# 中山大学移动信息工程学院本科生实验报告

## （2017 年秋季学期）

课程名称：移动应用开发　　任课教师： 郑贵锋

| 年级 | 2015 | 专业（方向） | 互联网 |
|---|---|---|---|
| 学号 | 15352194 | 姓名 | 梁杰鑫 |
| 电话 | 15113959962 | Email | Alcanderian@gmail.com |
| 开始日期 | 2017.11.25 | 完成日期 | 2017.12.1 |

## 一、 实验题目

实现一个简单的播放器。

1. 学会使用 MediaPlayer；
2. 学会简单的多线程编程，使用 Handle 更新 UI；
3. 学会使用 Service 进行后台工作；
4. 学会使用 Service 与 Activity 进行通信。

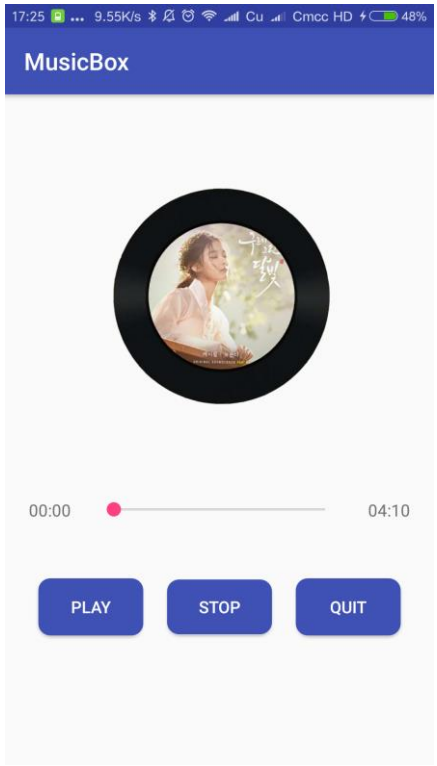
## 二、 实现内容

要求功能有：

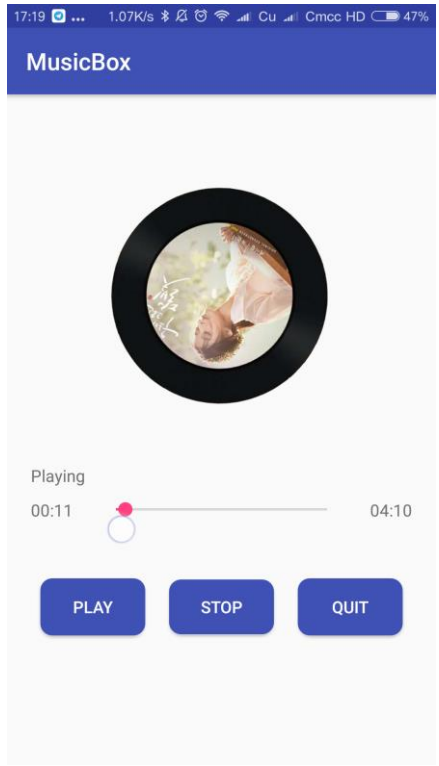
1. 播放、暂停，停止，退出功能；
2. 后台播放功能；
3. 进度条显示播放进度、拖动进度条改变进度功能；
4. 播放时图片旋转，显示当前播放时间功能；

## 三、 课堂实验结果

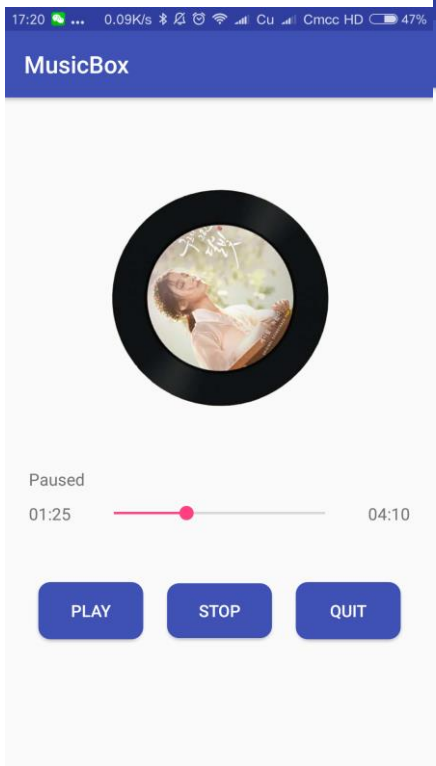
（1） 实验截图



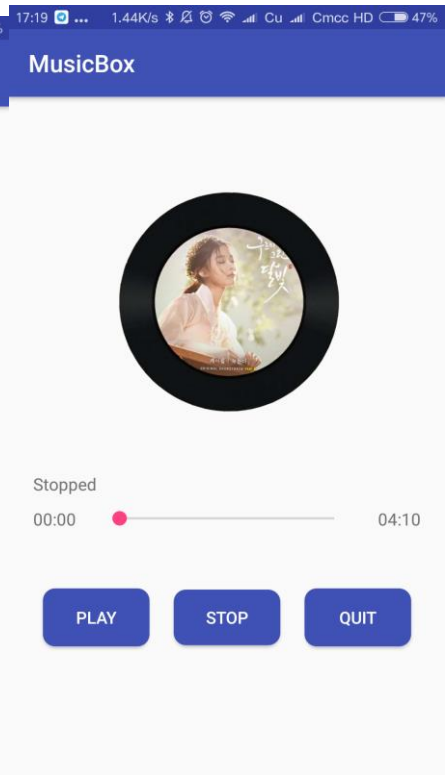

初始界面




播放状态




暂停状态




停止状态

（2） 实验步骤以及关键代码

- 申请权限

在本次实验中，我使用了内置 SD 卡作为音乐的存放路径，所以需要申请读取 SD 卡的权限。首先我们在 manifest 中添加我们需要申请的权限

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

因为安卓 6.0 上的系统是需要动态申请权限的，所以我们新建一个申请权限的类。

```
public class MyPermission {

    private static final int REQUEST_EXTERNAL_STORAGE = 1;
    private static String[] PERMISSIONS_STORAGE = {
            Manifest.permission.READ_EXTERNAL_STORAGE,
            Manifest.permission.WRITE_EXTERNAL_STORAGE
    };

    public static void verifyStoragePermissions(Activity activity) {
        int permission = ActivityCompat.checkSelfPermission(activity,
                Manifest.permission.READ_EXTERNAL_STORAGE);

        if (permission != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(activity,
                    PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
        }
    }
}
```

之后我们可以通过调用该类的静态方法来动态申请权限，并在 MainAcitivy 中重写 onRequestPermissionResult 函数。

```
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                           @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (!(grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED))
            onDestroy();
    }
```

- MusicService

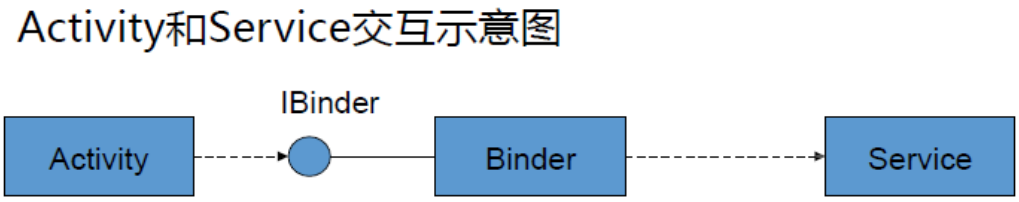
MusicService 是一个继承 Service 的类，和 Activity 一样需要在 manifest 中注册才能运行。和 activity 不同的是，service 是运行在非 UI 线程的类，通常用于音乐播放，后台下载等任务。本次实验中我们用 Service 来完成后台播放音乐的任务，播放音乐需要调用 MediaPlayer，所以要在 Service 中添加静态的 MediaPlayer 成员。

为什么不要非静态呢，我们可能会多次调用 Service 的构造函数，然后可能生成多个 MediaPlayer，这样会造成同时有多首歌在后台播放的情况。

```
public class MusicService extends Service {

    public static MediaPlayer mp = new MediaPlayer();
    public IBinder my_binder = new MusicBinder();
    public static int state = 3;

    public MusicService() {
        try {
            mp.setDataSource(Environment.getExternalStorageDirectory() + "/melt.mp3");
            mp.prepare();
            mp.setLooping(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public IBinder onBind(Intent intent) { return my_binder; }
```

- Binder

Binder 是 activity 与 service 之间用于通信的工具。




我们通过重写 Biner 的传输函数来进行自定义的通信逻辑。

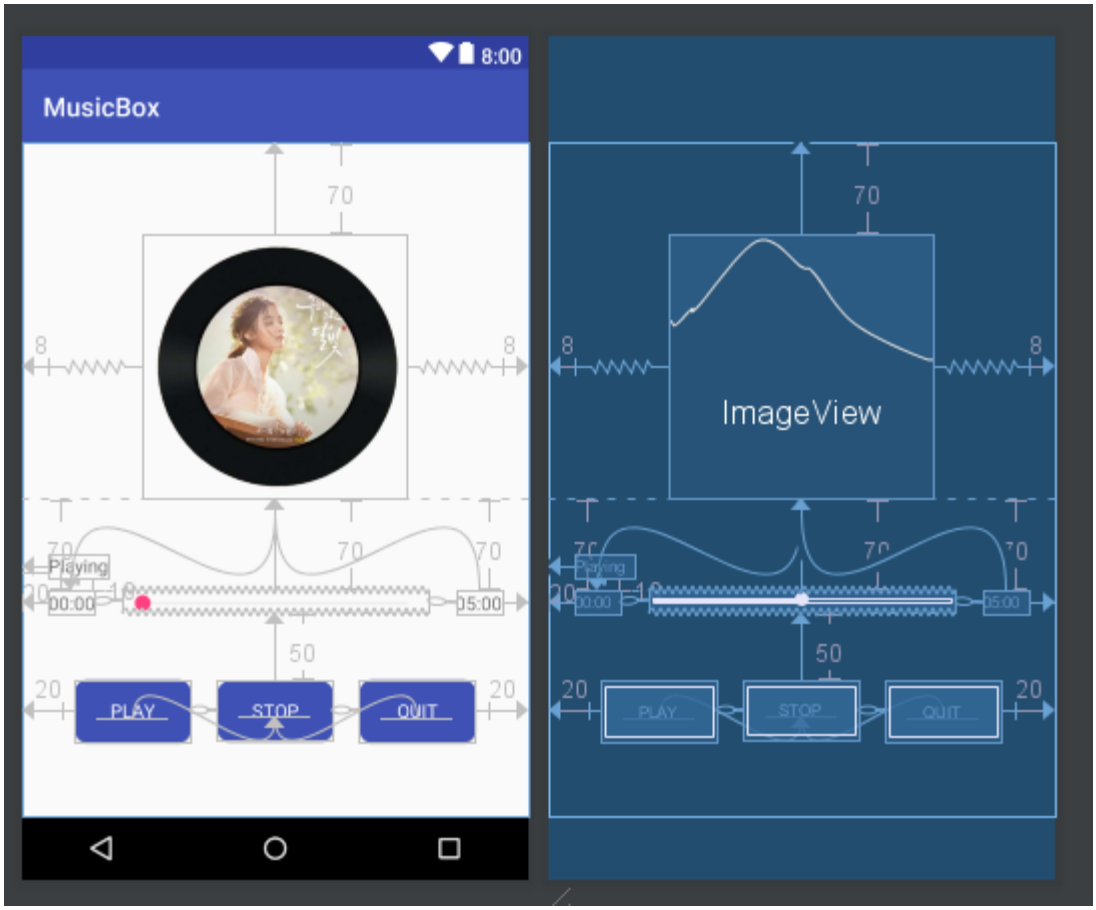
```
@Override
public IBinder onBind(Intent intent) { return my_binder; }

public class MusicBinder extends Binder {
    @Override
    public boolean onTransact(int code, Parcel data, Parcel reply, int flags)
            throws RemoteException {
        switch (code) {
            case 1: // start/pause
                if (mp.isPlaying()) {
                    mp.pause();
                    state = 1;
                    reply.writeInt(state);
                } else {
                    mp.start();
                    state = 2;
                    reply.writeInt(state);
                }
                break;
            case 2: // stop
                if (mp != null && state != 0) {
                    mp.stop();
                    state = 0;
                    reply.writeInt(state);
                    try {
                        mp.reset();
                        mp.setDataSource(Environment.getExternalStorageDirectory() + "/melt.mp3");
                        mp.prepare();
                        mp.setLooping(true);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                break;
            case 3: // exit
                mp.release();
                mp = null;
                break;
            case 4: // refresh
                reply.writeInt(mp.getCurrentPosition());
                break;
            case 5: // move the bar
                mp.seekTo(data.readInt());
                break;
            case 6: // get max
                reply.writeInt(mp.getDuration());
                break;
            case 7:
                reply.writeInt(state);
                break;
        }
        return super.onTransact(code, data, reply, flags);
    }
}
```

值得注意的是，因为要调用 Service 里面的资源，Binder 应该成为 service 的成员类，并且是非静态的。参数 code 用于获取指令的编码，每个编码对应的逻辑我们可以自己制定。两个 Parcel 类是用于数据通信的，我们可以在里面放入 int，double，甚至 bundle。

本次实验中，我们利用编码来命令 service 播放、暂停、停止音乐。还用于获取和设置音乐播放进度，以及获取音乐的播放状态。

- MainActivity

MainActivity 的布局如下




我们知道，播放器的进度条是实时更新的，但是我们不能在 UI 线程来实时更新 UI，这样会导致用户的操作不流畅。

我们利用 Java 的 Thread 调用一个 Handle 来向 service 询问音乐的播放进度，然后更新进度条。在 Thread 中除了更新进度条之外，我们还需要在 thread 中更新图片旋转的状态，因为音乐的播放状态是由 service 及控制的而不是 UI 线程控制的，所以要不停地询问 service，而不是 UI 触发事件去更改播放状态。

Thread 和 Handle 的定义如下

```
final Handler hd_refresher = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        switch (msg.what) {
            case 1:
                if (is_changing_pos == 0) {
                    try {
                        Parcel reply = Parcel.obtain();
                        ib_binder.transact( code: 4,  data: null, reply,  flags: 0);
                        int position = reply.readInt();
                        skb_pos.setProgress(position);
                        reply = Parcel.obtain();
                        ib_binder.transact( code: 7,  data: null, reply,  flags: 0);
                        play_status = reply.readInt();
                        refreshStatus();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                break;
        }
    }
};
```

可以看到是同时更新进度条和播放状态的，refreshStatus 的函数如下：

```
public void refreshStatus() {
    switch (play_status) {
        case 0:
            tv_status.setText("Stopped");
            oa_collection.pause();
            break;
        case 1:
            tv_status.setText("Paused");
            oa_collection.pause();
            break;
        case 2:
            tv_status.setText("Playing");
            if (is_init == 1)
                oa_collection.start();
            else
                oa_collection.resume();
            is_init = 0;
            break;
        case 3:
            tv_status.setText("");
            break;
    }
}
}
```

然后是 Thread 的定义，每隔 100ms 调用一次 Handle

```
Thread t_refresh = run() -> {
        while (true) {
            try {
                Thread.sleep( millis: 100);
            } catch (Exception e) {
                e.printStackTrace();
            }
            if (serv_conn != null) {
                hd_refresher.obtainMessage( what: 1).sendToTarget();
            }
        }
};
t_refresh.start();
```

最后是在 UI 界面绑定 Service，以及增加按钮的逻辑。

定义 ServiceConnection，并用 Intent 启动 Service，将 Intent 绑定到 ServiceConnection 上

```
serv_conn = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        Log.d( tag: "service",  msg: "connected");
        ib_binder = service;
        try {
            Parcel reply = Parcel.obtain();
            ib_binder.transact( code: 6,  data: null, reply,  flags: 0);
            int position = reply.readInt();
            skb_pos.setMax(position);
            tv_end.setText(sdf_pos.format(position));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onServiceDisconnected(ComponentName name) { serv_conn = null;
};

Intent it_serv = new Intent( packageContext: this, MusicService.class);
startService(it_serv);
bindService(it_serv, serv_conn, Context.BIND_AUTO_CREATE);
```

进度条的逻辑。Is_changeing_pos 变量是用于判断用户是否在拖动进度条，拖动的时候，Thread 不再向 Service 询问播放进度，当拖动停止的时候，就向 Service 发送跳转指令。

```
skb_pos.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        tv_now.setText(sdf_pos.format(progress));
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) { is_changing_pos = 1; }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
        try {
            Parcel data = Parcel.obtain();
            data.writeInt(skb_pos.getProgress());
            ib_binder.transact( code: 5, data,  reply: null,  flags: 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        is_changing_pos = 0;
    }
});
```

## 四、 思考及感想

本次实验其实在一定程度上对 MVC 架构进行了实现，将 UI 以及后台逻辑分离，异步通信，可以为用户带来更加流畅的体验。在商业化 APP 中，是一定会采用 MVC 架构的，尤其是在淘宝这样的 APP 上，有海量的 UI 数据和网络通信，还有数据的存取，就要将各个模块的工作分离出来，才能使得体验流畅。