移动应用开发实验报告

lab6：服务与多线程--简单音乐播放器

姓名	学号	年级	专业	电话	邮箱	起止日期
王若曦	15352319	15级	软件工程	15354222552	1511330303@qq.com	2017.11.25-2017.11.28

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和Activity通信；

实验题目

实现一个简单的播放器，功能包括：

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，拖动进度条改变播放进度；
4. 播放时图片旋转，显示当前播放时间功能；

实验过程

整体思路分析：
首先由于我们要进行音乐播放，所以要想设备申请权限，用来进行音乐的播放。由于需要利用Service实现后台播放功能，所以要在MainActivity中设计好界面，然后与后台Service绑定，将用户输入传到Service进行通信并取回值，并由Service对音乐播放MediaPlayer和seekBar等进行设置，实现音乐的播放并支持后台运行。然后通过ObjectAnimator设置图片的旋转动画。

动态权限申请：
由于手机型号的不同，有的版本比较低的手机在进行音乐播放时不需要申请权限，但是版本比较高的手机必须要先申请权限，获得权限后才能进行播放，所以我们这里要先进行动态申请权限。因为我的手机不申请权限的话是不能播放的（其中project留下的坑）。
动态申请权限的代码在实验文档中直接给出，我在网上又找到了一个申请的方法，不过其实差不多。

```
//动态申请权限
if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED)
{
ActivityCompat.requestPermissions(MainActivity.this, new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, 1);
} else {
  //initMediaPlayer();//初始化播放器 MediaPlayer
}
```

如果此时没有获取权限的话，那么进行权限申请。

然后会弹出对话框，询问是否赋予权限，用户赋予权限后会回调如下函数

```
public void onRequestPermissionsResult(int requestCode, String[] permission, int[] grantResult)
{
    if ((grantResult.length > 0) && (grantResult[0] == PackageManager.PERMISSION_GRANTED))
    {
        try
        {
            Parcel localParcel1 = Parcel.obtain();
            Parcel localParcel2 = Parcel.obtain();
            this.mBinder.transact(106, localParcel1, localParcel2, 0);
            hasPermission = true;
            return;
        }
```

```
        catch (RemoteException localRemoteException)
        {
            while (true)
                localRemoteException.printStackTrace();
        }
    }
    else{
        System.exit(0);
    }
}
```

如果用户赋予了权限，则向Service通信，传递106服务码，即令Service初始化MediaPlayer载入文件；否则退出程序。这里的通信是利用IBinder实现与Service的通信，其方法在后面具体介绍。
这里要注意的是申请权限时还要在AndroidManifest文件中申请并授权：

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

绑定Service连接通信：
首先，定义intent并调用startService启动服务,然后调用bindService与Service绑定并保持通信。
绑定成功后回调onServiceConnected函数，通过 IBinder 获取 Service 对象，实现 Activity 与 Service 的绑定。之后我们利用IBinder和Service进行通信，传递服务码等信息。

```
Intent intent = new Intent(this,MusicService.class);
        startService(intent);//启动服务
        //绑定activity和服务
        bindService(intent,mConnection, Context.BIND_AUTO_CREATE);

        mConnection = new ServiceConnection() {
            @Override
            public void onServiceConnected(ComponentName name, IBinder service) {
                Log.d("service","connected");
                mBinder = service;
            }
            @Override
            public void onServiceDisconnected(ComponentName name) {
                mConnection = null;
            }
        };
```

线程与Handler更新UI：
为了更新进度条的信息，使其实时显示播放进度，我们在主界面定义一个线程，并创建Handler获取线程信息，从而进行UI的更新，主要是进度条的更新。

首先定义线程，这里在主界面UI定义一个新线程，不断运行，每次有一个延迟100。当与Service的连接成功并获取权限之后，向Handler发送信息，使Handler接受到信息后做出响应，即更新进度条。

```
final Thread mThread = new Thread()
{
    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if (mConnection != null && hasPermission == true) {
                mHandler.obtainMessage(123).sendToTarget();
            }
        }
    }
};
mThread.start();
```

然后定义Handler，由于Handler和UI是同一线程，所以可以通过Handler更新UI上的组件状态。
接受新定义的线程传来的信息后，进行处理，利用switch结构，这里传来的信息自定义，只要有信息传来即可。

```
mHandler = new Handler(){
            public void handleMessage(Message msg){
                super.handleMessage(msg);
                switch (msg.what){
                    case 123:
                        try {
                            int code = 104;
                            Parcel data = Parcel.obtain();
                            Parcel reply = Parcel.obtain();
```

```
                        mBinder.transact(code,data,reply,0);
                        Bundle bundle = reply.readBundle();
                        cur.setText(time.format(bundle.getInt("cur")));
                        dur.setText(time.format(bundle.getInt("dur")));
                        seekBar.setProgress(bundle.getInt("progress"));
                        seekBar.setMax(bundle.getInt("dur"));
                    } catch (RemoteException e) {
                        e.printStackTrace();
                    }
                    break;
                }
            }
        };
```

这里介绍一下主界面利用IBinder和Service通信的方法：
由于我们要Service处理多种事件，所以要定义一个服务码并传给Service，令其对不同服务码做出不同操作。比如这里我们要Service返回当前播放进度，然后用来更新进度条，所以我们定义服务码code为104，在Service类中则返回当前当前时间，总时长和progress。然后定义Parcel类型变量，data用来将信息传递给Service，reply用来取回Service传回来的信息。然后调用transact方法将code，data，reply传递到Service，最后一个参数0应该也是一个标志位一样的参数，具体作用没有查到，按照实验文档上写0即可运行。
然后定义bundle并用reply调用readBundle方法，获取Service传来的信息。然后通过定义的
`private SimpleDateFormat time = new SimpleDateFormat("mm:ss");`
来转换时间信息，并用其对两个TextView设置文本，即当前播放时间和总时间。
最后再通过seekBar调用setProgress方法利用取回的progress信息设置进度条的进度，然后setMax设置进度条全长。

播放、暂停、停止、退出及拖动进度条设置播放器状态：
首先接着上面说一下拖动进度条更改播放进度：
在Handler里通过Service传回的播放进度设置了进度条进度后，进度条可以跟随实时的播放进度进行变化，然后我们调用seekBar的setOnSeekBarChangeListener方法，进行拖动进度条的事件处理。
这里我们要重写三个函数：
`onProgressChanged`，`onStartTrackingTouch` 和 `onStopTrackingTouch`
前两个函数我们不需要具体实现，只需实现第三个停止拖动是触发的事件。这里当我们拖动进度条到某一位置后，先获取拖动到的位置progress，然后定义Parcel变量data储存当前progress。然后向Service发送105服务码，通信方法上面提到过。这里的105服务码是对媒体播放进度的设置，Service获取该信息后，从data中取出进度条改变后的进度，然后mediaPlayer调用seekTo方法，从而使播放进度设置到进度条拖动到的位置，即改变播放进度。

```
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener(){
    public void onProgressChanged(SeekBar seekBar,int Int,boolean Boolean){}
    public void onStartTrackingTouch(SeekBar seekBar){}
    public void onStopTrackingTouch (SeekBar seekBar){
        //停止拖动的时候读取数据
        try{
            int progress = seekBar.getProgress();
            int code = 105;
            Parcel data = Parcel.obtain();
            data.writeInt(progress);
            Parcel reply = Parcel.obtain();
            mBinder.transact(code,data,reply,0);
            return;
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
});
```

然后对于播放，暂停，停止，退出按钮的功能实现，是通过设置对应button的监听器实现。
如对播放button的监听器设置：
在布局中，我们对Button设置了文本显示按钮的功能，比如没有播放时，播放按钮上的文本显示为Play，即点击之后便开始播放音乐，然后把上面的文本显示改为Pause，即在此点击就会使播放暂停。 然后还有一个TextView来显示此时的播放状态，如没有播放时，文本显示为Paused或Stoped，播放时显示Playing。所以我们在按键之后除了进行播放器的播放与停止之外还要对这几个文本进行设置。
重写onClick方法，通过播放按钮显示的功能进行判断并做出对应操作：若按钮显示PLAY，则说明此时音乐未播放，则设置按钮文本和状态文本，同时令图片旋转起来；若按钮显示PAUSE,则音乐正在播放，则做出对应操作。
然后进行与Service的通信，传递服务码101，其功能为调用Service中的play函数，根据情况判断，具体在后面Service类中介绍。

```
play.setOnClickListener(new View.OnClickListener(){
    public void onClick(View v){
        String text = MainActivity.this.play.getText().toString();
        try{
            if (text.equals("PLAY")){
                enter = 1;
                play.setText("PAUSE");
                status.setText("Playing");
                if (MainActivity.this.animator.isPaused()) animator.resume();
                else animator.start();
```

```
            }
            else if (text.equals("PAUSE")){
                enter = 0;
                play.setText("PLAY");
                status.setText("Paused");
                animator.pause();
            }
            int code = 101;
            Parcel data = Parcel.obtain();
            Parcel reply = Parcel.obtain();
            mBinder.transact(code,data,reply,0);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
});
```

停止和退出按钮的监听器设置代码基本一致，退出代码直接调用finish结束当前的Activity，并调用system.exit；停止则向Service传送102服务码，令其调用meidaplayer的stop函数停止播放，并设置状态和按钮文本。

然后是图片旋转的实现：
图片的旋转动画通过ObjectAnimator类实现，其有多种动画可用使用，如平移，旋转，缩放等。这里我们调用旋转，设置匀速旋转，旋转一周时长20000，重复模式为一直重复。然后即可实现图片的旋转。然后要注意在按下按钮使音乐播放开始或暂停时也要对animator进行设置，令其开始旋转或暂停。调用resume使其开始旋转，调用pause令其在当前位置暂停，调用end令其复位。

```
//图片旋转
animator = ObjectAnimator.ofFloat((ImageView)findViewById(R.id.pic), "rotation", new float[] { 0.0F, 360.0F });
LinearInterpolator linearInterpolator = new LinearInterpolator();
animator.setInterpolator(linearInterpolator);
animator.setRepeatCount(-1);
animator.setDuration(20000);
animator.setRepeatMode(ValueAnimator.RESTART);
```

最后在MainActivity中要重写onDestroy函数，解除和Service的绑定，并调用finish方法退出。

```
protected void onDestroy() {
    super.onDestroy();
    unbindService(MainActivity.this.mConnection);
    mConnection = null;
    try {
        MainActivity.this.finish();
        return;
    } catch (Exception e){
        e.printStackTrace();
    }
}
```

Service的定义与功能实现：
为了实现后台播放，我们自定义MusicService类继自Service类。
这里我们主要要实现的是继承自Binder的MyBinder类，用来接收Activity传来的信息并传回数据。内部重写onTransact函数，对于Activity的通信做出响应。
这里onTransact函数获取Activity通过transact函数传来的信息，包括服务码，包含传来数据的data和要传回去的reply以及一个标志位。

```
public class MyBinder extends Binder {

    protected boolean onTransact(int code, Parcel data, Parcel reply, int flags) throws RemoteException {
        switch (code) {
            //服务端处理
            default:
            case 101:play();break;
            case 102:stop();break;
            case 103:onDestroy();break;
            case 104:
                if(mediaPlayer!=null) {
                    Bundle bundle = new Bundle();
                    bundle.putInt("cur", getCurrentTime());
                    bundle.putInt("dur", getDuration());
                    bundle.putInt("progress", getProgress());
                    reply.writeBundle(bundle);
                }
                break;
            case 105:
                int i = data.readInt();
                Log.d("jumpTime",String.valueOf(i));
                mediaPlayer.seekTo(i);
```

```
                break;
            case 106:
                try {//有权限了，就载入文件
                    MusicService.this.mediaPlayer.setDataSource(Environment.getExternalStorageDirectory() + "/Music/me  lt.mp3");
                    MusicService.this.mediaPlayer.prepare();
                    MusicService.this.mediaPlayer.setLooping(true);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                break;
        }
        return super.onTransact(code, data, reply, flags);
    }
}
```

然后进入switch结构，对不同服务码做出不同响应：

> 101：播放，调用play函数，进行音乐的播放。

若mediaPlayer为空，则进行文件的导入并开始；若不为空即之前已经导入了文件并播放过，则要进行判断，若正在播放isplaying，则调用pause暂停播放，反之开始播放。
这里说一下音乐文件的加载，如果使用虚拟机的话，则按照实验文档上的步骤将音乐文件添加到虚拟机的data文件夹中；若使用真机，则利用Environment调用getExternalStorageDirectory函数设置路径。

```
public void play(){
    if(mediaPlayer == null){
        mediaPlayer = new MediaPlayer();
        Log.d("service","create");
        try{
            //手机内置sd卡
            mediaPlayer.setDataSource(Environment.getExternalStorageDirectory()+"/Music/melt.mp3");
            //虚拟机内存
            //mediaPlayer.setDataSource("/data/melt.mp3");
            Log.d("location",Environment.getExternalStorageDirectory().toString()+"/Music/melt.mp3");
            mediaPlayer.prepare();
            mediaPlayer.setLooping(true);
            mediaPlayer.start();
            return;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    if (mediaPlayer != null) {
        if(mediaPlayer.isPlaying()){//当前状态为播放
            mediaPlayer.pause();
            status=1;//pause
        }
        else{
            mediaPlayer.start();
            status=0;//play
        }
    }
}
```

> 102：停止，调用stop函数，停止播放并复位。

调用stop函数令播放停止，再次点击复位，调用seekTo函数将进度置0，跳回开始位置。

```
private  void stop(){
    if (mediaPlayer!=null){
        mediaPlayer.stop();
        status=-1;
    }
    try {
        if (status==-1) {  // 在点击stop之后
            mediaPlayer.prepare();
            mediaPlayer.seekTo(0);      // 跳转到开始处
        }

    }catch (Exception e){
        e.printStackTrace();
    }
```

```
}
```

> 103：onDestroy，调用销毁函数，停止播放并清空资源。

```
public void onDestroy() {
    super.onDestroy();
    if (mediaPlayer != null) {
        mediaPlayer.stop();
        mediaPlayer.reset();
        mediaPlayer.release();
    }
}
```

> 104：返回当前播放进度，当前时长和总时长，用于在MainActivity中设置进度条。

利用mediaPlayer调用对应方法获取当前时间，当前进度和总时长并写进bundle利用reply传回去设置进度条。 li

```
case 104:
if(mediaPlayer!=null) {
    Bundle bundle = new Bundle();
    bundle.putInt("cur", mediaPlayer.getCurrentPosition());
    bundle.putInt("dur", mediaPlayer.getDuration());
    bundle.putInt("progress", mediaPlayer.getCurrentPosition());
    reply.writeBundle(bundle);
}
break;
```

> 105：根据进度条被拖动的位置设置播放位置

i读取传来的data即进度条被拖动到的进度，然后利用mediaPlayer调用seekTo将播放进度设置到传来的位置，实现拖动进度条改变播放进度。

```
case 105:
    int i = data.readInt();
    Log.d("jumpTime",String.valueOf(i));
    mediaPlayer.seekTo(i);
    break;
```

> 106：获取权限后加载音乐文件

在Activity动态申请权限后若获取了权限，则会在onRequestPermissionsResult与Service通信，并传递106服务码，Service在接收到信息后进行音乐文件的导入，并准备播放。

```
case 106:
    try {//有权限了，就载入文件
        MusicService.this.mediaPlayer.setDataSource(Environment.getExternalStorageDirectory() + "/Music/melt.mp3");
        MusicService.this.mediaPlayer.prepare();
        MusicService.this.mediaPlayer.setLooping(true);
    } catch (IOException e) {
        e.printStackTrace();
    }
    break;
```

然后重写一个onBind函数，只要返回一个mBinder即可。

退出程序并返回后的界面效果：
由于退出程序验证后台播放后，之前的Activity就会被杀死，所以我们之前设置的图片旋转，button的文本等显示内容就会重置，如图片不会继续旋转，状态变为停止等。所以这里我们要定义一个全局的静态变量，用来表示是否之前使用过程序且没有彻底杀死，即后台仍在运行单Acitivity被杀死。然后如果是第一次运行程序，即enter=0，则初始化显示内容；若不是第一次，则enter=1，设置当前应该显示的内容，并令图片旋转起来。

```
if (enter==0){
    play.setText("Play");
    status.setText("Paused");
}
else{
    play.setText("Pause");
    status.setText("Playing");
    animator.start();
```

```
}
```

这里重新返回程序之后图片是从头开始转的，如果想让图片从之前转到的位置继续旋转的话，就要保存之前转到的位置，重新启动之后从这个位置开始旋转。

实验总结

实验过程中遇到的问题：

1. 权限的申请。写期中project的时候一开始测试的时候，用一个华为的老一点的手机，可以直接播放音乐，此时还没有申请权限。然后用另一台三星的手机测试的时候就播放不出来，就很迷不知道为什么。虽然当时做了lab6，但是以为不申请也可以播放，因为另一台手机可以。后来知道不光是android版本号，手机型号低也会不同。所以后来就都加了动态权限申请。这里动态申请需要注意的是不光要在主界面申请，还有在AndroidManifest文件中申请并授权，这样用户在对话框授权后才能播放音乐。
2. 线程与Handler的使用。一开始不是很理解线程和Handler的用法，他们是用来干什么的。但是知道主界面是跑在一个线程里的，如果我们要实时的更新其UI上的组件，如果这个更新操作和主界面写在一起那么是不能实时更新的，所以我们要再开一个线程，让他一直跑，每次都发一个消息，然后通过Handler接受消息后对我们要更新的控件进行更新。这就是线程和Handler的作用，这里我们主要用来进行进度条的设置。因为进度条要跟随播放进度实时更新不能在主界面实现更新。
3. Activity与Service的通信。我们要实现后天播放是通过让音乐在Service播放实现的，因为如果在Activity播放的话其会随着界面的关闭而停止，所以要在Service这个没有可视化界面的地方进行播放，只要该程序没有被杀死则一直会播放。然后我们要在主界面进行操作，如点击按钮开始播放，停止播放或拖动进度条改变播放进度，这就需要与Service通信，而不是只让Service一直播放。这里我们通过IBinder实现通信，定义IBinder类型的变量，然后令其调用transact函数，参数包含服务码，我们要传到Service的数据，要传回的数据以及标志位，然后实现通信和功能的实现。
4. Service类的内容。因为Service是没有界面的，所以一开始不太清楚在这个类里到底要写什么，比如onCreate函数怎么写。然后查了一下，感觉既然是在后台，通过与Activity的通信实现功能，那么肯定要写一个Binder函数来接受传来的数据并传回，即onTransact函数，并根据服务码的不同做出不容响应。而onCreate不需要写什么，因为界面什么的都没有，所以super.oncreate空即可。
5. 有个bug一开始没有注意，就是退出程序验证其能后台播放之后，再次回到程序图片和显示的文本内容就会初始化，即图片不会继续旋转等情况。这里其实如果只是让图片接着转很简单，只要判断一下之前是不是运行过程序并且其一直在后台运行即可。即定义一个全局的静态变量，当播放音乐的时候就将其置为1，由于onCreate函数每次运行程序都会执行一次，所以在onCreate里对这个变量进行判断，如果为0则表示第一次进入程序，如果为1则表示之前运行过程序并且一直还在后台运行。然后对这两种情况进行相应的设置即可，比如在变量为1时令图片旋转，即实现了返回之后图片仍旋转。如果想让图片接着上次的位置接着转的话则保存上次转到的位置，重新启动之后接着转即可。关于这个问题后来我发现，苹果手机根本没有返回键的，只有一个home键，所以按home键退出再返回之后图片肯定是接着转的，因为按home键根本连Activity也没有杀死，没有任何效果，但返回键会把Activity杀死，所以ios的话根本不需要考虑这个问题。

本次实验一方面掌握了媒体播放器的功能实现播放音乐，另一方面掌握了后台运行程序的方法。这两部分通常都结合在一起，所以还是比较有收获的。这次主要的坑就是不管怎么样，调用手机的硬件功能时要先申请权限，然后把文件等资源配置好，最后再开始功能是实现，调试等。权限是很重要的。