**基于Android系统的手机音视频播放器开发**

1. **绪论**

**（一）音视频播放器开发背景**

Android做为一个开源的移动设备操作系统，极大的推动了智能设备的发展，其开源性允许各平台厂商进行任意修改，导致众多的平台和芯片厂商提供的硬件解码器API不能很好的相互兼容，引起Android平台的多媒体播放器解码效率低下，功耗过高，系统资源消耗大。本人结合所学的通信工程中多媒体相关课程和所从事的工作，开发一款可以尽最大可能利用平台硬件解码器的全格式支持的音视频播放器，并且提供高度优化的软解码库。

**（二）音视频播放器开发目标**

本毕业设计是开发一个基于Android系统的音视频播放器，主要解决播放器跨设备所带来的硬件解码器互不兼容和软件解码器效率低下的问题。虽然各大芯片厂的解码器遵循OpenMAX标准，但是由于Android的开放性，芯片厂商或多或少都会对OMXCodec API做一些修改，导致同一款播放器在不同的平台性能差别很大。另外硬件解码器提供的Codec比较少，主要是H.264和VPX，其他的Codec，比如H.263等会采用软解码，由于优化力度不够，导致视频在播放中功耗过高，效率低下。基于此，该播放器还会集成一个高度优化的软解码库，降低功耗，提高播放器性能和用户体验。

**（三）音视频播放器开发现状**

在音视频播放方面，Android默认支持的文件格式很有限，基本只能支持MP4，WebM等欧美主流的格式，这对中国市场是远远不够的。本毕业设计中的播放器会集成ffmpeg，达到全格式支持。

在Codec方面，Android默认支持AAC，AVC等基本的音视频Codec，本毕业设计结合目前国内多媒体市场现状，集成所需的开源解码器。

**（四）音视频播放器应用特点**

结合开发目标，该应用的主要特点：

a. 尽最大可能使用硬件解码器

b. 软件解码器高度优化

c. 自动搜索媒体资源

d. 简单易用的操作界面

**二、需求分析**

**（一） 功能需求**

主要功能：

1. 多媒体资源搜索（可以自动或手动搜索）

2. 音视频播放

3. 音视频播放控制（播放，暂停，快进，快退）

**（二）开发环境需求**

1．开发主机：Ubuntu 14.04.4

Ubuntu（乌班图）是一个以桌面应用为主的Linux操作系统，其名称来自非洲南部祖鲁语或豪萨语的“ubuntu”一词，意思是“人性”、“我的存在是因为大家的存在”，是非洲传统的一种价值观，类似华人社会的“仁爱”思想。Ubuntu基于Debian发行版和GNOME桌面环境，而从11.04版起，Ubuntu发行版放弃了Gnome桌面环境，改为Unity，与Debian的不同在于它每6个月会发布一个新版本。Ubuntu的目标在于为一般用户提供一个最新的、同时又相当稳定的主要由自由软件构建而成的操作系统。Ubuntu具有庞大的社区力量，用户可以方便地从社区获得帮助。

2. Android应用开发工具：Android Studio

Android Studio 是一个Android开发环境，基于IntelliJ IDEA. 类似 Eclipse ADT，Android Studio 提供了集成的 Android 开发工具用于开发和调试。

**三、 UI设计**

**（一） UI**

软件设计可分为两个部分：编码设计与UI设计。随着软件应用的广泛普及，人们对于其要求也逐步提高，客户不止看中其功能实用性，更是需要UI来提升用户体验性，在操作享受软件带来的方便之余也不乏其美观性带来的愉悦感。一款软件的成功不仅仅在于其功能的强大，界面设计也占了其成功因素的半壁江山。

**（二） 软件界面设计作用：**

* 1. 让软件更加生动，有自己的个性。正如微软和苹果一样，各自都有自己鲜明的风格。
* 2. 使用者操作便捷，易上手。设计良好的界面能够引导用户自己完成相应的操作，起到向导的作用。

3. 延长软件使用寿命，让软件深入人心，提升软件市场竞争力。

**（三）本毕业设计的UI**

本毕业设计的重点是播放功能，所以UI做的比较简略，如图所示：

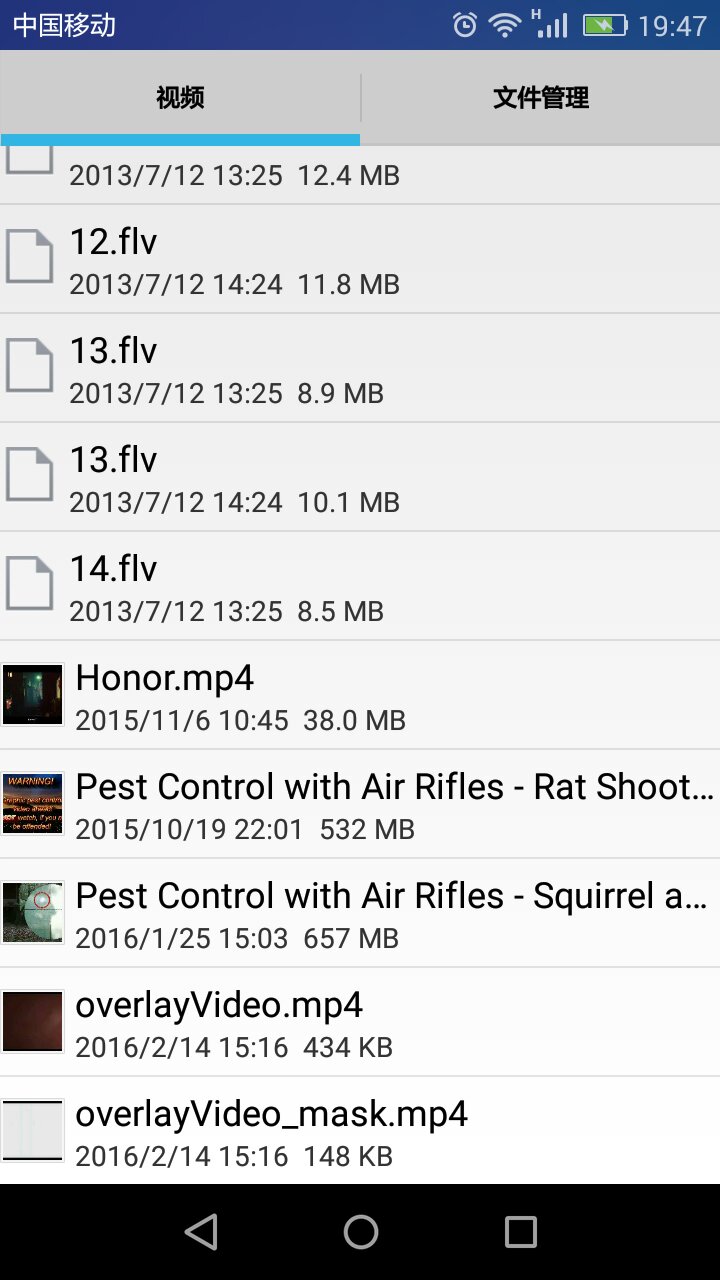


图3－1 音视频文件显示UI

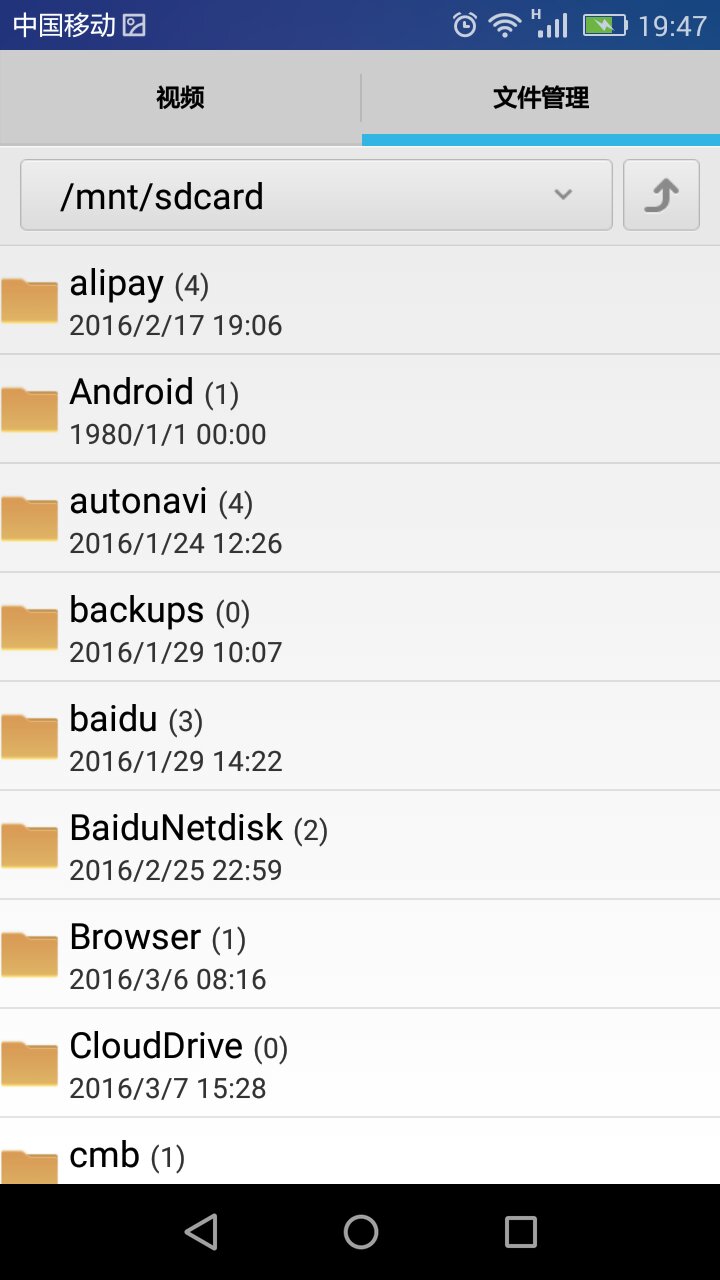


图 3－2 文件管理器UI



图 3－3 播放器UI

**四、功能逻辑设计**

**（一）类功能设计**

本次毕业设计中用到的类有Activity类、VideoViewDemo类、VideoView类和MediaPlayer类。

1. Activity简介

Activity是Android组件中最基本也是最为常见用的四大组件（Activity，Service服务,Content Provider内容提供者，BroadcastReceiver广播接收器）之一。

Activity是一个应用程序组件，提供一个屏幕，用户可以用来交互为了完成某项任务。

Activity中所有操作都与用户密切相关，是一个负责与用户交互的组件，可以通过setContentView(View)来显示指定控件。

在一个android应用中，一个Activity通常就是一个单独的屏幕，它上面可以显示一些控件也可以监听并处理用户的事件做出响应。Activity之间通过Intent进行通信。

（1）Active/Runing

一个新 Activity 启动入栈后，它显示在屏幕最前端，处理是处于栈的最顶端（Activity栈顶），此时它处于可见并可和用户交互的激活状态,叫做活动状态或者运行状态（active or running）。

（2）Paused

当 Activity失去焦点， 被一个新的非全屏的Activity 或者一个透明的Activity 被放置在栈顶，此时的状态叫做暂停状态（Paused）。此时它依然与窗口管理器保持连接，Activity依然保持活力（保持所有的状态，成员信息，和窗口管理器保持连接），但是在系统内存极端低下的时候将被强行终止掉。所以它仍然可见，但已经失去了焦点故不可与用户进行交互。

（3）Stoped

如果一个Activity被另外的Activity完全覆盖掉，叫做停止状态（Stopped）。它依然保持所有状态和成员信息，但是它不再可见，所以它的窗口被隐藏，当系统内存需要被用在其他地方的时候，Stopped的Activity将被强行终止掉。

（4）Killed

如果一个Activity是Paused或者Stopped状态，系统可以将该Activity从内存中删除，Android系统采用两种方式进行删除，要么要求该Activity结束，要么直接终止它的进程。当该Activity再次显示给用户时，它必须重新开始和重置前面的状态。

（5）状态转换

当一个 Activity 实例被创建、销毁或者启动另外一个 Activity 时，它在这四种状态之间进行转换，这种转换的发生依赖于用户程序的动作。下图说明了 Activity 在不同状态间转换的时机和条件：



2. VideoViewDemo

该类主要继承Activity类，实现其onCreate函数，初始化layout。另外该类还实现了函数playFunction主要完成播放功能。

3. VideoView

在Android中播放视频的方式有两种：

（1）第一种方式是使用MediaPlayer 结合SurfaceView 来播放，通过MediaPlayer来控制视频的播放、暂停、进度等；通过SurfaceView 来显示视频内容；优点：灵活性高，可以进行自定义；缺点：难度比较大；

（2）第二种方式是使用VideoView 来播放，这个类其实也是继承了SurfaceView 类，并且实现了MediaController. MediaPlayerController 这个用于控制媒体播放的接口，另外在VideoView上还有一个用于对媒体播放进行控制的面板，包括快进、快退、播放、暂停按钮以及一个进度条； 优点：比较简单，可以直接进行使用；缺点：灵活性不高；

（3）本毕业设计采用的是VideoView。其使用方法如下：

setVideoViewPath(String path);                设置播放视频的路径；

setVideoURL(Uri uri);                            设置视频的URI

start();                                                开始播放；

seekTo(int msec);                                   定位；

pause();                                              暂停播放；

stop();                                                停止播放；

setMediaController(MediaController);         设置MediaController 对象；

（4）媒体控制面版常用方法：MediaController:

   hide();                                 隐藏MediaController;

   show();                                 显示MediaController

   show(int timeout);         设置MediaController显示的时间，以毫秒计算，如果设置为0则一直到调用hide()时隐藏；

4. MediaPlayer

（1）如何获得MediaPlayer实例：

1. 可以使用直接new的方式：

MediaPlayer mp = new MediaPlayer();

2. 也可以使用create的方式，如：

MediaPlayer mp = MediaPlayer.create(this, R.raw.test);

（2）如何设置要播放的文件：

MediaPlayer要播放的文件主要包括3个来源：

1. 用户在应用中事先自带的resource资源

例如：MediaPlayer.create(this, R.raw.test);

1. 存储在SD卡或其他文件路径下的媒体文件

例如：mp.setDataSource("file:///sdcard/test.mp3");

1. 网络上的媒体文件

例如：mp.setDataSource("http://www.citynorth.cn/confucius.mp3");

（3）对播放器的主要控制方法：

Android通过控制播放器的状态的方式来控制媒体文件的播放，其中：

prepare()和prepareAsync() 提供了同步和异步两种方式设置播放器进入prepare状态，需要注意的是，如果MediaPlayer实例是由create方法创建的，那么第一次启动播放前不需要再调用prepare（）了，因为create方法里已经调用过了。

start()是真正启动文件播放的方法，

pause()和stop()比较简单，起到暂停和停止播放的作用，

seekTo()是定位方法，可以让播放器从指定的位置开始播放，需要注意的是该方法是个异步方法，也就是说该方法返回时并不意味着定位完成，尤其是播放的网络文件，真正定位完成时会触发OnSeekComplete.onSeekComplete()，如果需要是可以调用setOnSeekCompleteListener(OnSeekCompleteListener)设置监听器来处理的。

release()可以释放播放器占用的资源，一旦确定不再使用播放器时应当尽早调用它释放资源。

reset()可以使播放器从Error状态中恢复过来，重新会到Idle状态。

（4）设置播放器的监听器：

MediaPlayer提供了一些设置不同监听器的方法来更好地对播放器的工作状态进行监听，以期及时处理各种情况，如：

setOnCompletionListener(MediaPlayer.OnCompletionListener listener)、

setOnErrorListener(MediaPlayer.OnErrorListener listener)等,设置播放器时需要考虑到播放器可能出现的情况设置好监听和处理逻辑，以保持播放器的健壮性。

**（2）状态转换图**

对播放音频/视频文件和流的控制是通过一个状态机来管理的。下图显示一个MediaPlayer对象被支持的播放控制操作驱动的生命周期和状态。椭圆代表MediaPlayer对象可能驻留的状态。弧线表示驱动MediaPlayer在各个状态之间迁移的播放控制操作。**这里有两种类型的弧线。由一个箭头开始的弧代表同步的方法调用，而以双箭头开头的代表的弧线代表异步方法调用**。通过这张图，我们可以知道一个**MediaPlayer**对象有以下的状态：

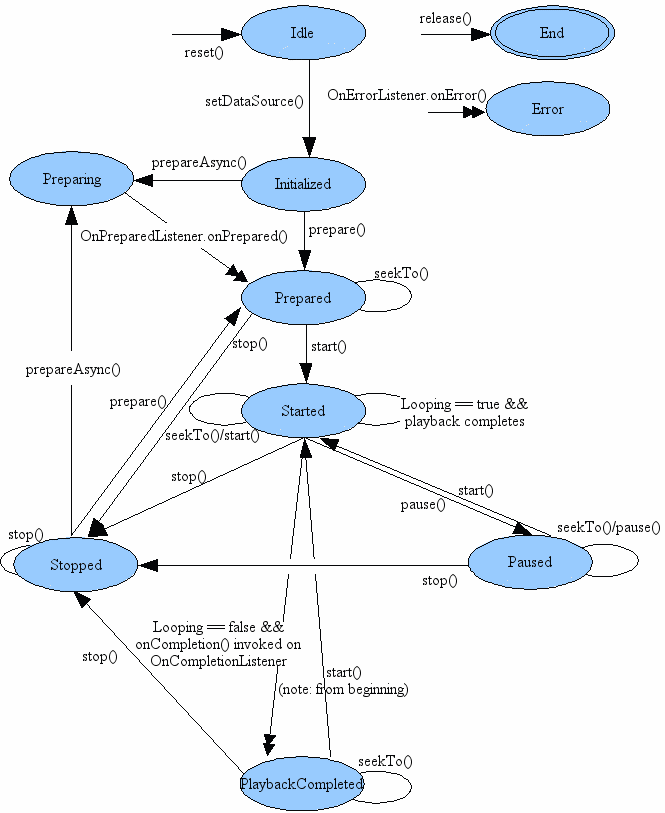


图4-1 MediaPlayer状态转换图

1. **功能实现**

（1）VideoViewDemo类的实现

VideoViewDemo类继承Activity类，主要包含以下两个主要函数，分别是onCreate和playFunction，实现如下：

1. onCreate函数

**public** **void** onCreate (Bundle icicle) {

**super**.onCreate(icicle);

Vitamio.*isInitialized*(**this**);

setContentView(R.layout.***videoview***);

playFunction ();

}

1. playFunction函数

**void** playfunction() {

String path = getIntent().getDataString();

VideoView mVideoView;

mVideoView = (VideoView) findViewById(R.id.***surface\_view***);

**if** (path == "") {

// Tell the user to provide a media file URL/path.

Toast.*makeText*(

VideoViewDemo.**this**,

"Please edit VideoViewDemo Activity, and set path" + " variable to your media file URL/path",

Toast.***LENGTH\_LONG***).show();

**return**;

} **else** {

/\*

\* Alternatively,for streaming media you can use

\* mVideoView.setVideoURI(Uri.parse(URLstring));

\*/

mVideoView.setVideoPath(path);

mVideoView.setMediaController(**new** MediaController(**this**));

mVideoView.requestFocus();

mVideoView

.setOnPreparedListener(**new** MediaPlayer.OnPreparedListener() {

@Override

**public** **void** onPrepared(MediaPlayer mediaPlayer) {

// optional need Vitamio 4.0

mediaPlayer.setPlaybackSpeed(1.0f);

}

});

}

}

（2）VideoView的实现

核心函数为openVideo，代码如下：

**private** **void** openVideo() {

**if** (mUri == **null** || mSurfaceHolder == **null** )

**return**;

Intent i = **new** Intent("com.android.music.musicservicecommand");

i.putExtra("command", "pause");

mContext.sendBroadcast(i);

release(**false**);

**try** {

mDuration = -1;

mCurrentBufferPercentage = 0;

mMediaPlayer = **new** MediaPlayer(mContext, mHardwareDecoder);

mMediaPlayer.setOnPreparedListener(mPreparedListener);

mMediaPlayer.setOnVideoSizeChangedListener(mSizeChangedListener);

mMediaPlayer.setOnCompletionListener(mCompletionListener);

mMediaPlayer.setOnErrorListener(mErrorListener);

mMediaPlayer.setOnBufferingUpdateListener(mBufferingUpdateListener);

mMediaPlayer.setOnInfoListener(mInfoListener);

mMediaPlayer.setOnSeekCompleteListener(mSeekCompleteListener);

mMediaPlayer.setOnTimedTextListener(mTimedTextListener);

HashMap<String, String> options = **new** HashMap<String, String>();

options.put("rtsp\_transport", "tcp"); // udp

options.put("analyzeduration", "1000000");

mMediaPlayer.setDataSource(mContext, mUri, options);

mMediaPlayer.setDisplay(mSurfaceHolder);

mMediaPlayer.setBufferSize(mBufSize);

mMediaPlayer.setVideoChroma(mVideoChroma == MediaPlayer.***VIDEOCHROMA\_RGB565*** ? MediaPlayer.***VIDEOCHROMA\_RGB565*** : MediaPlayer.***VIDEOCHROMA\_RGBA***);

mMediaPlayer.setScreenOnWhilePlaying(**true**);

mMediaPlayer.prepareAsync();

mCurrentState = ***STATE\_PREPARING***;

attachMediaController();

} **catch** (IOException ex) {

Log.*e*("Unable to open content: " + mUri, ex);

mCurrentState = ***STATE\_ERROR***;

mTargetState = ***STATE\_ERROR***;

mErrorListener.onError(mMediaPlayer, MediaPlayer.***MEDIA\_ERROR\_UNKNOWN***, 0);

**return**;

} **catch** (IllegalArgumentException ex) {

Log.*e*("Unable to open content: " + mUri, ex);

mCurrentState = ***STATE\_ERROR***;

mTargetState = ***STATE\_ERROR***;

mErrorListener.onError(mMediaPlayer, MediaPlayer.***MEDIA\_ERROR\_UNKNOWN***, 0);

**return**;

}

}

（3）MediaPlayer类的实现

MediaPlayer类是本毕业设计的最关键的类，它主要和native库打交道，该native库里面封装了对平台内置解码器的封装，同时也包含了自己开发的高性能解码器。主要函数如下：

**public** MediaPlayer(Context ctx, **boolean** preferHWDecoder) {

mContext = ctx;

String LIB\_ROOT = Vitamio.*getLibraryPath*();

**if** (preferHWDecoder) {

**if** (!*NATIVE\_OMX\_LOADED*.get()) {

**if** (Build.VERSION.***SDK\_INT*** > 17)

*loadOMX\_native*( LIB\_ROOT + "libOMX.18.so");

**else** **if** (Build.VERSION.***SDK\_INT*** > 13)

*loadOMX\_native*( LIB\_ROOT + "libOMX.14.so");

**else** **if** (Build.VERSION.***SDK\_INT*** > 10)

*loadOMX\_native*( LIB\_ROOT + "libOMX.11.so");

**else**

*loadOMX\_native*( LIB\_ROOT + "libOMX.9.so");

*NATIVE\_OMX\_LOADED*.set(**true**);

}

} **else** {

**try** {

*unloadOMX\_native*();

} **catch** (UnsatisfiedLinkError e) {

Log.*e*("unloadOMX failed %s", e.toString());

}

*NATIVE\_OMX\_LOADED*.set(**false**);

}

Looper looper;

**if** ((looper = Looper.*myLooper*()) != **null**)

mEventHandler = **new** EventHandler(**this**, looper);

**else** **if** ((looper = Looper.*getMainLooper*()) != **null**)

mEventHandler = **new** EventHandler(**this**, looper);

**else**

mEventHandler = **null**;

native\_init();

}

**static** {

String LIB\_ROOT = Vitamio.*getLibraryPath*();

**try** {

System.*load*( LIB\_ROOT + "libstlport\_shared.so");

System.*load*( LIB\_ROOT + "libvplayer.so");

*loadFFmpeg\_native*( LIB\_ROOT + "libffmpeg.so");

**boolean** vvo\_loaded = **false**;

**if** (Build.VERSION.***SDK\_INT*** > 8)

vvo\_loaded = *loadVVO\_native*( LIB\_ROOT + "libvvo.9.so");

**else** **if** (Build.VERSION.***SDK\_INT*** > 7)

vvo\_loaded = *loadVVO\_native*( LIB\_ROOT + "libvvo.8.so");

**else**

vvo\_loaded = *loadVVO\_native*( LIB\_ROOT + "libvvo.7.so");

**if** (!vvo\_loaded) {

vvo\_loaded = *loadVVO\_native*( LIB\_ROOT + "libvvo.j.so");

Log.*d*("FALLBACK TO VVO JNI " + vvo\_loaded);

}

*loadVAO\_native*( LIB\_ROOT + "libvao.0.so");

} **catch** (java.lang.UnsatisfiedLinkError e) {

Log.*e*("Error loading libs", e);

}

}

1. **测试**

（1）测试的意义

软件测试的意义在于，保证发布出去的产品达到了一定的质量标准。软件测试工程师的工作就是利用测试工具按照测试方案和流程对产品进行功能和性能测试，甚至根据需要编写不同的测试工具，设计和维护测试系统，对测试方案可能出现的问题进行分析和评估。执行测试用例后，需要跟踪故障，以确保开发的产品适合需求。 使用人工或者自动手段来运行或测试某个系统的过程,其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别. 它是帮助识别开发完成（中间或最终的版本）的计算机软件（整体或部分）的正确度(correctness) 、完全度(completeness)和质量(quality)的软件过程；是SQA(software quality assurance)的重要子域。

1. 功能测试

主要为手动测试。测试步骤：

（1）基本功能测试

1. 打开应用程序

2. 点击视频播放

3 播放过程中随机点击播放，暂停，快进，快退等操作。

4. 关闭应用

5. 重复1～4

2. 稳定性测试

连续播放视频2小时以上，观察播放效果，手机发热情况。

3. 音视频格式支持测试

随机打开mov，mp4，flv，avi等格式的视频，观察播放情况。

经测试，该播放器符合要求。

4. 性能测试

观察播放过程中CPU使用率和功耗情况。可以使用adb shell ps和adb shell top等命令。具体如图所示：

经测试，该应用程序符合要求。

**七、附录源程序**

（1）源程序

<https://github.com/iiiCode/BYSJ.git>

（2）下载方法

1. 安装git，以Ubuntu为例：

sudo apt-get install git

2. 下载源代码

git clone https://github.com/iiiCode/BYSJ.git

**参考文献**

1. Java编程思想（第4版） [thinking in java] [美] Bruce Eckel 著；陈昊鹏 译 机械工业出版社

2. ava核心技术·卷1 基础知识（原书第9版） [Core Java Volume I—Fundamentals (Ninth Edition)] Cay S. Horstmann，Gary Cornell 著；周立新，陈波，叶乃文 等 译 机械工业出版社

3. Java核心技术（卷2）：高级特性（原书第9版） [Core Java, Volume II--Advanced Features] [美] Cay S. Horstmann，[美] Gary Cornell 著；陈昊鹏，王浩，姚建平 译 机械工业出版社

4. 移动与嵌入式开发技术·Android 4编程入门经典：开发智能手机与平板电脑应用 [Beginning Android 4 Application Development] [美] Wei-Meng Lee 著；何晨光，李洪刚 译 清华大学出版社

5. Android 4高级编程（第3版） [美] Reto Meier 著；佘建伟，赵凯 译 清华大学出版社

6. 精通Android 5 多媒体开发 王石磊 著 人民邮电出版社

7. <https://www.vitamio.org/en/>