package cn.c;

import java.io.File;

import android.app.Activity;

import android.media.MediaPlayer;

import android.media.MediaPlayer.OnCompletionListener;

import android.media.MediaPlayer.OnErrorListener;

import android.media.MediaPlayer.OnInfoListener;

import android.media.MediaPlayer.OnPreparedListener;

import android.media.MediaPlayer.OnSeekCompleteListener;

import android.media.MediaPlayer.OnVideoSizeChangedListener;

import android.os.Bundle;

import android.os.Environment;

import android.view.Display;

import android.view.MotionEvent;

import android.view.SurfaceHolder;

import android.view.SurfaceView;

import android.widget.LinearLayout;

import android.widget.MediaController;

import android.widget.MediaController.MediaPlayerControl;

/\*\*

\* 参考资料:http://www.embedu.org/Column/Column503.htm

\* 利用MidiaPlayer播放视频大致的流程:

\* 创建播放器对象-->设置需要播放的视频的来源-->

\* 对视频进行解码等准备工作-->开始播放-->(暂停/重启)停止播放

\* 对应这些流程,在MediaPlayer中都定义了相关方法;

\* 而且,在调用这些方法后,MediaPlayer可以进入不同的状态.

\* 这些方法和状态都很好理解,读者可以参照上图来了解.

\* 另外,因为在设置播放的视频的视频数据源之后,它需要对其进行解码(调用prepare()方法)

\* 这是一个比较耗时的操作,为了避免应用程序阻止,

\* 可以使用它的prepareAsync()方法配合OnPreparedListener监听器来让其异步操作.

\* 注意,只有MediaPlayer的准备工作做完后,才能开始播放

\* MediaPlayer本身不能显示视频内容,它只能控制视频的播放

\* 如果要使用MediaPlayer来播放视频，还需要配合SurfaceView来显示视频

\* 且将MediaPlayer中解码的视频数据在SurfaceView中播放,需要调用MediaPlayer中的setDisplay()

\* 它接受一个SurfaceHolder对象,可将我们用于

\* 播放视频的SurfaceView对应的SurfaceHolder对象作为参数

\* 这样,MediaPlayer和SurfaceView就建立起了关联.

\*

\* MediaPlayer播放视频步骤

\* 1 设置MediaPlayer将要播放的视频

\* mMediaPlayer.setDataSource(path);

\* 2 设置MediaPlayer的视频输出界面

\* mMediaPlayer.setDisplay(mSurfaceHolder);

\* 3 MediaPlayer解码

\* mMediaPlayer.prepareAsync()且结合

\* public void onPrepared(MediaPlayer mediaPlayer)

\* 4 MediaPlayer开始播放

\* mediaPlayer.start();

\*

\*

\*/

public class MainActivity extends Activity implements

OnCompletionListener,OnErrorListener,OnInfoListener,OnPreparedListener,

OnSeekCompleteListener,OnVideoSizeChangedListener,SurfaceHolder.Callback,MediaPlayerControl{

private SurfaceView mSurfaceView;

private SurfaceHolder mSurfaceHolder;

private MediaController mMediaController;

private Display mDisplay;

private MediaPlayer mMediaPlayer;

int videoWidth=0;

int videoHeight=0;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

mDisplay=getWindowManager().getDefaultDisplay();

mSurfaceView=(SurfaceView) findViewById(R.id.surfaceView);

mSurfaceHolder=mSurfaceView.getHolder();

//SurfaceHolder是用于管理SurfaceView对象

//那它是怎么做到管理的呢?即用mSurfaceHolder

//添加回调即addCallback.

//该类实现了SurfaceHolder.Callback接口,所以addCallback(this)

//在SurfaceHolder.Callback接口中一共有三个方法来实现对于

//SurfaceView对象的管理,即:

//surfaceCreated()

//surfaceChanged()

//surfaceDestroyed()

//概况:SurfaceHolder主要作用是监控底层的情况

mSurfaceHolder.addCallback(this);

mSurfaceHolder.setType(SurfaceHolder.SURFACE\_TYPE\_PUSH\_BUFFERS);

mMediaPlayer=new MediaPlayer();

mMediaPlayer.setOnCompletionListener(this);

mMediaPlayer.setOnErrorListener(this);

mMediaPlayer.setOnInfoListener(this);

mMediaPlayer.setOnPreparedListener(this);

mMediaPlayer.setOnSeekCompleteListener(this);

mMediaPlayer.setOnVideoSizeChangedListener(this);

String path=Environment.getExternalStorageDirectory().getPath()

+File.separator+"Test\_Movie.m4v";

try {

//设置MediaPlayer将要播放的视频

mMediaPlayer.setDataSource(path);

} catch (Exception e) {

e.printStackTrace();

finish();

}

mMediaController=new MediaController(MainActivity.this);

}

@Override

public boolean onTouchEvent(MotionEvent event) {

if (mMediaController.isShowing()) {

mMediaController.hide();

} else {

mMediaController.show();

}

return false;

}

//来自于SurfaceHolder.Callback接口

//创建SurfaceView完成时,会调用该方法

public void surfaceCreated(SurfaceHolder holder) {

//设置MediaPlayer的视频输出界面

//只显示音频而不显示视频时可不调用这个方法

mMediaPlayer.setDisplay(mSurfaceHolder);

try {

//准备播放,调用mMediaPlayer.prepareAsync();

//方法后会执行覆写后的

//public void onPrepared(MediaPlayer mp)

//注意:

//也可调用prepare()解码,但为一个同步的操作

mMediaPlayer.prepareAsync();

} catch (Exception e) {

e.printStackTrace();

finish();

}

}

//来自于SurfaceHolder.Callback接口

//当SurfaceView的宽度,高度或其他参数发生变化时,会调用该方法

public void surfaceChanged(SurfaceHolder holder,

int format, int width,int height) {

}

//来自于SurfaceHolder.Callback接口

//当SurfaceView摧毁时,会调用该方法

public void surfaceDestroyed(SurfaceHolder holder) {

}

//来自于MediaPlayer.OnVideoSizeChangedListener接口

//当视频的宽度或高度发生变化时调用该方法

public void onVideoSizeChanged(MediaPlayer mp, int width, int height) {

}

//来自于MediaPlayer.OnSeekCompleteListener接口

public void onSeekComplete(MediaPlayer mp) {

}

//来自于MediaPlayer.OnPreparedListener接口

//一旦调用该方法,MediaPlayer就进入了"准备就绪"

//状态,准备开始播放.

//此处可用于动态设置SurfaceView的宽和高!!!

public void onPrepared(MediaPlayer mediaPlayer) {

videoWidth=mediaPlayer.getVideoWidth();

videoHeight=mediaPlayer.getVideoHeight();

if (videoWidth > mDisplay.getWidth()|| videoHeight > mDisplay.getHeight()) {

float heightRatio = (float) videoHeight/ (float) mDisplay.getHeight();

float widthRatio = (float) videoWidth/ (float) mDisplay.getWidth();

if (heightRatio > 1 || widthRatio > 1) {

if (heightRatio > widthRatio) {

videoHeight = (int) Math.ceil((float) videoHeight/ (float) heightRatio);

videoWidth = (int) Math.ceil((float) videoWidth/ (float) heightRatio);

} else {

videoHeight = (int) Math.ceil((float) videoHeight/ (float) widthRatio);

videoWidth = (int) Math.ceil((float) videoWidth/ (float) widthRatio);

}

}

}

//设置SurfaceView的宽和高

mSurfaceView.setLayoutParams

(new LinearLayout.LayoutParams(videoWidth,videoHeight));

//只有在MediaPlayer准备好之后才可以对

//MediaController进行设置和使用

mMediaController.setMediaPlayer(this);

mMediaController.setAnchorView(findViewById(R.id.mainView));

mMediaController.setEnabled(true);

mMediaController.show();

//MediaPlayer开始播放

mediaPlayer.start();

}

//来自于MediaPlayer.OnInfoListener接口

//当出现关于播放媒体的特定信息或者需要发出警告的时候

//将调用该方法

//比如开始缓冲、缓冲结束、下载速度变化(该行待验证)

//小结:这些Info都是以MediaPlayer.MEDIA\_INFO\_开头的

public boolean onInfo(MediaPlayer mp, int what, int extra) {

if (what==MediaPlayer.MEDIA\_INFO\_BAD\_INTERLEAVING) {

//音频和视频数据不正确地交错时将出现该提示信息.在一个

//正确交错的媒体文件中,音频和视频样本将依序排列,从而

//使得播放可以有效和平稳地进行

}

if (what==MediaPlayer.MEDIA\_INFO\_NOT\_SEEKABLE) {

//当媒体不能正确定位时将出现该提示信息.

//此时意味着它可能是一个在线流

}

if (what==MediaPlayer.MEDIA\_INFO\_VIDEO\_TRACK\_LAGGING) {

//当设备无法播放视频时将出现该提示信息

//比如视频太复杂或者码率过高

}

if (what==MediaPlayer.MEDIA\_INFO\_METADATA\_UPDATE) {

//当新的元数据可用时将出现该提示信息

}

if (what==MediaPlayer.MEDIA\_INFO\_UNKNOWN) {

//其余不可知提示信息

}

return false;

}

//来自于MediaPlayer.OnErrorListener接口

//MediaPlayer发生错误时会调用该方法

//只有如下这三种错误.

//小结:这些错误都是以MediaPlayer.MEDIA\_ERROR.开头的

public boolean onError(MediaPlayer mp, int what, int extra) {

if (what==MediaPlayer.MEDIA\_ERROR\_NOT\_VALID\_FOR\_PROGRESSIVE\_PLAYBACK) {

System.out.println("第一种错误");

}

if (what==MediaPlayer.MEDIA\_ERROR\_SERVER\_DIED) {

System.out.println("第二种错误");

}

if (what==MediaPlayer.MEDIA\_ERROR\_UNKNOWN) {

System.out.println("第三种错误");

}

return false;

}

//来自于MediaPlayer.OnCompletionListener接口

//当MediaPlayer播放完文件时,会调用该方法.

//此时可以进行一些其他的操作比如:播放下一个视频

public void onCompletion(MediaPlayer mp) {

finish();

}

//以下方法均来自MediaPlayerControl接口

public void start() {

mMediaPlayer.start();

}

public void pause() {

if (mMediaPlayer.isPlaying()) {

mMediaPlayer.pause();

}

}

public int getDuration() {

return mMediaPlayer.getDuration();

}

public int getCurrentPosition() {

return mMediaPlayer.getCurrentPosition();

}

public void seekTo(int pos) {

mMediaPlayer.seekTo(pos);

}

public boolean isPlaying() {

return mMediaPlayer.isPlaying();

}

public int getBufferPercentage() {

// TODO Auto-generated method stub

return 0;

}

public boolean canPause() {

return true;

}

public boolean canSeekBackward() {

return true;

}

public boolean canSeekForward() {

return true;

}

}