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：

年级	15352272	专业（方向）	软件工程（移动信息工程）
学号	15352272	姓名	彭国栋
电话	15626066058	Email	578291308@qq.com
开始日期		完成日期	

一、 实验题目

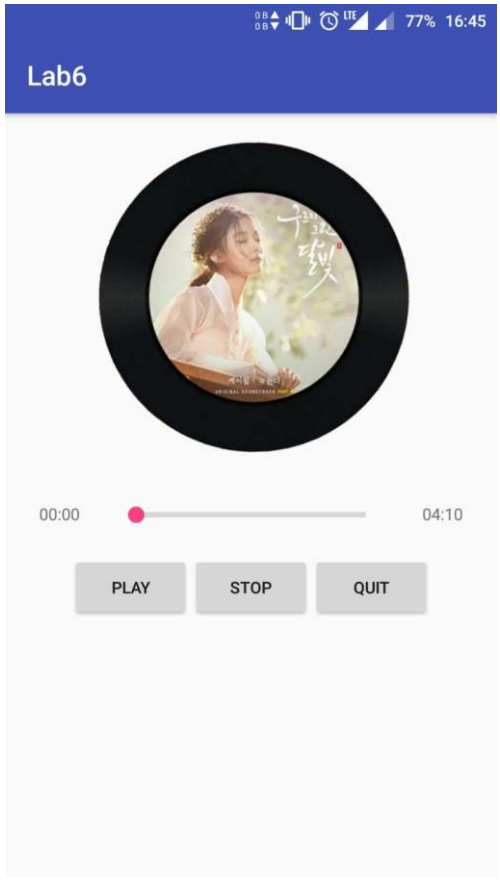
1 学会使用 MediaPlayer；
2. 学会简单的多线程编程，使用 Handle 更新 UI；
3. 学会使用 Service 进行后台工作；
4. 学会使用 Service 与 Activity 进行通信。

二、 实现内容

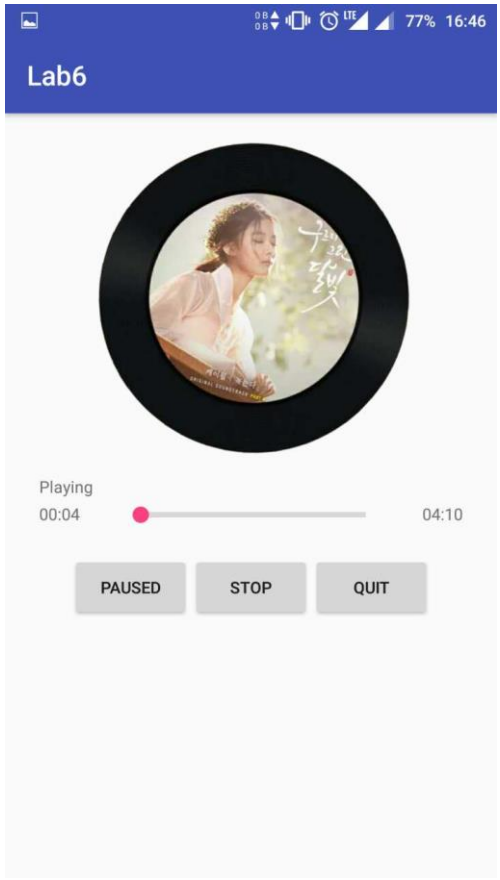
实现一个简单的播放器，要求功能有：
1. 播放、暂停，停止，退出功能；
2. 后台播放功能；
3. 进度条显示播放进度、拖动进度条改变进度功能；
4. 播放时图片旋转，显示当前播放时间功能；

三、 课堂实验结果

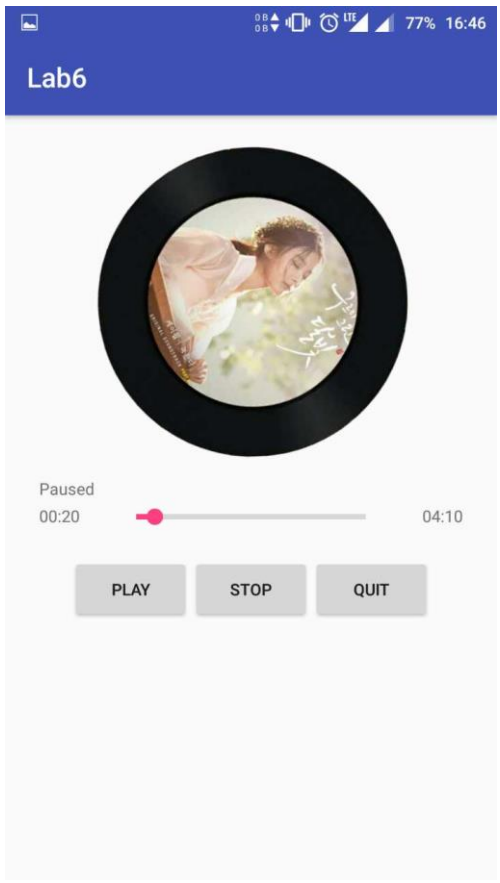
（1） 实验截图



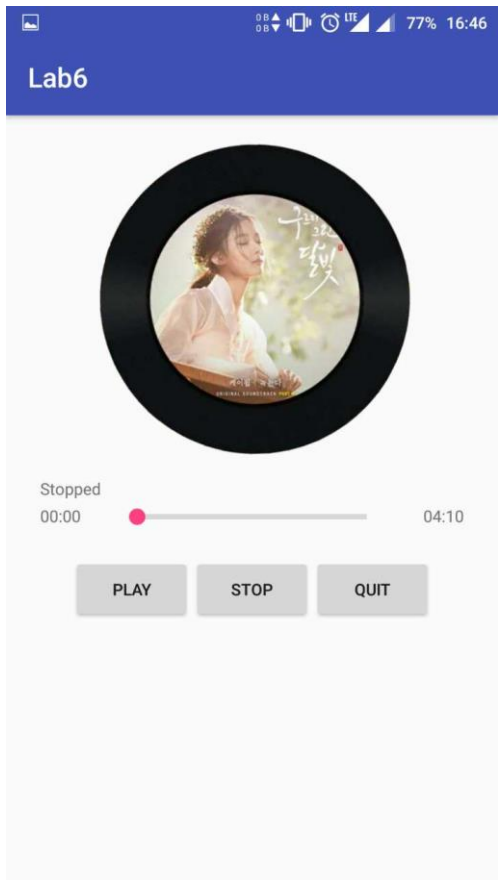
（打开程序主页面）



（开始播放）



（暂停）



（停止）

（2） 实验步骤以及关键代码

布局：

代码：

1、MainActivity.java

主要分为以下几个功能：verifyStoragePermission（确认权限），init（初始化各个空间和动画），pressEvent（各个控件的点击事件），bindServer（绑定服务），initHandle（刷新播放的时候的当前播放时间），Refresh（刷新界面状态）。

（1）verifyStoragePermission 确认权限，当权限已经获得之后，将 hasPermission 设置为 true：

```
public static void verifyStoragePermission(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();
    }
}
@Override
public void onRequestPermissionsResult(int requestCode,String permission[],int[] grantResults)
{
    if(grantResults.length>0&&grantResults[0]==PackageManager.PERMISSION_GRANTED)
    {
        Toast.makeText(this,"已授权",Toast.LENGTH_SHORT).show();
    }
    else
    {
        System.exit(0);
    }
}
```

（2）init，初始化各个控件和动画效果：

```
private void init()
{
    seekBar=(SeekBar)findViewById(R.id.seekbar);
    imageView=(ImageView)findViewById(R.id.image);
    music_hint=(TextView)findViewById(R.id.music_hint);
    music_begin=(TextView)findViewById(R.id.music_begin);
    music_end=(TextView)findViewById(R.id.music_end);
    play_or_pause=(Button)findViewById(R.id.play_or_pause);
    stop=(Button)findViewById(R.id.stop);
    quit=(Button)findViewById(R.id.quit);
    state=INITIAL_STATE;
    flag=false;

    //旋转
    objectAnimator=ObjectAnimator.ofFloat(imageView,"rotation",0,359);
    objectAnimator.setDuration(250000);
    objectAnimator.setInterpolator(new LinearInterpolator());
    objectAnimator.setRepeatCount(ObjectAnimator.INFINITE);//无限
    objectAnimator.end();
}
```

（3）pressEvent，设置各个控件的点击事件，包括四个控件，开始和暂停按钮、停止按钮、退出按钮、SeekBar 拖动栏。这几个点击事件由于都和音乐播放的 service 有关，所以主要是使用了 transact 函数来进行 activity 和 service 的交互。为了方便，先定义了几个状态常量：

```
//发送的状态
private final static int INITIAL_STATE=100;//初始状态
private final static int PLAY_OR_PAUSE_STATE=101;//播放或者暂停被按下
private final static int STOP_STATE=102;//停止被按下
private final static int GET_LENGTH_STATE=103;//获取音乐的长度
private final static int REFLASH_STATE=104;//更新音乐当前状态
private final static int DRAG_STATE=105;//拖动
```

```
//播放或者暂停被按下
play_or_pause.setOnClickListener((v) -> {
        try
        {
            Parcel data=Parcel.obtain();
            Parcel reply=Parcel.obtain();
            iBinder.transact(PLAY_OR_PAUSE_STATE,data,reply,0);
            state=reply.readInt();
            Refresh();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

});
```

```
//停止按钮被按下
stop.setOnClickListener((v) -> {
        try
        {
            Parcel data=Parcel.obtain();
            Parcel reply=Parcel.obtain();
            iBinder.transact(STOP_STATE,data,reply,0);
            state=reply.readInt();
            Refresh();
            //shuaxin
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
});
```

```
//停止按钮被按下
quit.setOnClickListener((v) -> { onDestroy(); });
```

```
//拖动
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        music_begin.setText(simpleDateFormat.format(progress));
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) { flag=true; }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
        try
        {
            Parcel data=Parcel.obtain();
            Parcel reply=Parcel.obtain();
            data.writeInt(seekBar.getProgress());
            iBinder.transact(DRAG_STATE,data,reply,0);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        flag=false;
    }
});
```

（4）bindServer，绑定服务，设置好读取音乐的长度，并将服务启动。

```
private void bindServer()
{
    serviceConnection=new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            iBinder=service;
            try
            {
                Parcel data=Parcel.obtain();
                Parcel reply=Parcel.obtain();
                iBinder.transact(GET_LENGTH_STATE,data,reply,0);
                int length=reply.readInt();
                seekBar.setMax(length);
                music_end.setText(simpleDateFormat.format(length));
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }

        @Override
        public void onServiceDisconnected(ComponentName name) { serviceConnection=null; }
    };
    Intent intent=new Intent(this,MusicService.class);
    startService(intent);
    bindService(intent,serviceConnection,BIND_AUTO_CREATE);
}
```

（5）initHandle，主要作用是为了更新播放的音乐的播放状态，即更新音乐当前的播放位置。每隔一秒发送一个线程：

```
Thread thread=run() → {
        while(true)
        {
            try
            {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if(serviceConnection!=null)
            {
                handler.obtainMessage(THREAD).sendToTarget();//从
            }
        }

};
thread.start();
```

获取线程，并告诉播放器需要更新状态。

```
final Handler handler=new Handler()
{
    @Override
    public void handleMessage(Message msg)
    {
        super.handleMessage(msg);
        switch (msg.what)
        {
            case THREAD:
                if(!flag)
                {
                    try
                    {
                        Parcel data=Parcel.obtain();
                        Parcel reply=Parcel.obtain();
                        iBinder.transact(REFLASH_STATE,data,reply,0);
                        int current=reply.readInt();
                        seekBar.setProgress(current);

                    }
                    catch (RemoteException e) {
                        e.printStackTrace();
                    }

                }
                break;
        }

    }
};
```

（6）Refresh，用于根据接收到 service 的信息，来进行对界面的更新。

为了方便，先定义了几个常量表示接收到的信息：

```
//接收到的状态
private final static int PLAYING_STATE=106;//更新为正在播放状态
private final static int PAUSE_STATE=107;//更新为暂停状态
private final static int EXIT=-1;//退出状态
```

```
switch (state)
{
    case PLAYING_STATE://设置为开始
        play_or_pause.setText("PAUSED");
        music_hint.setVisibility(View.VISIBLE);
        music_hint.setText("Playing");
        if(objectAnimator.isStarted())
        {
            objectAnimator.resume();
        }
        else
        {
            objectAnimator.start();
        }
        break;
    case PAUSE_STATE://设置为暂停
        play_or_pause.setText("PLAY");

        music_hint.setVisibility(View.VISIBLE);
        music_hint.setText("Paused");
        objectAnimator.pause();
        break;
    case STOP_STATE://设置为停止
        play_or_pause.setText("play");
        music_hint.setVisibility(View.VISIBLE);
        music_hint.setText("Stopped");
        objectAnimator.pause();
        objectAnimator.end();
        break;
```

（7）重写 onKeyDown，为了设置返回仍能在后台运行，使用了这样的方式进行处理，但是这样的方式按了返回之后并没有关闭这个 activity，所以并不算严格意义上的返回，虽然效果是一样的。

```
//返回不停止
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
        moveTaskToBack(false);
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
```

2、MusicService.java

主要是实现 MyBinder 类的重写 onTransact 函数。主要是根据 activity 中传过来的状态，做出相应的响应。

```
@Override
protected boolean onTransact(int code, Parcel data, Parcel reply, int flags) throws RemoteException
{
    switch (code)
    {
        //播放或者暂停被点击
        case PLAY_OR_PAUSE_STATE:
            if(mp.isPlaying())
            {
                mp.pause();
                reply.writeInt(PAUSE_STATE);
            }
            else
            {
                mp.start();
                reply.writeInt(PLAYING_STATE);
            }
            break;
```

```
//停止被点击
case STOP_STATE:
    try
    {
        mp.stop();
        mp.prepare();
        mp.seekTo(0);
    } catch (IOException e) {
        e.printStackTrace();
    }
    reply.writeInt(STOP_STATE);
    break;
//更新状态
case REFLASH_STATE:
    reply.writeInt(mp.getCurrentPosition());
    break;
//拖动状态
case DRAG_STATE:
    mp.seekTo(data.readInt());
    break;
//获取音乐长度
case GET_LENGTH_STATE:
    reply.writeInt(mp.getDuration());
    break;
//退出被按下
case EXIT:
    onDestroy();
    break;
```

（3） 实验遇到困难以及解决思路

图片不知道如何旋转。解决方法：使用 ObjectAnimator 可以实现这个效果。

还有就是在音乐播放的时候，发现只更新了一次状态，并没有一直更新状态。解决方法：在发送线程来实现更新的时候，发现一开始没有一直发送线程，后面添加了个 while(true)循环，每隔一秒发送线程之后即可一直更新。

四、 课后实验结果

五、 实验思考及感想

感觉这次实验最大的收获是线程的使用和 service 和 activity 之间的通信。音乐播放器的使用，当真正实现的时候也是十分兴奋，不过这个播放器还是比较简单，因为暂时只能读取一个文件，而且图片的封面也是预先设定好的，而不是从网络上获取，与实际中使用播放器还是存在很大的差异。