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）	移动互联网
学号	15352344	姓名	吴文标
电话	13112312320	Email	wwb.bill@qq.com
开始日期	2017.11.13	完成日期	2017.11.14

一、 实验题目

服务与多线程--简单音乐播放器

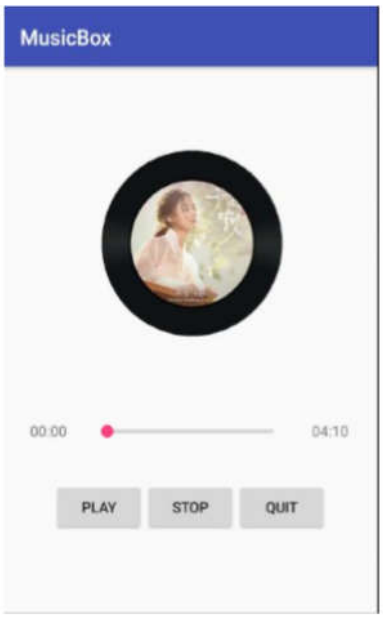
二、 实验目的

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。

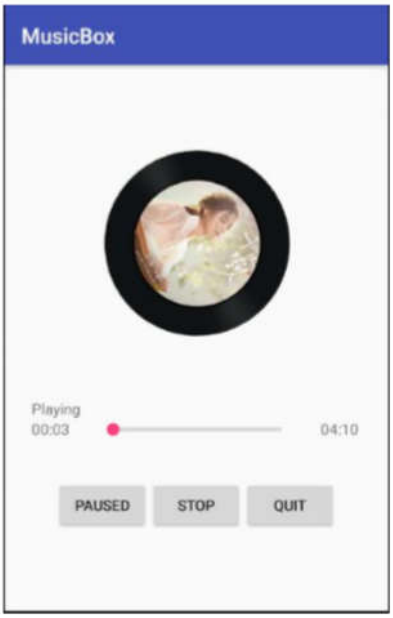
三、 实现内容

实现一个简单的播放器，要求功能有：

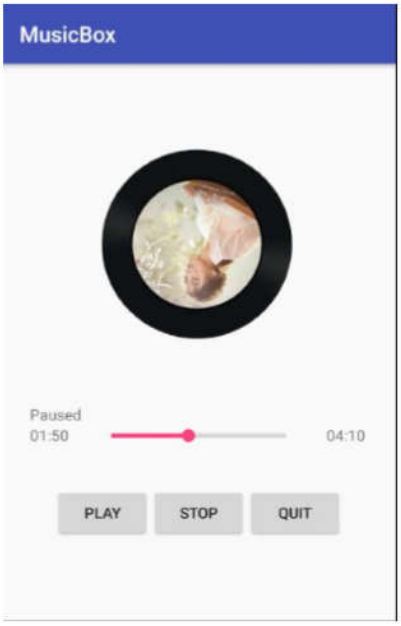
1. 播放、暂停，停止，退出功能；
2. 后台播放功能；
3. 进度条显示播放进度、拖动进度条改变进度功能；
4. 播放时图片旋转，显示当前播放时间功能；



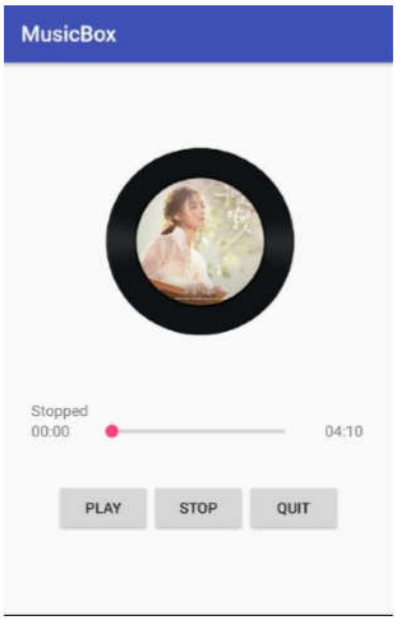
打开程序主页面



开始播放



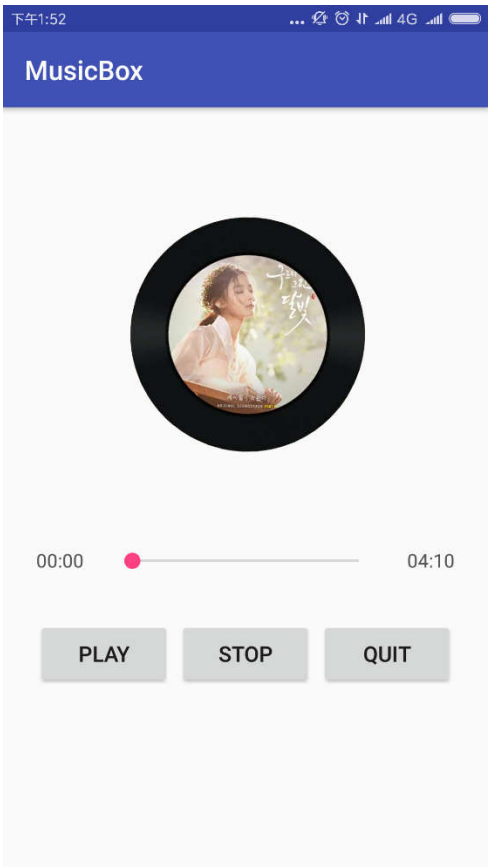
暂停

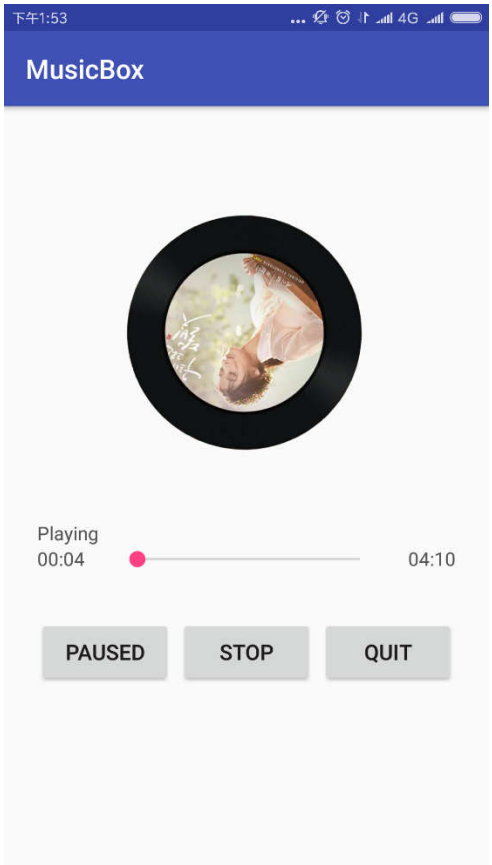


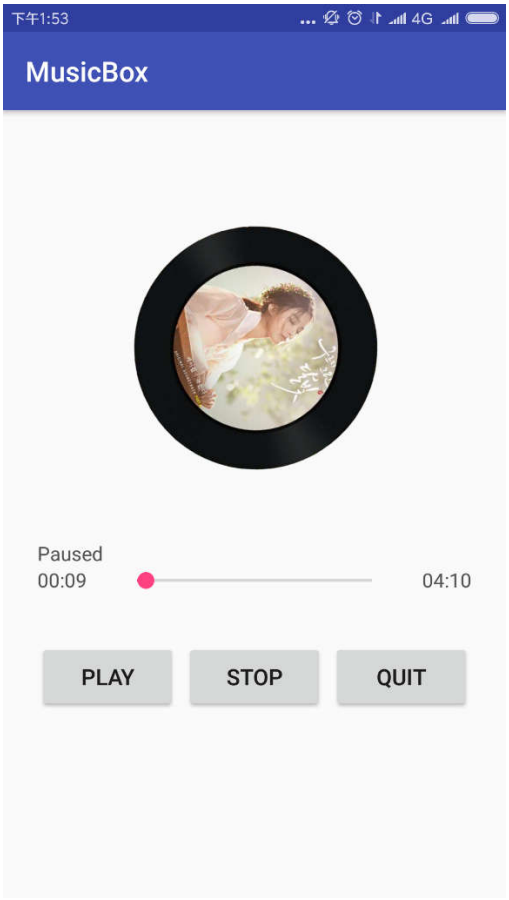
停止

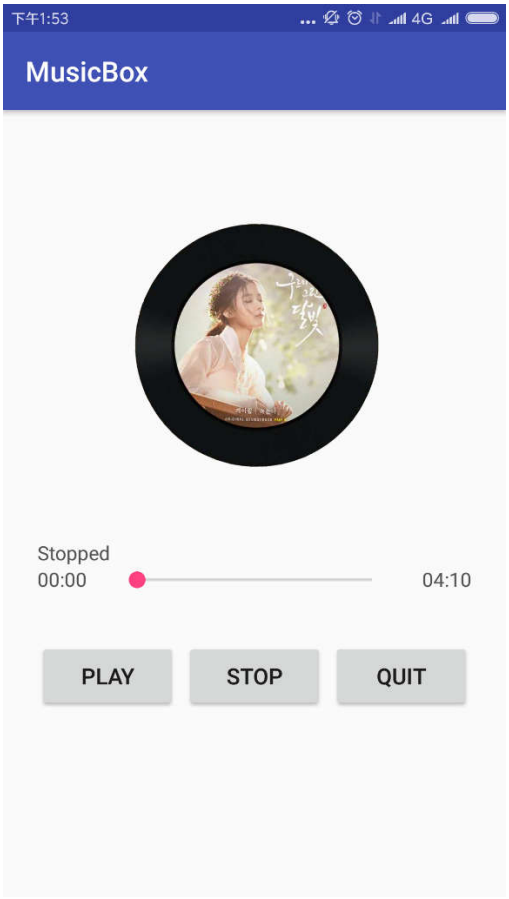
四、 课堂实验结果

（1） 实验截图





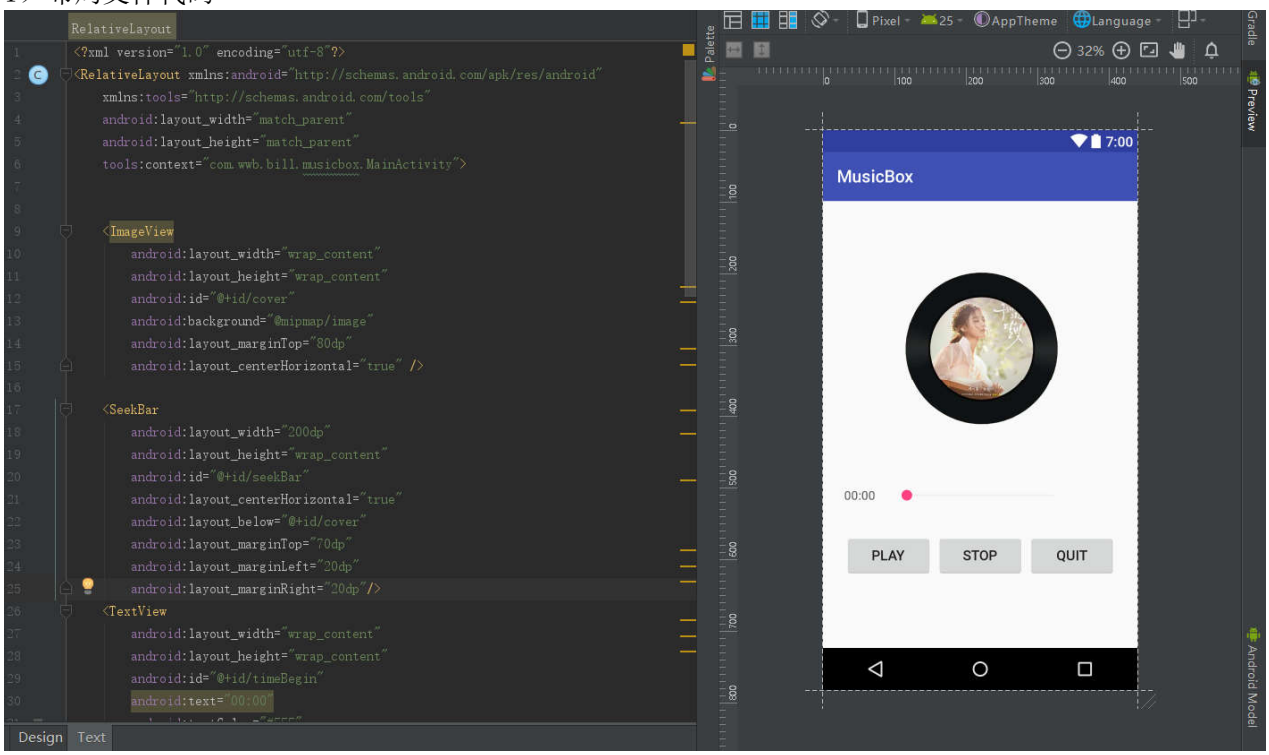




（2） 实验步骤以及关键代码

步骤 1：新建 project，界面实现。

1）布局文件代码



步骤 2：创建 Service 类。

1） 在清单文件 AndroidMainfest.xml 中进行注册

添加以下部分即可：

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

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

android:exported--是否支持其它应用调用当前组件。

2） 创建 Service 子类 MusicService。

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

onBind 是必须实现的一个方法，这个方法其实就是用于和 Activity 建立关联的。

这里我们新增了一个 MyBinder 类继承自 Binder 类，用于实现 IBinder 类：

```
//定义onBind方法所返回对象1
public final IBinder binder = new MyBinder();
//继承Binder实现IBinder类
public class MyBinder extends Binder {
    @Override
    protected boolean onTransact(int code, Parcel data, Parcel reply, int flags) throws RemoteException{
        switch (code){
            case 101:
                //播放按钮，服务处理函数
                playAndpause();
                break;
            case 102:
                //停止按钮，服务处理函数
                stop();
                break;
            case 103:
                //退出按钮，服务处理函数
                quit();
                break;
```

注：创建一个 Binder 对象的时候，Binder 内部就会启动一个隐藏线程，该线程的主要作用就是接收 Binder 驱动发送来的消息。因为我们客户端在调用服务器端的时候并不能直接调用其响应的类和方法，只能通过 Binder 驱动来调用，当服务端的隐藏线程收到 Binder 驱动发来的消息之后，就会回调服务端的 onTransact 方法。该方法接收四个参数，这四个参数都是客户端的 Binder 传来的，第一个参数用来指定客户端要调用服务端的哪一个方法，第二个参数是客户端传来的参数，第三个参数表示服务端返回的参数，最后一个 flags 表示客户端的调用是否有返回值，0 表示服务端执行完成之后有返回值，1 表示服务端执行完后没有返回值。

3） 客户端获取 Binder 驱动中的 Binder 对象，然后调用该对象中的 transact 方法进行数据传递，代码模板如下：

```
try{
    int code = 101;
    Parcel data = Parcel.obtain();
    Parcel reply = Parcel.obtain();
    mBinder.transact(code, data, reply, 0);
} catch (RemoteException e) {
    e.printStackTrace();
}
```

4） 客户端绑定服务端（这里是 MainActivity）

```
{//绑定Service
    Intent intent = new Intent(this, MusicService.class);
    startService(intent);//启动Service
    bindService(intent, sc, BIND_AUTO_CREATE);  //绑定指定的Service
}
```

bindService 成功后回调 onServiceConnected 函数，通过 IBinder 获取 Service 对象，实现 Activity 与 Service 的绑定：

```
private ServiceConnection sc = new ServiceConnection() {
    //当Activity和Service连接成功时回调该方法
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        Log.d("service", "connected");
        mBinder = service;
    }
    @Override
    public void onServiceDisconnected(ComponentName name) { sc = null; }
};
```

5） Activity 销毁时解除绑定：

```
@Override
public void onDestroy() {
    super.onDestroy();
    unbindService(sc);
}
```

步骤 3：MediaPlayer 使用与设置

1） 创建 MediaPlayer 对象并初始化

```
public static MediaPlayer mediaPlayer = new MediaPlayer();
public MusicService() {
    try{
        //模拟器测试时歌曲路径mediaPlayer.setDataSource("/data/elt.mp3");
        //手机测试路径
        mediaPlayer.setDataSource(Environment.getExternalStorageDirectory().getAbsolutePath() + "/melt.mp3");
        mediaPlayer.prepare();
        mediaPlayer.setLooping(true);
    } catch (Exception e){
        e.printStackTrace();
    }
}
```

2) 播放器相关功能实现：

```
//播放按钮，服务处理函数
public void playAndpause(){
    if(mediaPlayer.isPlaying()){
        mediaPlayer.pause();
    }else{
        mediaPlayer.start();
    }
}
```

```
//停止按钮，服务处理函数
public void stop(){
    if(mediaPlayer != null){
        mediaPlayer.stop();
        try{
            mediaPlayer.prepare();
            //按钮事件被触发后，进度条回到最开始的位置
            mediaPlayer.seekTo(0);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

```
//退出按钮，服务处理函数
public void quit(){
    mediaPlayer.stop();
    mediaPlayer.release();
}
```

```
//拖动进度条，服务处理函数
public void TrackingTouch( int position) {
    mediaPlayer.seekTo(position);
}
```

函数使用（Binder 驱动处理）：

```
protected boolean onTransact(int code, Parcel data, Parcel reply, int flags) throws RemoteException{
    switch (code){
        case 101:
            //播放按钮，服务处理函数
            playAndpause();
            break;
        case 102:
            //停止按钮，服务处理函数
            stop();
            break;
        case 103:
            //退出按钮，服务处理函数
            quit();
            break;
        case 104:
            //界面刷新。服务返回数据函数
            int playTimeNow = mediaPlayer.getCurrentPosition();
            int playTime = mediaPlayer.getDuration();
            int isPlay = 0;
            if(mediaPlayer.isPlaying())isPlay = 1;
            reply.writeInt(playTimeNow);
            reply.writeInt(playTime);
            reply.writeInt(isPlay);
            return true;
        case 105:
            //拖动进度条，服务处理函数
            TrackingTouch(data.readInt());
            break;
    }
    return super.onTransact(code, data, reply, flags);
```

步骤 4：MainActivity 实现

1）控件 ID 获取

```
private void ViewID(){
    isCreate = true;
    musicCover = (ImageView)findViewById(R.id.cover);
    playStatu = (TextView)findViewById(R.id.playStatus);
    playTimeNow = (TextView)findViewById(R.id.timeBegin);
    playTime = (TextView)findViewById(R.id.playtime);
    seekBar = (SeekBar)findViewById(R.id.seekBar);
    playBtn = (Button)findViewById(R.id.playBtn);
    stopBtn = (Button)findViewById(R.id.stopBtn);
    quitBtn = (Button)findViewById(R.id.quitBtn);
}
```

2）按钮点击事件

```
playBtn.setOnClickListener((v) -> {
        isCreate = false;
        isStopped = false;
        try{
            int code = 101;
            Parcel data = Parcel.obtain();
            Parcel reply = Parcel.obtain();
            mBinder.transact(code, data, reply, 0);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
});
stopBtn.setOnClickListener((v) -> {
        isStopped = true;
        isCreate = false;
        try{
            int code = 102;
            Parcel data = Parcel.obtain();
            Parcel reply = Parcel.obtain();
            mBinder.transact(code, data, reply, 0);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
});
quitBtn.setOnClickListener((v) -> {
        unbindService(sc);
        try {
            int code = 103;
            Parcel data = Parcel.obtain();
            Parcel reply = Parcel.obtain();
            mBinder.transact(code, data, reply, 0);

            MainActivity.this.finish();
            System.exit(0);
        } catch (Exception e) {
            e.printStackTrace();
        }
});
```

调用 Binder 驱动中的 transact 方法进行消息发送，传递四个参数，之前描述过了不再详说，

这里的 data，reply 都是 Parcal 类型的，Parcel 就是一个存放读取数据的容，obtain()获得一个新的 parcel ，相当于 new 一个对象。

3）多线程与 Handler 的使用

如果在 UI 界面上进行某项操作要执行一段很耗时的代码，为了保证不影响 UI 线程，所以我们会创建一个新的线程去执行我们的耗时的代码。继承 Thread 类可以实现多线程，start 方法启动一个子线程，当满足子线程执行条件时，线程便会执行，结束或者中断后线程便会销毁：

```
private void MyThread(){
    Thread mThread = new Thread(){
        @Override
        public void  run(){
            while(true){
                try{
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                if(sc != null && hasPermission==true){
                    mHandler.obtainMessage(123).sendToTarget();//发送essage到关联线程的消息队列
                }
            }
        }
    };
    mThread.start();
}
```

当我们的耗时操作完成时，我们需要更新 UI 界面以告知用户操作完成了。子线程无法修改定义在 UI 线程（主线程）的组件，而 handler 与 UI 是同一线程，所以可以通过 Handler 更新组件状态。Handler 是 Android 中引入的一种让开发者参与处理线程中消息循环的机制。每个 Hanlder 都关联了一个线程，每个线程内部都维护了一个消息队列 MessageQueue，这样 Handler 实际上也就关联了一个消息队列。可以通过 Handler 将 Message 和 Runnable 对象发送到该 Handler 所关联线程的 MessageQueue（消息队列）中，然后该消息队列一直在循环拿出一个 Message，对其进行处理，处理完之后拿出下一个 Message，继续进行处理，周而复始。当创建一个 Handler 的时候，该 Handler 就绑定了当前创建 Hanlder 的线程。从这时起，该 Hanlder 就可以发送 Message 和 Runnable 对象到该 Handler 对应的消息队列中，当从 MessageQueue 取出某个 Message 时，会让 Handler 对其进行处理。

```
final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        switch (msg.what) {
            case 123:
                // UI 更新相关内容
                break;
```

4）UI 更新

通过 Binder 驱动在 Service 获取 mediaPlayer 的状态：

```
int SplayTimeNow = 0;
int SplayTime = 0;
int isPlay = 0;
try{
    int code = 104;
    Parcel data = Parcel.obtain();
    Parcel reply = Parcel.obtain();
    mBinder.transact(code, data, reply, 0);
    SplayTimeNow = reply.readInt();
    SplayTime = reply.readInt();
    isPlay = reply.readInt();
} catch (RemoteException e) {
    e.printStackTrace();
}
```

动画实现：

```
private void Rotate() {
    objectAnimator = ObjectAnimator.ofFloat(musicCover, "rotation", 0f, 360f);
    objectAnimator.setInterpolator(new LinearInterpolator());
    objectAnimator.setDuration(8000);
    objectAnimator.setRepeatCount(-1);
}
```

音乐播放状态以及音乐封面动画的更新：

```
//UI更新
//初始Status
if(isCreate){
    playStatu.setText("  ");
}
//点击stop按钮
else if(isStopped){
    playStatu.setText("Stopped");
    objectAnimator.end();
}
```

```
//音乐播放/暂停
else if(isPlay == 1){
    playStatu.setText("Playing");
    playBtn.setText("PAUSED");
    if(objectAnimator.isRunning()){
        objectAnimator.resume();
    }else{
        objectAnimator.start();
    }
} else{
    playStatu.setText("Paused");
    playBtn.setText("PLAY");
    objectAnimator.pause();
}
```

对进度条进行监听：

```
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        //更新播放时间，progress用于获取当前数值的大小，单位为毫秒
        playTimeNow.setText(time.format(progress));
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {

    }
    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
        try{
            int code = 105;
            Parcel data = Parcel.obtain();
            Parcel reply = Parcel.obtain();
            data.writeInt(seekBar.getProgress());
            mBinder.transact(code, data, reply, 0);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
```

步骤 5：动态内置存储文件读取权限申请

对于安卓 6.0 以上机型，需要动态获取文件阅读权限，征得用户权限认可。

1）先在 AndroidManifest.xml 文件里注册权限：

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

2）在 MainActivity 添加权限确认函数：

```
private static final int REQUEST_EXTERNAL_STORAGE = 0;
private static String[] PERMISSIONS_STORAGE = {"android.permission.READ_EXTERNAL_STORAGE"};
public static void verifyStoragePermissions(Activity activity) {
    try {
        int permission = ActivityCompat.checkSelfPermission(activity,
                "android.permission.READ_EXTERNAL_STORAGE");
        if (permission != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(activity, PERMISSIONS_STORAGE, REQUEST_EXTERNAL_STORAGE);
        }
        else {
            hasPermission = true;
        }
    } catch (Exception e){
        e.printStackTrace();
    }
}
```

3）请求权限会弹出询问框，用户选择后，系统会调用如下回调函数：

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    if(grantResults.length >0 &&grantResults[0]==PackageManager.PERMISSION_GRANTED){
        //用户同意授权
    }else{
        //用户拒绝授权
        System.exit(0);
    }
    return;
}
```

4）在 onCreate 方法中调用 verifyStoragePermissions 即可。

```
verifyStoragePermissions(MainActivity.this); //动态获取读取内置存储权限
```

（3） 实验遇到困难以及解决思路

1）RelativeLayout 布局 id 找不到的问题。一直以来写布局文件都没有太在意顺序的问题，这次就遇上了，前期看着报错一脸茫然，经过一番 debug 也找不到原因，后来直接复制 log 到 google 查找才在多个查询中找到端倪，大概就是因为安卓读取布局的时候是从上往下读的，如果放在上面的控件布局需要用到下面的控件的 id 就会出现找不到 id 的情况，调整一下顺序便可解决。

五、 课后实验

1） 真机首次安装 app 程序会出问题，退出再打开就好。

问题在于动态验证权限后用户选择确认时没有刷新，播放器还未初始化。

解决：

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    if(grantResults.length >0 &&grantResults[0]==PackageManager.PERMISSION_GRANTED){
        //用户同意授权
        startPlaying();
        hasPermission = true;
    }else{
        //用户拒绝授权
        System.exit(0);
    }
    return;
}
 private void startPlaying(){
     try{
         int code = 106;
         Parcel data = Parcel.obtain();
         Parcel reply = Parcel.obtain();
         mBinder.transact(code, data, reply, 0);
         reply.recycle();
         data.recycle();
     } catch (RemoteException e) {
         e.printStackTrace();
     }

     int SplayTimeNow = 0;
     int SplayTime = 0;
     int isPlay = 0;
     try{
         int code = 104;
         Parcel data = Parcel.obtain();
         Parcel reply = Parcel.obtain();
         mBinder.transact(code, data, reply, 0);

         SplayTimeNow = reply.readInt();
         SplayTime = reply.readInt();
         isPlay = reply.readInt();
         reply.recycle();
         data.recycle();
     } catch (RemoteException e) {
         e.printStackTrace();
     }
     seekBar.setProgress(SplayTimeNow);
     seekBar.setMax(SplayTime);
     playTime.setText(time.format(SplayTime));
case 106:
    try {
        mediaPlayer.setDataSource(Environment.getExternalStorageDirectory().getAbsolutePath() + "/melt.mp3")
        mediaPlayer.prepare();
        mediaPlayer.setLooping(true);
    } catch (Exception e){
        e.printStackTrace();
    }
    break;
```

六、 实验思考及感想

实验的难度在于 Binder 驱动的的数据传输以及多线程和 handler 的理解，也就是服务端与客户端的联系以及多线程运作相关知识的学习，这方面之前有过一些了解，在这一次实验中通过亲手操作感觉学到了不少，加深了印象，不过在更多更深层的理解还是不够的，还是需要加强学习，特别是 java 基础。

作业要求：

1. 命名要求： 学号_姓名_实验编号，例如 15330000_林 XX_lab1。
2. 实验报告提交格式为 pdf。
3. 实验内容不允许抄袭，我们要进行代码相似度对比。如发现抄袭，按 0 分处理。