# 中山大学移动信息工程学院本科生实验报告

| 年级 | 1501 | 专业（方向） | 移动（互联网） |
| --- | --- | --- | --- |
| 学号 | 15352015 | 姓名 | 曹广杰 |
| 电话 | 13727022190 | Email | 1553118845@qq.com |

## Content

## 一、实验题目

# Appwidget及Broadcast 使用

## 二、实现内容

本次实验模拟实现一个播放器。

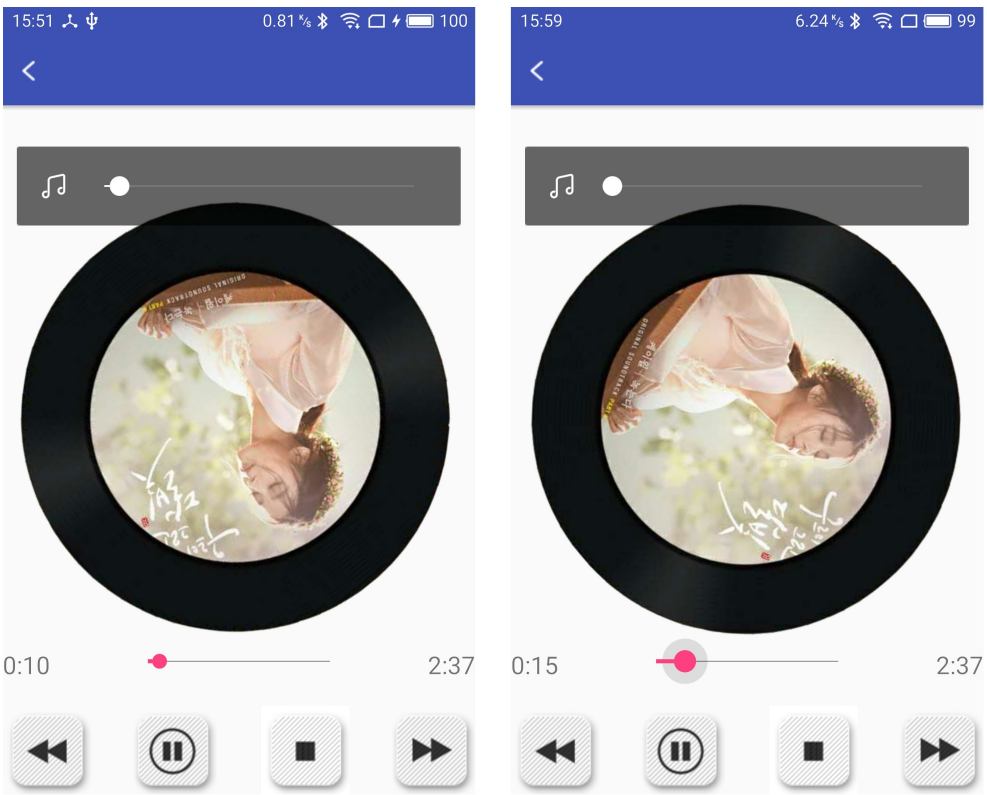
- 学会使用 MediaPlayer
- 学会简单的多线程编程，使用 Handle 更新 UI
- 学会使用 Service 进行后台工作
- 学会使用 Service 与 Activity 进行通信

实现方式要求:

- 播放、暂停，停止，退出功能
- 后台播放功能
- 进度条显示播放进度、拖动进度条改变进度功能
- 播放时图片旋转，显示当前播放时间功能

## 三、课堂实验结果

### 实验截图




### 实验步骤以及关键代码

实现布局动画的效果

在布局文件中，设置了一些触发的按钮。会根据用户的操作反映出不同的动画效果。这里的动画效果为音乐碟片的旋转动画：

音乐碟片的转动

首先需要为音乐碟片选择合适的组件，这里笔者接受建议使用ObjectAnimator。该组件将会实现对ImageView的动画操作：

```
private void setplate(){
        ImageView imageView = (ImageView) findViewById(R.id.plate_img);
        animator = ObjectAnimator.ofFloat(imageView, "rotation", 0f, 360f);
        animator.setDuration(10000);
        animator.setInterpolator(new LinearInterpolator());
        animator.setRepeatCount(ValueAnimator.INFINITE);
    }
```

在上述代码中，对碟片的布局对象获取，然后通过ObjectAnimator的函数接口对布局文件对应的图像进行操作，包括但不限于设置动画的运行时间、运行方式以及重复次数。

**实现Activity与service的联系**

本次实验的要求为根据Activity界面上用户对按钮的操作实现对后台service的控制。首先需要声明service的存在，在Manifest的application中，添加对service的声明语句：

```
        <service
            android:name=".PlayerService"
            android:exported="true"/>
```

接下来的实现分为两个步骤，Activity与service保持通信，service返回数据给Activity。

**Activity与service通信**

笔者在本次实验中使用的通信方式是绑定Activity与service，以便进行后续操作。首先在OnCreate函数中初始化实现操作：

```
        Intent intent = new Intent(this, PlayerService.class);
        startService(intent);
        bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE);
```

该部分中的函数bindService正是对Activity与service绑定的函数调用。此时绑定的操作可能成功也可能失败，这正是参数serviceConnection存在的意义。serviceConnection将会根据绑定的结果采取不同的操作：

```
private IBinder binder;
ServiceConnection serviceConnection = new ServiceConnection() {
  @Override
  public void onServiceConnected(ComponentName name, IBinder service) {
    binder = service;
  }
  @Override
  public void onServiceDisconnected(ComponentName name) {
    serviceConnection = null;
  }
};
```

这里一定要重写两个函数：onServiceConnected与onServiceDisconnected用于分别处理绑定操作中可能出现的两种情况。绑定若成功，则本地的binder获得service的binder所有权，可以实现在activity与service之间的调度，否则免谈。

在任何时候，如果Activity需要向service传递数据，则可以通过以下的方式：

```
try {
    binder.transact(102, Parcel.obtain(), Parcel.obtain(), 0);
} catch (Exception e){
    e.printStackTrace();
}
```

这一切需要发生在binder真正成为activity与service的服务代理之后，也就是绑定操作成功之后。binder的通信机制就是使用函数transact。函数transact的参数意义如下：

- 第一个参数code：由开发者自己定义，只要使用了这个函数，service端就会对不同的CODE进行不同的操作；
- 第二个参数data：Parcel类型，用于发送数据，分别使用特定的函数进行书写与读取。
- 第三个参数reply：Parcel类型，来自service端的参数，可以使用特定的函数读取内部的信息。
- 最后一个参数写0就行，我也不知道怎么用。

**service与Activity的通信**

前文述说IBinder可以实现针对activity与service之间的通信。但是用于IBinder有虚函数的存在所以需要在service类中基于原有类完成自己实现的类，以便后续的调用和接口的实现：

```
public class mybinder extends Binder {
  @Override
  protected boolean onTransact(int code, Parcel data, Parcel reply, int flags) throws
RemoteException {
    switch (code){
      case 101:
//        TODO
        break;
      default:
//        TODO
        break;
    }
    return super.onTransact(code, data, reply, flags);
  }
}
```

此处使用switch语句对code参数进行了分类处理的操作，开发者可以根据不同的情况进行不同的操作。至于回传数据，与activity中的情形基本一致，都是使用Parcel作为容器，完成对数据的封装和传输。不同的是，这里操作的参数是reply，而在发送数据过程中是修改参数data。

添加进度条

添加进度条的操作需要先修改布局文件，添加对于seekBar——进度条的组件信息：

```
<SeekBar
        android:id="@+id/seekbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
```

至于布局信息需要开发者自己定义。

进度条有两个过程需要考虑：随着播放进度的变化而变化，通过进度条的拖动操作影响service中播放的进度。

进度条与service同步

这个过程实际上就是一个service与activity通信的过程，控件对象得到来自service的信息，并将其展示在activity的界面上：

```
try {
  binder.transact(105, data, reply, 0);
  int currentP = reply.readInt()/1000;
  seekBar.setProgress(currentP);
} catch (Exception e){
  e.printStackTrace();
}
```

这时候进度条可以更新一次，但是如果为了实现实时更新，或者频繁刷新，则需要使用更加复杂的机制，在另一个线程中添加更新动画的操作：

```
Thread thread = new Thread(){
    @Override
    public void run() {
      super.run();
      while (true){
      // 设置时间间隔
        if (serviceConnection!= null&&hasPermission){
          handler.obtainMessage(111).sendToTarget();// 更新操作对应的代码
        }
      }
    }
  };
  thread.start();
```

此时已经创建了一个新的线程，并在线程中按照一定的间隔提交申请操作的代码，Handler接收到对应的代码之后需要进行对应的操作：

```
private void updateUI(){
  final Handler handler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
      super.handleMessage(msg);
      switch (msg.what){
        case 111:// 笔者编辑的更新代码
//            更新操作
          break;
      }
    }
  };
}
```

内部存在一些细小的实现方式，如实现延时，以保证当前的操作已经完成或者控制更新的频率：

```
        try {
          Thread.sleep(100);
        } catch (Exception e){
          e.printStackTrace();
        }
```

进度条的拖动操作

进度条拖动操作控制service属于activity向service的通信。从seekBar的角度来说，首先需要设置一个监听器以获取当前的用户操作，并在内部完成需要自己定义的三个函数：

```
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
  @Override
  public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) { }
  @Override
  public void onStartTrackingTouch(SeekBar seekBar) { }
  @Override
  public void onStopTrackingTouch(SeekBar seekBar) { }
});
```

从上往下分别表示拖动进度条的中、前、后阶段的操作，正常的Android版本，修改onProcessChange函数即可，因为监听器是实时监听的：

```
    if (fromUser) {
      Parcel data = Parcel.obtain();
      try {
        播放时间【TextView】.setText(duration/60 +":"+ duration%60);
        binder.transact(104, process, Parcel.obtain(), 0);
      } catch (Exception e){
        e.printStackTrace();
      }
```

### 实验遇到困难以及解决思路

#### 虚拟机中不能添加MP3文件

解决方案：通过查询获知是由于权限不足，通过Windows操作系统的命令行操作修改权限，就可以访问虚拟机的data文件夹。

#### 动画效果不能暂停

解决方案：笔者之前使用的控件是Animation，原始控件中没有暂停的接口，将控件修改为ObjectAnimator即可调用对应的接口。

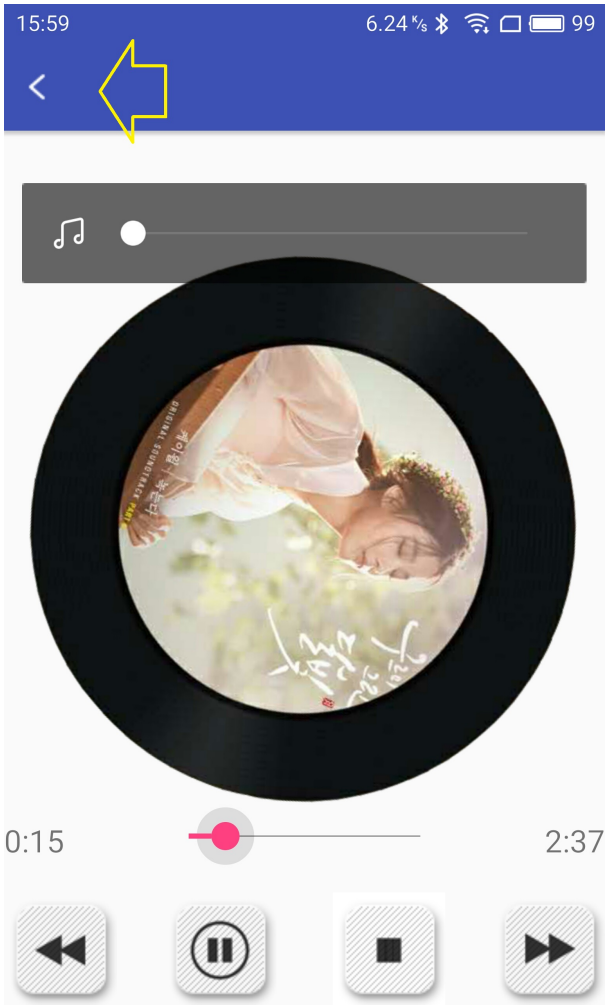
#### 绑定操作导致卡顿

解决方案：使用debug工具的时候，绑定操作处会直接终止，没有回调函数的运行。但是Android的设计应该是允许debug每一行代码的，无论是回调函数还是非回调函数。所以，推测出可能是Manifest文件中的绑定操作出问题，质疑当前的开发环境对debug来讲毫无意义。

## 四、课后实验结果

#### 添加toolbar控件

笔者将返回键（quit）添加到了左上角的toolbar上，修改了原有的布局界面：




## 五、实验思考及感想

本次实验的主要任务是service以及多线程的设计。实验中遇到过一些问题，发现debug也并不是一件非常简单的事情。

### xml的bug很难找出

可能是由于版本不支持，或者是引用的控件是不存在的，这种情况都会直接导致程序闪退，不留任何debug的余地。所以尽量使用提供的或者推荐的控件。

### 报错信息的提取

报错信息有时候有许多，其中一定要提取最为关键的信息，才可能找到对应的解决方案。有时候实在是看不懂，也一定要注意第一句提示信息，说不定一阵所搜之后就别有洞天。

### 全局变量的更新问题

全局变量在局部函数中进行修改，跳出函数后仍然是原来的函数值，原因是在声明的时候将其设置为了static状态。取消这一属性，就可以在局部函数中修改，而在全局范围内作用。

### 实现绑定操作

IBinder的信息在网络上很难找到，主要是使用的范围比较受限，该工具主要用作Linux的开发中，而在Android studio中的使用较少，所以如果需要获得相应的信息，就需要针对性地查询，询问某些指定性的人物，某些关注者。

service的布局问题，内容问题，触发问题都是相对独立的，完全可以各自独立地完成。此外，IBinder的使用还是相对独立的，至于内容的设定则是在重构函数内部实现。这就是开发体系结构上的层次化，使得开发过程结构清晰，分级完整。而拼接组合也非常有趣，尽管以上面对的函数功能千差万别，或者是系统本身的，或者是自己定义的，但是它们在传入数据的时候，都在遵循函数运行的基本架构——每一个函数的使用，就是简单清晰、容易理解的。至于底层的部分，很大一部分并不需要我们去操作。然而，正是由于不了解底层的实现，又是使用其他人完成的包，兼容性的问题只恐怕不能够完全地考虑到，因此，对于不同的机型，可能会有一些无法解释的bug，这就使得同一份代码在不同的平台上尽管可以运行，但是运行的效果却未必让人满意。