Google、在奋斗

我只有很努力、你看我才觉毫不费力。

GL音乐播放器--代码实现<三>

分类： android开发-GL音乐播放器系列 2014-02-18 12:22 155人阅读 评论(0) 收藏 举报

今天，我就把我的musicService类贴出来，这个是播放器能运行的主要机制。由于昨天才将博客搬家过来，其实现在我的软件项目已经完成，本打算边写项目边写博客，看样子是做不到了，接下来的几篇博客会将我的主要功能代码贴出来，如果大家想直接看代码，就去我的资源那去下载，如果想看看具体解释，讲解，就继续跟我的博客，因为快开学的缘故，我会尽快完成。
话不多说，上代码：

首先是 MusicService类主要获取文件状态，这是我总的，你可以有选择的看，里面好多我会在后面的博客里提到

[cpp] view plain copy print ?

```
<span style="font-size:12px;color:#000000;">package com.genius.service;

import java.util.List;

import com.genius.aidl.MusicConnect;
import com.genius.aidl.MusicConnect.Stub;
import com.genius.musicplay.MusicData;
import com.genius.musicplay.MusicPlayer;

import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.os.RemoteException;
import android.util.Log;

public class MusicService extends Service{

    private static final String TAG = "MusicService";

    private MusicPlayer m_mMusicPlayer;

    private SDStateBrocast mSDStateBrocast;

    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return mBinder;
    }

    @Override
    public void onCreate() {
        // TODO Auto-generated method stub
        super.onCreate();

        m_mMusicPlayer = new MusicPlayer(this);
```

个人资料

Google_acmer

访问： 2572次
积分： 423分
排名： 千里之外

原创： 40篇 转载： 2篇
译文： 0篇 评论： 5条

公告栏

风过留痕
雁过留声
如果您真的喜欢我的文章
可以留下评论
我们一起讨论
希望对您的学习有所帮助
转载请注明出处、谢谢您的浏览

文章搜索

博客专栏

GL音乐播放器开发
文章：12篇
阅读：1726

文章分类

Android开发小知识积累系列 (16)
android开发-GL音乐播放器系列 (11)
ACM--字符串 (1)
ACM--省赛集 (1)
ACM--图论 (1)
ACM--排序 (1)
心情随笔 (3)
人生感悟 (3)
博客始记 (1)

```
39.         mSDStateBrocast = new SDStateBrocast();
40.         // 指定BroadcastReceiver监听的Action 检查SD卡情况
41.         IntentFilter intentFilter = new IntentFilter();
42.         intentFilter.addAction(Intent.ACTION_MEDIA_MOUNTED);
43.         intentFilter.addAction(Intent.ACTION_MEDIA_UNMOUNTED);
44.         intentFilter.addAction(Intent.ACTION_MEDIA_SCANNER_FINISHED);
45.         intentFilter.addAction(Intent.ACTION_MEDIA_EJECT);
46.         intentFilter.addDataScheme("file");
47.         // 注册BroadcastReceiver
48.         registerReceiver(mSDStateBrocast, intentFilter);
49. 
50.     }
51. 
52.     @Override
53.     public void onDestroy() {
54.         // TODO Auto-generated method stub
55. 
56.         unregisterReceiver(mSDStateBrocast);
57. 
58.         super.onDestroy();
59. 
60. 
61.     }
62. 
63. 
64. 
65. 
66. 
67.     private MusicConnect.Stub mBinder = new Stub() {
68. 
69.         @Override
70.         public void refreshMusicList(List<MusicData> musicFileList)
71.                 throws RemoteException {
72.             // TODO Auto-generated method stub
73.             m_mMusicPlayer.refreshMusicList(musicFileList);
74.         }
75. 
76. 
77.         @Override
78.         public void getFileList(List<MusicData> musicFileList)
79.                 throws RemoteException {
80.             // TODO Auto-generated method stub
81.             List<MusicData> tmp = m_mMusicPlayer.getFileList();
82.             int count = tmp.size();
83.             for(int i = 0; i < count; i++)
84.             {
85.                 musicFileList.add(tmp.get(i));
86.             }
87.         }
88. 
89.         @Override
90.         public int getCurPosition() throws RemoteException {
91.             // TODO Auto-generated method stub
92.             return m_mMusicPlayer.getCurPosition();
93.         }
94. 
95.         @Override
96.         public int getDuration() throws RemoteException {
97.             // TODO Auto-generated method stub
98.             return m_mMusicPlayer.getDuration();
99.         }
100. 
101.         @Override
102.         public boolean pause() throws RemoteException {
103.             // TODO Auto-generated method stub
104.             return m_mMusicPlayer.pause();
105.         }
106. 
107.         @Override
108.         public boolean play(int position) throws RemoteException {
109.             // TODO Auto-generated method stub
110. 
111.             Log.i(TAG, "play pos = " + position);
112.             return m_mMusicPlayer.play(position);
113.         }
114. 
115.         @Override
116.         public boolean playNext() throws RemoteException {
117.             // TODO Auto-generated method stub
```

```
118.             return m_mMusicPlayer.playNext();
119.         }
120. 
121.         @Override
122.         public boolean playPre() throws RemoteException {
123.             // TODO Auto-generated method stub
124.             return m_mMusicPlayer.playPre();
125.         }
126. 
127.         @Override
128.         public boolean rePlay() throws RemoteException {
129.             // TODO Auto-generated method stub
130.             return m_mMusicPlayer.replay();
131.         }
132. 
133.         @Override
134.         public boolean seekTo(int rate) throws RemoteException {
135.             // TODO Auto-generated method stub
136.             return m_mMusicPlayer.seekTo(rate);
137.         }
138. 
139.         @Override
140.         public boolean stop() throws RemoteException {
141.             // TODO Auto-generated method stub
142.             return m_mMusicPlayer.stop();
143.         }
144. 
145.         @Override
146.         public int getPlayState() throws RemoteException {
147.             // TODO Auto-generated method stub
148.             return m_mMusicPlayer.getPlayState();
149.         }
150. 
151.         @Override
152.         public void exit() throws RemoteException {
153.             // TODO Auto-generated method stub
154.             m_mMusicPlayer.exit();
155.         }
156. 
157. 
158.         @Override
159.         public void sendPlayStateBrocast() throws RemoteException {
160.             // TODO Auto-generated method stub
161.             m_mMusicPlayer.sendPlayStateBrocast();
162.         }
163. 
164. 
165.         @Override
166.         public void setPlayMode(int mode) throws RemoteException {
167.             // TODO Auto-generated method stub
168.             m_mMusicPlayer.setPlayMode(mode);
169.         }
170. 
171. 
172.         @Override
173.         public int getPlayMode() throws RemoteException {
174.             // TODO Auto-generated method stub
175.             return m_mMusicPlayer.getPlayMode();
176.         }
177. 
178. 
179.     };
180. 
181. 
182.     //自定义的BroadcastReceiver，负责监听从Service传回来的广播
183.     class  SDStateBrocast extends BroadcastReceiver
184.     {
185. 
186.         @Override
187.         public void onReceive(Context context, Intent intent) {
188.             // TODO Auto-generated method stub
189.              String action = intent.getAction();
190.               if (action.equals(Intent.ACTION_MEDIA_MOUNTED))//插入SD卡并且已正确安装（识
别）
191.               {
192. 
193.               }else if (action.equals(Intent.ACTION_MEDIA_UNMOUNTED))//SD卡存在，但是还没
有被挂载
194.               {
```

```
195.
196.                }else if (Intent.ACTION_MEDIA_SCANNER_FINISHED.equals(action))//已经扫描SD
的目录
197.                {
198.
199.                }else if (Intent.ACTION_MEDIA_EJECT.equals(action))//已拔掉SD卡
200.                {
201.                    m_mMusicPlayer.exit();
202.                }
203.
204.            }
205.        }
206.    }
207.    </span>
```

接下来是MusicPlayer，主要控制播放状态模式改变对应的响应

```
[java] view plain copy print ?
01.  <span style="font-size:12px;color:#000000;">package com.genius.musicplay;
02.
03.  import java.util.ArrayList;
04.  import java.util.List;
05.  import java.util.Random;
06.
07.  import android.content.Context;
08.  import android.content.Intent;
09.  import android.media.MediaPlayer;
10.  import android.media.MediaPlayer.OnCompletionListener;
11.  import android.media.MediaPlayer.OnErrorListener;
12.  import android.os.Bundle;
13.  import android.os.Environment;
14.  import android.util.Log;
15.
16.  public class MusicPlayer implements OnCompletionListener, OnErrorListener{
17.
18.      private final static String TAG = "MusicPlayer";
19.
20.      private final String BROCAST_NAME = "com.genius.musicplay.brocast";
21.
22.      private MediaPlayer mMediaPlayer;                   // 播放器对象
23.
24.      private List<MusicData> mMusicFileList;               // 音乐文件列表
25.
26.      private int mCurPlayIndex;                       // 当前播放索引
27.
28.      private int mPlayState;                              // 播放器状态
29.
30.      private int mPLayMode;                         // 歌曲播放模式
31.
32.      private Random mRandom;
33.
34.      private Context mContext;
35.
36.      private void defaultParam()
37.      {
38.          mMediaPlayer = new MediaPlayer();
39.
40.          mMediaPlayer.setOnCompletionListener(this);
41.
42.          mMediaPlayer.setOnErrorListener(this);
43.
44.          mMusicFileList = new ArrayList<MusicData>();
45.
46.          mCurPlayIndex = -1;
47.
48.          mPlayState = MusicPlayState.MPS_NOFILE;
49.
50.          mPLayMode = MusicPlayMode.MPM_LIST_LOOP_PLAY;
51.
52.
53.      }
54.
55.      public MusicPlayer(Context context){
56.          mContext = context;
57.          defaultParam();
58.
59.          mRandom = new Random();
60.          mRandom.setSeed(System.currentTimeMillis());
```

```
61.     }
62.
63.     public void exit()
64.     {
65.         mMediaPlayer.reset();
66.         mMusicFileList.clear();
67.         mCurPlayIndex = -1;
68.         mPlayState = MusicPlayState.MPS_NOFILE;
69.     }
70.
71.     public void refreshMusicList(List<MusicData> FileList)
72.     {
73.
74.         if (FileList == null)
75.         {
76.             mMusicFileList.clear();
77.             mPlayState = MusicPlayState.MPS_NOFILE;
78.             mCurPlayIndex = -1;
79.             return ;
80.         }
81.
82.         mMusicFileList = FileList;
83.
84.         if (mMusicFileList.size() == 0)
85.         {
86.             mPlayState = MusicPlayState.MPS_NOFILE;
87.             mCurPlayIndex = -1;
88.             return ;
89.         }
90.
91.
92.         switch(mPlayState)
93.         {
94.         case MusicPlayState.MPS_NOFILE:
95.             prepare(0);
96.             break;
97.         case MusicPlayState.MPS_INVALID:
98.             prepare(0);
99.             break;
100.        case MusicPlayState.MPS_PREPARE:
101.            prepare(0);
102.            break;
103.        case MusicPlayState.MPS_PLAYING:
104.            break;
105.        case MusicPlayState.MPS_PAUSE:
106.            break;
107.        default:
108.            break;
109.        }
110.    }
111.
112.    public List<MusicData> getFileList()
113.    {
114.        Log.i(TAG, "getFileList mMusicFileList.size = " + mMusicFileList.size());
115.        return mMusicFileList;
116.    }
117.
118.    public int getPlayState()
119.    {
120.        return mPlayState;
121.    }
122.
123.
124.    public boolean replay()
125.    {
126.
127.        if (mPlayState == MusicPlayState.MPS_NOFILE || mPlayState == MusicPlayState.MPS_I
128.        {
129.            return false;
130.        }
131.
132.        mMediaPlayer.start();
133.        mPlayState = MusicPlayState.MPS_PLAYING;
134.        sendPlayStateBrocast();
135.
136.        return true;
137.    }
138.
139.    public boolean play(int position)
```

```
140.     {
141.         if (mPlayState == MusicPlayState.MPS_NOFILE)
142.         {
143.             return false;
144.         }
145. 
146.         if (mCurPlayIndex == position)
147.         {
148.             if (mMediaPlayer.isPlaying() == false)
149.             {
150.                 mMediaPlayer.start();
151.                 mPlayState = MusicPlayState.MPS_PLAYING;
152.                 sendPlayStateBrocast();
153.             }
154. 
155.             return true;
156.         }
157. 
158.         mCurPlayIndex = position;
159.         if (prepare(mCurPlayIndex) == false)
160.         {
161.             return false;
162.         }
163. 
164.         return replay();
165. 
166.     }
167. 
168.     public boolean pause()
169.     {
170.         if (mPlayState != MusicPlayState.MPS_PLAYING)
171.         {
172.             return false;
173.         }
174. 
175.         mMediaPlayer.pause();
176.         mPlayState = MusicPlayState.MPS_PAUSE;
177.         sendPlayStateBrocast();
178. 
179.         return true;
180.     }
181. 
182.     public boolean stop()
183.     {
184.         if (mPlayState != MusicPlayState.MPS_PLAYING && mPlayState != MusicPlayState.MPS_
185.         {
186.             return false;
187.         }
188. 
189. 
190.         return prepare(mCurPlayIndex);
191. 
192.     }
193. 
194.     public boolean playNext()
195.     {
196.         if (mPlayState == MusicPlayState.MPS_NOFILE)
197.         {
198.             return false;
199.         }
200. 
201. 
202.         mCurPlayIndex++;
203.         mCurPlayIndex = reviceIndex(mCurPlayIndex);
204. 
205. 
206.         if (prepare(mCurPlayIndex) == false)
207.         {
208.             return false;
209.         }
210. 
211.         return replay();
212.     }
213. 
214.     public boolean playPre()
215.     {
216.         if (mPlayState == MusicPlayState.MPS_NOFILE)
217.         {
218.             return false;
```

```
219.         }
220.
221.
222.         mCurPlayIndex--;
223.         mCurPlayIndex = reviceIndex(mCurPlayIndex);
224.
225.
226.         if (prepare(mCurPlayIndex) == false)
227.         {
228.             return false;
229.         }
230.
231.         return replay();
232.     }
233.
234.     public boolean seekTo(int rate)
235.     {
236.         if (mPlayState == MusicPlayState.MPS_NOFILE || mPlayState == MusicPlayState.MPS_I
237.         {
238.             return false;
239.         }
240.
241.         int r = reviceSeekValue(rate);
242.         int time = mMediaPlayer.getDuration();
243.         int curTime = (int) ((float)r / 100 * time);
244.
245.         mMediaPlayer.seekTo(curTime);
246.         return true;
247.     }
248.
249.     public int getCurPosition()
250.     {
251.         if (mPlayState == MusicPlayState.MPS_PLAYING || mPlayState == MusicPlayState.MPS_
252.         {
253.             return mMediaPlayer.getCurrentPosition();
254.         }
255.
256.         return 0;
257.     }
258.
259.     public int getDuration()
260.     {
261.
262.         if (mPlayState == MusicPlayState.MPS_NOFILE || mPlayState == MusicPlayState.MPS_I
263.         {
264.             return 0;
265.         }
266.
267.         return mMediaPlayer.getDuration();
268.
269.
270.     }
271.
272.
273.     public void setPlayMode(int mode)
274.     {
275.         switch(mode)
276.         {
277.         case MusicPlayMode.MPM_SINGLE_LOOP_PLAY:
278.         case MusicPlayMode.MPM_ORDER_PLAY:
279.         case MusicPlayMode.MPM_LIST_LOOP_PLAY:
280.         case MusicPlayMode.MPM_RANDOM_PLAY:
281.             mPLayMode = mode;
282.             break;
283.         }
284.
285.     }
286.
287.     public int getPlayMode()
288.     {
289.         return mPLayMode;
290.     }
291.
292.
293.
294.     private int reviceIndex(int index)
295.     {
296.         if (index < 0)
297.         {
```

```
298.                index = mMusicFileList.size() - 1;
299.            }
300.
301.            if (index >= mMusicFileList.size())
302.            {
303.                index = 0;
304.            }
305.
306.            return index;
307.        }
308.
309.        private int reviceSeekValue(int value)
310.        {
311.            if (value < 0)
312.            {
313.                value = 0;
314.            }
315.
316.            if (value > 100)
317.            {
318.                value = 100;
319.            }
320.
321.            return value;
322.        }
323.
324.        private int getRandomIndex()
325.        {
326.            int size = mMusicFileList.size();
327.            if (size == 0)
328.            {
329.                return -1;
330.            }
331.            return Math.abs(mRandom.nextInt() % size);
332.        }
333.
334.        private boolean prepare(int index)
335.        {
336.            Log.i(TAG, "prepare index = " + index);
337.            mCurPlayIndex = index;
338.            mMediaPlayer.reset();
339.
340.            String path = mMusicFileList.get(index).mMusicPath;
341.            try {
342.                mMediaPlayer.setDataSource(path);
343.                mMediaPlayer.prepare();
344.                mPlayState = MusicPlayState.MPS_PREPARE;
345.                Log.i(TAG, "mMediaPlayer.prepare    path = " + path);
346.                sendPlayStateBrocast();
347.            } catch (Exception e) {
348.                // TODO: handle exception
349.                e.printStackTrace();
350.                mPlayState = MusicPlayState.MPS_INVALID;
351.                sendPlayStateBrocast();
352.                return false;
353.            }
354.
355.            return true;
356.        }
357.
358.        @Override
359.        public void onCompletion(MediaPlayer mp) {
360.            // TODO Auto-generated method stub
361.            Log.i(TAG, "mPLayMode = " + mPLayMode);
362.            switch (mPLayMode) {
363.            case MusicPlayMode.MPM_SINGLE_LOOP_PLAY:
364.                play(mCurPlayIndex);
365.                break;
366.            case MusicPlayMode.MPM_ORDER_PLAY:
367.                {
368.                    if (mCurPlayIndex != mMusicFileList.size() - 1)
369.                    {
370.                        playNext();
371.                    }else{
372.                        prepare(mCurPlayIndex);
373.                    }
374.                }
375.                break;
376.            case MusicPlayMode.MPM_LIST_LOOP_PLAY:
```

```
            playNext();
            break;
        case MusicPlayMode.MPM_RANDOM_PLAY:
            {
                int index = getRandomIndex();
                if (index != -1)
                {
                    mCurPlayIndex = index;
                }else{
                    mCurPlayIndex++;
                }
                mCurPlayIndex = reviceIndex(mCurPlayIndex);


                if (prepare(mCurPlayIndex))
                {
                    replay();
                }
            }
            break;
        default:
            prepare(mCurPlayIndex);
            break;
        }



    }


    public void sendPlayStateBrocast()
    {

        if (mContext != null)
        {
            Intent intent = new Intent(BROCAST_NAME);
            intent.putExtra(MusicPlayState.PLAY_STATE_NAME, mPlayState);
            intent.putExtra(MusicPlayState.PLAY_MUSIC_INDEX, mCurPlayIndex);

            if (mPlayState != MusicPlayState.MPS_NOFILE)
            {
                Bundle bundle = new Bundle();
                MusicData data = mMusicFileList.get(mCurPlayIndex);

                bundle.putParcelable(MusicData.KEY_MUSIC_DATA, data);
                intent.putExtra(MusicData.KEY_MUSIC_DATA, bundle);
            }


            mContext.sendBroadcast(intent);
        }


    }

    @Override
    public boolean onError(MediaPlayer mp, int what, int extra) {
        // TODO Auto-generated method stub

        Log.e(TAG, "MusicPlayer     onError!!!\n");


        return false;
    }
}
</span>
```

在接下来便是在主要的Activity里的运用

```
[java] view plain  copy  print  ?
<span style="font-size:12px;">package com.genius.demo;

import java.util.ArrayList;
import java.util.List;

import android.view.KeyEvent;
import com.genius.adapter.GridViewAdapter;
```

```
08. import com.genius.adapter.ListViewAdapter;
09. import com.genius.adapter.MenuAdapter;
10. import com.genius.interfaces.IOnServiceConnectComplete;
11. import com.genius.interfaces.IOnSliderHandleViewClickListener;
12. import com.genius.musicplay.MediaAlbum;
13. import com.genius.musicplay.MusicData;
14. import com.genius.musicplay.MusicPlayMode;
15. import com.genius.musicplay.MusicPlayState;
16. import com.genius.service.ServiceManager;
17. import com.genius.widget.GalleryFlow;
18. import com.genius.widget.ImageAdapter;
19. import com.genius.widget.ImageUtil;
20. import com.genius.widget.MySlidingDrawer;
21. import com.genius.widget.Settings;
22. import com.genius.demo.Menu;
23. import com.genius.demo.MusicPlayActivity.UIManager.PopMenuManager;
24. import com.genius.demo.MusicPlayActivity.UIManager.SliderDrawerManager;
25. import android.R.bool;
26. import android.app.Activity;
27. import android.app.AlertDialog;
28. import android.app.Dialog;
29. import android.app.SearchManager.OnDismissListener;
30. import android.app.Service;
31. import android.content.BroadcastReceiver;
32. import android.content.ContentResolver;
33. import android.content.ContentUris;
34. import android.content.Context;
35. import android.content.DialogInterface;
36. import android.content.Intent;
37. import android.content.IntentFilter;
38. import android.content.SharedPreferences;
39. import android.database.Cursor;
40. import android.graphics.Bitmap;
41. import android.graphics.BitmapFactory;
42. import android.graphics.BitmapFactory.Options;
43. import android.graphics.drawable.ColorDrawable;
44. import android.graphics.drawable.LevelListDrawable;
45. import android.hardware.Sensor;
46. import android.hardware.SensorEvent;
47. import android.hardware.SensorEventListener;
48. import android.hardware.SensorManager;
49. import android.media.AudioManager;
50. import android.net.Uri;
51. import android.opengl.Visibility;
52. import android.os.Bundle;
53. import android.os.Environment;
54. import android.os.Handler;
55. import android.os.Message;
56. import android.os.ParcelFileDescriptor;
57. import android.os.Vibrator;
58.
59. import android.provider.MediaStore;
60. import android.telephony.PhoneStateListener;
61. import android.telephony.TelephonyManager;
62. import android.util.Log;
63. import android.view.Gravity;
64. import android.view.KeyEvent;
65. import android.view.LayoutInflater;
66. import android.view.View;
67. import android.view.View.OnClickListener;
68. import android.view.View.OnKeyListener;
69. import android.view.ViewGroup.LayoutParams;
70. import android.view.animation.Animation;
71. import android.view.animation.AnimationUtils;
72. import android.widget.AdapterView;
73. import android.widget.Button;
74. import android.widget.GridView;
75. import android.widget.ImageButton;
76. import android.widget.ImageView;
77. import android.widget.ListView;
78. import android.widget.PopupWindow;
79. import android.widget.RelativeLayout;
80. import android.widget.SeekBar;
81. import android.widget.SlidingDrawer;
82. import android.widget.TextView;
83. import android.widget.Toast;
84. import android.widget.AdapterView.OnItemClickListener;
85. import android.widget.SeekBar.OnSeekBarChangeListener;
86. import android.widget.SlidingDrawer.OnDrawerCloseListener;
```

```
87. import android.widget.SlidingDrawer.OnDrawerOpenListener;
88. 
89. /**
90.  * @author Google_acmer
91.  *
92.  */
93. public  class MusicPlayActivity extends Activity implements IOnServiceConnectComplete, Se
94.     /** Called when the activity is first created. */
95.     private final static String TAG = "MusicPlayActivity";
96.     private SensorManager sensorManager;
97.     private boolean mRegisteredSensor;
98.     private final String BROCAST_NAME = "com.genius.musicplay.brocast";
99.     private  int isplay=0;
100.    private final static int REFRESH_PROGRESS_EVENT = 0x100;
101.    private final static int ABOUT_DIALOG_ID = 1;
102.    private Menu xmenu; // 自定义菜单
103.    private ImageView musicAlbum;   //音乐专辑封面
104.    private ImageView musicAblumReflection; //倒影反射
105.    Vibrator vibrator;//声明振动器
106.    private Handler mHandler;
107.
108.    private UIManager mUIManager;
109.
110.    private ServiceManager mServiceManager;
111.
112.    private MusicTimer mMusicTimer;
113.
114.    private MusicPlayStateBrocast mPlayStateBrocast;
115.
116.    private SDStateBrocast mSDStateBrocast;
117.
118.    private List<MusicData> m_MusicFileList;
119.
120.    private ListViewAdapter mListViewAdapter;
121.
122.    private boolean mIsSdExist = false;
123.    private boolean mIsHaveData = false;
124.
125.    private int mCurMusicTotalTime = 0;
126.
127.    private int mCurPlayMode = MusicPlayMode.MPM_LIST_LOOP_PLAY;
128.    private int listPosition=0; // 播放歌曲在mp3Infos的位置
129.    SharedPreferences preferences;
130.    private String listPosition1;
131.    private int Shake_pause;
132.    private int Shake_next;
133.    public void onCreate(Bundle savedInstanceState) {
134.        super.onCreate(savedInstanceState);
135.        setContentView(R.layout.main);
136.        //创建data文件存取数据
137.        preferences = getSharedPreferences("data", 0);
138.        int position=preferences.getInt("background_id", 0);//改变后的皮肤ID
139.        Shake_pause=preferences.getInt("Shake_id", 0);//甩歌暂停
140.        Shake_next=preferences.getInt("Shake1_id", 0);//甩歌切歌
141.
142.        Settings mSetting = new Settings(this, true);
143.        this.getWindow().setBackgroundDrawableResource(Settings.SKIN_RESOURCES[position])
初始化壁纸
144.        long time1 = System.currentTimeMillis();
145.        init();
146.        long time2 = System.currentTimeMillis();
147.
148.        sensorManager=(SensorManager) getSystemService(SENSOR_SERVICE);//获取系统的传感器
管理服务
149.        TelephonyManager telManager = (TelephonyManager) getSystemService(Context.TELEPHO
取得TelephonyManager服务
150.        telManager.listen(new MobliePhoneStateListener(),PhoneStateListener.LISTEN_CALL_S
151.        vibrator =(Vibrator)getSystemService(Service.VIBRATOR_SERVICE);//获取系统的震动服
务
152.        LoadMenu();
153.    }
154.  //   定义菜单
155.    private void LoadMenu() {
156.        xmenu = new Menu(this);
157.        List<int[]> data1 = new ArrayList<int[]>();
158.        data1.add(new int[] { R.drawable.btn_menu_skin, R.string.skin_settings });
159.        data1.add(new int[] { R.drawable.btn_menu_exit, R.string.menu_exit_txt });
160.
161.        xmenu.addItem("常用", data1, new MenuAdapter.ItemListener() {
```

```
162.
163.            public void onClickListener(int position, View view) {
164.                xmenu.cancel();
165.                if (position == 0) {
166.                    Intent intent = new Intent(MusicPlayActivity.this,SkinSettingActivity
167.                    startActivityForResult(intent, 1);
168.                    //Toast.makeText(MusicPlayActivity.this, "正在开
发！", Toast.LENGTH_SHORT).show();
169.                } else if (position == 1) {
170.                    exit();
171.                }
172.            }
173.        });
174.        List<int[]> data2 = new ArrayList<int[]>();
175.        data2.add(new int[] { R.drawable.btn_menu_setting,
176.                R.string.menu_settings });
177.        data2.add(new int[] { R.drawable.btn_menu_sleep, R.string.menu_time_txt});
178.        data2.add(new int[] { R.drawable.btn_menu_sleep, R.string.menu_time_txt1});
179.
180.        xmenu.addItem("工具", data2, new MenuAdapter.ItemListener() {
181.
182.            @Override
183.            public void onClickListener(int position, View view) {
184.            //  xmenu.cancel();
185.                if (position == 0) {
186.                    Toast.makeText(MusicPlayActivity.this, "此版本暂无！请下个版本再
试！", Toast.LENGTH_SHORT).show();
187.                } else if (position == 1) {
188.                    if(Shake_pause!=20)
189.                    {Shake_pause=20;//甩歌暂停启动
190.                    preferences.edit().putInt("Shake_id", Shake_pause).commit();
191.                    Toast.makeText(MusicPlayActivity.this, "甩歌暂停已启动，甩甩试
试", Toast.LENGTH_SHORT).show();
192.                    }
193.                    else
194.                    {Shake_pause=10;//甩歌暂停暂停
195.                    preferences.edit().putInt("Shake_id", Shake_pause).commit();
196.                    Toast.makeText(MusicPlayActivity.this, "甩歌暂停已关
闭", Toast.LENGTH_SHORT).show();
197.                    }
198.                } else if (position == 2) {
199.                    if(Shake_next!=20)
200.                    {Shake_next=20;//甩歌切歌启动
201.                    preferences.edit().putInt("Shake1_id", Shake_next).commit();
202.                    Toast.makeText(MusicPlayActivity.this, "甩歌切歌已启动，甩甩试
试", Toast.LENGTH_SHORT).show();
203.                    }
204.                    else
205.                    {Shake_next=10;//甩歌切歌暂停
206.                    preferences.edit().putInt("Shake1_id", Shake_next).commit();
207.                    Toast.makeText(MusicPlayActivity.this, "甩歌切歌已关
闭", Toast.LENGTH_SHORT).show();
208.                    }
209.
210.                }
211.            }
212.
213.        });
214.        List<int[]> data3 = new ArrayList<int[]>();
215.        data3.add(new int[] { R.drawable.ic_menu_about_default, R.string.about_title });
216.        data3.add(new int[] { R.drawable.ic_menu_back_pressed, R.string.back_message });
217.        xmenu.addItem("帮助", data3, new MenuAdapter.ItemListener() {
218.            @Override
219.            public void onClickListener(int position, View view) {
220.                if (position == 0) {
221.                    showDialog(ABOUT_DIALOG_ID);
222.                } else if (position == 1) {
223.                    Intent intent = new Intent(MusicPlayActivity.this,SendMessage.class);
224.                    startActivity(intent);
225.                }
226.
227.            }
228.        });
229.        xmenu.create();     //创建菜单
230.    }
231.    //得到改变后皮肤的数据
232.    @Override
233.    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
234.        super.onActivityResult(requestCode, resultCode, data);
```

```
235.         if (requestCode == 1 && requestCode == 1) {
236.             Settings setting = new Settings(this, false);
237.             this.getWindow().setBackgroundDrawableResource(setting.getCurrentSkinResId())
238.         }
239.     }
240.     @Override
241.     public boolean onCreateOptionsMenu(android.view.Menu menu) {
242.         menu.add("menu");
243.         return super.onCreateOptionsMenu(menu);
244.     }
245.
246.
247.     @Override
248.     public boolean onMenuOpened(int featureId, android.view.Menu menu) {
249.         // 菜单在哪里显示。参数1是该布局总的ID，第二个位置，第三，四个是XY坐标
250.         xmenu.showAtLocation(findViewById(R.id.mainLayout), Gravity.BOTTOM
251.                 | Gravity.CENTER_HORIZONTAL, 0,0);
252.         // 如果返回true的话就会显示系统自带的菜单，反之返回false的话就是显示自己写的
253.         return false;
254.     }
255.
256.
257.     @Override
258.     protected void onDestroy() {
259.         // TODO Auto-generated method stub
260.         super.onDestroy();
261.
262.         mMusicTimer.stopTimer();
263.         unregisterReceiver(mPlayStateBrocast);
264.         unregisterReceiver(mSDStateBrocast);
265.         mServiceManager.disconnectService();
266.
267.
268.     }
269.
270.     public void onResume()
271.     {
272.         super.onResume();
273.         //为系统的重力传感器注册监听器
274.         List<Sensor> sensors=sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
275.         if(sensors.size()>0){
276.             Sensor sensor=sensors.get(0);
277.             mRegisteredSensor=sensorManager.registerListener(this, sensor, SensorManager.
278.         }
279.     }
280.
281.     public void onStart()
282.     {
283.         super.onStart();
284.     }
285.
286.
287.
288.
289.     @Override
290.     protected Dialog onCreateDialog(int id) {
291.         // TODO Auto-generated method stub
292.
293.         switch(id)
294.         {
295.         case ABOUT_DIALOG_ID:
296.         {
297.             Dialog aboutDialog = new AlertDialog.Builder(MusicPlayActivity.this)
298.             .setIcon(R.drawable.about_dialog_icon)
299.             .setTitle(R.string.about_title_name)
300.             .setMessage(R.string.about_content)
301.             .setPositiveButton("确定", new DialogInterface.OnClickListener() {
302.                 public void onClick(DialogInterface dialog, int whichButton) {
303.
304.                     /* User clicked OK so do some stuff */
305.                 }
306.             }).create();
307.
308.             return aboutDialog;
309.         }
310.         default:
311.             break;
312.         }
313.
```

```
314.         return null;
315.     }
316.
317.
318.
319.     @Override
320.     public void onBackPressed() {
321.         // TODO Auto-generated method stub
322.
323.         mUIManager.Back();
324.     }
325.
326.
327.
328.     public void init()
329.     {
330.         mHandler = new Handler(){
331.
332.             @Override
333.             public void handleMessage(Message msg) {
334.                 // TODO Auto-generated method stub
335.
336.                 switch(msg.what)
337.                 {
338.                     case REFRESH_PROGRESS_EVENT:
339.                         mUIManager.setPlayInfo(mServiceManager.getCurPosition(), mCurMusi
340.                         break;
341.                     default:
342.                         break;
343.                 }
344.             }
345.
346.         };
347.
348.         mUIManager = new UIManager();
349.
350.         mServiceManager = new ServiceManager(this);
351.         mServiceManager.setOnServiceConnectComplete(this);
352.         mServiceManager.connectService();
353.
354.
355.         mMusicTimer = new MusicTimer(mHandler, REFRESH_PROGRESS_EVENT);
356.
357.         m_MusicFileList = new ArrayList<MusicData>();
358.         mListViewAdapter = new ListViewAdapter(this, m_MusicFileList);
359.         mUIManager.mListView.setAdapter(mListViewAdapter);
360.
361.
362.         mPlayStateBrocast = new MusicPlayStateBrocast();
363.         IntentFilter intentFilter1 = new IntentFilter(BROCAST_NAME);
364.         registerReceiver(mPlayStateBrocast, intentFilter1);
365.
366.         mSDStateBrocast = new SDStateBrocast();
367.         IntentFilter intentFilter2 = new IntentFilter();
368.         intentFilter2.addAction(Intent.ACTION_MEDIA_MOUNTED);
369.         intentFilter2.addAction(Intent.ACTION_MEDIA_UNMOUNTED);
370.         intentFilter2.addAction(Intent.ACTION_MEDIA_SCANNER_FINISHED);
371.         intentFilter2.addAction(Intent.ACTION_MEDIA_EJECT);
372.         intentFilter2.addDataScheme("file");
373.         registerReceiver(mSDStateBrocast, intentFilter2);
374.     }
375.     //读取音乐列表
376.     private List<MusicData> getMusicFileList()
377.     {
378.         List<MusicData> list = new ArrayList<MusicData>();
379.         //获取专辑封面的Uri
380.
381.         String[] projection = new String[]{MediaStore.Audio.Media._ID,
382.                                         MediaStore.Audio.Media.TITLE,
383.                                         MediaStore.Audio.Media.DURATION,
384.                                         MediaStore.Audio.Media.DATA,
385.                                         MediaStore.Audio.Media.ARTIST,
386.                                         MediaStore.Audio.Media.ALBUM_ID,
387.                                         };
388.
389.         long time1 = System.currentTimeMillis();
390.         Cursor cursor = getContentResolver().query(MediaStore.Audio.Media.EXTERNAL_CONTEN
391.         if (cursor != null)
392.         {
```

```
393.            cursor.moveToFirst();
394.            int colIdIndex = cursor.getColumnIndex(MediaStore.Audio.Media._ID);
395.            int colNameIndex = cursor.getColumnIndex(MediaStore.Audio.Media.TITLE);
396.            int colTimeIndex = cursor.getColumnIndex(MediaStore.Audio.Media.DURATION);
397.            int colPathIndex = cursor.getColumnIndex(MediaStore.Audio.Media.DATA);
398.            int colArtistIndex = cursor.getColumnIndex(MediaStore.Audio.Media.ARTIST);
399.            int albumId = cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM_ID);
400.            int fileNum = cursor.getCount();
401.            for(int counter = 0; counter < fileNum; counter++){
402.
403.                MusicData data = new MusicData();
404.                data.mMusicName = cursor.getString(colNameIndex);
405.                data.mMusicTime = cursor.getInt(colTimeIndex);
406.                data.mMusicPath = cursor.getString(colPathIndex);
407.                data.mMusicAritst = cursor.getString(colArtistIndex);
408.                data.MusicAlbumId = cursor.getInt(albumId);
409.                data.MusicId= cursor.getInt(colIdIndex);
410.                list.add(data);
411.                cursor.moveToNext();
412.            }
413.
414.            cursor.close();
415.        }
416.        long time2 = System.currentTimeMillis();
417.
418.        Log.i(TAG, "seach filelist cost = " + (time2 - time1));
419.        return list;
420.    }
421.
422.    public void showNoData()
423.    {
424.        //Toast.makeText(this, "No Music Data...", Toast.LENGTH_SHORT).show();
425.    }
426.
427.    public void rePlay()
428.    {
429.        if (mIsHaveData == false)
430.        {
431.            showNoData();
432.        }else{
433.            mServiceManager.rePlay();
434.        }
435.
436.
437.    }
438.
439.    public void play(int position)
440.    {
441.        if (mIsHaveData == false)
442.        {
443.            showNoData();
444.        }else{
445.            mServiceManager.play(position);
446.        }
447.
448.    }
449.
450.    public void pause()
451.    {
452.        mServiceManager.pause();
453.    }
454.
455.    public void stop()
456.    {
457.        mServiceManager.stop();
458.    }
459.
460.
461.    public void playPre()
462.    {
463.        if (mIsHaveData == false)
464.        {
465.            showNoData();
466.        }else{
467.            mServiceManager.playPre();
468.        }
469.
470.    }
471.
```

```
472.    public void playNext()
473.    {
474.        if (mIsHaveData == false)
475.        {
476.            showNoData();
477.        }else{
478.            mServiceManager.playNext();
479.        }
480.
481.    }
482.
483.
484.    public void seekTo(int rate)
485.    {
486.        mServiceManager.seekTo(rate);
487.    }
488.
489.    public void exit()
490.    {
491.        mServiceManager.exit();
492.        finish();
493.    }
494.
495.    public void modeChange()
496.    {
497.        mCurPlayMode++;
498.        if (mCurPlayMode > MusicPlayMode.MPM_RANDOM_PLAY)
499.        {
500.            mCurPlayMode = MusicPlayMode.MPM_SINGLE_LOOP_PLAY;
501.        }
502.
503.        mServiceManager.setPlayMode(mCurPlayMode);
504.        mUIManager.setPlayMode(mCurPlayMode, true);
505.    }
506.
507.    @Override
508.    public void OnServiceConnectComplete() {
509.        // TODO Auto-generated method stub
510.
511.        String state = Environment.getExternalStorageState().toString();
512.        if (state.equals(Environment.MEDIA_MOUNTED))
513.        {
514.            mIsSdExist = true;
515.        }else{
516.            Toast.makeText(this, "SD卡未安装，建议安装SD卡", Toast.LENGTH_SHORT).show();
517.            return ;
518.        }
519.
520.        int playState = mServiceManager.getPlayState();
521.        switch(playState)
522.        {
523.            case MusicPlayState.MPS_NOFILE:
524.                m_MusicFileList = getMusicFileList();
525.
526.                mServiceManager.refreshMusicList(m_MusicFileList);
527.                break;
528.            case MusicPlayState.MPS_INVALID:
529.            case MusicPlayState.MPS_PREPARE:
530.            case MusicPlayState.MPS_PLAYING:
531.            case MusicPlayState.MPS_PAUSE:
532.                long time1 = System.currentTimeMillis();
533.                m_MusicFileList = mServiceManager.getFileList();
534.                long time2 = System.currentTimeMillis();
535.                Log.i(TAG, "mServiceManager.getFileList()   cost = " + (time2 - time1));
536.                mServiceManager.sendPlayStateBrocast();
537.                break;
538.            default:
539.                break;
540.        }
541.
542.        if (m_MusicFileList.size() > 0)
543.        {
544.            mIsHaveData = true;
545.        }
546.
547.        mListViewAdapter.refreshAdapter(m_MusicFileList);
548.
549.        mUIManager.setPlayMode(mServiceManager.getPlayMode(), false);
550.
```

```
    }


    class  SDStateBrocast extends BroadcastReceiver
    {

        @Override
        public void onReceive(Context context, Intent intent) {
            // TODO Auto-generated method stub
             String action = intent.getAction();

            if (action.equals(Intent.ACTION_MEDIA_MOUNTED))
              {

                  mIsSdExist = true;
              }else if (action.equals(Intent.ACTION_MEDIA_UNMOUNTED))
              {

                  mIsSdExist = false;

              }else if (Intent.ACTION_MEDIA_SCANNER_FINISHED.equals(action))
              {

                  if (mIsSdExist)
                  {
                      m_MusicFileList = getMusicFileList();
                      mServiceManager.refreshMusicList(m_MusicFileList);
                      if (m_MusicFileList.size() > 0)
                      {
                        mIsHaveData = true;
                      }
                      mListViewAdapter.refreshAdapter(m_MusicFileList);
                  }

              }else if (Intent.ACTION_MEDIA_EJECT.equals(action))
              {
                  m_MusicFileList.clear();
                  mListViewAdapter.refreshAdapter(m_MusicFileList);
                  mIsHaveData = false;
                  mUIManager.emptyPlayInfo();
              }

        }

    }


    class  MusicPlayStateBrocast extends BroadcastReceiver
    {

        @Override
        public void onReceive(Context context, Intent intent) {
            // TODO Auto-generated method stub

              String action = intent.getAction();
              if (action.equals(BROCAST_NAME))
              {
                  TransPlayStateEvent(intent);
              }

        }


        public void TransPlayStateEvent(Intent intent)
        {
            MusicData data = new MusicData();
            int playState = intent.getIntExtra(MusicPlayState.PLAY_STATE_NAME, -1);
            Bundle bundle = intent.getBundleExtra(MusicData.KEY_MUSIC_DATA);
            if (bundle != null)
            {
                data = bundle.getParcelable(MusicData.KEY_MUSIC_DATA);
            }
            int playIndex = intent.getIntExtra(MusicPlayState.PLAY_MUSIC_INDEX, -1);

            switch (playState) {
            case MusicPlayState.MPS_INVALID:
                mMusicTimer.stopTimer();
                Toast.makeText(MusicPlayActivity.this, "当前音乐文件无
效", Toast.LENGTH_SHORT).show();
```

```
629.                mUIManager.setPlayInfo(0, data.mMusicTime, data.mMusicName);
630.                mUIManager.showPlay(true);
631.                break;
632.            case MusicPlayState.MPS_PREPARE:
633.                mMusicTimer.stopTimer();
634.                mCurMusicTotalTime = data.mMusicTime;
635.                if (mCurMusicTotalTime == 0)
636.                {
637.                    mCurMusicTotalTime = mServiceManager.getDuration();
638.                }
639.                mUIManager.setPlayInfo(0, data.mMusicTime, data.mMusicName);
640.                mUIManager.showPlay(true);
641.                break;
642.            case MusicPlayState.MPS_PLAYING:
643.                mMusicTimer.startTimer();
644.                if (mCurMusicTotalTime == 0)
645.                {
646.                    mCurMusicTotalTime = mServiceManager.getDuration();
647.                }
648.                mUIManager.setPlayInfo(mServiceManager.getCurPosition(), data.mMusicTime,
649.                mUIManager.showPlay(false);
650.                break;
651.            case MusicPlayState.MPS_PAUSE:
652.                mMusicTimer.stopTimer();
653.                if (mCurMusicTotalTime == 0)
654.                {
655.                    mCurMusicTotalTime = mServiceManager.getDuration();
656.                }
657.                mUIManager.setPlayInfo(mServiceManager.getCurPosition(), data.mMusicTime,
658.                mUIManager.showPlay(true);
659.                break;
660.            default:
661.                break;
662.            }
663.
664.            mUIManager.setSongNumInfo(playIndex, m_MusicFileList.size());
665.
666.            mListViewAdapter.setPlayState(playIndex, playState);
667.        }
668.
669.
670.    }
671.
672.    //显示专辑照片
673.    private void showArtwork(MusicData mp3Info) {
674.        Bitmap bm = MediaAlbum.getArtwork(this, mp3Info.MusicId, mp3Info.MusicAlbumId, tr
675.        //切换播放时候专辑图片出现透明效果
676.                Animation albumanim = AnimationUtils.loadAnimation(MusicPlayActivity.this
677.                //开始播放动画效果
678.                musicAlbum.startAnimation(albumanim);
679.                if(bm != null) {
680.                    musicAlbum.setImageBitmap(bm);  //显示专辑封面图片
681.                    musicAblumReflection.setImageBitmap(ImageUtil.createReflectionBitmapF
    显示倒影
682.                } else {
683.                    bm = MediaAlbum.getDefaultArtwork(this, false);
684.                    musicAlbum.setImageBitmap(bm);  //显示专辑封面图片
685.                    musicAblumReflection.setImageBitmap(ImageUtil.createReflectionBitmapF
    显示倒影
686.                }
687.    }
688.
689.
690.
691.
692.
693.    class UIManager implements OnItemClickListener{
694.
695.        private final static String TAG = "UIManager";
696.
697.        private SliderDrawerManager mSliderDrawerManager;
698.
699.        private PopMenuManager      mPopMenuManager;
700.
701.        public ListView             mListView;
702.
703.        private View                mMusicListView;
704.
705.
```

```
706.         private int mModeDrawableIDS[] = {  R.drawable.mode_single_loop,
707.                 R.drawable.mode_order,
708.                 R.drawable.mode_list_loop,
709.                 R.drawable.mode_random
710.         };
711.
712.         private String modeToasts[] = {"单曲循环",
713.                                        "顺序播放",
714.                                        "列表循环",
715.                                        "随机播放"
716.         };
717.
718.         public UIManager()
719.         {
720.             initView();
721.         }
722.
723.         private void initView()
724.         {
725.
726.             mListView = (ListView) findViewById(R.id.listView);
727.             mListView.setOnItemClickListener(this);
728.             mMusicListView = findViewById(R.id.ListLayout);
729.
730.             mSliderDrawerManager = new SliderDrawerManager();
731.             mPopMenuManager = new PopMenuManager();
732.             //Toast.makeText(MusicPlayActivity.this, listPosition, Toast.LENGTH_SHORT).sh
733.
734.         }
735.
736.         public void setPlayInfo(int curTime, int totalTime, String musicName)
737.         {
738.             curTime /= 1000;
739.             totalTime /= 1000;
740.             int curminute = curTime/60;
741.             int cursecond = curTime%60;
742.
743.             String curTimeString = String.format("%02d:%02d", curminute,cursecond);
744.
745.             int totalminute = totalTime/60;
746.             int totalsecond = totalTime%60;
747.             String totalTimeString = String.format("%02d:%02d", totalminute,totalsecond);
748.
749.             int rate = 0;
750.             if (totalTime != 0)
751.             {
752.                 rate = (int) ((float)curTime / totalTime * 100);
753.             }
754.
755.             mSliderDrawerManager.mPlayProgress.setProgress(rate);
756.
757.             mSliderDrawerManager.mcurtimeTextView.setText(curTimeString);
758.             mSliderDrawerManager.mtotaltimeTextView.setText(totalTimeString);
759.             if (musicName != null)
760.             {
761.                 mSliderDrawerManager.mPlaySongTextView.setText(musicName);
762.             }
763.
764.         }
765.
766.
767.         public void emptyPlayInfo()
768.         {
769.             mSliderDrawerManager.mPlayProgress.setProgress(0);
770.             mSliderDrawerManager.mcurtimeTextView.setText("00:00");
771.             mSliderDrawerManager.mtotaltimeTextView.setText("00:00");
772.             mSliderDrawerManager.mPlaySongTextView.setText(R.string.default_title_name);
773.
774.         }
775.
776.         public void setSongNumInfo(int curPlayIndex, int totalSongNum)
777.         {
778.             String str = String.valueOf(curPlayIndex + 1) + "/" + String.valueOf(totalSon
779.             listPosition=Integer.parseInt(String.valueOf(curPlayIndex ));
780.             m_MusicFileList = getMusicFileList();
781.             MusicData data = m_MusicFileList.get(listPosition);
782.             showArtwork(data);
783.             //Toast.makeText(MusicPlayActivity.this, a, Toast.LENGTH_SHORT).show();
784.             mSliderDrawerManager.mSongNumTextView.setText(str);
```

```
785.         }
786.
787.         public void showPlay(boolean flag)
788.         {
789.             if (flag)
790.             {
791.                 mSliderDrawerManager.mBtnPlay.setVisibility(View.VISIBLE);
792.                 mSliderDrawerManager.mBtnPause.setVisibility(View.GONE);
793.                 mSliderDrawerManager.mBtnHandlePlay.setVisibility(View.VISIBLE);
794.                 mSliderDrawerManager.mBtnHandlePause.setVisibility(View.INVISIBLE);
795.
796.             }else{
797.                 mSliderDrawerManager.mBtnPlay.setVisibility(View.GONE);
798.                 mSliderDrawerManager.mBtnPause.setVisibility(View.VISIBLE);
799.                 mSliderDrawerManager.mBtnHandlePlay.setVisibility(View.INVISIBLE);
800.                 mSliderDrawerManager.mBtnHandlePause.setVisibility(View.VISIBLE);
801.             }
802.
803.         }
804.
805.         public void ShowHandlePanel(boolean flag)
806.         {
807.             if (flag)
808.             {
809.                 mSliderDrawerManager.mHandlePane.setVisibility(View.VISIBLE);
810.             }else{
811.                 mSliderDrawerManager.mHandlePane.setVisibility(View.INVISIBLE);
812.             }
813.
814.         }
815.
816.         public void setPlayMode(int mode, Boolean bShowToast)
817.         {
818.             if (mode >= MusicPlayMode.MPM_SINGLE_LOOP_PLAY && mode <= MusicPlayMode.MPM_R
819.             {
820.                 mSliderDrawerManager.mBtnModeSet.setImageResource(mModeDrawableIDS[mode])
821.
822.                 if (bShowToast)
823.                 {
824.                     Toast.makeText(MusicPlayActivity.this, modeToasts[mode], Toast.LENGTH
825.                 }
826.
827.             }
828.
829.         }
830.
831.         public void Back()
832.         {
833.             if (mSliderDrawerManager.mSlidingDrawer.isOpened())
834.             {
835.                 mSliderDrawerManager.mSlidingDrawer.close();
836.             }else{
837.                 finish();
838.             }
839.         }
840.
841.
842.
843.         @Override
844.         public void onItemClick(AdapterView<?> arg0, View arg1, int pos,
845.                 long arg3) {
846.             // TODO Auto-generated method stub
847.
848.             play(pos);
849.
850.         }
851.
852.
853.
854.
855.
856.
857.         class SliderDrawerManager  implements OnClickListener, OnSeekBarChangeListener,
858.         OnDrawerOpenListener, OnDrawerCloseListener, IOnSliderHandleViewClickListener
859.         {
860.             public MySlidingDrawer mSlidingDrawer;
861.
862.
863.             public ImageButton   mBtnHandle;
```

```
    public ImageButton   mBtnHandlePlay;
    public ImageButton   mBtnHandlePause;
    public TextView       mSongNumTextView;
    public View           mHandlePane;
    public TextView mPlaySongTextView;



    public ImageButton mBtnModeSet;
    public ImageButton mBtnVolumnSet;
    public SeekBar mPlayProgress;
    public TextView mcurtimeTextView;
    public TextView mtotaltimeTextView;


    public ImageButton mBtnPlay;
    public ImageButton mBtnPause;
    public ImageButton mBtnStop;
    public ImageButton mBtnPlayNext;
    public ImageButton mBtnPlayPre;


    private boolean mPlayAuto = true;


    private GalleryFlow mGalleryFlow1;
    private GalleryFlow mGalleryFlow2;
    private AudioManager am;        //音频管理引用，提供对音频的控制
    RelativeLayout ll_player_voice; //音量控制面板布局
    int currentVolume;              //当前音量
    int maxVolume;                  //最大音量
    public ImageButton mBtnMenu;    //显示音量控制面板的按钮
    SeekBar GL_player_voice;        //控制音量大小
    // 音量面板显示和隐藏动画
    private Animation showVoicePanelAnimation;
    private Animation hiddenVoicePanelAnimation;

    public SliderDrawerManager()
    {
        initView();
    }

    private void initView()
    {
        GL_player_voice = (SeekBar) findViewById(R.id.GL_player_voice);
        mBtnPlay = (ImageButton) findViewById(R.id.buttonPlay);
        mBtnPause = (ImageButton) findViewById(R.id.buttonPause);
        mBtnStop = (ImageButton) findViewById(R.id.buttonStop);
        mBtnPlayPre = (ImageButton) findViewById(R.id.buttonPlayPre);
        mBtnPlayNext = (ImageButton) findViewById(R.id.buttonPlayNext);
        mBtnMenu = (ImageButton) findViewById(R.id.buttonMenu);
        mBtnModeSet = (ImageButton) findViewById(R.id.buttonMode);
        mBtnVolumnSet = (ImageButton) findViewById(R.id.buttonVolumn);
        mBtnHandle = (ImageButton) findViewById(R.id.handler_icon);
        mBtnHandlePlay= (ImageButton) findViewById(R.id.handler_play);
        mBtnHandlePause = (ImageButton) findViewById(R.id.handler_pause);
        musicAlbum = (ImageView) findViewById(R.id.iv_music_ablum);
        musicAblumReflection = (ImageView) findViewById(R.id.iv_music_ablum_refle
        mBtnPlay.setOnClickListener(this);
        mBtnPause.setOnClickListener(this);
        mBtnStop.setOnClickListener(this);
        mBtnPlayPre.setOnClickListener(this);
        mBtnPlayNext.setOnClickListener(this);
        mBtnStop.setOnClickListener(this);
        mBtnMenu.setOnClickListener(this);
        mBtnModeSet.setOnClickListener(this);
        mBtnVolumnSet.setOnClickListener(this);
        GL_player_voice.setOnSeekBarChangeListener(new SeekBarChangeListener());
        ll_player_voice = (RelativeLayout) findViewById(R.id.ll_player_voice);
        mPlaySongTextView = (TextView) findViewById(R.id.textPlaySong);
        mcurtimeTextView = (TextView) findViewById(R.id.textViewCurTime);
        mtotaltimeTextView = (TextView) findViewById(R.id.textViewTotalTime);
        mSongNumTextView = (TextView) findViewById(R.id.textSongNum);



        mPlayProgress = (SeekBar) findViewById(R.id.seekBar);
        mPlayProgress.setOnSeekBarChangeListener(new SeekBarChangeListener());

```

```
943.
944.                 mSlidingDrawer = (MySlidingDrawer) findViewById(R.id.slidingDrawer);
945.                 mSlidingDrawer.setOnDrawerOpenListener(this);
946.                 mSlidingDrawer.setOnDrawerCloseListener(this);
947.                 mSlidingDrawer.setHandleId(R.id.handler_icon);
948.                 mSlidingDrawer.setTouchableIds(new int[]
{R.id.handler_play, R.id.handler_pause});
949.                 mSlidingDrawer.setOnSliderHandleViewClickListener(this);
950.
951.
952.
953.                 mHandlePane = findViewById(R.id.handle_panel);
954.                 //音量调节面板显示和隐藏的动画
955.                 showVoicePanelAnimation = AnimationUtils.loadAnimation(MusicPlayActivity.
956.                 hiddenVoicePanelAnimation = AnimationUtils.loadAnimation(MusicPlayActivit
957.
958.                 //获得系统音频管理服务对象
959.                 am = (AudioManager)getSystemService(Context.AUDIO_SERVICE);
960.                 currentVolume = am.getStreamVolume(AudioManager.STREAM_MUSIC);
961.                 maxVolume = am.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
962.                 GL_player_voice.setProgress(currentVolume);
963.
964.                 m_MusicFileList = getMusicFileList();
965.                 MusicData data = m_MusicFileList.get(listPosition);
966.                 showArtwork(data);
967.                 am.setStreamVolume(AudioManager.STREAM_MUSIC, currentVolume, 0);
968.             }
969.
970.             /**
971.              * 显示专辑封面
972.              */
973.
974.
975.             @Override
976.             public void onClick(View v) {
977.                 // TODO Auto-generated method stub
978.                 switch(v.getId())
979.                 {
980.                 case R.id.buttonPlay:
981.                     rePlay();isplay=1;
982.                     break;
983.                 case R.id.buttonPause:
984.                     pause();isplay=0;
985.                     break;
986.                 case R.id.buttonStop:
987.                     stop();isplay=0;
988.                     break;
989.                 case R.id.buttonPlayPre:
990.                     playPre();isplay=1;
991.                     break;
992.                 case R.id.buttonPlayNext:
993.                     playNext();isplay=1;
994.                     break;
995.                 case R.id.buttonMenu:
996.                     voicePanelAnimation();
997.                     break;
998.                 case R.id.buttonMode:
999.                     modeChange();
1000.                    break;
1001.                case R.id.buttonVolumn:
1002.                    Toast.makeText(MusicPlayActivity.this, "别摁了,这个按钮摆设用
的", Toast.LENGTH_SHORT).show();                    ;
1003.                    break;
1004.                default:
1005.                    break;
1006.                }
1007.            }
1008.            //控制显示音量控制面板的动画
1009.            public void voicePanelAnimation() {
1010.                if(ll_player_voice.getVisibility() == View.GONE) {
1011.                    ll_player_voice.startAnimation(showVoicePanelAnimation);
1012.                    ll_player_voice.setVisibility(View.VISIBLE);
1013.                }
1014.                else{
1015.                    ll_player_voice.startAnimation(hiddenVoicePanelAnimation);
1016.                    ll_player_voice.setVisibility(View.GONE);
1017.                }
1018.            }
1019.            private class SeekBarChangeListener implements OnSeekBarChangeListener {
```

```
1020.
1021.                 @Override
1022.                 public void onProgressChanged(SeekBar seekBar, int progress,
1023.                         boolean fromUser) {
1024.                     switch(seekBar.getId()) {
1025.                     case R.id.seekBar:
1026.                         if (fromUser) {
1027.                             mServiceManager.seekTo(progress); // 用户控制进度的改变
1028.                         }
1029.                         break;
1030.                     case R.id.GL_player_voice:
1031.                         // 设置音量
1032.                         am.setStreamVolume(AudioManager.STREAM_MUSIC, progress, 0);
1033.                         System.out.println("am--->" + progress);
1034.                         break;
1035.                     }
1036.                 }
1037.
1038.                 @Override
1039.                 public void onStartTrackingTouch(SeekBar seekBar) {
1040.
1041.                 }
1042.
1043.                 @Override
1044.                 public void onStopTrackingTouch(SeekBar seekBar) {
1045.
1046.                 }
1047.
1048.             }
1049.
1050.
1051.             @Override
1052.             public void onDrawerOpened() {
1053.                 // TODO Auto-generated method stub
1054.                 mMusicListView.setVisibility(View.INVISIBLE);
1055.                 mBtnHandle.setImageResource(R.drawable.handle_down);
1056.                 ShowHandlePanel(false);
1057.             }
1058.
1059.             @Override
1060.             public void onDrawerClosed() {
1061.                 // TODO Auto-generated method stub
1062.                 mMusicListView.setVisibility(View.VISIBLE);
1063.                 mBtnHandle.setImageResource(R.drawable.handle_up);
1064.                 ShowHandlePanel(true);
1065.             }
1066.
1067.             @Override
1068.             public void onViewClick(View view) {
1069.                 // TODO Auto-generated method stub
1070.                 switch(view.getId())
1071.                 {
1072.                 case R.id.handler_play:
1073.                     rePlay();isplay=1;
1074.                     break;
1075.                 case R.id.handler_pause:
1076.                     pause();isplay=0;
1077.                     break;
1078.                 default:
1079.                     break;
1080.                 }
1081.             }
1082.
1083.             @Override
1084.             public void onProgressChanged(SeekBar seekBar, int progress,
1085.                     boolean fromUser) {
1086.                 // TODO Auto-generated method stub
1087.
1088.             }
1089.
1090.             @Override
1091.             public void onStartTrackingTouch(SeekBar seekBar) {
1092.                 // TODO Auto-generated method stub
1093.
1094.             }
1095.
1096.             @Override
1097.             public void onStopTrackingTouch(SeekBar seekBar) {
1098.                 // TODO Auto-generated method stub
```

```
1099.
1100.            }
1101.
1102.
1103.
1104.
1105.        }
1106.
1107.
1108.
1109.
1110.
1111.
1112.
1113.
1114.        class PopMenuManager implements android.widget.PopupWindow.OnDismissListener{
1115.
1116.            MenuItemData            mMenuItemData;
1117.            private GridView        mMenuGrid;          // 弹出菜单GRIDVIEW
1118.            private View            mMenuView;          // 弹出菜单视图
1119.            private GridViewAdapter mGridViewAdapter;   // 弹出菜单适配器
1120.            private PopupWindow     mPopupWindow;       // 弹出菜单WINDOW
1121.
1122.            private View            mPopBackgroundView;
1123.
1124.            public PopMenuManager()
1125.            {
1126.                initView();
1127.            }
1128.
1129.            private void initView()
1130.            {
1131.                mPopBackgroundView = findViewById(R.id.VirtualLayout);
1132.
1133.                String []menu_name_array1 = getResources().getStringArray(R.array.menu_it
1134.                LevelListDrawable levelListDrawable1 = (LevelListDrawable) getResources()
1135.                mMenuItemData = new MenuItemData(levelListDrawable1, menu_name_array1, me
1136.
1137.                LayoutInflater inflater = getLayoutInflater();
1138.                mMenuView = inflater.inflate(R.layout.menu, null);
1139.                mMenuGrid = (GridView)mMenuView.findViewById(R.id.menuGridView);
1140.
1141.
1142.                mGridViewAdapter = new GridViewAdapter(MusicPlayActivity.this, mMenuItemD
1143.                mMenuGrid.setAdapter(mGridViewAdapter);
1144.
1145.
1146.                mMenuGrid.setOnItemClickListener(new OnItemClickListener() {
1147.
1148.                    @Override
1149.                    public void onItemClick(AdapterView<?> parent, View view,
1150.                            int position, long id) {
1151.                        // TODO Auto-generated method stub
1152.                    //  onMenuItemClick(position);
1153.                    }
1154.
1155.
1156.                });
1157.
1158.
1159.
1160.                mPopupWindow = new PopupWindow(mMenuView, LayoutParams.FILL_PARENT,  Layo
1161.                mPopupWindow.setFocusable(true);        // 如果没有获得焦点menu菜单中的控件
     事件无法响应
1162.
1163.                //  以下两行加上去后就可以使用BACK键关闭POPWINDOW
1164.                ColorDrawable dw = new ColorDrawable(0x00);
1165.                mPopupWindow.setBackgroundDrawable(dw);
1166.
1167.                mPopupWindow.setAnimationStyle(android.R.style.Animation_Toast);
1168.                mPopupWindow.setOnDismissListener(this);
1169.            }
1170.
1171.
1172.
1173.
1174.            @Override
1175.            public void onDismiss() {
1176.                // TODO Auto-generated method stub
```

```
1177.                 mPopBackgroundView.setVisibility(View.INVISIBLE);
1178.             }
1179.         }
1180.     }
1181.     //重力感应代码
1182.     private static final int SHAKE_THRESHOLD = 4000;//这个控制精度，越小表示反应越灵敏
1183.     private long lastUpdate=0;
1184.     private double last_x=0;
1185.     private double last_y= 4.50;
1186.     private double last_z=9.50;
1187.     //这个控制精度，越小表示反应越灵敏
1188.     public void onAccuracyChanged(Sensor sensor, int accuracy) {
1189.         // TODO Auto-generated method stub
1190.         //处理精准度改变
1191.     }
1192.
1193.
1194.     public void onSensorChanged(SensorEvent event) {
1195.         // TODO Auto-generated method stub
1196.         if(event.sensor.getType()==Sensor.TYPE_ACCELEROMETER){
1197.             long curTime = System.currentTimeMillis();
1198.
1199.             // 每100毫秒检测一次
1200.             if ((curTime - lastUpdate) > 100) {
1201.             long diffTime = (curTime - lastUpdate);
1202.             lastUpdate = curTime;
1203.             double x=event.values[SensorManager.DATA_X];
1204.             double y=event.values[SensorManager.DATA_Y];
1205.             double z=event.values[SensorManager.DATA_Z];
1206.             float speed = (float) (Math.abs(x+y+z - last_x - last_y - last_z) / diffTime
1207.             if (speed > SHAKE_THRESHOLD) {
1208.
1209.                         //检测到摇晃后执行的代码
1210.                 if(isplay==1){
1211.                     if(Shake_pause==20&&Shake_next!=20)
1212.                     {
1213.                         vibrator.vibrate(1000);//手机震动一秒
1214.
1215.                      pause();
1216.                     }
1217.                     if(Shake_next==20&&Shake_pause!=20)
1218.                      playNext();
1219.                      isplay=0;
1220.
1221.                  }else {
1222.                      if(Shake_pause==20)
1223.                      rePlay();
1224.                      isplay=1;
1225.                  }
1226.                 if(Shake_pause==20&&Shake_next==20)
1227.                 {
1228.                     Toast.makeText(MusicPlayActivity.this, "不好意思，甩动切歌和甩动暂停只
能开启一个！", Toast.LENGTH_SHORT).show();
1229.                 }
1230.             }
1231.             last_x = x;
1232.             last_y = y;
1233.             last_z = z;
1234.             }
1235.         }
1236.     }
1237.     //电话监听
1238.     private class MobliePhoneStateListener extends PhoneStateListener {
1239.
1240.         @Override
1241.         public void onCallStateChanged(int state, String incomingNumber) {
1242.             switch (state) {
1243.             case TelephonyManager.CALL_STATE_IDLE: /* 无任何状态时 */
1244.                 rePlay();
1245.
1246.                 break;
1247.             case TelephonyManager.CALL_STATE_OFFHOOK: /* 接起电话时 */
1248.
1249.             case TelephonyManager.CALL_STATE_RINGING: /* 电话进来时 */
1250.                   pause();
1251.                 break;
1252.             default:
1253.                 break;
1254.
```

```
            }

        }

    }

    /**
     * 按返回键弹出对话框确定退出
     */
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK
                && event.getAction() == KeyEvent.ACTION_DOWN) {
            new com.genius.widget.CustomDialog.Builder(MusicPlayActivity.this)
                    .setTitle(R.string.info)
                    .setMessage(R.string.dialog_messenge)
                    .setPositiveButton(R.string.confrim,
                            new DialogInterface.OnClickListener() {

                                @Override
                                public void onClick(DialogInterface dialog,
                                        int which) {
                                    exit();

                                }
                            }).setNeutralButton(R.string.cancel, null).show();
            return false;
        }
        return false;
    }


}</span>
```

更多 0

上一篇：Android 广播大全 Intent Action 事件

下一篇：GL音乐播放器<三>--界面设计之专辑照片的实现

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

专区推荐内容

使用 Thread 类创建与启动...
下载QNX白皮书助您完成卓越的嵌...
在HTML5中，如何从C#执行网...
如何从Javascript传送字...
Hadoop4Win+Eclip...
开启 Android 应用开发新...

<< >>

更多招聘职位

【啪啪】前端工程师
【广州聚多网络科技有限公司】iOS 开
【深圳开饭喇信息科技有限公司（Openl
发）
【软通动力技术服务有限公司深圳分公
【上海财华保网络科技有限公司】移动

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC
WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK
IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails
QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6
Rackspace Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby
Hibernate ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala
Django Bootstrap