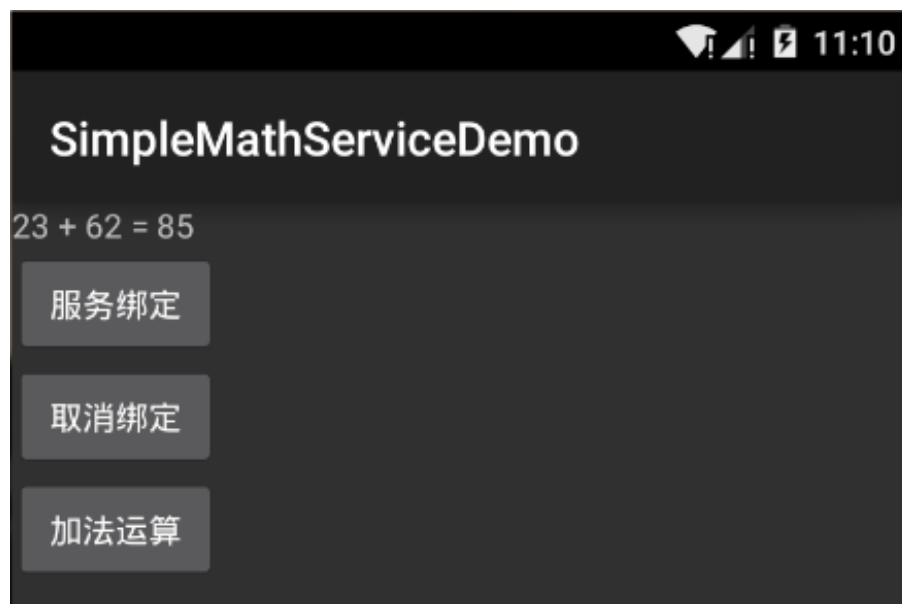
• 7.2.3 服务绑定

- SimpleMathServiceDemo是绑定方式使用Service的示例
 - 在示例中创建了MathService服务，用来完成简单的数学运算，这里的数学运算仅指加法运算，虽然没有实际意义，但可以说明如何使用绑定方式调用Service中的公有方法
 - 在服务绑定后，用户可以点击“加法运算”，将两个随机产生的数值传递给MathService服务，并从MathService实例中获取到加法运算的结果，然后显示在屏幕的上方
 - “取消绑定”按钮可以解除与MathService的绑定关系，在取消绑定后，点击“加法运算”按钮将无法获取运算结果

7.2 本地服务

• 7.2.3 服务绑定

- SimpleMathServiceDemo是绑定方式使用Service的示例
- SimpleMathServiceDemo的用户界面如下图所示:



7.2 本地服务

• 7.2.3 服务绑定

- SimpleMathServiceDemo是绑定方式使用Service的示例
 - 在SimpleMathServiceDemo示例中， MathService.java文件是Service的定义文件
 - SimpleMathServiceDemoActivity.java文件是界面的Activity文件， 绑定服务和取消绑定服务的代码在这个文件中
 - 下面是MathService.java和SimpleMathServiceDemoActivity.java文件的完整代码

7.2 本地服务

• 7.2.3 服务绑定

• MathService.java 文件代码

```
1  package edu.hrbeu.SimpleMathServiceDemo;  
2  
3  import android.app.Service;  
4  import android.content.Intent;  
5  import android.os.Binder;  
6  import android.os.IBinder;  
7  import android.widget.Toast;  
8  
9  public class MathService extends Service{  
10  
11      private final IBinder mBinder = new LocalBinder();  
12
```

7.2 本地服务

• 7.2.3 服务绑定

• MathService.java 文件代码

```
13     public class LocalBinder extends Binder{  
14         MathService getService() {  
15             return MathService.this;  
16         }  
17     }  
18  
19     @Override  
20     public IBinder onBind(Intent intent) {  
21         Toast.makeText(this, "本地绑定: MathService",  
22             Toast.LENGTH_SHORT).show();  
23         return mBinder;  
24     }
```

7.2 本地服务

• 7.2.3 服务绑定

• MathService.java 文件代码

```
25
26     @Override
27     public boolean onUnbind(Intent intent){
28         Toast.makeText(this, "取消本地绑定: MathService",
29             Toast.LENGTH_SHORT).show();
30         return false;
31     }
32
33
34     public long Add(long a, long b){
35         return a+b;
36     }
37
38 }
```


7.2 本地服务

• 7.2.3 服务绑定

• SimpleMathServiceDemoActivity.java 文件代码

```
1 package edu.hrbeu.SimpleMathServiceDemo;  
2  
3 import android.app.Activity;  
4 import android.content.ComponentName;  
5 import android.content.Context;  
6 import android.content.Intent;  
7 import android.content.ServiceConnection;  
8 import android.os.Bundle;  
9 import android.os.IBinder;  
10 import android.view.View;  
11 import android.widget.Button;  
12 import android.widget.TextView;
```

7.2 本地服务

• 7.2.3 服务绑定

• SimpleMathServiceDemoActivity.java 文件代码

```
13
14 public class SimpleMathServiceDemoActivity extends Activity {
15     private MathService mathService;
16     private boolean isBound = false;
17     TextView labelView;
18     @Override
19     public void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.main);
22
23         labelView = (TextView)findViewById(R.id.label);
24         Button bindButton = (Button)findViewById(R.id.bind);
```

7.2 本地服务

• 7.2.3 服务绑定

• SimpleMathServiceDemoActivity.java 文件代码

```
25     Button unbindButton = (Button)findViewById(R.id.unbind);
26     Button computeButton = (Button)findViewById(R.id.compute);
27
28     bindButton.setOnClickListener(new View.OnClickListener(){
29         @Override
30         public void onClick(View v) {
31             if(!isBound){
32                 final Intent serviceIntent = new
33                 Intent(SimpleMathServiceDemoActivity.this,MathService.class);
34                 bindService(serviceIntent,mConnection,Context.BIND_AUTO_CREATE);
35                 isBound = true;
36             }
37         }
38     })
```

7.2 本地服务

• 7.2.3 服务绑定

• SimpleMathServiceDemoActivity.java 文件代码

```
37     });  
38  
39     unbindButton.setOnClickListener(new View.OnClickListener(){  
40         @Override  
41         public void onClick(View v) {  
42             if(isBound){  
43                 isBound = false;  
44                 unbindService(mConnection);  
45                 mathService = null;  
46             }  
47         }  
48     });
```

7.2 本地服务

• 7.2.3 服务绑定

• SimpleMathServiceDemoActivity.java 文件代码

```
49
50     computButton.setOnClickListener(new View.OnClickListener(){
51         @Override
52         public void onClick(View v) {
53             if (mathService == null){
54                 labelView.setText("未绑定服务");
55                 return;
56             }
57             long a = Math.round(Math.random()*100);
58             long b = Math.round(Math.random()*100);
59             long result = mathService.Add(a, b);
60             String msg = String.valueOf(a)+" + "+String.valueOf(b)+
```

7.2 本地服务

• 7.2.3 服务绑定

• SimpleMathServiceDemoActivity.java 文件代码

```
61         msg = "="+String.valueOf(result);
62         labelView.setText(msg);
63     }
64 });
65 }
66
67 private ServiceConnection mConnection = new ServiceConnection() {
68     @Override
69     public void onServiceConnected(ComponentName name, IBinder service) {
70         mathService = ((MathService.LocalBinder)service).getService();
71     }
72 }
```


7.2 本地服务

• 7.2.3 服务绑定

- SimpleMathServiceDemoActivity.java 文件代码

```
73      @Override
74      public void onServiceDisconnected(ComponentName name) {
75          mathService = null;
76      }
77  };
78 }
```

7.3 远程服务

• 7.3.1 进程间通信

- Android系统中，每个应用程序在各自的进程中运行，且出于安全原因的考虑，这些进程之间彼此是隔离的，进程之间传递数据和对象，需要使用Android支持的进程间通信（Inter-Process Communication，IPC）机制
- 在Unix/Linux系统中，传统的IPC机制包括共享内存、管道、消息队列和socket等等，这些IPC机制虽然被广泛使用，但仍然存在着固有的缺陷，如容易产生错误、难于维护等等
- 在Android系统中，没有使用传统的IPC机制，而是采用Intent和远程服务的方式实现IPC，使应用程序具有更好的独立性和鲁棒性

7.3 远程服务

• 7.3.1 进程间通信

- Android系统允许应用程序使用Intent启动Activity和Service，同时Intent可以传递数据，是一种简单、高效、易于使用的IPC机制
- Android系统的另一种IPC机制就是远程服务，服务和调用者在不同的两个进程中，调用过程需要跨越进程才能实现
- 在Android系统中使用远程服务，一般按照以下三个步骤实现
 - 使用AIDL语言定义远程服务的接口
 - 根据AIDL语言定义的接口，在具体的Service类中实现接口中定义的方法和属性
 - 在需要调用远程服务组件中，通过相同的AIDL接口文件，调用远程服务

7.3 远程服务

• 7.3.2 服务创建与调用

- 在Android系统中，进程之间不能直接访问相互的内存控件，因此为了使数据能够在不同进程间传递，数据必须转换成能够穿越进程边界的系统级原语，同时，在数据完成进程边界穿越后，还需要转换回原有的格式
- AIDL（Android Interface Definition Language）是Android系统自定义的接口描述语言，可以简化进程间数据格式转换和数据交换的代码，通过定义Service内部的公共方法，允许在不同进程间的调用者和Service之间相互传递数据
- AIDL的IPC机制、COM和Corba都是基于接口的轻量级进程通信机制

7.3 远程服务

• 7.3.2 服务创建与调用

- AIDL语言的语法与Java语言的接口定义非常相似，唯一不同之处在于，AIDL允许定义函数参数的传递方向
- AIDL支持三种方向：in、out和inout
 - 标识为in的参数将从调用者传递到远程服务中
 - 标识为out的参数将从远程服务传递到调用者中
 - 标识为inout的参数将先从调用者传递到远程服务中，再从远程服务返回给调用者
- 如果不标识参数的传递方向，默认所有函数的传递方向为in
- 出于性能方面的考虑，不要在参数中标识不需要的传递方向

7.3 远程服务

•7.3.2 服务创建与调用

- 远程服务的创建和调用需要使用AIDL语言，一般分为以下几个过程
 - 使用AIDL语言定义远程服务的接口
 - 通过继承Service类实现远程服务
 - 绑定和使用远程服务

7.3 远程服务

• 7.3.2 服务创建与调用

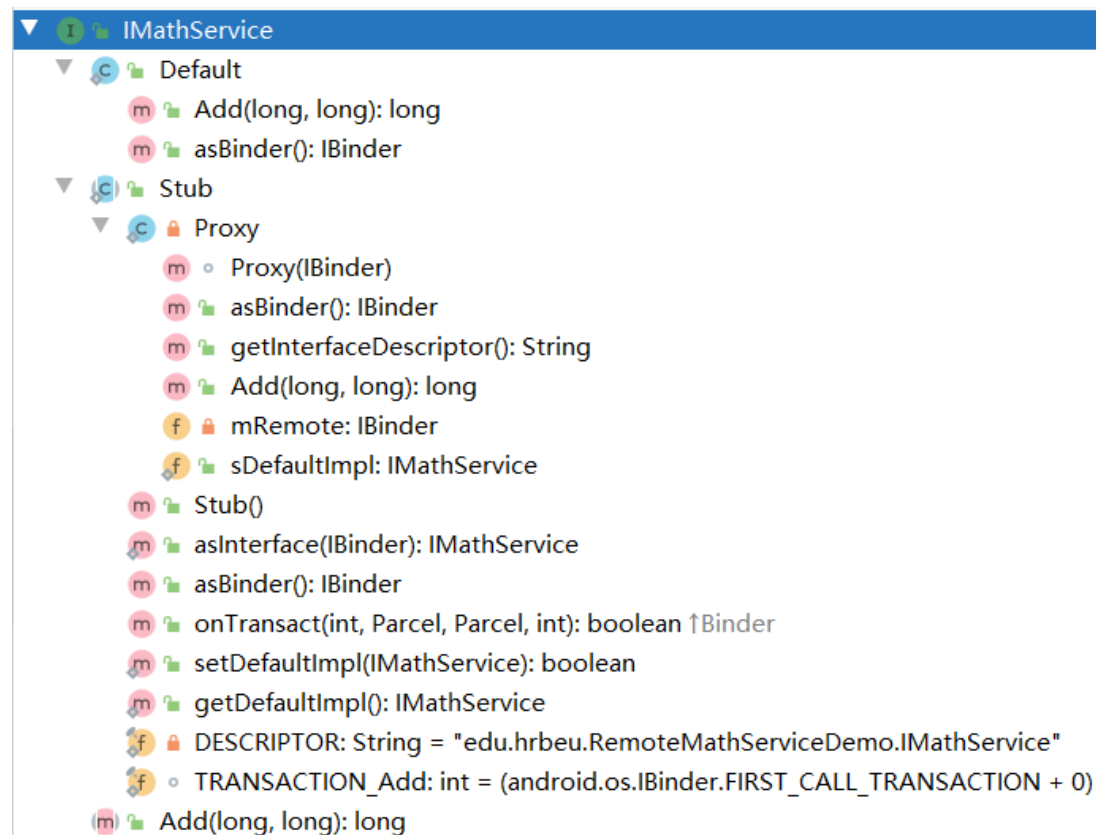
- 下面以RemoteMathServiceDemo示例为参考，说明如何创建远程服务
- 在这个示例中定义了MathService服务，可以为远程调用者提供加法服务
- 使用AIDL语言定义远程服务的接口
 - 首先使用AIDL语言定义MathService的服务接口，服务接口文件的扩展名为.aidl，使用的包名称与Android项目所使用的相同
 - 在src目录下的建立IMathService.aidl文件，代码如下：

```
1 package edu.hrbeu.RemoteMathServiceDemo;
2 interface IMathService {
3     long Add(long a, long b);
4 }
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 从上面的代码中可以看出，IMathService 接口仅包含一个add()方法，传入的参数是两个长型整数，返回值也是长型整数
- 使用Android Studio编辑IMathService.aidl文件，当保存文件后点击Android Studio 菜单栏上的Build->Make Project，根据AIDL文件在java(generated)目录下生成java接口文件IMathService.java。
- 右图为IMathService.java文件结构



7.3 远程服务

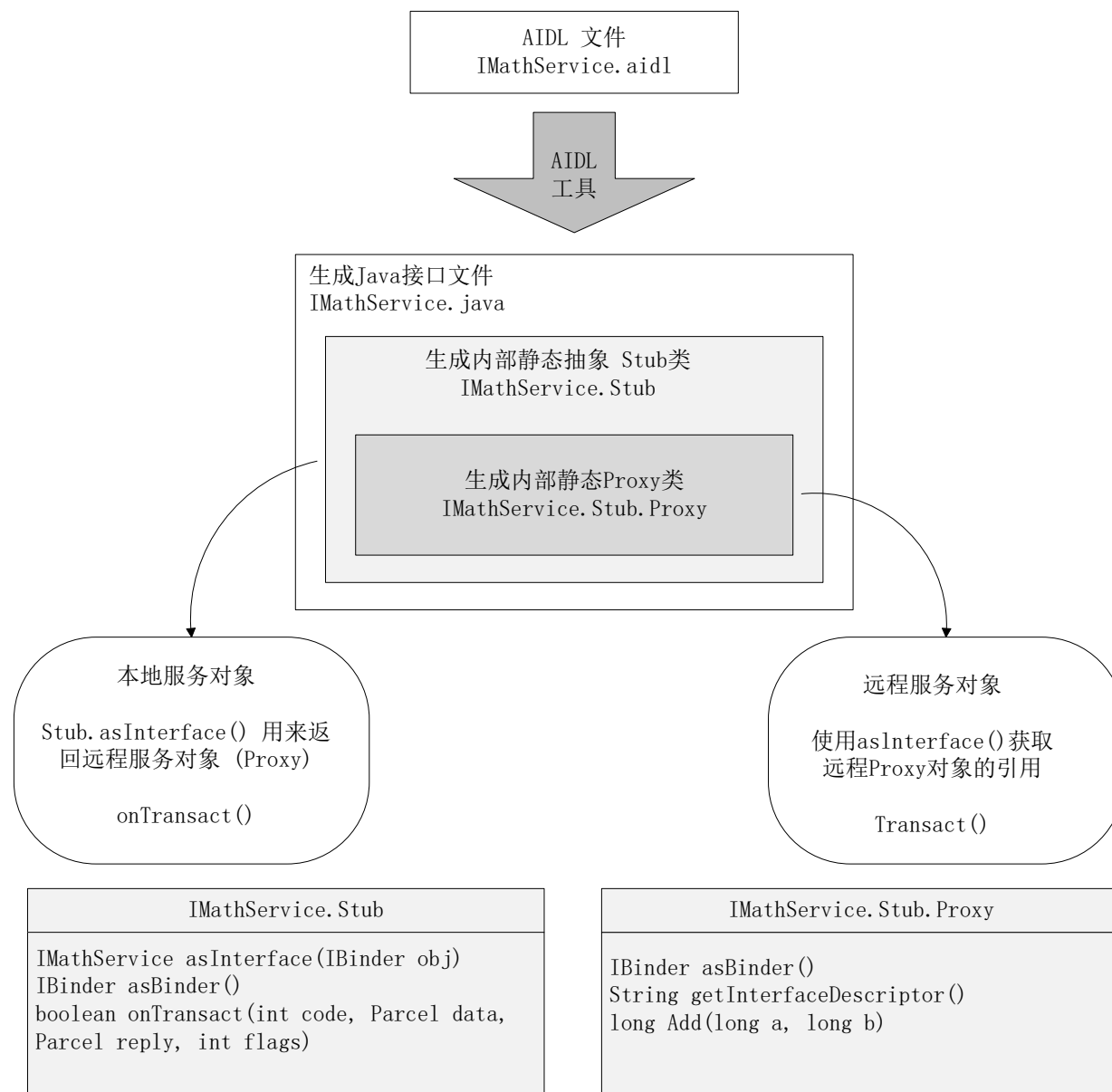
• 7.3.2 服务创建与调用

- 使用AIDL语言定义远程服务的接口
 - IMathService.java文件根据IMathService.aidl的定义，生成了两个内部静态抽象类Default和Stub，如图7.7所示，Default和Stub都继承了Binder类，并实现IMathService接口。
 - Default类是接口IMathService的默认实现，Default类的默认实现方法基本都是空方法，一般用不到。
 - 在Stub类中，还包含一个重要的静态类Proxy。可以认为Stub类用来实现本地服务调用，Proxy类用来实现远程服务调用，将Proxy作为Stub的内部类完全是出于使用方便的目的。

7.3 远程服务

• 7.3.2 服务创建与调用

- 使用AIDL语言定义远程服务的接口
 - Stub类和Proxy类关系图



7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
1  /*
2   * This file is auto-generated. DO NOT MODIFY.
3   */
4  package edu.hrbeu.RemoteMathServiceDemo;
5  public interface IMathService extends android.os.IInterface
6  {
7      /** Default implementation for IMathService. */
8      public static class Default implements edu.hrbeu.RemoteMathServiceDemo.IMathService
9      {
10         @Override public long Add(long a, long b) throws android.os.RemoteException
11         {
12             return 0L;
13         }
14         @Override
```

7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
15  public android.os.IBinder asBinder() {  
16      return null;  
17  }  
18  }  
19  /** Local-side IPC implementation stub class. */  
20  public static abstract class Stub extends android.os.Binder implements  
    edu.hrbeu.RemoteMathServiceDemo.IMathService  
21  {  
22      private static final java.lang.String DESCRIPTOR =  
        "edu.hrbeu.RemoteMathServiceDemo.IMathService";  
23      /** Construct the stub at attach it to the interface. */  
24      public Stub()  
25      {  
26          this.attachInterface(this, DESCRIPTOR);  
27      }
```


7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
28  /**
29   * Cast an IBinder object into an edu.hrbeu.RemoteMathServiceDemo.IMathService
   interface,
30   * generating a proxy if needed.
31   */
32   public static edu.hrbeu.RemoteMathServiceDemo.IMathService
   asInterface(android.os.IBinder obj)
33   {
34       if ((obj==null)) {
35           return null;
36       }
37       android.os.IInterface iin = obj.queryLocalInterface(DESCRIPTOR);
```

7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
38     if (((iin!=null)&&(iin instanceof edu.hrbeu.RemoteMathServiceDemo.IMathService))){
39         return ((edu.hrbeu.RemoteMathServiceDemo.IMathService)iin);
40     }
41     return new edu.hrbeu.RemoteMathServiceDemo.IMathService.Stub.Proxy(obj);
42 }
43 @Override public android.os.IBinder asBinder()
44 {
45     return this;
46 }
47 @Override public boolean onTransact(int code, android.os.Parcel data, android.os.Parcel
    reply, int flags) throws android.os.RemoteException
48 {
49     java.lang.String descriptor = DESCRIPTOR;
```

7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
50  switch (code)
51  {
52    case INTERFACE_TRANSACTION:
53    {
54      reply.writeString(descriptor);
55      return true;
56    }
57    case TRANSACTION_Add:
58    {
59      data.enforceInterface(descriptor);
60      long _arg0;
61      _arg0 = data.readLong();
```

7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
62     long _arg1;  
63     _arg1 = data.readLong();  
64     long _result = this.Add(_arg0, _arg1);  
65     reply.writeNoException();  
66     reply.writeLong(_result);  
67     return true;  
68 }  
69 default:  
70 {  
71     return super.onTransact(code, data, reply, flags);  
72 }  
73 }
```

7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
74  }
75  private static class Proxy implements edu.hrbeu.RemoteMathServiceDemo.IMathService
76  {
77      private android.os.IBinder mRemote;
78      Proxy(android.os.IBinder remote)
79      {
80          mRemote = remote;
81      }
82      @Override public android.os.IBinder asBinder()
83      {
84          return mRemote;
85      }
86      public java.lang.String getInterfaceDescriptor()
87      {
```

7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
88     return DESCRIPTOR;
89 }
90 @Override public long Add(long a, long b) throws android.os.RemoteException
91 {
92     android.os.Parcel _data = android.os.Parcel.obtain();
93     android.os.Parcel _reply = android.os.Parcel.obtain();
94     long _result;
95     try {
96         _data.writeInterfaceToken(DESCRIPTOR);
97         _data.writeLong(a);
98         _data.writeLong(b);
99         boolean _status = mRemote.transact(Stub.TRANSACTION_Add, _data, _reply, 0);
100         if (!_status && getDefaultImpl() != null) {
101             return getDefaultImpl().Add(a, b);
```


7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
102     }
103     _reply.readException();
104     _result = _reply.readLong();
105 }
106 finally {
107     _reply.recycle();
108     _data.recycle();
109 }
110 return _result;
111 }
112 public static edu.hrbeu.RemoteMathServiceDemo.IMathService sDefaultImpl;
113 }
114 static final int TRANSACTION_Add = (android.os.IBinder.FIRST_CALL_TRANSACTION + 0);
```

7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
115 public static boolean setDefaultImpl(edu.hrbeu.RemoteMathServiceDemo.IMathService impl) {  
116     // Only one user of this interface can use this function  
117     // at a time. This is a heuristic to detect if two different  
118     // users in the same process use this function.  
119     if (Stub.Proxy.sDefaultImpl != null) {  
120         throw new IllegalStateException("setDefaultImpl() called twice");  
121     }  
122     if (impl != null) {  
123         Stub.Proxy.sDefaultImpl = impl;  
124         return true;  
125     }  
126     return false;  
127 }
```

7.3 远程服务

• 7.3.2 服务创建与调用

- IMathService.java的完成代码

```
128 public static edu.hrbeu.RemoteMathServiceDemo.IMathService getDefaultImpl() {  
129     return Stub.Proxy.sDefaultImpl;  
130 }  
131 }  
132 public long Add(long a, long b) throws android.os.RemoteException;  
133 }
```

7.3 远程服务

• 7.3.2 服务创建与调用

- `IMathService`继承了`android.os.IInterface`（第5行），这是所有使用AIDL建立的接口都必须继承基类接口，这个基类接口中定义了`asBinder()`方法，用来获取Binder对象
- 在代码的第43行到第46行，实现了`android.os.IInterface`接口所定义的`asBinder()`方法
- 在`IMathService`中，绝大多数的代码是用来实现Stub这个抽象类的
- 每个远程接口都包含Stub类，因为是内部类，所有并不会产生命名冲突
- `asInterface(IBinder)`是Stub内部的远程服务接口，调用者可以通过该方法获取到远程服务的实例

7.3 远程服务

• 7.3.2 服务创建与调用

- 仔细观察asInterface(IBinder)实现方法，首先判断IBinder对象obj是否为null（第34行），如果是则立即返回
- 然后使用DESCRIPTOR构造android.os.IInterface实例（第37行），并判断android.os.IInterface实例是否为本地服务，如果是本地服务，则无需进行进程间通信，返回android.os.IInterface实例（第39行）；如果不是本地服务，则构造并返回Proxy对象（第41行）
- Proxy内部包含与IMathService.aidl相同签名的函数（第90行），并且在该函数中以一定的顺序将所有参数写入Parcel对象（第96~104行），以供Stub内部的onTransact()方法能够正确获取到参数

7.3 远程服务

• 7.3.2 服务创建与调用

- 当数据以Parcel对象的形式传递到远程服务的内部时，onTransact()方法（第477行）将从Parcel对象中逐一的读取每个参数，然后调用Service内部制定的方法，并再将结果写入另一个Parcel对象，准备将这个Parcel对象返回给远程的调用者
- Parcel是Android系统应用程序进程间数据传递的容器，能够在两个进程中完成数据的打包和拆包的工作，但Parcel不同于通用意义上的序列化，Parcel的设计目的是用于高性能IPC传输，因此不能够将Parcel对象保存在任何持久存储设备上

7.3 远程服务

• 7.3.2 服务创建与调用

- 通过继承Service类实现远程服务
 - IMathService.aidl是对远程服务接口的定义，自动生成的IMathService.java内部实现了远程服务数据传递的相关方法，下一步介绍如何实现远程服务
 - 实现远程服务需要建立一个继承android.app.Service的类，并在该类中通过onBind()方法返回IBinder对象，调用者使用返回的IBinder对象访问远程服务
 - IBinder对象的建立通过使用IMathService.java内部的Stub类实现，并逐一实现在IMathService.aidl接口文件定义的函数
 - 在RemoteMathServiceDemo示例中，远程服务的实现类是MathService.java，下面是MathService.java的完整代码

7.3 远程服务

• 7.3.2 服务创建与调用

- 通过继承Service类实现跨进程服务
 - MathService.java的完整代码

```
1  package edu.hrbeu.RemoteMathServiceDemo;  
2  
3  import android.app.Service;  
4  import android.content.Intent;  
5  import android.os.IBinder;  
6  import android.widget.Toast;  
7  
8  public class MathService extends Service{  
9      private final IMathService.Stub mBinder = new IMathService.Stub() {  
10          public long Add(long a, long b) {  
11              return a + b;  
12          }  
13      };  
14 }
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 通过继承Service类实现跨进程服务
 - MathService.java的完整代码

```
14  @Override
15  public IBinder onBind(Intent intent) {
16      Toast.makeText(this, "远程绑定: MathService",
17          Toast.LENGTH_SHORT).show();
18      return mBinder;
19  }
20  @Override
21  public boolean onUnbind (Intent intent){
22      Toast.makeText(this, "取消远程绑定: MathService",
23          Toast.LENGTH_SHORT).show();
24      return false;
25  }
26  }
```

7.3 远程服务

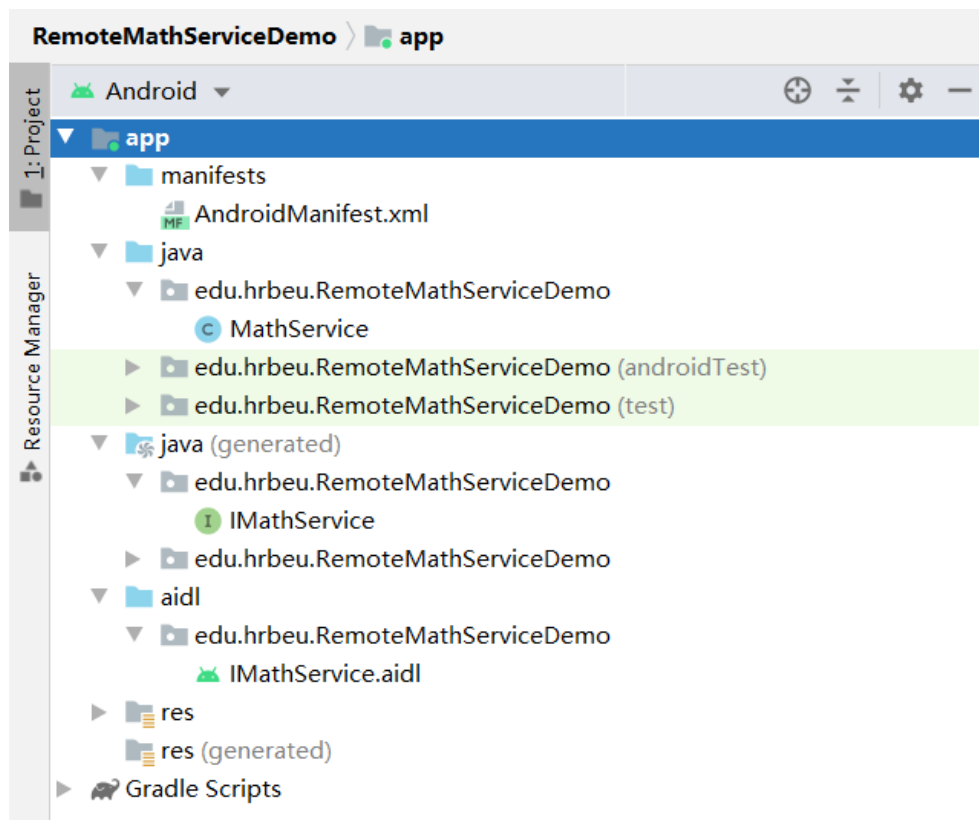
• 7.3.2 服务创建与调用

- 通过继承Service类实现跨进程服务
 - 第8行代码表明MathService继承于android.app.Service
 - 第9行建立IMathService.Stub的实例mBinder，并在第10行实现了AIDL文件定义的远程服务接口
 - 第18行在onBind()方法中，将mBinder返回给远程调用者
 - 第16行和第22行分别是在绑定和取消绑定时，为用户产生的提示信息

7.3 远程服务

• 7.3.2 服务创建与调用

- 通过继承Service类实现跨进程服务
 - RemoteMathServiceDemo示例文件结构



7.3 远程服务

• 7.3.2 服务创建与调用

- 通过继承Service类实现跨进程服务
 - 示例中只有远程服务的类文件MathService.java和接口文件IMathService.aidl，没有任何显示用户界面的Activity文件
 - 因此在调试RemoteMathServiceDemo示例时，模拟器上不会有任何用户界面出现，但在控制台会有“安装成功完成，4秒160毫秒。无法识别启动Activity：找不到默认Activity”提示信息，如下图所示，表明.apk文件已经上传到模拟器中

```
04/18 14:34:02: Launching 'app' on No Devices.
```

```
Install successfully finished in 4 s 160 ms.
```

```
Could not identify launch activity: Default Activity not found
```

```
Error while Launching activity
```


7.3 远程服务

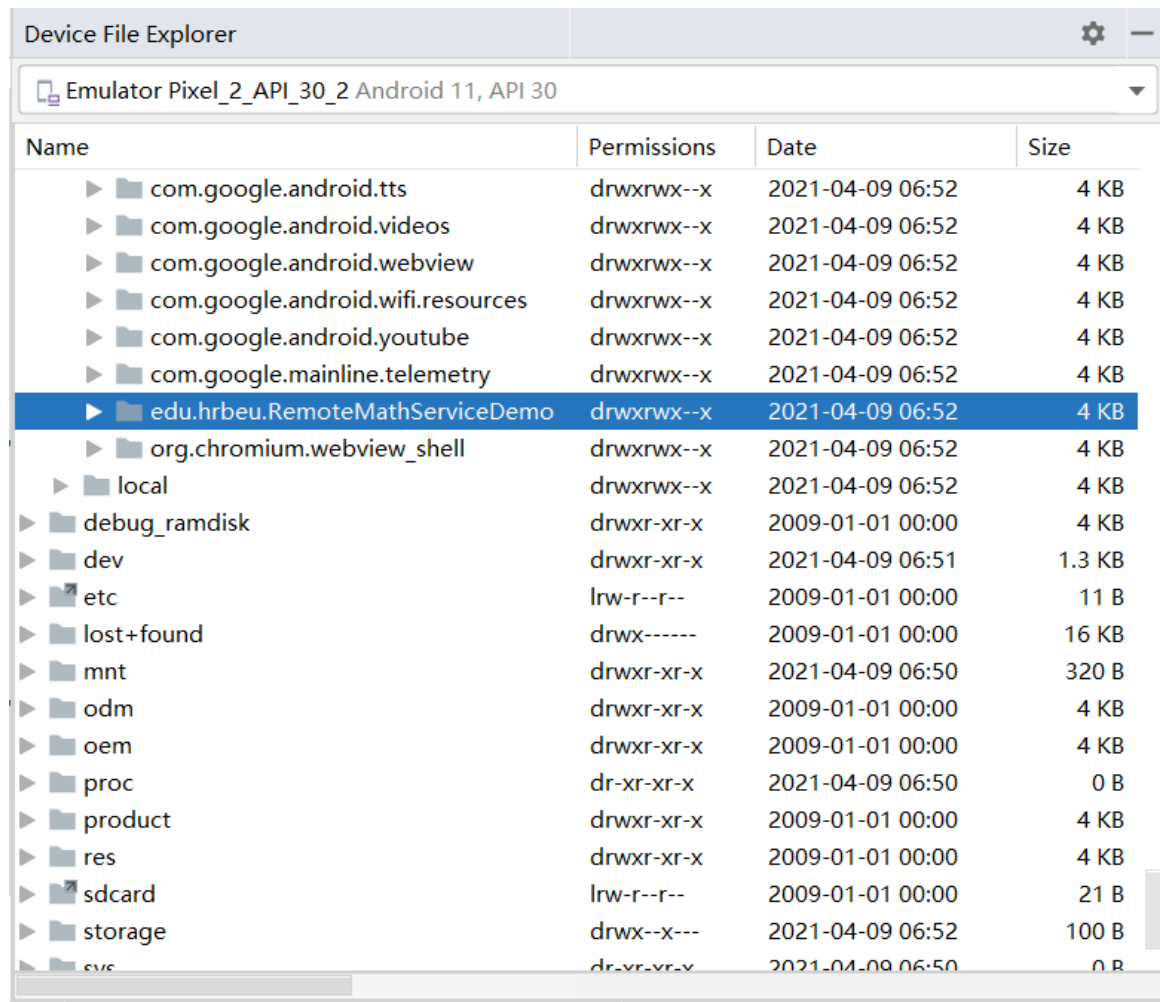
• 7.3.2 服务创建与调用

- 通过继承Service类实现跨进程服务
 - 为了进一步确认编译好的.apk文件是否正确上传到模拟器中，可以使用Device File Explorer查看模拟器的文件系统
 - 如果能在/data/data/下找到edu.hrbeu.RemoteMathServiceDemo.apk文件，说明提供远程服务的.apk文件已经正确上传。RemoteMathServiceDemo示例无法在Android模拟器的程序启动栏中找到，只能够通过其它应用程序调用该示例中的远程服务
 - 下图显示了edu.hrbeu.RemoteMathServiceDemo.apk文件的保存位置

7.3 远程服务

• 7.3.2 服务创建与调用

- 通过继承Service类实现跨进程服务
 - RemoteMathServiceDemo.apk文件位置



Device File Explorer

Emulator Pixel_2_API_30_2 Android 11, API 30

Name	Permissions	Date	Size
▶ com.google.android.tts	drwxrwx--x	2021-04-09 06:52	4 KB
▶ com.google.android.videos	drwxrwx--x	2021-04-09 06:52	4 KB
▶ com.google.android.webview	drwxrwx--x	2021-04-09 06:52	4 KB
▶ com.google.android.wifi.resources	drwxrwx--x	2021-04-09 06:52	4 KB
▶ com.google.android.youtube	drwxrwx--x	2021-04-09 06:52	4 KB
▶ com.google.mainline.telemetry	drwxrwx--x	2021-04-09 06:52	4 KB
▶ edu.hrbeu.RemoteMathServiceDemo	drwxrwx--x	2021-04-09 06:52	4 KB
▶ org.chromium.webview_shell	drwxrwx--x	2021-04-09 06:52	4 KB
▶ local	drwxrwx--x	2021-04-09 06:52	4 KB
▶ debug_ramdisk	drwxr-xr-x	2009-01-01 00:00	4 KB
▶ dev	drwxr-xr-x	2021-04-09 06:51	1.3 KB
▶ etc	lrw-r--r--	2009-01-01 00:00	11 B
▶ lost+found	drwx-----	2009-01-01 00:00	16 KB
▶ mnt	drwxr-xr-x	2021-04-09 06:50	320 B
▶ odm	drwxr-xr-x	2009-01-01 00:00	4 KB
▶ oem	drwxr-xr-x	2009-01-01 00:00	4 KB
▶ proc	dr-xr-xr-x	2021-04-09 06:50	0 B
▶ product	drwxr-xr-x	2009-01-01 00:00	4 KB
▶ res	drwxr-xr-x	2009-01-01 00:00	4 KB
▶ sdcard	lrw-r--r--	2009-01-01 00:00	21 B
▶ storage	drwx--x---	2021-04-09 06:52	100 B
▶ sys	dr-xr-xr-x	2021-04-09 06:50	0 B

7.3 远程服务

• 7.3.2 服务创建与调用

- 通过继承Service类实现跨进程服务
 - RemoteMathServiceDemo是本书中第一个没有Activity的示例，在AndroidManifest.xml文件中，在<application>标签下只有一个<service>标签
 - AndroidManifest.xml文件的完代码如下
 - 这里注意第10行代码，edu.hrbeu.RemoteMathServiceDemo.MathService是远程调用MathService的标识，调用者使用Intent.setAction()函数将标识加入Intent中，然后隐式启动或绑定服务

7.3 远程服务

• 7.3.2 服务创建与调用

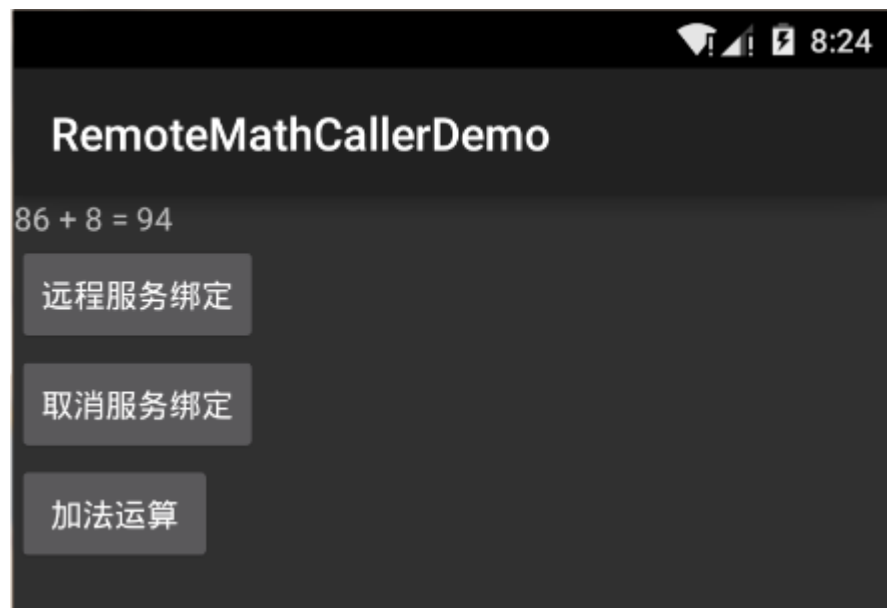
- AndroidManifest.xml文件的完代码如下

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="edu.hrbeu.RemoteMathServiceDemo"
4      android:versionCode="1"
5      android:versionName="1.0">
6      <application android:icon="@drawable/icon" android:label="@string/app_name">
7          <service android:name=".MathService" />
8      </application>
9      <uses-sdk android:minSdkVersion="14" />
10 </manifest>
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务
 - RemoteMathCallerDemo示例说明如何调用RemoteMathServiceDemo示例中的远程服务。RemoteMathCallerDemo的界面如下图所示



7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务
 - 用户可以绑定远程服务，也可以取消服务绑定
 - 在绑定远程服务后，调用RemoteMathServiceDemo中的MathService服务进行加法运算，运算的输入由RemoteMathCallerDemo随机产生，运算的输入和结果显示在屏幕的上方

7.3 远程服务

• 7.3.2 服务创建与调用

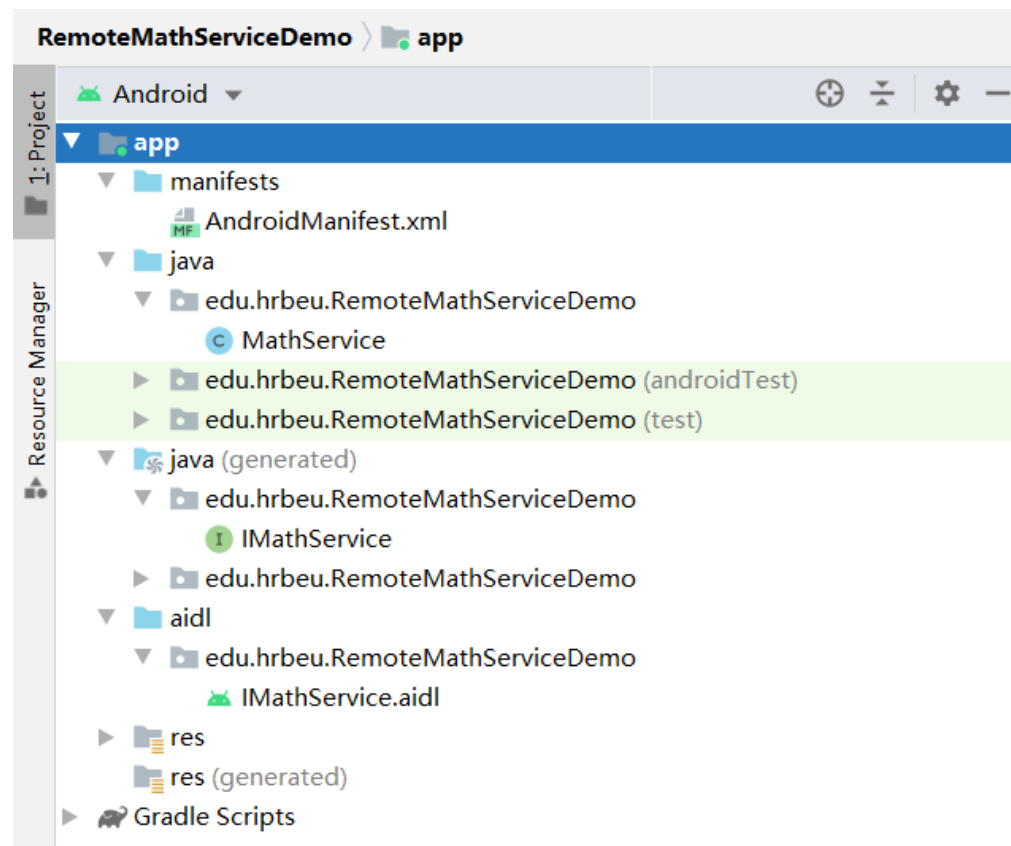
- 绑定和使用远程服务

- 应用程序在调用远程服务时，需要具有相同的Proxy类和签名调用函数，这样才能够使数据在调用者处打包后，可以在远程服务处正确拆包，反之亦然
- 从实践角度来讲，调用者需要使用与远程服务端相同的AIDL文件
- RemoteMathCallerDemo示例中，在edu.hrbeu.RemoteMathServiceDemo包下引入与RemoteMathServiceDemo相同的AIDL文件IMathService.aidl，保存文件后点击Android Studio菜单栏上的Build->Make Project，根据AIDL文件在java(generated)目录下生成java接口文件IMathService.java

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务
 - 下图是RemoteMathServiceDemo的文件结构



7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务
 - RemoteMathCallerDemoActivity.java是Activity的文件，远程服务的绑定和使用方法与7.2.3节的本地服务绑定示例SimpleMathServiceDemo相似
 - 同之处主要包括以下两处
 - 一是使用IMathService声明远程服务实例（代码第1行）
 - 二是通过IMathService.Stub的asInterface()方法实现获取服务实例（代码第6行）

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务

```
1  private IMathService mathService;  
2  
3  private ServiceConnection mConnection = new ServiceConnection() {  
4      @Override  
5      public void onServiceConnected(ComponentName name, IBinder service) {  
6          mathService = IMathService.Stub.asInterface(service);  
7      }  
8      @Override  
9      public void onServiceDisconnected(ComponentName name){  
10         mathService = null;  
11     }  
12 };
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务
 - 绑定服务时，首先在Intent中指明Service所在的类，然后调用bindService()绑定服务。

```
1  final Intent serviceIntent = new Intent(RemoteMathCallerDemoActivity.this,  
                                           edu.hrbeu.RemoteMathServiceDemo.MathService.class);  
2  bindService(serviceIntent,mConnection,Context.BIND_AUTO_CREATE);
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务
 - 下面是RemoteMathCallerDemoActivity.java文件的完整代码

```
1  package edu.hrbeu.RemoteMathCallerDemo;  
2  
3  import edu.hrbeu.RemoteMathServiceDemo.IMathService;  
5  import android.app.Activity;  
6  import android.content.ComponentName;  
7  import android.content.Context;  
8  import android.content.Intent;  
9  import android.content.ServiceConnection;  
10 import android.os.Bundle;  
11 import android.os.IBinder;  
12 import android.os.RemoteException;
```


7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务

- 下面是RemoteMathCallerDemoActivity.java文件的完整代码

```
13  import android.view.View;
14  import android.widget.Button;
15  import android.widget.TextView;
16
17  public class RemoteMathCallerDemoActivity extends Activity {
18      private IMathService mathService;
20      private ServiceConnection mConnection = new ServiceConnection() {
21          @Override
22          public void onServiceConnected(ComponentName name, IBinder service) {
23              mathService = IMathService.Stub.asInterface(service);
24          }
25      }
26  }
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务

- 下面是RemoteMathCallerDemoActivity.java文件的完整代码

```
25      @Override
26      public void onServiceDisconnected(ComponentName name) {
27          mathService = null;
28      }
29  };
31  private boolean isBound = false;
32  TextView labelView;
33  @Override
34  public void onCreate(Bundle savedInstanceState) {
35      super.onCreate(savedInstanceState);
36      setContentView(R.layout.main);
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务

- 下面是RemoteMathCallerDemoActivity.java文件的完整代码

```
38     labelView = (TextView)findViewById(R.id.label);
39     Button bindButton = (Button)findViewById(R.id.bind);
40     Button unbindButton = (Button)findViewById(R.id.unbind);
41     Button computButton = (Button)findViewById(R.id.compute_add);
42
43     bindButton.setOnClickListener(new View.OnClickListener(){
44         @Override
45         public void onClick(View v) {
46             if(!isBound){
47                 final Intent serviceIntent = new
48                     Intent(RemoteMathCallerDemoActivity.this,
                        edu.hrbeu.RemoteMathServiceDemo.MathService.class);
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务
 - 下面是RemoteMathCallerDemoActivity.java文件的完整代码

```
49         bindService(serviceIntent,mConnection,Context.BIND_AUTO_CREATE);
50         isBound = true;
51     }
52 }
53 });
54
55 unbindButton.setOnClickListener(new View.OnClickListener(){
56     @Override
57     public void onClick(View v) {
58         if(isBound){
59             isBound = false;
60             unbindService(mConnection);
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务
 - 下面是RemoteMathCallerDemoActivity.java文件的完整代码

```
61         mathService = null;
62     }
63 }
64 });
66 computButton.setOnClickListener(new View.OnClickListener(){
67     @Override
68     public void onClick(View v) {
69         if (mathService == null){
70             labelView.setText("未绑定远程服务");
71             return;
72         }
```

7.3 远程服务

• 7.3.2 服务创建与调用

- 绑定和使用远程服务

- 下面是RemoteMathCallerDemoActivity.java文件的完整代码

```
73         long a = Math.round(Math.random()*100);
74         long b = Math.round(Math.random()*100);
75         long result = 0;
76         try {
77             result = mathService.Add(a, b);
78         } catch (RemoteException e) {
79             e.printStackTrace();
80         }
81         String msg = String.valueOf(a)+" + "+String.valueOf(b)+
82             " = "+String.valueOf(result);
83         labelView.setText(msg);
84     }
85 });
86 }
87 }
```


7.3 远程服务

• 7.3.3 数据传递

- 在Android系统中，进程间传递的数据包括：
 - Java语言支持的基本数据类型
 - 用户自定义的数据类型
- 为了使数据能够穿越进程边界，所有数据都必须是“可打包”的
- 对于Java语言的基本数据类型，打包过程是自动完成的
- 但对于自定义的数据类型，用户则需要实现Parcelable接口，使自定义的数据类型能够转换为系统级原语保存在Parcel对象中，穿越进程边界后可再转换为初始格式

7.3 远程服务

• 7.3.3 数据传递

• AIDL支持的数据类型表

类型	说明	需要引入
<i>Java 语言的基本类型</i>	包括boolean、byte、short、int、float和double等	否
<i>String</i>	java.lang.String	否
<i>CharSequence</i>	java.lang.CharSequence	否
<i>List</i>	其中所有的元素都必须是AIDL支持的数据类型	否
<i>Map</i>	其中所有的键和元素都必须是AIDL支持的数据类型	
<i>其它AIDL接口</i>	任何其它使用AIDL语言生成的接口类型	是
<i>Parcelable对象</i>	实现Parcelable接口的对象	是

7.3 远程服务

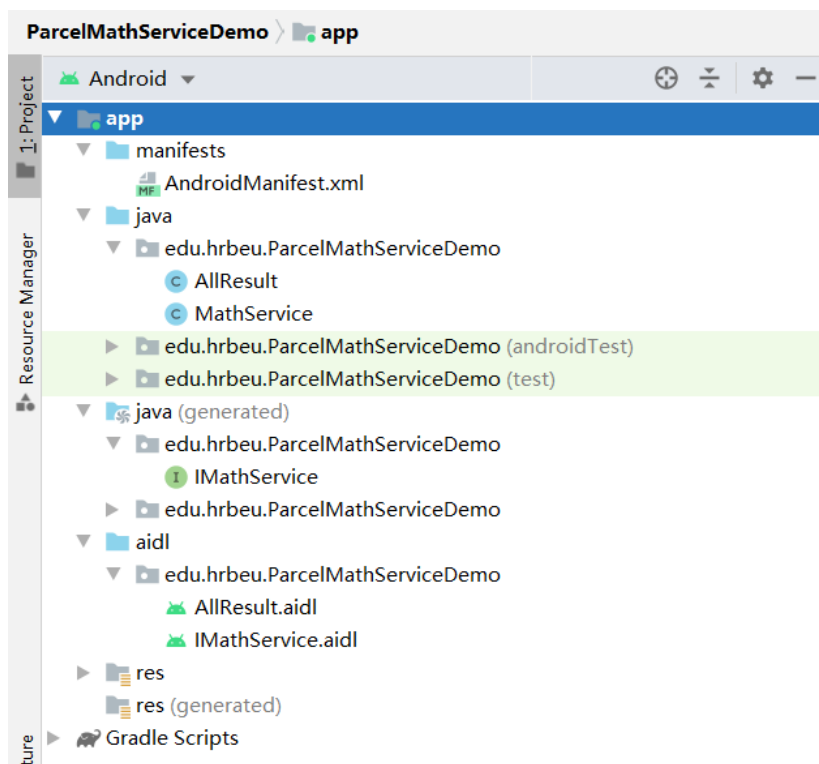
• 7.3.3 数据传递

- 下面以ParcelMathServiceDemo示例为参考，说明如何在远程服务中使用自定义数据类型
- 这个示例是RemoteMathServiceDemo示例的延续
 - 定义了MathService服务，同样可以为远程调用者提供加法服务
 - 同样也是没有启动界面，因此在模拟器的调试过程与RemoteMathServiceDemo示例相同
- 不同之处在于MathService服务增加了“全运算”功能
 - 接收到输入参数后，将向调用者返回一个包含“加、减、乘、除”全部运算结果的对象。这个对象是一个自定义的类，为了能够使其它AIDL文件可使用这个自定义类，需要使用AIDL语言声明这个类

7.3 远程服务

• 7.3.3 数据传递

- ParcelMathServiceDemo示例的文件结构如下图所示:



7.3 远程服务

• 7.3.3 数据传递

- 首先建立AllResult.aidl文件，声明AllResult类

```
1 package edu.hrbeu.ParcelMathServiceDemo;
```

```
2 parcelable AllResult;
```

- 在IMathService.aidl文件中，代码第6行为全运算增加了新的函数ComputeAll()，该函数的返回值就是在AllResult.aidl文件中定义AllResult
- 同时，为了能够使用自定义数据结构AllResult，在代码中需引入了edu.hrbeu.ParcelMathServiceDemo.AllResult包
- 第2行和第6行是新增的代码，其它的代码与RemoteMathServiceDemo示例相同

7.3 远程服务

• 7.3.3 数据传递

```
1 package edu.hrbeu.ParcelMathServiceDemo;  
2 import edu.hrbeu.ParcelMathServiceDemo.AllResult;  
3  
4 interface IMathService {  
5     long Add(long a, long b);  
6     AllResult ComputeAll(long a, long b);  
7 }
```


7.3 远程服务

• 7.3.3 数据传递

- 在AIDL文件定义完毕后，下一步来介绍如何构造AllResult类
- AllResult类除了基本的构造函数以外，还需要有以Parcel对象为输入的构造函数，并且需要重载打包函数writeToParcel()
- AllResult.java完整代码如下：

```
1  package edu.hrbeu.ParcelMathServiceDemo;
2
3  import android.os.Parcel;
4  import android.os.Parcelable;
5
6  public class AllResult implements Parcelable {
```

7.3 远程服务

• 7.3.3 数据传递

• AllResult.java的完整代码

```
7    public long AddResult;
8    public long SubResult;
9    public long MulResult;
10   public double DivResult;
11
12   public AllResult(long addRusult, long subResult, long mulResult, double divResult){
13       AddResult = addRusult;
14       SubResult = subResult;
15       MulResult = mulResult;
16       DivResult = divResult;
17   }
18
```

7.3 远程服务

• 7.3.3 数据传递

• AllResult.java的完整代码

```
19     public AllResult(Parcel parcel) {  
20         AddResult = parcel.readLong();  
21         SubResult = parcel.readLong();  
22         MulResult = parcel.readLong();  
23         DivResult = parcel.readDouble();  
24     }  
25  
26     @Override  
27     public int describeContents() {  
28         return 0;  
29     }  
30
```

7.3 远程服务

• 7.3.3 数据传递

• AllResult.java的完整代码

```
31     @Override
32     public void writeToParcel(Parcel dest, int flags) {
33         dest.writeLong(AddResult);
34         dest.writeLong(SubResult);
35         dest.writeLong(MulResult);
36         dest.writeDouble(DivResult);
37     }
38
39     public static final Parcelable.Creator<AllResult> CREATOR =
40         new Parcelable.Creator<AllResult>(){
41         public AllResult createFromParcel(Parcel parcel){
42             return new AllResult(parcel);
```

7.3 远程服务

• 7.3.3 数据传递

- AllResult.java的完整代码

```
43     }  
44     public AllResult[] newArray(int size){  
45         return new AllResult[size];  
46     }  
47 };  
48 }
```

7.3 远程服务

• 7.3.3 数据传递

- 代码第6行说明了AllResult类继承于Parcelable
- 代码第7行到第10行用来保存全运算的运算结果
- 第12行是AllResult类的基本构造函数
- 第19行也是类的构造函数，支持Parcel对象实例化AllResult
- 代码第32行的writeToParcel()是“打包”函数，将AllResult类内部的数据，按照特定的顺序写入Parcel对象，写入的顺序必须与构造函数的读取顺序一致（代码第20行到第23行）
- 第39行实现了静态公共字段Creator，用来使用Parcel对象构造AllResult对象

7.3 远程服务

• 7.3.3 数据传递

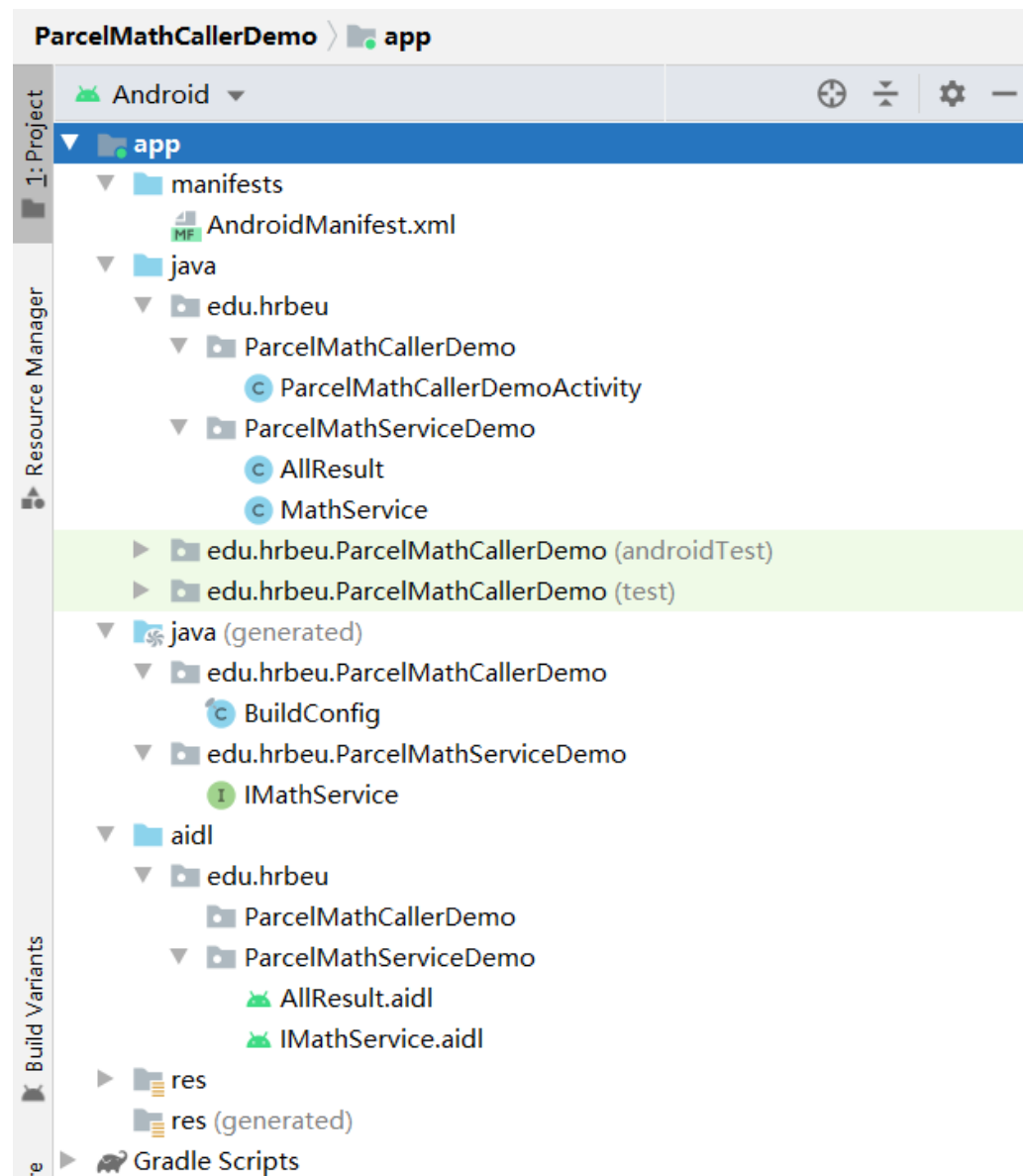
- 在MathService.java文件中，增加了用来进行全运算的ComputeAll()函数，并将运算结果保存在AllResult对象中
- MathService.java中的ComputeAll()函数实现代码如下：

```
1  @Override
2  public AllResult ComputeAll(long a, long b) throws RemoteException {
3      long addResult = a + b;
4      long subResult = a - b;
5      long mulResult = a * b;
6      double divResult = (double) a / (double) b;
7      AllResult allResult = new AllResult(addResult, subResult, mulResult, divResult);
8      return allResult;
9  }
```

7.3 远程服务

• 7.3.3 数据传递

- ParcelMathCallerDemo示例是ParcelMathServiceDemo示例中MathService服务的调用
- ParcelMathCallerDemo文件结构如右图所示:
 - 其中，AllResult.aidl、AllResult.java和IMathService.aidl文件务必与ParcelMathServiceDemo示例的三个文件完全一致，否则会出现错误



7.3 远程服务

• 7.3.3 数据传递

- 下图是ParcelMathCallerDemo用户界面



7.3 远程服务

• 7.3.3 数据传递

- 上图的ParcelMathCallerDemo界面中可以发现，原来的“加法运算”按钮改为了“全运算”按钮，运算结果显示在界面的上方
- 下面也仅给出ParcelMathCallerDemo.java文件与RemoteMathCallerDemo示例RemoteMathCallerDemoActivity.java文件不同的代码段
 - 定义了“全运算”按钮的监听函数
 - 随机产生输入值
 - 调用远程服务
 - 获取运算结果，并将运算结果显示在用户界面上

7.3 远程服务

•7.3.3 数据传递

```
10 computAllButton.setOnClickListener(new View.OnClickListener(){
11     @Override
12     public void onClick(View v) {
13         if (mathService == null){
14             labelView.setText("未绑定远程服务");
15             return;
16         }
17         long a = Math.round(Math.random()*100);
18         long b = Math.round(Math.random()*100);
19         AllResult result = null;
20         try {
21             result = mathService.ComputeAll(a, b);
22         } catch (RemoteException e) {
23             e.printStackTrace();
```

7.3 远程服务

• 7.3.3 数据传递

```
24     }
25     String msg = "";
26     if (result != null){
27         msg += String.valueOf(a)+" + "+String.valueOf(b)+" =
           "+String.valueOf(result.AddResult)+"\n";
28         msg += String.valueOf(a)+" - "+String.valueOf(b)+" =
           "+String.valueOf(result.SubResult)+"\n";
29         msg += String.valueOf(a)+" * "+String.valueOf(b)+" =
           "+String.valueOf(result.MulResult)+"\n";
30         msg += String.valueOf(a)+" / "+String.valueOf(b)+" =
           "+String.valueOf(result.DivResult);
31     }
32     labelView.setText(msg);
33 }
34 });
```




习题：

- 1.简述Service的基本原理和用途。
- 2.编程建立一个简单的进程内服务，实现比较两个整数大小功能。服务提供Int Compare(Int, Int)函数，输入两个整数，输出较大的整数。
- 3.使用AIDL语言实现功能与第2题相同的跨进程服务。

THANKS

谢 谢 观 看

