移动平台应用开发

实
验
报
告

学　　院 计算机学院

年　　级 2017

班　　级 1 班

学　　号 1711425

姓　　名 曹元议

2020 年 5 月 10 日

目录

一、实验目的

熟悉 Intent 和 Activity 的使用，并利用 Intent 实现一个简易的 mp3 播放器。

二、实验内容

2.1 实验要求及功能设计总览

本次实验要求利用学习的 Intent 和 Activity 创建并实现一个简易的 mp3 播放器，通过将 mp3 的描述信息放入到 Intent 中的方法，让安卓系统自动调用系统播放器播放，具体要求：

1. 采用一个 ListView。
2. 扫描指定目录下的 mp3 文件，显示到 ListView 控件中。
3. 当用户点击 ListView 中显示的某个 mp3 文件时，调用系统播放器播放。

我在这次实验要设计的内容除了基本要求的使用 ListView 显示某个文件夹下的 mp3 文件、通过点击 ListView 的某一项调用系统播放器播放相应的 mp3 文件外，还有以下为优化用户体验而设计的拓展功能：

1. 对 ListView 下显示的所有 mp3 文件提供了按文件名搜索的功能（搜索框使用 SearchView），通过自定义 Adapter（继承 ArrayAdapter）来自定义过滤规则
2. 对 ListView 的每一项设计了 ContextMenu，用户长按 ListView 的某一项弹出该项对应的 ContextMenu，通过 ContextMenu 可以点击查看显示在自定义对话框中的该音乐文件的信息（例如所在路径、修改时间、大小、音乐人、专辑名、专辑图片等信息）或者删除该文件
3. 利用自定义 Adapter 在另一个 Activity 的 ListView 中显示一个简单的设置菜单
4. 从自定义的简单的文件系统中自由选择音乐文件所在路径的 Activity
5. 通过传递参数和返回值实现 Activity 之间的交互
6. 音乐格式过滤，可选择显示 mp3 或者 m4a 音乐文件。

其中包含的部分上次实验的功能如下，本次实验中将不再赘述具体实现细节：

1. 自定义 Activity 的背景
2. 从欢迎页通过淡出淡入效果切换到程序的主界面

3. 简单的国际化功能
4. 再按一次退出程序功能
5. 自定义对话框显示版权信息

2.2 Intent 及其用途

在 Android 系统中，Intent 提供了一种通用的消息系统，它允许在用户的应用程序与其他的应用程序之间传递 Intent 来执行动作和产生事件。

Intent 用来协助完成各应用或组件间的交互与通信。Intent 负责对应用中一次操作的动作、动作涉及到的数据、附加数据等进行描述，Android 则根据此 Intent 的描述，负责找到对应的组件，将相应数据传递给调用的组件并完成组件的调用。

Intent 的主要用途有：绑定应用程序组件、启动其它 Activity 或者 Service、发送广播消息。

2.3 Intent 对象的属性

Intent 就是一个动作的完整描述，包含了动作的产生组件、接收组件、动作的特征和传递的消息数据，这些都构成了 Intent 对象的属性，Intent 对象的属性如下：

Component（组件名称） ：Component 属性用于指定 Intent 的目标组件，一般由相应组件的包名与类名组合而成。通常 Android 会根据 Intent 中包含的其它属性的信息，例如 Action、Data/Type、Category 及 Intent filter 的过滤条件进行查找，最终找到一个与之匹配的目标组件。但是，如果 Component 这个属性有指定值的话，将直接使用它指定的组件，而不再执行上述查找过程。指定了 Component 属性值之后，Intent 的其他属性值都是可选的，此时该 Intent 就是一个显式 Intent；如果不指定 Component 属性值，则该 Intent 就是个隐式 Intent。

Action（动作）：Action 属性用来指明要实施的动作是什么，其属性值是 Intent 即将触发动作名称的字符串。可使用 SDK 中预定义的一些标准的动作属性值（例如：ACTION_MAIN、ACTION_CALL、ACTION_VIEW、ACTION_DIAL、ACTION_EDIT），这些动作由 Intent 类中预先定义好的常量字符串描述；也可以自定义 Action 的属性值。调用 Intent 对象的 getAction()方法，可以获取动作字符串；调用 setAction()方法，可以设置动作。

Data（数据）：Data 属性一般用一个 Uri 变量来表示。Data 属性主要完成对 Intent 消息中数据的封装，描述 Intent 动作所操作数据的 URI 及 MIME（类型），不同类型的 Action 会有不同的 Data 封装（如打电话的 Intent 会封装成“tel://”格式的 URI，ACTION_VIEW 的 Intent 中的 Data 则会封装成“http://”格式的 URI。）正确的 Data 封装对 Intent 请求的匹配很重要，Android 系统会根据 Data 的 URI 和 MIME 找到能处理该 Intent 的最佳目标组件。

Category（类别）：用于描述目标组件的类别信息，是一个字符串对象。它指定了将要执行的这个 Action 的其他一些额外信息。一个 Intent 中可以包含多个 Category。Android 系统同样定义了一组静态字符常量（例如 CATEGORY_GADGET、CATEGORY_HOME、CATEGORY_TAB、CATEGORY_LAUNCHER、CATEGORY_PREFERENCE）来表示 Intent 的不同类别。如果没有设置 Category 属性值，Intent 会与在 Intent filter 中包含“android.category.DEFAULT”的 Activity 匹配。调用 getCategories()得到一个 Category，调用 addCategory ()方法可以添加一个 Category，调用 removeCategory ()方法可以删除一个已经添加到 Intent 的 Category。

Type（数据类型）：用于显式指定 Intent 的数据类型（MIME）。一般 Intent 的数据类型能够根据数据本身进行判定，但是通过设置这个属性，可以强制采用显式指定的类型而不再进行隐式的判定。

Extra（附加信息）：是其它所有附加信息的集合。使用 extras 属性可以为组件提供扩展信息、可以实现在 Activity 之间传递一些参数或数据。Extra 属性值以“键-值”对形式保存。Intent 通过调用 putExtras()方法来添加一个新的键-值对，而在目标 Activity 中调用 getExtras()方法来获取 Extra 属性值。在 Android 系统的 Intent 类中，同样对一些常用的 Extra 键值进行了定义，如：EXTRA_BCC、EXTRA_EMAIL、EXTRA_UID、EXTRA_TEXT。

Flag（标志位）：有关系统如何启动组件的标志位，Android 同样对其进行了封装。

2.4 ListView 控件

本次实验使用最频繁的控件是 ListView。

ListView 是 android.view.GroupView 的间接子类，在 android.Widget 包中。ListView 以列表的形式展示内容，可以按设定的规则自动填充并展示一组列表信息，并且能够根据数据的长度自适应显示。如果显示内容过多，会出现垂

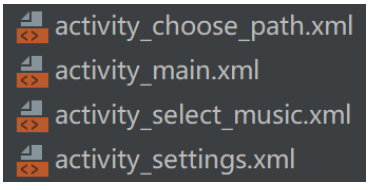
直滚动条。ListView 能够通过适配器 Adapter 将数据和自身绑定，Adapter 将数据映射到 TextView 上，在有限的屏幕上提供大量内容供用户选择。
ListView 的 Item 被点击会触发 OnItemClick 事件，Item 被选择（相当于长按）会触发 OnItemSelected 事件。
响应上述事件会调用 onItemClick()或 onItemSelected()。
列表的适配器主要有三种类型：（也可以通过继承 BaseAdapter 自定义）
ArrayAdapter
SimpleAdapter
SimpleCursorAdapter

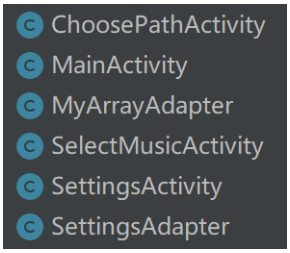
三、实验步骤及实验结果

本次实验的完整项目文件和演示视频在下面的百度网盘链接：
链接：https://pan.baidu.com/s/14k_eM1Y_N-r7TeSp2N3-EQ
提取码：98hp
（报告中对功能的演示若有说不清楚的地方可以按需取用在 examples 文件夹中的演示视频。为了防止链接被和谐，部分视频文件的文件名和格式后缀已经被处理，具体规则见 readme.txt）

3.1 实验步骤及项目文件总览

本次实验的步骤为：编写代码→测试并展示实验结果。
本次实验的编程部分也是主要分为两个步骤：设计页面布局、实现控件处理函数。
本次实验涉及到的 Activity 布局文件和与布局相对应的 Java 文件和自定义的 Adapter 类的 Java 文件如下图所示：





其中，上述的 Java 文件都在包 com.example.musicplayer 中。
Activity_choose_path.xml 和对应的 ChoosePathActivity.java 文件对应选择音乐文件所在路径的 Activity；activity_main.xml 和

MainActivity.java 对应欢迎页的 Activity；activity_select_music.xml 和 SelectMusicActivity.java 对应选择音乐的 Activity，也就是本次实验所对应的程序的主界面，可以通过这个界面调用系统音乐播放器来播放从 ListView 中选择的音乐文件；activity_settings.xml 和 SettingsActivity.java 对应设置程序参数的 Activity。
SettingsAdapter.java 用在 SettingsActivity 的 ListView 中，继承自 BaseAdapter 类；MyArrayAdapter.java 用在 SelectMusicActivity 的 ListView 中，继承自 ArrayAdapter 类。
由 AndroidManifest.xml 的 intent-filter 可见，MainActivity 依然是启动时加载的界面（欢迎页），本次实验将不再赘述。

```
<activity
    android:name=".MainActivity"
    android:theme="@style/Theme.AppCompat.DayNight.NoActionBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

欢迎页的效果如下图所示：

在 com.example.musicinfo 包中也定义了一个音乐信息的类 MusicInfo，用于在 SelectMusicActivity 的 ListView 中显示以及在自定义对话框中显示音乐的详细信息。其中，成员变量及类变量的定义如下：

```java
public class MusicInfo {
    private File file; // 对应的音乐文件
    private String absPath; // 音乐的绝对路径
    private String musicName; // 音乐文件名
    private String musicDate; // 音乐的修改日期
    private double musicSize; // 音乐文件大小
    private String title; // 音乐的标题
    private String artist; // 音乐人
    private String album; // 专辑名
    private String duration; // 音乐的时长
    private String musicFormat; // 音乐文件格式
    private byte[] pic; // 专辑图片对应的字节数组
    /* 获取音乐元数据信息，包括标题、音乐人、专辑名等 */
    private static MediaMetadataRetriever mmr = new MediaMetadataRetriever();
```

其中所有的私有成员变量都有一个获取它们的方法 `getxxx()`。构造方法定义如下：

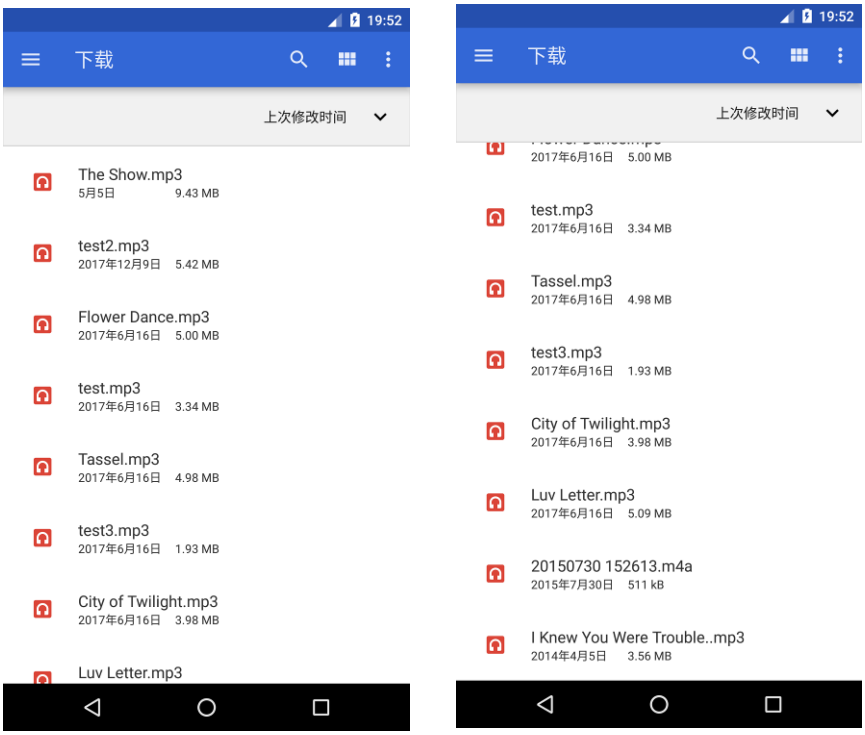
```java
    public MusicInfo(File file) {
        this.file = file;
        this.absPath  = file.getAbsolutePath();
        String illegal = "Illegal music file format!";
        // 先要验证文件的存在性和合法性
        if (!file.exists() || file.isDirectory() || !absPath.contains("."))
            throw new IllegalArgumentException(illegal);
        if (!this.absPath.toLowerCase().endsWith(".mp3") &&
                !this.absPath.toLowerCase().endsWith(".m4a"))
            throw new IllegalArgumentException(illegal);
        this.musicFormat = absPath.toLowerCase().substring(absPath.lastIndexOf('.') + 1);
        this.musicName = file.getName();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        Date date = new Date(file.lastModified());
        this.musicDate = dateFormat.format(date); // 将修改日期格式转换为可读的日期格式
        this.musicSize = (double)file.length() / 1024 /1024; // 单位从字节转换为MB
        try {
            mmr.setDataSource(absPath);
            this.title = mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_TITLE);
            this.album = mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ALBUM);
```

```
21            this.artist =
    mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ARTIST);
22            this.duration =
    mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_DURATION);
23            this.pic = mmr.getEmbeddedPicture();
24        }
25        catch (Exception e) {
26            e.printStackTrace();
27        }
28        if (this.duration != null) { // 由于从 mmr 中获取的播放时长单位为毫秒，
    所以需要将其格式转换为时间
29            SimpleDateFormat timeFormat = new SimpleDateFormat("HH:mm:ss");
30            timeFormat.setTimeZone(TimeZone.getTimeZone("GMT+0"));
31            Date time = new Date(Long.parseLong(duration));
32            this.duration = timeFormat.format(time);
33        }
34    }
```

其中通过 Android 提供的 `android.media.MediaMetadataRetriever` 类可以解析一个音乐文件的元数据：标题、音乐人、专辑名、专辑图片、时长等信息。重写 `MusicInfo` 类的 `toString()`方法如下，`Adapter` 会通过该方法将通过 `file.getName()`获取到的文件名显示到 `ListView` 中：

```
@NonNull
@Override
public String toString() {
    return getMusicName();
}
```

本次实验选择的测试文件在模拟器的路径为：`/sdcard/Download`。由于现在已经进入付费音乐时代，从各个音乐平台上下载的很多音乐很多都是收费的并且收费的音乐下载下来很有可能无法在其他地方播放，因此我使用的音乐文件绝大多数是很久以前存的音乐文件。如下图所示（9 个 mp3 文件 1 个 m4a 文件）：



3.2 选择音乐界面的实现

SelectMusicActivity 用于从 ListView 中选择将要通过系统音乐播放器进行播放的音乐文件。其中，SelectMusicActivity 类的成员变量定义如下：

```
public class SelectMusicActivity extends AppCompatActivity {

    private String musicPath;
    private int format = 0;
    private boolean allMusic = true;
    private MyArrayAdapter adapter;
    private List<MusicInfo> musicList = new ArrayList<~>();
    private TextView textView;
    private ListView listView;
    private SearchView searchView;
    private boolean searchMode = false;
    private String[] strings;
    private String searchText;
```

其中，最核心的成员变量为 musicList、adapter 和 musicPath。musicList 是一个容纳类 MusicInfo 实例的 ArrayList，作为本界面中 ListView 的数据源，被成员变量 adapter 所绑定。adapter 是绑定 musicList 数据源的适配器，

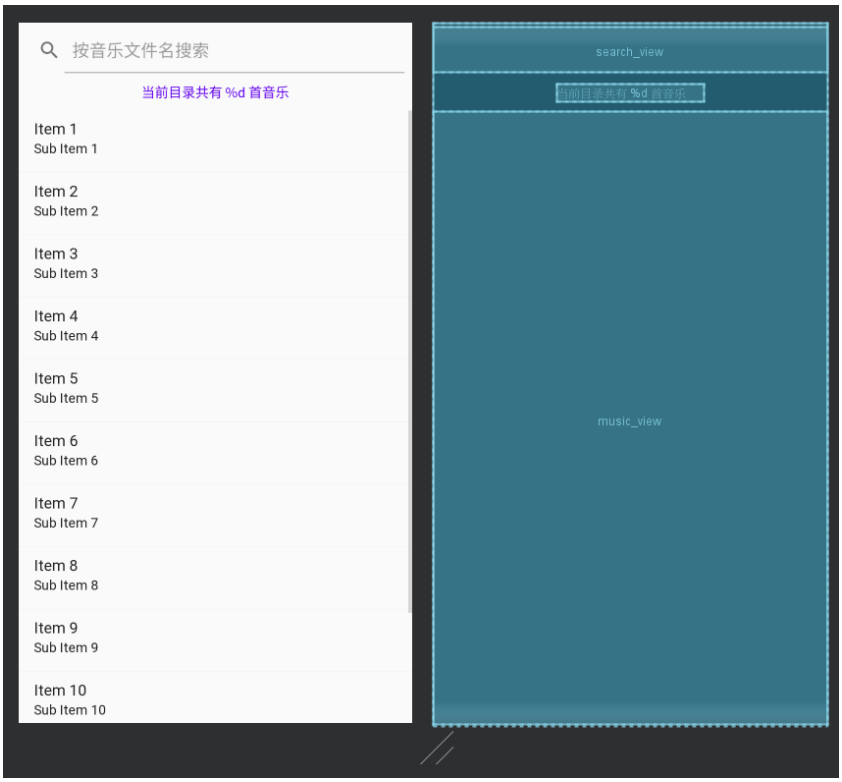
这个适配器类是 MyArrayAdapter，是我自己继承的 ArrayAdapter 的自定义适配器。musicPath 即是程序扫描音乐文件的路径。

allMuaic 变量代表是否扫描 SD 卡上所有的音乐文件；searchMode 代表是不是处于搜索模式（搜索框的内容不为空即是搜索模式）；format 是选择音乐格式的变量（0 为仅显示 mp3 文件，1 为仅显示 m4a 文件，2 为显示 mp3 和 m4a 文件；当然，m4a 文件的部分不在本次实验要求的范围之内）；strings 变量是由 format 变量索引的字符串数组，显示在 id 为 music_count 的 TextView 中，下面是在 onCreate()函数中 strings 变量被赋的值：

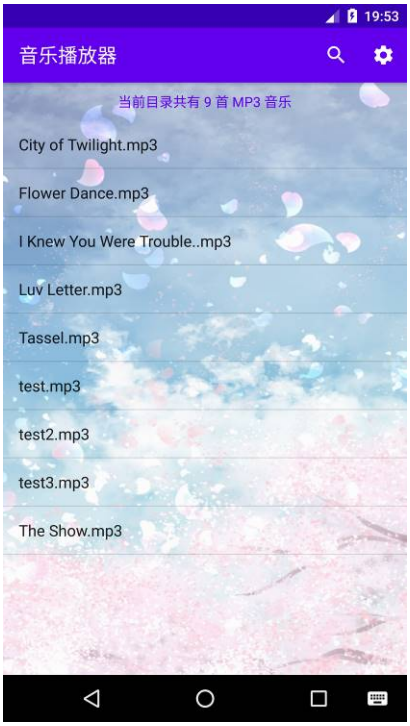
```
strings = new String[]{ "当前目录共有 %d 首 MP3 音乐", "当前目录共有 %d 首 M4A 音乐",
"当前目录共有 %d 首音乐" };
```

3.2.1 界面布局大致情况

从 activity_select_music.xml 文件的设计视图可以看到这个界面的大致布局为：



可以看出控件数量较少。以下是运行程序之后这个界面的实际显示效果：（我设置的初始状态是扫描 SD 卡上的所有 mp3 文件）

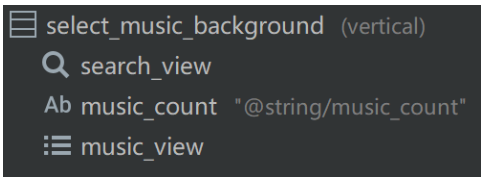


可以看见所有 9 个 mp3 文件都被显示出来了。

其中，标题“音乐播放器”通过 `SelectMusicActivity.java` 的 `SelectMusicActivity` 类的 `onCreate()`函数调用的 `setTitle()`函数动态设置。

进入 `activity_select_music.xml` 的编辑和预览视图，整体布局结构如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/select_music_background"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:fadingEdge="vertical"
    android:gravity="center"
    tools:context=".SelectMusicActivity">

    <SearchView...>
    <TextView...>
    <ListView...>

</LinearLayout>
```



可以发现最外层的布局为 LinearLayout，方向为 vertical。在布局里面只有一个 SearchView、TextView 和 ListView。

Id 为 search_view 的 SearchView 用于搜索下面的 ListView 中显示的 MP3 文件，其属性设置如下：

```
<SearchView
    android:id="@+id/search_view"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dip"
    android:iconifiedByDefault="false"
    android:queryHint="按音乐文件名搜索">
</SearchView>
```

其中属性 android:queryHint 的作用和 EditText 的 android:hint 属性类似，在输入框为空的时候显示。android:iconifiedByDefault 属性表示如果当前焦点不在 SearchView 上，则 SearchView 会缩小为一个图标，这里将属性值设置为 false 则使 SearcchView 始终是展开的状态。这里将控件的宽设置为 fill_parent 表示 SearcchView 始终占据屏幕的整行。

Id 的 music_count 的 TextView 用于显示下面的 ListView 有多少个列表项（即文件夹下有多少个音乐文件或者搜索到多少个文件），其属性设置如下：

```
<TextView
    android:id="@+id/music_count"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dip"
    android:layout_marginBottom="10dip"
    android:text="当前目录共有 %d 首音乐"
    android:textColor="@color/colorPrimary">
</TextView>
```

Id 为 music_view 的 ListView 用于显示该文件夹下所有的音乐文件，其属性设置如下：

```
<ListView
    android:id="@+id/music_view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
</ListView>
```

控件的宽和高都设置为 fill_parent 是为了使 ListView 能够占据所属布局尽可能大的空间。

3.2.2 调用系统音乐播放器的功能实现

SelectMusicActivity类中的initMusicList()函数是初始化ListView要显示的音乐列表的函数：

```
private void initMusicList() {
    musicPath = Environment.getExternalStorageDirectory().getPath();
    musicList.clear();
    getMusicList(musicPath);
    adapter = new MyArrayAdapter<MusicInfo>( context: this, android.R.layout.simple_list_item_1,musicList);
    listView.setAdapter(adapter);
    setTextView();
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            playMusic((int)(adapter.getItemId(position)));
        }
    });
}
```

首先需要通过 getMusicList()函数来获取要显示的音乐的信息到数据源 musicList 中（调用之前需要先清空 musicList 中的内容），然后将 musicList 和 adapter 绑定才能让 ListView 显示通过 getMusicList()函数获取的文件夹下的音乐文件。getMusicList()函数的定义如下：

```
1     private void getMusicList(String path) {
2         File dir = new File(path);
3         File[] files = dir.listFiles(); // 列出文件夹下的所有文件
4         if (files != null) {
5             for (File file : files) { // 遍历一下这些文件
6                 // 如果这个文件是一个文件夹并且 allMusic 为 true 就需要递归
7                 // 即遍历子文件夹下的文件
8                 if (file.isDirectory() && allMusic) {
9                     getMusicList(file.getAbsolutePath());
10                }
11                else { // 如果不是文件夹就要判断这个文件是什么格式的，根据格式决
    定是否加入到 musicList 中
12                    String currPath = file.getAbsolutePath();
13                    switch (format) {
14                        case 0:
15                            if (currPath.toLowerCase().endsWith(".mp3"))
16                                musicList.add(new MusicInfo(file));
17                            break;
18                        case 1:
19                            if (currPath.toLowerCase().endsWith(".m4a"))
20                                musicList.add(new MusicInfo(file));
21                            break;
22                        case 2:
```

```
                        if (currPath.toLowerCase().endsWith(".mp3") ||
                                currPath.toLowerCase().endsWith(".m4a"))
                            musicList.add(new MusicInfo(file));
                        break;
                    }
                }
            }
        }
    }
```

playMuaic()函数即为调用系统音乐播放器的功能实现，也是本次实验的核心内容：

```
    private void playMusic(int position) {
        final String TAG = Tag.getTag(getClass().getSimpleName());
        MusicInfo info = musicList.get(position);
        String currAbsPath = info.getAbsPath();
        Log.d(TAG, currAbsPath);
        Intent intent = new Intent(Intent.ACTION_VIEW);
        try {
            Uri uri;
            // Android 7.0 及以上需要用 FileProvider 创建uri
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
                Context context = getApplicationContext();
                File file = info.getFile();
                intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
                uri =
    FileProvider.getUriForFile(context,context.getPackageName() +
    ".provider", file);
            }
            else {
                uri = Uri.parse("file://" + currAbsPath);
            }
            intent.setDataAndType(uri, "audio/" + info.getMusicFormat());
            startActivity(intent);
        }
        catch (Exception e) {
            Toast.makeText(SelectMusicActivity.this,
                    String.format(getString(R.string.play_music_error),
    currAbsPath),
                    Toast.LENGTH_SHORT).show();
            Log.e(TAG, Log.getStackTraceString(e));
        }
    }
```

其中 position 参数对应在 ListView（id 为 music_view）中的位置，因为首

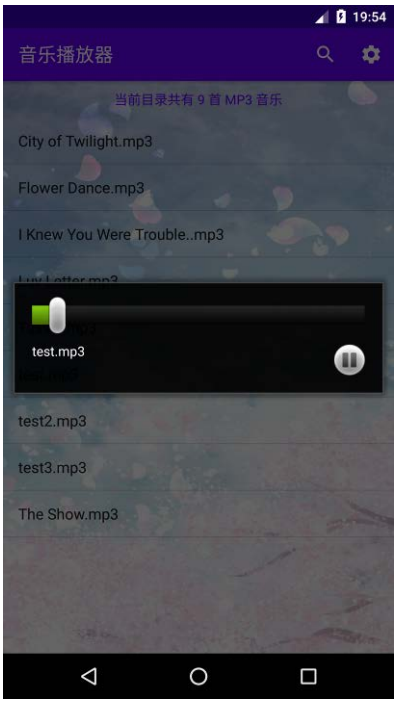
先需要通过这个 position 变量从 musicList 中获取要播放的音乐的 MusicInfo 实例。然后从这个实例提取出要播放的音乐所在的路径到局部变量 currAbsPath 中。下面就是通过 Intent 调用系统播放器进行播放音乐的功能了，由于这个过程会出现一些意想不到的问题（例如所在的 Android 系统中并没有安装播放器或者是卸载了播放器、要播放的文件不存在或已损坏等等），因此就用了 try…catch 块来包围，如果捕获了异常则通过 Toast 输出一个错误提示信息。

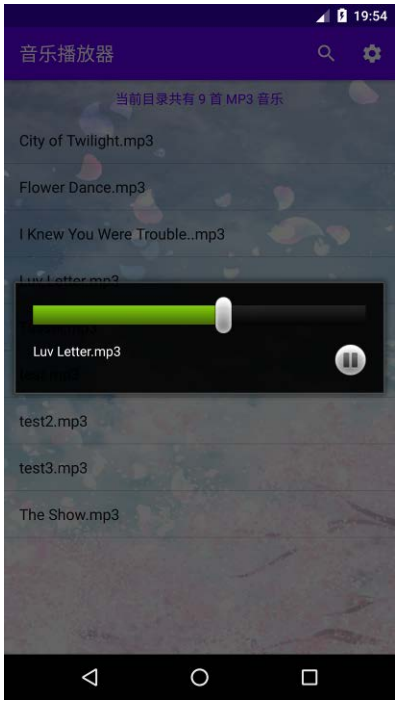
在这里通过 Intent 调用系统播放器的代码是区分了 Android 系统版本的。在 Android 7.0 及以上的版本需要动态申请 Uri 的权限并通过 FileProvider 和对应音乐文件的 File 类实例来获取相应的 Uri；而低于 Android 7.0 的系统则参考了 ppt 课件中提供的方法来创建 Uri。最终要通过 intent.setDataAndType 来设置 Intent 的数据类型（音乐文件以及格式），这样才能通过 tartActivity(intent)直接打开系统音乐播放器。

在这里，可以通过 ListView 的监听事件 onItemClick 以及与 ListView 绑定的上下文菜单中的一项来调用 playMusic()函数。

ListView 的监听事件 onItemClick 的代码设置在函数 initMusicList()中，上面的图已经列出.只需要调用 playMusic()函数，并传入正确的 position 参数即可（该音乐在数据源列表中的位置）。

这样，点击 ListView 的某一项就可以调用系统音乐播放器去播放相应的音乐文件了，效果截图如下（以 test.mp3 和 Luv Letter.mp3 为例）：





3.2.3 搜索音乐功能的实现

在 SelectMusicActivity 类中，onCreate()函数中与 SearchView 相关的代码如下：

```
searchView = (SearchView) findViewById(R.id.search_view);
searchView.setVisibility(View.VISIBLE);
openSearchView();
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
    @Override
    public boolean onQueryTextSubmit(String query) { return false; }

    @Override
    public boolean onQueryTextChange(String newText) {
        searchText = newText;
        searchMode = newText.length() != 0;
        // 使用用户输入的内容对ListView的item进行过滤，无输入则取消过滤，且不会显示黑色弹框
        adapter.getFilter().filter(newText);
        setTextView();
        return true;
    }
});
```

SearchView 需要响应的监听事件为 OnQueryText，监听事件函数 onQueryTextChange()设置当搜索框中的内容发生改变时的动作，而监听事件函数 onQueryTextSubmit()设置当搜索框的内容提交的时候发生的动作。为了优化用户的体验，我选择直接在搜索框内容改变的时候就进行搜索动作，而不是按下回车键或者点击提交按钮（我没有设置 SearchView 要显示这个按钮），因此就只定义了 onQueryTextChange()函数而 onQueryTextSubmit()并未定义而保持其默认定义。onQueryTextChange()函数的参数 newText 即搜索框的新内容，由这个来决定是否处于搜索模式（newText 长度不为 0，即搜索内容不为空为搜索模式，设置成员变量 searchMode 值为 true）。

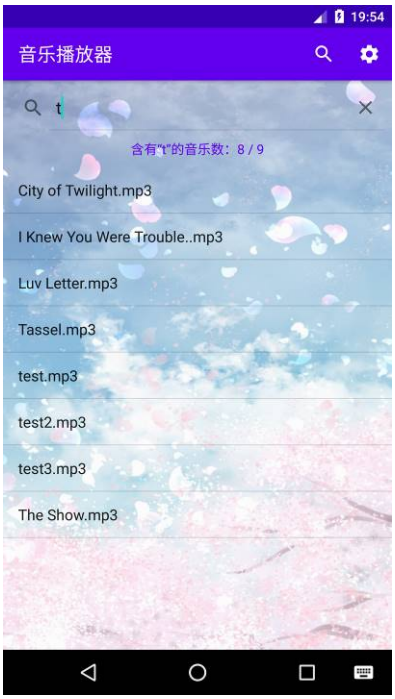
adapter.getFilter().filter(newText)是通过 Adapter 对 musicList 中匹配了 newText 的内容进行过滤，这样就是进行了一次搜索操作，Listview 只会显示被匹配到的音乐文件。如果 newText 长度为 0，则不处于搜索模式，取消对 musicList 的过滤，恢复 ListView 显示的所有数据。

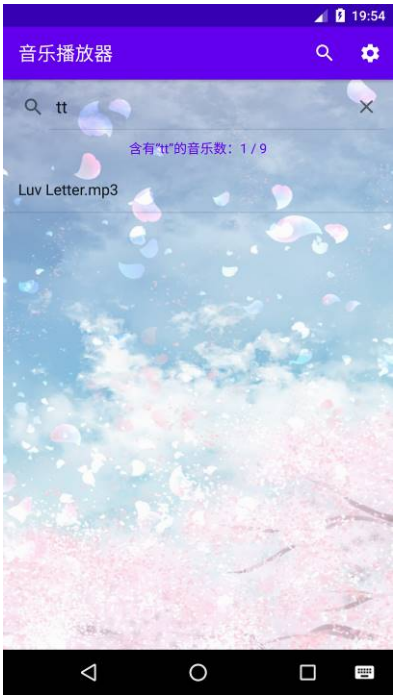
setTextView()函数是根据是否为搜索模式（searchMode 是否为 true）来在 TextView 中显示特定的内容：

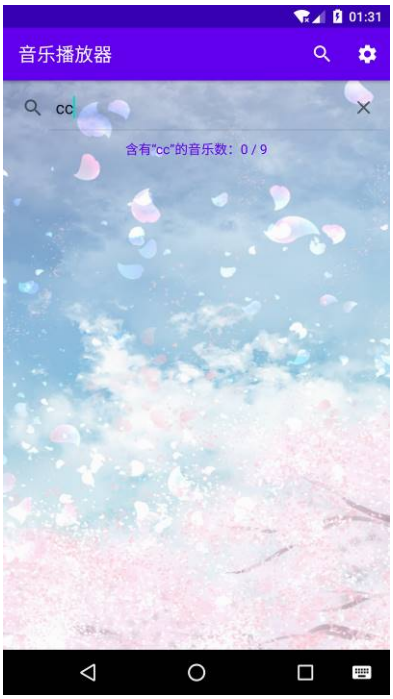
```
private void setTextView() {
    if (searchMode)
        textView.setText(String.format("含有“%s”的音乐数: %d / %d", searchText,
                adapter.getCount(), musicList.size()));
    else
        textView.setText(String.format(strings[format], musicList.size()));
}
```

以下是搜索操作的效果截图：

在搜索框中输入不同的内容将会得到不同的效果：（清空搜索框的效果与程序运行后从未在搜索框中输入的效果相同，故不重复截图）







3.2.4 菜单及其他功能实现

我在这个界面定义的菜单有 Options Menu 和 Context Menu 两种。前者是显示在顶部标题栏上的；后者是绑定到 ListView 的每一个 item 上的（长按 ListView 的某个 item 就可以弹出）。

Options Menu 的菜单项定义在 res/menu/select_music_menu.xml 中：

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/search_item"
        android:title="搜索"
        android:icon="@drawable/search"
        app:showAsAction="always"/>
    <item
        android:id="@+id/settings_item"
        android:title="设置"
        android:icon="@drawable/settings"
        app:showAsAction="always"/>
</menu>
```

其中属性 app:showAsAction="always"表示菜单项的标题文字（或图标）直接显示在菜单栏上。

在 SelectMusicActivity.java 的 onCreateOptionsMenu()方法设置 Options Menu 要绑定的菜单项源文件为 res/menu/select_music_menu.xml：

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.select_music_menu, menu);
    return super.onCreateOptionsMenu(menu);
}
```

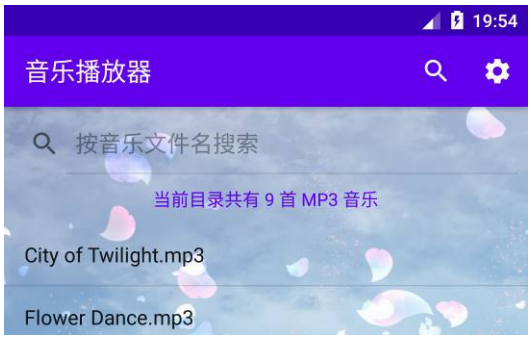
在 SelectMusicActivity.java 的 onOptionsItemSelected()方法中设置点击 Options Menu 的菜单项的响应事件：

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    Intent intent;
    switch (item.getItemId()){
        case R.id.search_item:
            openSearchView();
            break;
        case R.id.settings_item:
            intent = new Intent();
            intent.setClass( packageContext: SelectMusicActivity.this, SettingsActivity.class);
            intent.putExtra( name: "currentPath", musicPath);
            intent.putExtra( name: "allMusic", allMusic);
            intent.putExtra( name: "format", format);
            SelectMusicActivity.this.startActivityForResult(intent,  requestCode: 0);
            break;
        default:
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

第一个菜单项 R.id.search_item 对应的是搜索按钮，通过调用 openSearchView()函数来实现点击一下弹出搜索框，再点击一下收起搜索框的效果：

```
private void openSearchView() {
    if (searchView.getVisibility() == View.VISIBLE)
        searchView.setVisibility(View.GONE);
    else
        searchView.setVisibility(View.VISIBLE);
}
```

openSearchView()函数通过设置 SearchView 是否可见（visibility）来决定搜索框是否显示。如果这次点击时 visibility 为 View.VISIBLE（即搜索框已经显示），则需要 visibility 为 View.GONE 来收起搜索框（View.GONE 可以让控件不可见并且该控件在不可见的时候不可见不占据布局的任何空间）；反之则显示 visibility 为 View.VISIBLE 来显示搜索框。演示效果如下：（由于我在 onCreate()中调用了一下 openSearchView()函数，而 SearchView 的 visibility 默认为 View.VISIBLE，则界面第一次加载时搜索框是隐藏的）



如果搜索框处于隐藏状态，点击一下左上角的搜索按钮，将会弹出搜索框，如上图所示。再点击一下就会再次隐藏搜索框（效果与前面的图相同就不截图了）。

第二个菜单项 R.id.settings_item 对应的是设置按钮，点击该菜单项将会向 SettingsActivity（设置界面）传递参数并打开它。

Context Menu 的菜单项定义在 res/menu/music_item_menu.xml 中：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/play_music_item" android:title="播放"/>
    <item android:id="@+id/music_info_item" android:title="查看详细信息"/>
    <item android:id="@+id/delete_music_item" android:title="删除"/>
</menu>
```

在 SelectMusicActivity.java 的 onCreateContextMenu()方法设置

Context Menu 要绑定的菜单项源文件为 res/menu/music_item_menu.xml：

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {
    getMenuInflater().inflate(R.menu.music_item_menu, menu);
    super.onCreateContextMenu(menu, v, menuInfo);
}
```

由于 Context Menu 要绑定到 ListView 的菜单项上，因此要在 onCreate()函数中为 ListView 的每个菜单项注册 Context Menu，然后在 onDestroy()函数中取消注册：

```
else {
    // Android 版本低于6.0则直接初始化音乐列表（要申请的权限已在AndroidManifest.xml中列出）
    initMusicList();
}
SelectMusicActivity.this.registerForContextMenu(listView); // 为ListView的每一项注册上下文菜单
```

```
@Override
protected void onDestroy() {
    SelectMusicActivity.this.unregisterForContextMenu(listView);
    super.onDestroy();
}
```

在 SelectMusicActivity.java 的 onContextItemSelected()方法中设置点击 ContextMenu 的菜单项的响应事件：

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterView.AdapterContextMenuInfo menuInfo = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo()
    // 获取对应于ListView的position
    int position = (int)(adapter.getItemId(menuInfo.position));
    switch(item.getItemId()) {
        case R.id.play_music_item:
            playMusic(position);
            break;
        case R.id.music_info_item:
            showMusicInfo(position);
            break;
        case R.id.delete_music_item:
            deleteMusic(position);
        default:
            break;
    }
    return super.onContextItemSelected(item);
}
```

第一个菜单项 R.id.play_music_item 就是通过调用 playMusic()函数进行播放音乐的操作。

第二个菜单项 R.id.music_info_item 是通过调用 showMusicInfo()函数来在自定义对话框中显示该音乐文件的详细信息，showMusicInfo()函数定义如下（涉及到的自定义对话框布局文件 music_info_dialog.xml 不再列出，可以查阅源代码），就是通过 position 参数获取到 musicList 中对应的 MusicInfo 类实例，将这个实例中的内容显示在自定义对话框中：

```
1    private void showMusicInfo(int position) {
```

```
2         MusicInfo info = musicList.get(position);
3         AlertDialog.Builder infoDialog = new
    AlertDialog.Builder(SelectMusicActivity.this);
4         final View dialogView =
    LayoutInflater.from(SelectMusicActivity.this)
5                 .inflate(R.layout.music_info_dialog,null);
6         infoDialog.setTitle(getString(R.string.music_details));
7         infoDialog.setView(dialogView); // 向对话框加载自定义布局文件
8         TextView fileNameText = (TextView)
    dialogView.findViewById(R.id.file_name_text);
9         TextView fileFormatText = (TextView)
    dialogView.findViewById(R.id.file_format_text);
10        TextView fileDateText = (TextView)
    dialogView.findViewById(R.id.file_date_text);
11        TextView fileSizeText = (TextView)
    dialogView.findViewById(R.id.file_size_text);
12        TextView titleText = (TextView)
    dialogView.findViewById(R.id.title_text);
13        TextView artistText = (TextView)
    dialogView.findViewById(R.id.artist_text);
14        TextView albumText = (TextView)
    dialogView.findViewById(R.id.album_text);
15        TextView durationText = (TextView)
    dialogView.findViewById(R.id.duration_text);
16        TextView absPathText = (TextView)
    dialogView.findViewById(R.id.abs_path_text);
17        ImageView albumPic = (ImageView)
    dialogView.findViewById(R.id.album_pic);
18
    fileNameText.setText(String.format(getString(R.string.file_name_text),
    info));
19
    fileFormatText.setText(String.format(getString(R.string.file_format_tex
    t), info.getMusicFormat()));
20
    fileDateText.setText(String.format(getString(R.string.file_date_text),
    info.getMusicDate()));
21
    fileSizeText.setText(String.format(getString(R.string.file_size_text),
    info.getMusicSize())); // .2f：保留2位小数
22
    absPathText.setText(String.format(getString(R.string.abs_path_text),
    info.getAbsPath()));
```

```
23        String unknown = getString(R.string.unknown); // 如果字符串为null 则显示“未知”
24        if (info.getTitle() != null) {
25            titleText.setText(String.format(getString(R.string.title_text), info.getTitle()));
26        }
27        else {
28            titleText.setText(String.format(getString(R.string.title_text), unknown));
29        }
30        if (info.getArtist() != null) {
31
artistText.setText(String.format(getString(R.string.artist_text), info.getArtist()));
32        }
33        else {
34
artistText.setText(String.format(getString(R.string.artist_text), unknown));
35        }
36        if (info.getAlbum() != null) {
37            albumText.setText(String.format(getString(R.string.album_text), info.getAlbum()));
38        }
39        else {
40            albumText.setText(String.format(getString(R.string.album_text), unknown));
41        }
42        if (info.getDuration() != null) {
43
durationText.setText(String.format(getString(R.string.duration_text), info.getDuration()));
44        }
45        else {
46
durationText.setText(String.format(getString(R.string.duration_text), unknown));
47        }
48        if (info.getPic() != null) {
49            if (info.getPic().length != 0) {
50                // 将byte[] 转为 drawable 并显示在 ImageView 控件中
51                Bitmap bitmap = BitmapFactory.decodeByteArray(info.getPic(), 0, info.getPic().length);
```

```
52              BitmapDrawable bd = new BitmapDrawable(getResources(),
    bitmap);
53              albumPic.setImageDrawable(bd);
54          }
55      }
56      infoDialog.setPositiveButton(getString(R.string.yes),
57              new DialogInterface.OnClickListener() {
58                  @Override
59                  public void onClick(DialogInterface dialog, int which) {
60                      dialog.dismiss();
61                  }
62              });
63      infoDialog.show();
64   }
```

显示效果如下：（方形的图片为专辑图片，为 ImageView 控件）

首先在 ListView 上的某一项长按会弹出如下的上下文菜单：

长按不同的项，点击“查看详细信息”的结果分别如下：

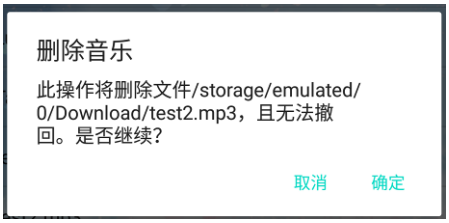
第三个菜单项 R.id.delete_music_item 是通过调用 deleteMusic()函数来弹出对话框并删除相应的音乐文件：

```
private void deleteMusic(int position) {
    final MusicInfo info = musicList.get(position);
    AlertDialog.Builder builder = new AlertDialog.Builder( context: SelectMusicActivity.this);
    builder.setTitle("删除音乐");
    builder.setMessage(String.format("此操作将删除文件%s，且无法撤回。是否继续？", info.getAbsPath()));
    final int pos = position;
    builder.setPositiveButton("确定", (dialog, which) → {
            File file = info.getFile();
            if (file.delete()) {
                musicList.remove(pos);
                adapter.notifyDataSetChanged();
                if (searchMode) // 如果还在搜索，就重新过滤一下
                    adapter.getFilter().filter(searchText);
                setTextView();
            }
    });
    builder.setNegativeButton("取消", listener: null);
    builder.create().show();
}
```

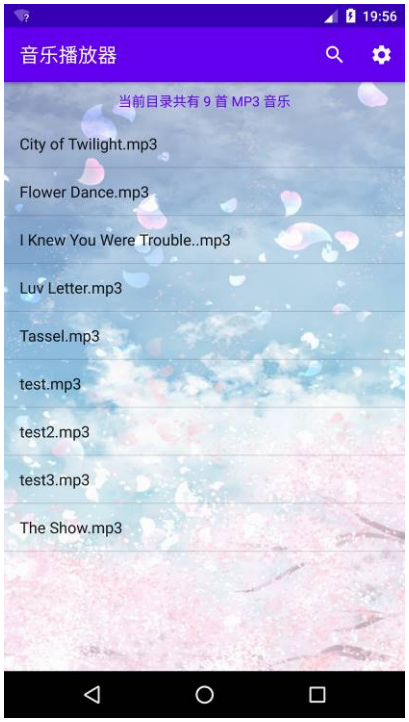
其中，点击对话框的“确定”按钮将从 musicList 中取得相应的音乐文件并删除它，删除这个文件之后需要将该 MusicInfo 类的实例通过 remove()函数从 musicList 中移除，还要通过 adapter.notifyDataSetChanged()来更新数据源从而更 ListView 中显示的内容，然后更新还剩下的音乐数。这里有一个细节就是如果处于搜索模式，需要再过滤一次数据源以实时更新搜索结果。

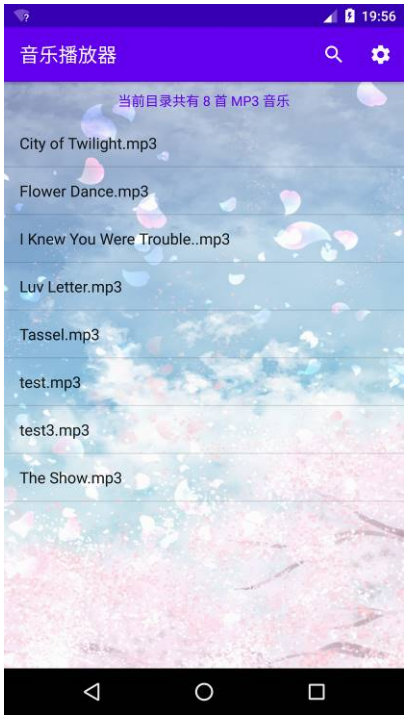
显示效果如下：（以 test2.mp3 为例）

删除对话框显示如下（点击“确定”即可删除）：

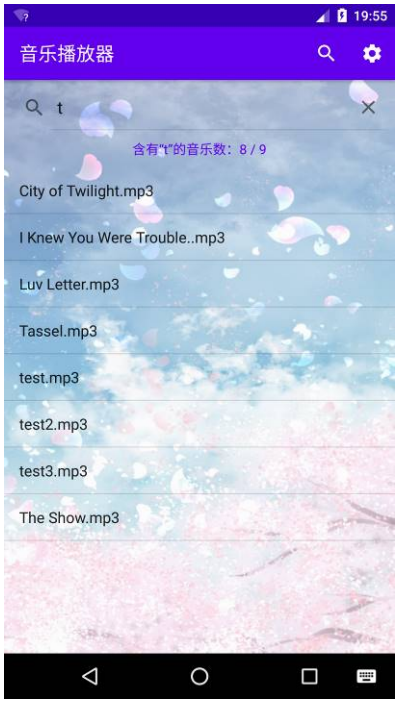


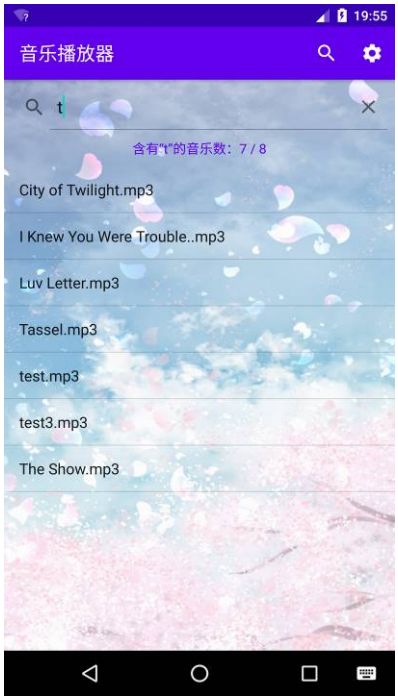
如果未处于搜索模式，删除前 vs 删除后：





如果处于搜索模式，删除前 vs 删除后：





（如果此时退出搜索模式，显示效果和未处于搜索模式并且删除了 test2.mp3 的效果相同，故不再重复截图）

3.3 设置界面的实现

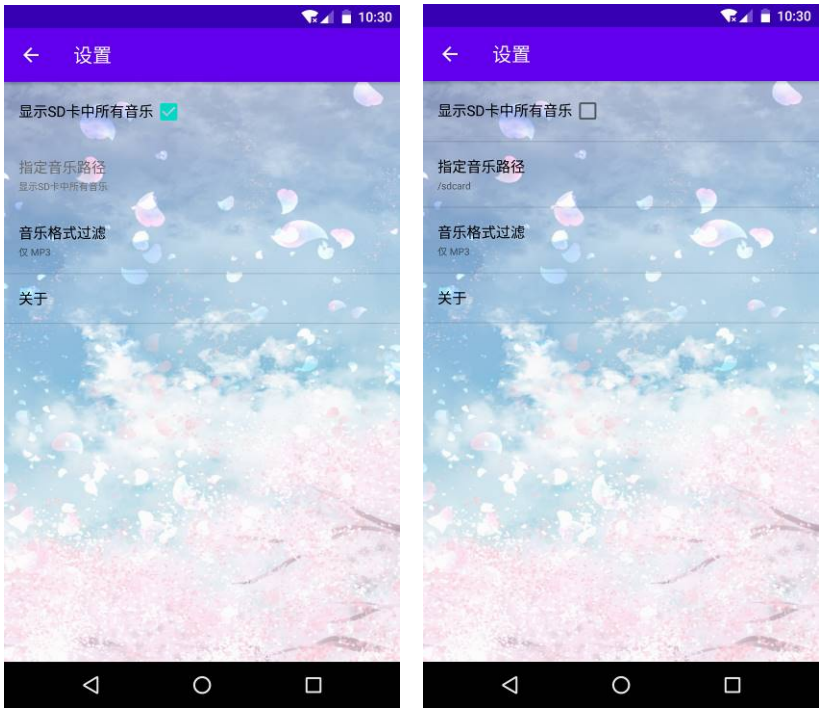
SettingsActivity 用于设置要显示什么路径的音乐文件和选择音乐文件的格式。其中，SettingsActivity 类的成员变量定义如下：

```
public class SettingsActivity extends AppCompatActivity {

    private List<String> list = new ArrayList<~>();
    private SettingsAdapter adapter;
    private int format = 0;
```

其中，list 数据源配合自定义的 adapter 在界面中仅有的 ListView 中显示出设置菜单。format 变量对应于选择音乐的格式（由于实验要求的是 mp3 格式因此默认为 0 代表只显示 mp3 格式的音乐文件）。

3.3.1 界面布局大致情况

由于 activity_settings.xml 只有一个 ListView 且设置属性的方式同 activity_select_music.xml，因此布局如何设计从略。
以下是运行程序之后该 Activity 显示的实际效果：（左：默认，右：取消选中 CheckBox）



点击“音乐格式过滤”并选中“MP3 / M4A”：

点击“关于”：

3.3.2 与 SelectMusicActivity 的数据交互

SelectMusicActivity 在准备启动 SettingsActivity 时会传值给 SettingsActivity，在 SettingsActivity 的 onCreate()函数中接收这些值，

并将这些值作为 list 的数据源，在 ListView 中会有所体现。并且设置希望接收的返回值为 0。

在 SelectMusicActivity 的 onOptionsItemSelected()函数中，对应设置菜单的响应事件设置：

```
case R.id.settings_item:
    intent = new Intent();
    intent.setClass( packageContext: SelectMusicActivity.this, SettingsActivity.class);
    intent.putExtra( name: "currentPath", musicPath);
    intent.putExtra( name: "allMusic", allMusic);
    intent.putExtra( name: "format", format);
    SelectMusicActivity.this.startActivityForResult(intent,  requestCode: 0);
    break;
```

首先设置要启动的 Activity 为 SettingsActivity。然后通过 intent 的 putExtra()函数传递图中的参数给 SettingsActivity，SettingsActivity 的 onCreate()接收并处理这些值：

```
1     @Override
2     protected void onCreate(Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         setContentView(R.layout.activity_settings);
5         setTitle(getString(R.string.settings));
6
7         /* 左上角返回按钮设置 */
8         getSupportActionBar().setDisplayHomeAsUpEnabled(true); // 设置显示返回按钮
9         getSupportActionBar().setHomeButtonEnabled(true); // 设置返回按钮允许按下
10
11        ListView listView = (ListView) findViewById(R.id.settings_view);
12        // 接收来自启动该 Activity 的 Activity 传来的值
13        boolean receiveBool = getIntent().getBooleanExtra("allMusic", false);
14        String receive = null;
15        if (!receiveBool) { // 通过 list 的第 0 个值设置上面的 CheckBox 是否选中
16            list.add("F"); // 如果 CheckBox 不选中才接收 currentPath 的值
17            receive = getIntent().getStringExtra("currentPath");
18        }
19        else {
20            list.add("T");
21        }
22        if (receive != null) // currentPath 的值不为空则将该值显示在设置菜单的第二项上
23            list.add(receive);
24        else
```

```
25          list.add("/sdcard"); // 默认值
26      int receiveFormat = getIntent().getIntExtra("format", -1);
27      if (receiveFormat != -1) // format 的值不为空则将该值对应的字符串显示
    在设置菜单的第三项上
28          format = receiveFormat;
29      int[] layoutType = { 0, 1, 1, 2 }; // 指定 ListView 不同的 item 使用不
    同的 Layout
30      list.add("");
31      list.add(null); // 设置菜单的第四项是关于项，不需要数据则将数据设置为
    null
32      switch (format) {
33          case 0:
34              list.set(2, getString(R.string.only_mp3));
35              break;
36          case 1:
37              list.set(2, getString(R.string.only_m4a));
38              break;
39          case 2:
40              list.set(2, getString(R.string.mp3_m4a));
41              break;
42          default:
43              break;
44      }
45      adapter = new SettingsAdapter(SettingsActivity.this, layoutType,
    list);
46      listView.setAdapter(adapter); // 将 adapter 与 listView 绑定，list 中的
    数据信息呈现在 listView 上
47      // 设置 listView 的 OnItemClick 监听事件
48      listView.setOnItemClickListener(new
    AdapterView.OnItemClickListener() {
49          @Override
50          public void onItemClick(AdapterView<?> parent, View view, int
    position, long id) {
51              switch (position) {
52                  case 0:
53                      String src = list.get(0);
54                      if (src.equals("T"))
55                          list.set(0, "F");
56                      else if (src.equals("F"))
57                          list.set(0, "T");
58                  case 1:  // 打开 ChoosePathActivity
59                      openFileManager();
60                      break;
61                  case 2:  // 弹出设置格式的对话框
```

```
62                      switchFormat();
63                      break;
64                  case 3:  // 第四个设置项弹出关于对话框
65                      new
    AboutDialog().showAboutDialog(SettingsActivity.this);
66                      break;
67                  default:
68                      break;
69              }
70          }
71      });
72   }
```

其中，参数 allMusic 的值与 list 的第 0 项相关联，用于控制 CheckBox 的选中状态和第二个设置项是否可点击的状态。currentPath 显示在第二个设置项上。通过参数 format 设置的字符串与 list 的第 2 项相关联，显示在第三个设置项上。List 的第 3 项对应设置项的第四项，是“关于”项，不需要与什么数据关联，因此设为 null。弹出格式对话框的函数 switchFormat()和上次实验切换主题对话框的设置同理，只展示“确定”按钮的监听事件设置：

```
builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        for (int i = 0; i < checkedItems.length; i++) {
            if (checkedItems[i]) {
                format = i;
                list.set(2, items[i]);
                adapter.notifyDataSetChanged();
                break;
            }
        }
        dialog.dismiss();
    }
});
```

在 SettingsActivity 的 onCreate()函数中，和上次实验一样启用了左上角返回按钮：

```
/* 左上角返回按钮设置 */
getSupportActionBar().setDisplayHomeAsUpEnabled(true); // 设置显示返回按钮
getSupportActionBar().setHomeButtonEnabled(true); // 设置返回按钮允许按下
```

对应于函数 onOptionsItemSelected()的响应事件：

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            saveSettings();
            break;
        default:
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

saveSettings()函数定义如下：

```java
private void saveSettings() {
    Intent intent = new Intent();
    intent.putExtra( name: "newCurrentPath", list.get(1));
    intent.putExtra( name: "format", format);
    intent.putExtra( name: "allMusic", list.get(0));
    setResult( resultCode: 0, intent);
    finish();
}
```

因为在 SettingsActivity 中这些值会发生改变，因此在返回到 SelectMusicActivity 之前要将这些新值传回给调用它的 SelectMusicActivity，并且要设置返回值为 SelectMusicActivity 所期望的返回值 0，最后关闭 SettingsActivity。

SelectMusicActivity 中的 onActivityResult()用于接收这些返回值，调整 ListView 和 TextView 的显示内容（因此每次从 SettingsActivity 到 SelectMusicActivity，ListView 和 TextView 的显示内容都会更新）。如果期望接收的返回值 requestCode 和实际返回值 resultcode 相匹配且值都为 0，才表明参数是从 SettingsActivity 传过来的：

```java
1    @Override
2    public void onActivityResult(int requestCode, int resultCode, Intent data)
   {
3        super.onActivityResult(requestCode, resultCode, data);
4        if (requestCode == 0 && resultCode == 0) { // 接收来自 SettingsActivity
   传来的值
5            String receiveAll = data.getStringExtra("allMusic");
6            if (receiveAll != null) {
7                if (receiveAll.equals("T"))
8                    allMusic = true;
9                else if (receiveAll.equals("F"))
```

```
                allMusic = false;
            }
            String receivePath = null;
            if (!allMusic) { // 如果CheckBox未选中，就要接收newCurrentPath
    参数的值作为指定显示音乐的路径
                receivePath = data.getStringExtra("newCurrentPath");
                if (receivePath != null) {
                    musicPath = receivePath;
                }
            }
            else {
                musicPath =
    Environment.getExternalStorageDirectory().getPath(); // 设为SD卡路径
            }
            int receiveFormat = data.getIntExtra("format", -1);
            if (receiveFormat != -1) {
                format = receiveFormat;
            }
            musicList.clear();
            getMusicList(musicPath); // 重新按照路径获取音乐列表
            adapter.notifyDataSetChanged(); // 更新数据源
            if (searchMode) // 如果还在搜索，就重新过滤一下
                adapter.getFilter().filter(searchText);
            setTextView();
        }
    }
```

SettingsActivity 的 onKeyDown()函数设置了屏幕底部返回键的动作，和左上角返回按钮的动作相同：

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
        saveSettings();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
```

3.4 选择路径界面的实现

ChoocePathActivity 用于设置要在 SelectMusicActivity 显示的音乐文件的路径。这也是本次实验的一个重点。原先我是想类比使用 Intent 打开系统播

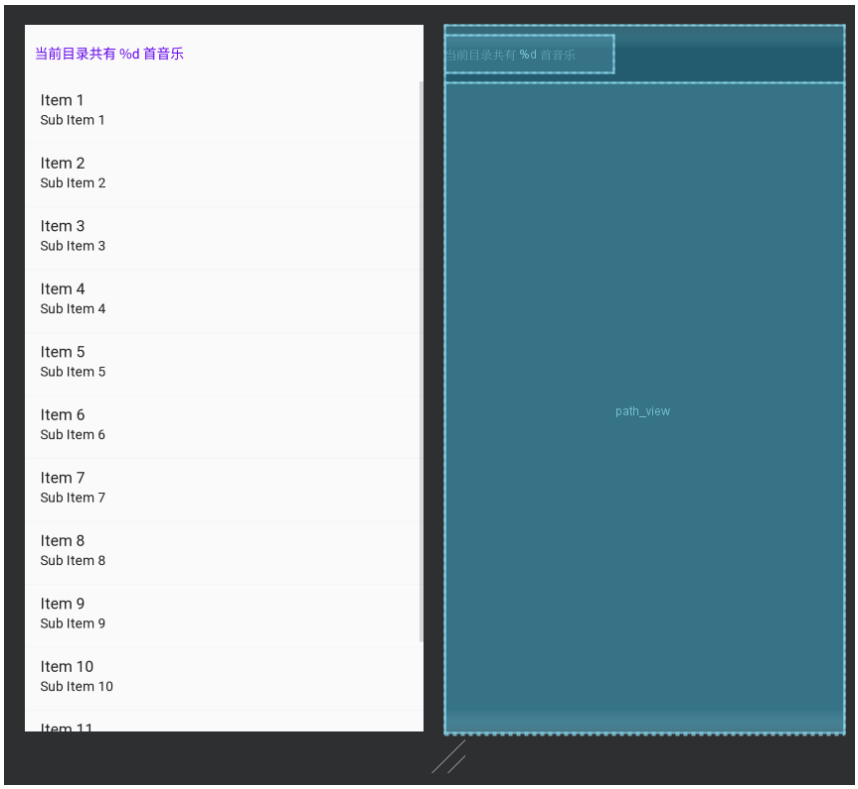
放器的方式，通过 Intent 来打开系统文件管理器，从这个文件管理器来选择路径，但是后来发现由于不同手机的 Android 系统定制程度和版本都不同，容易出现打不开文件管理器的情况，而且这种方式似乎不能单独选择路径。最后就只能在这个界面中实现一个最简单的文件管理系统——只能选择在 ListView 中显示的路径，不能做其他操作。ChoocePathActivity 类的成员变量和类变量定义如下：

```
public class ChoosePathActivity extends AppCompatActivity {

    private static final String ROOT_PATH = "/";
    private static String currentAbsPath = ROOT_PATH;
    private static String currentRelPath = "";
    private List<String> paths = new ArrayList<~>();
    private ArrayAdapter adapter;
    private TextView currentPath;
```

currentAbsPath 是当前的绝对路径，ListView 中会显示这个路径下的所有目录。currentRelPath 是当前的相对路径。paths 是 ListView 的数据源，将会以 ArrayAdapter 类型的 adapter 绑定。currentPath 对应于显示当前绝对路径的 TextView。

3.4.1 界面布局大致情况

从 activity_choose_path.xml 文件的设计视图可以看到这个界面的大致布局为：

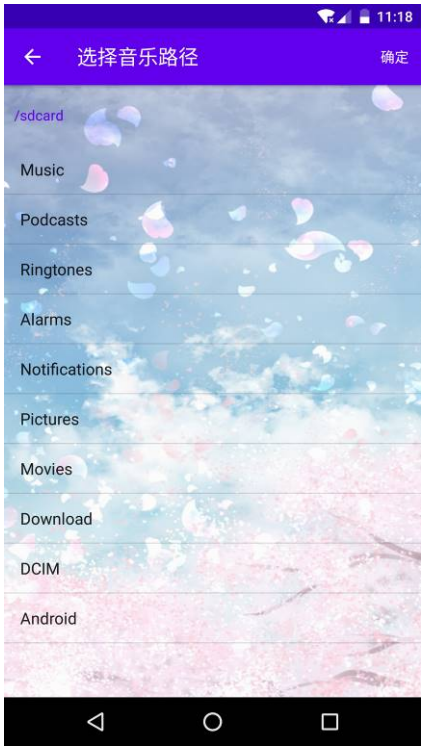


可以看出布局也较简单，切换到编辑视图可以看到外层布局是垂直的 LinearLayout，里面只有一个显示当前决定路径的 TextView 以及显示当前路径下的目录的 ListView：

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/choose_path_background"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ChoosePathActivity">

    <TextView
        android:id="@+id/current_path"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dip"
        android:layout_marginTop="10dip"
        android:layout_marginBottom="10dip"
        android:text="当前目录共有 %d 首音乐"
        android:textColor="@color/colorPrimary">
    </TextView>
    <ListView
        android:id="@+id/path_view"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    </ListView>
</LinearLayout>
```

TextView 的初始文字是随便填的，因为会在程序中进行动态设置。

运行程序之后界面显示的实际效果如下：（从 SettingsActivity 点击第二个菜单项弹出 ChoosePathActivity，初始时显示的路径对应于 SettingsActivity 的第二个设置项）



3.4.2 选择路径功能的实现

首先，SettingsActivity 的 ListView 在 SettingsActivity 的 onCreate()函数（上文已列出）中设置了 OnItemClicked 事件。如果第二个设置项被点击，则通过调用 openFileManager()函数启动 ChoosePathActivity 并向 ChoosePathActivity 传递路径参数并且设置期望的返回值为 0，以在 ChoosePathActivity 的 onCreate()对 TextView 和 ListView 的内容进行初始化。

```
private void openFileManager() {
    Intent intent = new Intent();
    intent.setClass( packageContext: SettingsActivity.this, ChoosePathActivity.class);
    intent.putExtra( name: "currentPath", list.get(1));
    SettingsActivity.this.startActivityForResult(intent,  requestCode: 0);
}
```

ChoosePathActivity 的 onCreate()函数如下：

```
1     @Override
2     protected void onCreate(Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         setContentView(R.layout.activity_choose_path);
5         setTitle(getString(R.string.choose_path));
```

```
6
7       /* 左上角返回按钮设置（这里的作用为返回到上一级目录） */
8       getSupportActionBar().setDisplayHomeAsUpEnabled(true); // 设置显示
    返回按钮
9       getSupportActionBar().setHomeButtonEnabled(true); // 设置返回按钮允
    许按下
10
11      // 接收来自启动该 Activity 的 Activity 传来的值
12      String receive = getIntent().getStringExtra("currentPath");
13      if (receive != null)
14          currentAbsPath = receive;
15      File initFile;
16      // 先要验证一下路径的合法性，如果不合法，就尝试回退到上级目录，直到根目录
    为止
17      do {
18          initFile = new File(currentAbsPath);
19          if (initFile.exists() && initFile.isDirectory())
20              break;
21          currentAbsPath = currentAbsPath.substring(0,
    currentAbsPath.lastIndexOf('/'));
22      } while (currentAbsPath.length() != 0);
23      if (currentAbsPath.length() == 0) {
24          currentAbsPath = ROOT_PATH;
25          initFile = new File(currentAbsPath);
26      }
27      final File[] initFiles = initFile.listFiles();
28      if (initFiles != null && initFiles.length > 0) {
29          for (File f : initFiles) {
30              if (f.isDirectory())
31                  paths.add(f.getName());
32          }
33      }
34      currentPath = (TextView) findViewById(R.id.current_path);
35      currentPath.setText(currentAbsPath);
36      ListView path = (ListView) findViewById(R.id.path_view);
37      adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, paths);
38      path.setAdapter(adapter); // 将 adapter 与 ListView 绑定
39      path.setOnItemClickListener(new AdapterView.OnItemClickListener() {
40          @Override
41          public void onItemClick(AdapterView<?> parent, View view, int
    position, long id) {
42              currentRelPath = paths.get(position);
43              if (!currentAbsPath.equals(ROOT_PATH))
```

```
                currentAbsPath += "/";
            currentAbsPath += currentRelPath;
            currentPath.setText(currentAbsPath);
            File file = new File(currentAbsPath);
            File[] files = file.listFiles();
            paths.clear();
            if (files != null && files.length > 0) {
                for (File f : files) {
                    if (f.isDirectory())
                        paths.add(f.getName());
                }
            }
            adapter.notifyDataSetChanged();
        }
    });
}
```

首先接收从 SettingsActivity 传来的路径参数，然后要检查这个参数的合法性（是否存在，是不是目录），如果不合法就要尝试回退到上级目录进行检查，直到根目录为止。然后使用合法的路径作为当前路径，设置 TextView 显示为当前路径并初始化 initFile 对象，通过 listFiles()列出当前路径。然后将 adapter 和数据源和 ListView 绑定，这样就可以将该路径下的目录显示在 ListView 中。

最后设置 ListView 的监听事件 OnItemClick，实现点击 ListView 的某一项就更新当前目录并显示新的当前目录中有什么子目录。

在 onCreate()函数中也设置了左上角返回按钮，但是我让它在这里的作用不是返回到上一级界面而是显示上一级目录。

在 ChoosePathActivity 设置的菜单在 res/menu/choose_path_menu.xml 中，只有一个右上角确定按钮：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/save_item"
        android:title="确定"
        app:showAsAction="always"/>
</menu>
```

ChoosePathActivity 的 onOptionsItemSelected()函数中设置了确定按钮和左上角返回按钮的响应事件：

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.choose_path_menu, menu);
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.save_item:
            Intent intent = new Intent();
            intent.putExtra( name: "newCurrentPath", currentAbsPath);
            setResult( resultCode: 0, intent);
            finish();
            break;
        case android.R.id.home:
            backToLast();
            break;
        default:
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

首先左上角返回按钮是通过 backToLast()函数返回到上一级目录，backToLast()函数定义如下：

```java
/** 回退到上级目录 */
private void backToLast() {
    if (!currentAbsPath.equals(ROOT_PATH)) {
        currentAbsPath = currentAbsPath.substring(0, currentAbsPath.lastIndexOf( ch: '/'));
        if (currentAbsPath.length() > 0)
            currentRelPath = currentAbsPath.substring(currentAbsPath.lastIndexOf( ch: '/') + 1);
        else {
            currentAbsPath = ROOT_PATH;
            currentRelPath = "";
        }
        File file = new File(currentAbsPath);
        File[] files = file.listFiles();
        paths.clear();
        if (files != null && files.length > 0) {
            for (File f : files) {
                if (f.isDirectory())
                    paths.add(f.getName());
            }
        }
        currentPath.setText(currentAbsPath);
        adapter.notifyDataSetChanged();
    }
}
```

取上级路径的思路就是取 0 到最后一个/的 currAbsPath 子字符串。如果 currAbsPath 最终为空，就要设置它为根目录。然后重新读取这个路径下的目录并显示即可。

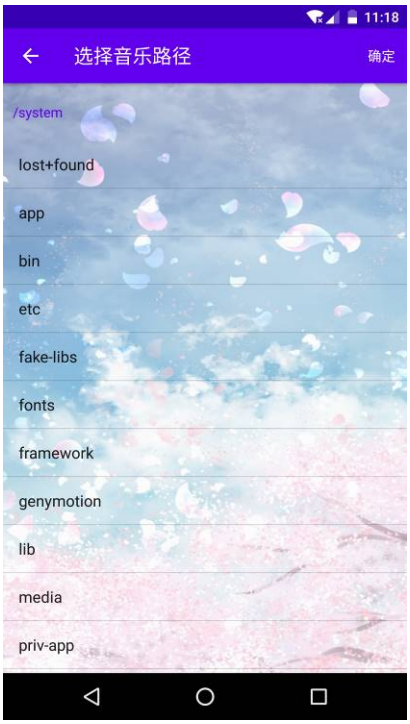
确定按钮就是选择 TextView 中显示的路径并传回给 SettingsActivity 并设置返回值为 SettingsActivity 所期望的返回值，然后关闭此 Activity。
在 SettingsActivity 的 onActivityResult()函数中，如果期望的返回值 requestCode 和实际返回值 resultCode 匹配且值都为 0，则接收这个新的路径参数并显示在第二个设置项中。

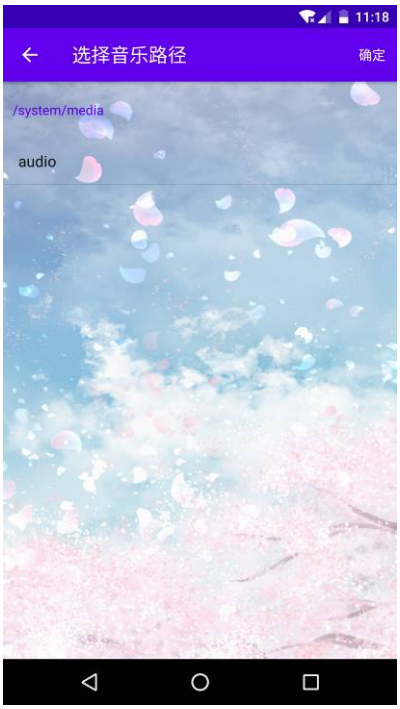
```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 0 && resultCode == 0) { // 接收来自 ChoosePathActivity 传来的值
        String receive = data.getStringExtra( name: "newCurrentPath");
        if (receive != null) {
            list.set(1, receive);
            adapter.notifyDataSetChanged();
        }
    }
}
```

ChoosePathActivity 的 onKeyDown()函数设置了屏幕底部返回键的动作，虽然也是返回到上一个界面 SettingsActivity，但是设置的返回值为 1，则 SettingsActivity 中的第二个设置项显示的内容不会改变。

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
        Intent intent = new Intent();
        setResult( resultCode: 1, intent);
        finish();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
```

以下是与 ChoosePathActivity 相关的运行效果：
从/sdcard 退回到根目录，并定位到/system/media 路径：
【bug 提醒：在部分模拟器或者手机上，如果退回到根目录/可能会没有任何显示，应该跟程序的权限有关】



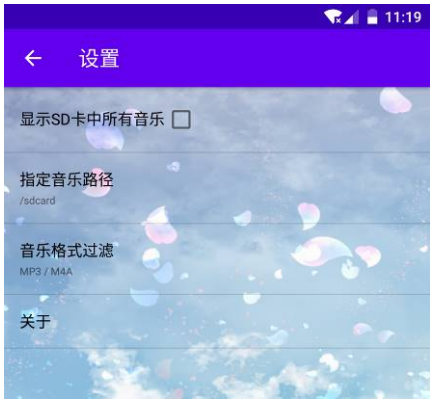


从/system/media 退回到根目录，并定位到/sdcard/Download，然后点击“确定”，可以看到第二个设置项的值已经变成了/sdcard/Download：

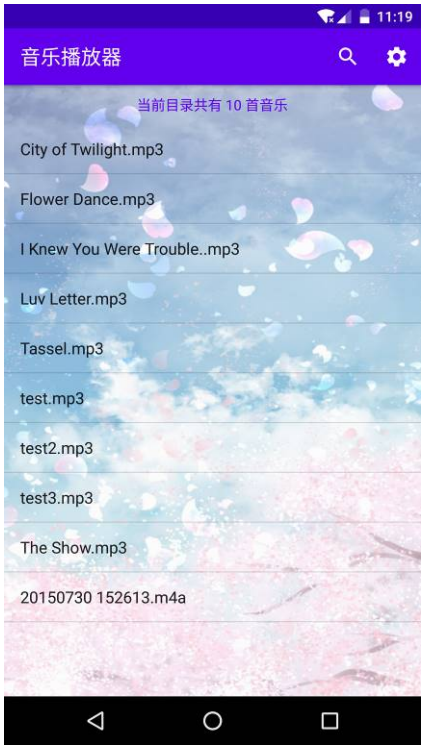
再次点击 SettingsActivity 的第二个设置项打开 ChoosePathActivity，定位到/sdcard/Music，然后点击左下角返回键，SettingsActivity 没有任何变化：

SettingsActivity 的设置内容如下，返回到 SelectMusicActivity 的显示如下：

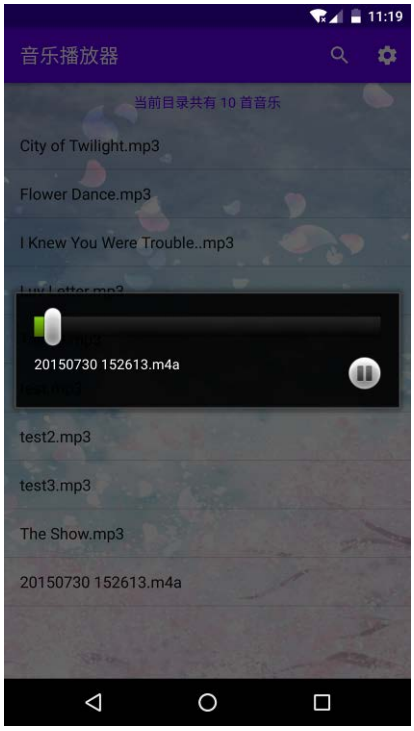
音乐路径为/sdcard（由于 CheckBox 未选中，不会显示子目录 Download 的文件）



音乐路径为/sdcard/Download：



查看 20150730 152613.m4a 的详细信息并播放：



四、实验遇到的问题及其解决方法

4.1 如何将音乐文件导入到模拟器中

这个对于像我一样平时只用苹果手机的人来说比较棘手，但是我用的是 Genymotion 模拟器，可以在打开模拟器之后直接把文件拖进去，Genymotion 模拟器会将文件默认放在文件夹`/sdcard/Download` 中。

但是一定要注意文件名最好不要包含中文，否则可能会出现解码失败的情况，导致文件无法在系统文件管理器中被正确显示（会有乱码字符），起初我想把这个文件删掉，改个文件名再重新拖移，但是这个文件居然删不掉。后来就没有再管这个问题。直到运行程序才发现这个会让程序无法正常运行，会闪退，并且报的异常是无法正确解码这个乱码字符。

无奈之下只好百度，参考网上的方案，打开模拟器之后在命令行中输入 `adb shell` 命令进入到模拟器系统的命令行模式。然后我通过命令定位到 `/sdcard/Download` 文件夹，在通过 `rm -rf *`命令删掉这个文件夹下所有的文件，再将我要测试的所有的音乐文件名都改成英文名，重新拖进去才解决。当然 Android 系统是基于 Linux 的，可以直接使用 Linux 的命令。

（这个问题没有具体截图）

4.2 如何获取 SD 卡访问权限

按照以前的方法直接在 `AndroidManifest.xml` 中声明一下要使用的权限即可：

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.musicplayer">

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
        android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"
        tools:ignore="ProtectedPermissions" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MusicPlayer"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
```

但是只采用这个方式在 Android 6.0 及以上的系统版本就不好使了，然后参考刘

庄同学在群里发的博客链接（https://www.cnblogs.com/zanzg/p/9129375.html），在 SelectMusicActivity 的 onCreate()函数中加入动态申请权限的代码：

```
// Android 6.0 及以上需要动态申请SD卡访问权限
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {

    int storagePermission = SelectMusicActivity.this.checkSelfPermission(
            Manifest.permission.WRITE_EXTERNAL_STORAGE);
    //检测是否有权限，如果没有权限，就需要申请
    if (storagePermission != PackageManager.PERMISSION_GRANTED) {
        //申请权限
        SelectMusicActivity.this.requestPermissions(STORAGE_PERMISSIONS, requestCode: 1);
    }
    else {
        // 如果有权限就直接初始化音乐列表
        initMusicList();
    }
}
else {
    // Android 版本低于6.0则直接初始化音乐列表（要申请的权限已在AndroidManifest.xml中列出）
    initMusicList();
}
```

其中，第一次运行程序的时候，resuestPermissions()函数调用后会弹出一个是否为该程序授予访问 SD 卡访问权限的对话框，只有用户点击了“同意”，程序才能够访问或修改 SD 卡的文件。并且在程序中只需要在其中一个位置动态申请权限即可，无需在其他界面的类或者其他地方再申请一次。initMusicList()函数就是初始化界面操作的代码，由于这段代码还要在 onRequestPermissionsResult()中使用，所以就封装成了函数。

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] gra
    if (requestCode == 1) {
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            String sdCard = Environment.getExternalStorageState();
            if (sdCard.equals(Environment.MEDIA_MOUNTED)) {
                initMusicList();
            }
        } else {
            runOnUiThread(() -> {
                    Toast.makeText( context: SelectMusicActivity.this,
                            "外置存储访问未授权或不支持外置存储访问", Toast.LENGTH_SHORT).show();
            });
        }
    }
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
```

onRequestPermissionsResult()用于获取申请权限的结果。如果申请成功就初始化界面。（最后一个参数是 grantResults，内容是申请权限的结果）

虽然我原来看的别的教程也说过 Android 6.0 及以上的系统版本需要动态申请 SD 卡访问权限，但是当时没做到这就暂时没理会这个问题。最后觉得这篇博客确实讲的比较好而且代码原作者还对此进行了一层封装，就拿来用了。

4.3 如何正确使用 Intent 调用系统音乐播放器

按照 ppt 上提供的代码示例可以调用系统音乐播放器：


【例6-4】 示例工程06_IntentOpenMP3Example演示了利用Intent播放MP3音频文件，MainActivity类的主要代码如下。

```
public class MainActivity extends Activity {
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button btnstart = (Button)findViewById(R.id.btn_1);
    btnstart.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        Uri myuri = Uri.parse("http://www.baidu.com");
        Intent myintent = new Intent(Intent.ACTION_VIEW);
        Uri uri = Uri.parse("file:///storage/sdcard/music/music01.mp3");
        myintent.setDataAndType(uri, "audio/mp3");
        startActivity(myintent);
      }
    });
}}
```

但是这个方式在 Android 7.0 及以上的系统版本又不好使了（模拟器系统版本是 Android 8.0），会报 `android.os.FileUriExposedException`，原因是 Android 7.0 及以上的系统版本还要动态申请 `Uri` 权限（使用 `FileProvider`）。需要在 `AndroidManifest.xml` 中的`<application>`标签中加入`<provider>`标签，从国内的博客上找到的 `android:name` 属性都是 `android.support.v4.content.FileProvider`，但是在我这里不行。在 StackOverFlow 查了一下之后改成 `androidx.core.content.FileProvider` 就可以了。

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="MusicPlayer"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

    <provider
        android:name="androidx.core.content.FileProvider"
        android:authorities="${applicationId}.provider"
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/provider_paths" />
    </provider>
```

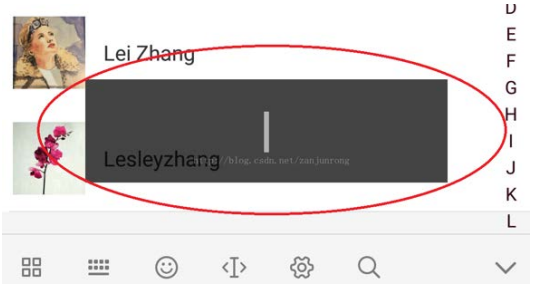
然后还要在 res/文件夹下新建 xml 文件夹，然后创建文件 res/xml/provider_paths.xml，填入如下内容：

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="external_files" path="."/>
</paths>
```

最终在 playMusic()中的代码已经在上面展示过了（大于等于 Android 7.0 的版本使用新方法，否则使用 ppt 提供的方法）。

4.4 搜索功能实现的相关问题

我想利用 SearchView 实现搜索的功能，并且希望在搜索框输入文字的过程中过滤并显示 ListView 的相应内容，但是原先是使用 listView.setTextFilterEnabled(true)启用过滤功能。然后在 SearchView 监听事件的 onQueryTextChange()函数中通过 listView.setFilterText(newText)设置新的过滤文本，如果新的过滤文本长度为 0，则通过 listView.clearTextFilter()取消过滤。但是在搜索框输入的时候会出现一个如下图的大黑框，非常碍眼（这是从网上找的图，自己这里没有截图）：



后来发现是 listView.setFilterText(newText)引起的。然后就使用 adapter.getFilter().filter(newText)进行过滤，不仅能达到同样的目的而且，如果 newText 的内容为空就取消过滤，而且还没有这个黑色框，非常方便。

但是问题又来了，在 TextView 中显示的搜索结果数总是上一次搜索的结果数，于是就给 TextView 加了 OnScroll 监听事件，在 OnScroll()函数更新 TextView 的内容：

```
listView.setOnScrollListener(new AbsListView.OnScrollListener() {
    @Override
    public void onScrollStateChanged(AbsListView view, int scrollState) {

    }

    @Override
    public void onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount, int totalItemCount)
        setTextView(); // 实时更新被过滤得到的item数
    }
});
```

当然第三个参数 visibleItemCount 也可以表示被过滤得到的 item 数（在 setTestView()函数中是使用 adapter 定义的 getCount()函数获取，也能达到同样的目的）。

```
private void setTextView() {
    if (searchMode)
        textView.setText(String.format("含有"%s"的音乐数: %d / %d", searchText,
                adapter.getCount(), musicList.size()));
    else
        textView.setText(String.format(strings[format], musicList.size()));
}
```

还有一个更大的问题，就是无论是使用 listView.setFilterText(newText)还是直接使用 ArrayAdapter 类中的 filter(newText)，过滤的结果均不符合预期（过滤结果数总比预期的少），我的预期是过滤得到文件名中包含该字符串的 item（仅仅是包含就行了）。这个在网上还真没搜到什么内容（当然我没去 Google 上搜，毕竟我英语真不好而且还被墙）。原来想放弃实现这个功能的，毕

竟不是要求实现的功能，而是我自己想加的。但是最后查阅 ArrayAdapter 的具体实现才知道原因。

可以发现 ArrayAdapter 是一个泛型类，继承了 BaseAdapter 类，实现了 FilterAble 和 ThemedSpinnerAdapter 接口（这个应该跟 Spinner 控件有关系）：

```
public class ArrayAdapter<T> extends BaseAdapter implements Filterable, ThemedSpinnerAdapter {
    /**
     * Lock used to modify the content of {@link #mObjects}. Any write operation
     * performed on the array should be synchronized on this lock. This lock is also
     * used by the filter (see {@link #getFilter()} to make a synchronized copy of
     * the original array of data.
     */
```

然后 getFilter()函数的实现如下（需要实现 FilterAble 接口），会返回一个 ArrayFilter 类的实例：

```
@NonNull @Override
public @NonNull Filter getFilter() {
    if (mFilter == null) {
        mFilter = new ArrayFilter();
    }
    return mFilter;
}
```

ArrayFilter 是 ArrayAdapter 的一个 private 内部类，继承了 Filter 类：

```
/**
 * <p>An array filter constrains the content of the array adapter with
 * a prefix. Each item that does not start with the supplied prefix
 * is removed from the list.</p>
 */
private class ArrayFilter extends Filter {
    @Override
    protected FilterResults performFiltering(CharSequence prefix) {
        final FilterResults results = new FilterResults();

        if (mOriginalValues == null) {
            synchronized (mLock) {
                mOriginalValues = new ArrayList<>(mObjects);
            }
        }
```

当 filter()函数被调用时，真正执行过滤操作的函数是 performFiltering()，可以发现过滤的逻辑如下：

```
    final int count = values.size();
    final ArrayList<T> newValues = new ArrayList<>();

    for (int i = 0; i < count; i++) {
        final T value = values.get(i);
        final String valueText = value.toString().toLowerCase();

        // First match against the whole, non-splitted value
        if (valueText.startsWith(prefixString)) {
            newValues.add(value);
        } else {
            final String[] words = valueText.split( regex: " ");
            for (String word : words) {
                if (word.startsWith(prefixString)) {
                    newValues.add(value);
                    break;
                }
            }
        }
    }

    results.values = newValues;
    results.count = newValues.size();
}
```

上图的过滤逻辑是根据 ListView 显示的每一项的字符串（对应数据源的某一项的 toString()方法返回的结果字符串）当中的空格将字符串切分，然后看被切分出的每一小部分是不是以过滤源字符串开头（不区分大小写），只要有一个小部分是，这个项就会被加入到过滤结果中。这显然不是我想要的逻辑。所以只能继承 ArrayAdapter 类然后重写一些函数了。根据我要达到的目的，com.example.musicplayer 包的 MyArrayAdapter 只实现了 Filterable 接口并且只重写了 getCount()、getItem()、getItemId()、getFilter()，定义了继承了 Filter 的内部类 MyArrayFilter 并重写了 performFiltering()和 publishResults()函数。

```
public class MyArrayAdapter<T> extends ArrayAdapter<T> implements Filterable {
    private MyArrayFilter filter;
    private List<T> mList; // 对应于ListView显示的数据源
    private List<T> originalList; // 数据源的备份
    /* 记录ListView的每个被过滤之后显示item所对应源数据的位置，每次过滤前都会清空
    *  如果是空的，则说明过滤结果不包含任何item，或者取消过滤模式 */
    private List<Integer> sortedList = new ArrayList<>();
```

比较重点的一个是 MyArrayFilter 的 performFiltering()中重写了过滤逻辑。和原来的 ArrayAdapter 一样有一个数据源的备份（originalList），在

过滤的时候将过滤结果替换原先的数据源（mList）的内容；取消过滤（源字符串长度为0或者源字符串为null）的时候使用备份数据源（originalList）恢复成原先数据源的内容。和原来不同的是重写了过滤逻辑，将过滤逻辑改成了显示的字符串是否包括源字符串（不区分大小写），如果包括源字符串就将该项加入到过滤结果中。

```
1     private class MyArrayFilter extends Filter {
2         @Override
3         protected FilterResults performFiltering(CharSequence charSequence)
{
4             FilterResults results = new FilterResults();
5             if (originalList == null) {
6                 originalList = new ArrayList<T>(mList);
7             }
8             sortedList.clear();
9             // 如果过滤的源字符串为null 或者长度为0，则取消过滤，从备份中恢复数
    据源
10            if (charSequence == null || charSequence.length() == 0) {
11                results.values = originalList;
12                results.count = originalList.size();
13            }
14            else {
15                final ArrayList<T> values;
16                values = new ArrayList<>(originalList); // 将originalList 拷
    贝到新的ArrayList 中
17                List<T> filteredList = new ArrayList<>();
18                int count = values.size();
19                for (int i = 0; i < count; i++) {
20                    final T value = values.get(i);
21                    final String valueText = value.toString().toLowerCase();
22                    if
    (valueText.contains(charSequence.toString().toLowerCase())) {
23                        filteredList.add(value);
24                        sortedList.add(i);
25                    }
26                }
27                results.values = filteredList;
28            }
29            return results;
30        }
31
32        @Override
33        protected void publishResults(CharSequence constraint, FilterResults
    results) {
```

```
34            //noinspection unchecked
35            mList = (List<T>) results.values; // 将过滤结果作为新的数据源
36            if (results.count > 0) {
37                notifyDataSetChanged();
38            } else {
39                notifyDataSetInvalidated();
40            }
41        }
42    }
```

最后一个问题就是过滤后得到的结果从 SelectMusicActivity 中无法根据 position 定位到真正要进行操作的音乐文件。因此在 MyArrayAdapter 中定义一个 sortedList，用于记录过滤结果对应于原先数据源中的位置，然后通过重写 getItemId()获取到音乐文件对应于原先数据源的位置：

```
@Override
public long getItemId(int position) {
    return sortedList.size() == 0 ? position : sortedList.get(position);
}
```

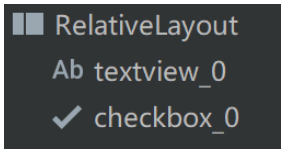
如果 sortedList 为空，则表明不处于过滤模式，直接返回原先的 position；反之返回该 position 在 sortedList 中的值（也就是对应原先数据源的位置）。这样 SelectMusicActivity 中就可以使用 getItemId()的返回值来索引 musicList 了。

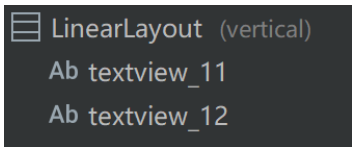
4.5 如何使 SettingsActivity 的 ListView 的每一项都有不同的布局

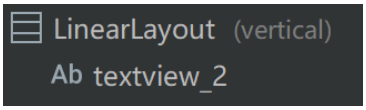
参考了网上的方法，在包 com.example.musicplayer 中自定义了一个 SettingsAdapter，继承 BaseAdapter，通过重写 getView()函数实现在 ListView 的每一项呈现不同的布局（首先为 ListView 的每个 item 创建控件，然后设置 ListView 的每个 item 中的控件应该显示的内容和监听事件）；通过重写 areAllItemsEnabled()和 isEnabled()函数来判断设置项是否可点击（主要是根据数据源的第 0 项来判断第二个设置项，也就是打开 ChoosePathActivity 的设置项是否可点击，数据源的第 0 项如果是"T"则第二个设置项不可点击，如果是"F"则第二个设置项可点击）。SettingsAdapter 的成员变量定义如下（变量的含义见注释）：

```
public class SettingsAdapter extends BaseAdapter {

    private Context context; // 使用此Adapter的源上下文
    private int[] layoutType; // 指定ListView的每一项要使用布局的类型
    private List<String> list; // 数据源
    private LayoutInflater inflater;
    /* 缓存ListView的每一个item所对应的控件，减少每次调用findViewById()的代价 */
    private View0 view0;
    private View1 view11;
    private View1 view12;
    private View2 view2;

    private class View0 {
        CheckBox checkBox;
        TextView textView;
    }

    private class View1 {
        TextView textView1;
        TextView textView2;
    }

    private class View2 {
        TextView textView;
    }
```

res/layout/view0.xml、res/layout/view1.xml、res/layout/view2.xml 是为不同的设置项准备的布局文件。三个文件的大致布局情况如下：







其中 view0.xml 对应第一个设置项，view1.xml 对应第二个和第三个设置项，view2.xml 对应最后一个设置项。

以下是 SettingsAdapter 的 areAllItemsEnabled()、isEnabled()、getView()函数的定义。其中为 CheckBox 设置的监听事件的含义是：如果 CheckBox 被选中，则设置数据源的第 0 项为"T"并且设置第二个设置项的文字颜色为灰色（表示不可点击）；否则设置数据源的第 0 项为"F"并且恢复第二个设置项文字本来的颜色。

```
1      @Override
2      public boolean areAllItemsEnabled() {
3          return false; // 表明并不是所有 item 都可点击
4      }
5
6      @Override
7      public boolean isEnabled(int position) {
8          if (position == 1) {
```

```
9             if (list.get(0).equals("F"))
10                return true;
11            else if (list.get(0).equals("T")) {
12                return false;
13            }
14        }
15        return true;
16    }
17
18    @Override
19    public View getView(int position, View convertView, ViewGroup parent)
{
20        int type = getItemViewType(position);
21        final int pos = position;
22        // 为ListView的每个item创建控件
23        if (convertView == null) {
24            switch (type) {
25                case 0:
26                    convertView = inflater.inflate(R.layout.view0, parent,
    false);
27                    view0 = new View0();
28                    view0.textView = (TextView)
    convertView.findViewById(R.id.textview_0);
29                    view0.checkBox = (CheckBox)
    convertView.findViewById(R.id.checkbox_0);
30                    convertView.setTag(view0); // 缓存
31                    break;
32                case 1:
33                    convertView = inflater.inflate(R.layout.view1, parent,
    false);
34                    if (position == 1) {
35                        view11 = new View1();
36                        view11.textView1 = (TextView)
    convertView.findViewById(R.id.textview_11);
37                        view11.textView2 = (TextView)
    convertView.findViewById(R.id.textview_12);
38                        convertView.setTag(view11);
39                    }
40                    else if (position == 2) {
41                        view12 = new View1();
42                        view12.textView1 = (TextView)
    convertView.findViewById(R.id.textview_11);
43                        view12.textView2 = (TextView)
    convertView.findViewById(R.id.textview_12);
```

```
44                      convertView.setTag(view12);
45                  }
46                  break;
47              case 2:
48                  convertView = inflater.inflate(R.layout.view2, parent,
    false);
49                  view2 = new View2();
50                  view2.textView = (TextView)
    convertView.findViewById(R.id.textview_2);
51                  convertView.setTag(view2);
52                  break;
53          }
54      }
55      else {
56          switch (type) { // 获取缓存的控件
57              case 0:
58                  view0 = (View0) convertView.getTag();
59                  break;
60              case 1:
61                  if (position == 1) {
62                      view11 = (View1) convertView.getTag();
63                  }
64                  else if (position == 2) {
65                      view12 = (View1) convertView.getTag();
66                  }
67                  break;
68              case 2:
69                  view2 = (View2) convertView.getTag();
70                  break;
71          }
72      }
73
74      // 在这里设置ListView的每个item中的控件应该显示的内容和监听事件
75      switch (position) {
76          case 0:
77              if (((String)getItem(position)).equals("T")) {
78                  view0.checkBox.setChecked(true);
79              }
80              else if (((String)getItem(position)).equals("F")) {
81                  view0.checkBox.setChecked(false);
82              }
83              view0.textView.setText(R.string.all_path);
84              final View finalConvertView = convertView;
```

```
85                view0.checkBox.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener() {
86                    @Override
87                    public void onCheckedChanged(CompoundButton buttonView,
    boolean isChecked) {
88                        if (isChecked) {
89                            list.set(0, "T");
90
    view11.textView1.setTextColor(finalConvertView.getResources().getColor(
    R.color.lightGray));
91
    view11.textView2.setTextColor(finalConvertView.getResources().getColor(
    R.color.lightGray));
92
    view11.textView2.setText(finalConvertView.getResources().getString(R.st
    ring.all_path));
93                        }
94                        else {
95                            list.set(0, "F");
96
    view11.textView1.setTextColor(finalConvertView.getResources().getColor(
    R.color.black));
97
    view11.textView2.setTextColor(finalConvertView.getResources().getColor(
    R.color.defaultGray));
98                            view11.textView2.setText((String)getItem(pos));
99                        }
100                       notifyDataSetChanged();
101                   }
102               });
103               break;
104           case 1:
105               view11.textView1.setText(R.string.set_path);
106               view11.textView2.setText((String)getItem(position));
107               if (list.get(0).equals("F")) {
108
    view11.textView1.setTextColor(convertView.getResources().getColor(R.col
    or.black));
109
    view11.textView2.setTextColor(convertView.getResources().getColor(R.col
    or.defaultGray));
110                   view11.textView2.setText((String)getItem(position));
111               }
112               else if (list.get(0).equals("T")) {
```

```
113
    view11.textView1.setTextColor(convertView.getResources().getColor(R.col
    or.gray));
114
    view11.textView2.setTextColor(convertView.getResources().getColor(R.col
    or.gray));
115
    view11.textView2.setText(convertView.getResources().getString(R.string.
    all_path));
116                 }
117                 break;
118             case 2:
119                 view12.textView1.setText(R.string.format_filter);
120                 view12.textView2.setText((String)getItem(position));
121                 break;
122             case 3:
123                 view2.textView.setText(R.string.about);
124                 break;
125         }
126         return convertView;
127     }
```

另外，在 SettingsActivity 的 onCreate()函数中，

```
55          if (receiveFormat != -1)
56              format = receiveFormat;
57          int[] layoutType = { 0, 1, 1, 2 }; // 指定ListView不同的item使用不同的Layout
58          list.add("");
59          list.add(null);
```

图中的第 57 行就是指定每个设置项使用什么布局文件。0 代表 view0.xml，1 代表 view1.xml，2 代表 view2.xml。

五、实验结论

5.1 实验结论与感想

1. 这次实验让我更加熟悉了 Intent 的使用，让我知道了 Intent 除了切换 Activity 之外其他的用处。ListView 是我在上次实验中没有使用到的控件，通过这次实验对 ListView 的各种使用方式的探索以及自定义 Adapter 也让我熟悉了 ListView 控件的使用。
2. 有了上次实验的基础，这次在界面布局和事件处理函数方面不需要像上次实

验那样花费太大精力去做并且在报告中描述了。当然这也跟这次使用到的控件本身就少也有关系。

3. 这两次实验还带给我一个经验就是如果想实现什么功能可以先去网上搜一搜有没有合适的解决方案，如果有可以参考。但是如果没有的话还是要查阅 API 文档或者源代码。如果已有的控件无法满足功能需求，尝试继承这个类并重写相应的函数是一个万能的手段，但是这需要较强的 Java 语言功底。

4. 这学期虽然快要结束了，但是从第一次配置环境的实验到现在学到的东西还是比较少，仍然只是 Android 开发最基本的东西，还不足以更好地实现一个功能完整的 app。因此要想在大作业实现一个功能相对完整的 app 还是要继续深入学习。

5. 还有，通过这两次实验，我感受到 Android 开发的一个难处就是不同手机不同版本的兼容性问题：一是原生 Android 系统不同版本之间差异大，二是不同品牌不同型号的手机对 Android 系统定制的程度也不同，三是不同品牌不同型号的手机硬件规格也不同（例如屏幕大小不同）。一个好的 Android app 一定是能够兼容绝大多数手机以及绝大多数系统版本的，无论是从程序能否正常运行的角度来看还从是界面实际显示效果的角度来看都是这样的。

6. 其中最明显的感受就是 Android 版本之间的差异，由于这次要通过 Intent 实现调用系统播放器，且不考虑不同定制版本的 Android 系统有没有内置播放器，就单是不同版本的 Android 系统获取 Uri 的方式就不同，而且申请 SD 卡访问权限的方式也不同。Android 6.0 需要动态申请权限，实际的效果就是在手机或者模拟器上第一次运行这个程序会弹出一个是否为该程序授予访问 SD 卡访问权限的对话框（忘记截图了），只有用户点击了“同意”，程序才能够访问或修改 SD 卡的文件，类似的功能 iOS 系统早就有了（例如是否允许程序访问通讯录）。早期的 Android 系统是没有这个功能的（直接在 `AndroidManifest.xml` 中写明要使用的权限即可），而 Android 软件生态又相对比较混乱（没有像苹果的 App Store 一样对上架的 app 有严格的监管措施），难免会有恶意程序使用了未经用户同意的权限，对用户的信息安全带来了很大的威胁，这就是后来的 Android 系统也引入了这个功能的原因吧。这个功能的引入一定程度上保护了用户的信息安全，但是对 Android 程序员来说就增加了编程的难度，因为要写不同的代码来兼容不同的系统版本，而且还要考虑不同版本会增加或者废除一些 API 函数。

5.2 其他建议

最后再为这门课程提一个小建议吧。无论是从第一次配置实验环境的文档来看，还是从这次实验内容对应于 ppt 的内容来看，都能看出来这些资料有过时的地方。

当然我可以理解，毕竟计算机的很多专业课的内容也都是相对来说比较过时的东西。如果说相应的知识或者实验环境多年以来都没什么大变化，过时一点也无伤大雅。但是我认为每次 Android 系统版本以及 Android Studio 版本更迭都会带来较大的变化，而这些很大的变化带来的问题主要体现在增加或者废除一些 API 函数的问题、有些方法在高版本并不适用的问题或者是安装和配置高版本的 Android Studio 与低版本 Android Studio 的方法上的差异。而如果每次实验都要把主要精力放在解决类似于这样的问题的话，很可能会打击一部分学习能力或者从网络上获取信息的能力较差的同学学习这门课的积极性，而且会偏离本来设置这些实验的目的。所以我觉得对于这门课而言，与时俱进是更好的选择。另外我也希望这门课可以设置一到两位可以与同学们沟通的助教，这样如果有同学遇到了问题而老师没空解决的话，助教可以帮忙解决一部分问题。

以上只是我个人的观点，如有不妥之处还请见谅。