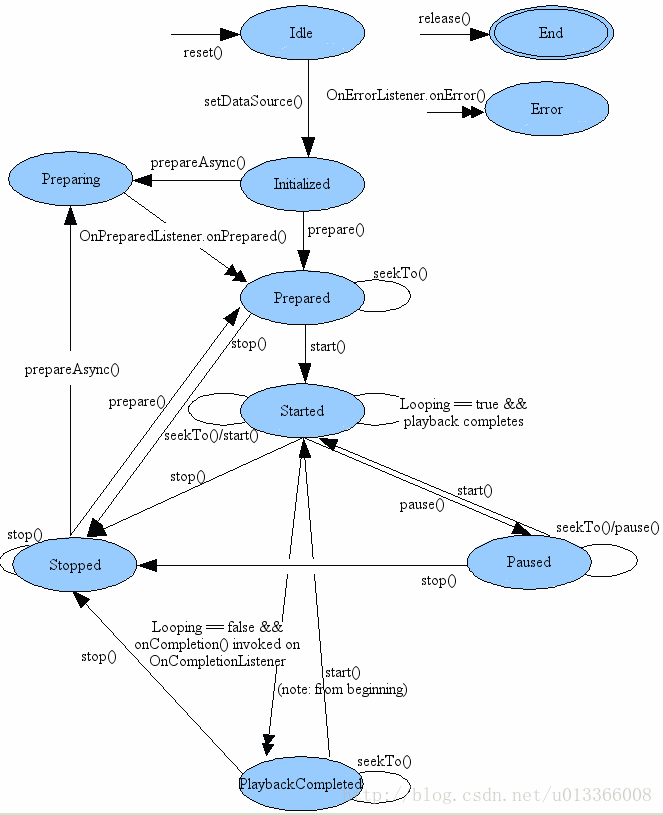
**2.播放音频**

Android提供了对常用音频和视频格式的支持，它所支持的音频格式有MP3(.mp3)、3GPP(.3gp)等，支持视频格式有3GPP和MPEG-4（**拓展名.m4a**）等。

**1）使用MediaPlayer类播放音频**



MediaPlayer类中有许多方法，如

create(Context content,int resid) 指定资源ID创建一个MediaPlayer对象

create(Context context,Uri uri) 指定URI创建一个MediaPlayer对象

setDataSource() 指定要装载的资源

prepare() 准备播放(播放前调用)

start() 开始播放

stop() 停止播放

pause() 暂停播放

reset() 恢复MediaPlayer到未初始化状态

代码：

//播放录音文件

privatevoidplayRecords(){

MediaPlayerplayer=newMediaPlayer();//无参的构造函数用于加载资源

try{

player.setDataSource(Environment.*getExternalStorageDirectory*()+"/test/20200325\_200159.m4a");

player.prepare();

player.start();//开始或恢复

}catch(Exceptione){

e.printStackTrace();

}

}

***android mediaplayer怎样判断播放结束***

给MediaPlayer对象加上播放完毕的监听：

mMediaPlayer.setOnCompletionListener(new OnCompletionListener() {

@Override

public void onCompletion(MediaPlayer mp) {

Log.d("tag", "播放完毕");

//根据需要添加自己的代码。。。

}

});

***1.1）vector***



***1.2）File***

**lastModified方法：**

public long lastModified()返回此抽象路径名表示的文件最后一次被修改的时间。

返回：

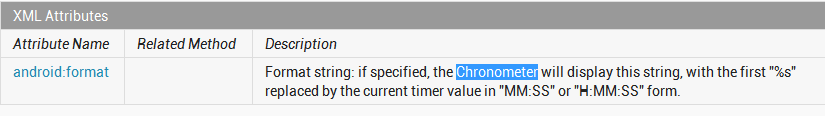
表示文件最后一次被修改的时间的 long 值，用该时间与历元（1970 年 1 月 1 日，00:00:00 GMT）的时间差来计算此值（以[毫秒](https://www.baidu.com/s?wd=%E6%AF%AB%E7%A7%92&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)为单位）。如果该文件不存在，或是发生 I/O 错误，则返回 0L

**2）使用SoundPool类播放音频**

可以同时播放多个短小的音频，而且占用资源较少.

**3)计时器的实现**

安卓提供了一个计时器组件：Chronometer，该组件extends TextView，因此都会显示一段文本，但是它显示的时间是从某个起始时间开始过去了多少时间，它只提供了android:format一个属性用于指定计时器的计数格式。



Chronometer的用法很简单，它支持如下用法：

getBase()：返回时间。

setBase(long base)：设置计时器的起始时间。

start()：开始计时。

stop()：停止计时。

setFormat(String format)：设置显示时间的格式。

setOnChronometerTickListener(Chronometer.OnChronometerTickListener listener)：为计时器绑定监听事件。

**4）声波振幅显示**

其实这个功能主要是依靠MediaRecorder的**getMaxAmplitude()方法**来获得声音的振幅，然后依据计算公式分贝的计算公式K=20lg(Vo/Vi) Vo当前的振幅值,Vi基准值为600来获得分贝数然后在根据分贝来显示ImageView上的不同图片。这样就实现了声波振幅显示了。

***4.1）Interface Runnable***

Runnable接口应由任何类实现，其实例将由线程执行。 该类必须定义一个无参数的方法，称为run 。

该接口旨在为希望在活动时执行代码的对象提供一个通用协议。 例如， Runnable由Thread类Thread 。 活跃的只是意味着一个线程已经启动，还没有被停止。

另外， Runnable提供了一个类被激活而不是Thread Thread类化的Thread 。 一个实现类Runnable可以在不继承运行Thread实例化一个Thread实例，并在传递本身作为目标。 在大多数情况下， Runnable接口应使用，如果你只打算重写run()方法并没有其他Thread方法。 这是重要的，因为类不应该被子类化，除非程序员打算修改或增强类的基本行为。

***4.2）getMaxAmplitude() 方法***

这个方法是用来获取在前一次调用此方法之后录音中出现的最大振幅，文档解释如下：

Returns the maximum absolute amplitude that was sampled since the last call to this method. Call this only after the setAudioSource().

the maximum absolute amplitude measured since the last call, or 0 when called for the first time。

所以这个方法是需要间隔一段时间调用一次的，也就是说，需要**放在线程里面调用的**。第一次调用会返回0。

**根据分贝的大小不断的切换图片**

主要代码：

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!显示录音分贝模块  
 private final Handler mHandler = new Handler();  
 //开启录音的线程  
 private Runnable mUpdateMicStatusTimer = new Runnable() {  
 public void run() {  
 updateMicStatus();  
 }  
 };  
 */\*\*  
 \* 更新话筒状态 分贝是也就是相对响度 分贝的计算公式K=20lg(Vo/Vi) Vo当前振幅值 Vi基准值为600：我是怎么制定基准值的呢？ 当20  
 \* \* Math.log10(mMediaRecorder.getMaxAmplitude() / Vi)==0的时候vi就是我所需要的基准值  
 \* 当我不对着麦克风说任何话的时候，测试获得的mMediaRecorder.getMaxAmplitude()值即为基准值。  
 \* Log.i("mic\_", "麦克风的基准值：" + mMediaRecorder.getMaxAmplitude());前提时不对麦克风说任何话  
 \*/* private void updateMicStatus() {  
 if (mMediaRecorder != null && view != null) {  
 // int vuSize = 10 \* mMediaRecorder.getMaxAmplitude() / 32768;  
 int ratio = mMediaRecorder.getMaxAmplitude() / BASE;  
 int db = 0;// 分贝  
 if (ratio > 1)  
 db = (int) (20 \* Math.*log10*(ratio));  
 Log.*d*("分贝值："+db+" "+Math.*log10*(ratio),"s");  
 switch (db / 4) {  
 case 0:  
 view.setImageResource(R.drawable.*f1*);  
 break;  
 case 1:  
 view.setImageResource(R.drawable.*f01*);  
 break;  
 case 2:  
 view.setImageResource(R.drawable.*f2*);  
 break;  
 case 3:  
 view.setImageResource(R.drawable.*f3*);  
 break;  
 case 4:  
 view.setImageResource(R.drawable.*f4*);  
 break;  
 case 5:  
 view.setImageResource(R.drawable.*f5*);  
 break;  
 default:  
 view.setImageResource(R.drawable.*ic\_launcher*);  
 break;  
 }  
 mHandler.postDelayed(mUpdateMicStatusTimer, SPACE);  
 /\*  
 \* if (db > 1) { vuSize = (int) (20 \* Math.log10(db)); Log.i("mic\_",  
 \* "麦克风的音量的大小：" + vuSize); } else Log.i("mic\_", "麦克风的音量的大小：" + 0);  
 \*/  
 }  
 }  
  
  
//!!!!!!!!!!!!!!!!!!!!!!！！！！！！！！!!!!播放录音模块  
 // 获取当前目录下所有的m4a文件并按时间排好顺序  
 public static Vector<String> GetAutioFileName(String fileAbsolutePath) {  
 Vector<String> vecFile = new Vector<>();  
 File file = new File(fileAbsolutePath);  
 File[] subFile = file.listFiles();  
  
 *sortFileByTime*(subFile);  
  
 for (int iFileLength = 0; iFileLength < subFile.length; iFileLength++) {  
 // 判断是否为文件夹  
 if (!subFile[iFileLength].isDirectory()) {  
 String filename = subFile[iFileLength].getName();  
 // 判断是否为m4a结尾  
 if (filename.trim().toLowerCase().endsWith(".m4a")) {  
 vecFile.add(filename);//将指定的元素追加到此Vector的末尾。  
 }  
 }  
 }  
 return vecFile;  
 }  
 //按时间升序排列  
 private static void sortFileByTime(File[] subFile) {  
  
 for (int i=1;i<subFile.length;i++){  
 for (int j=0;j<subFile.length-i;j++){  
 if (subFile[j].lastModified()<subFile[j+1].lastModified()){  
 File temp = subFile[j];  
 subFile[j] = subFile[j+1];  
 subFile[j+1] = temp;  
 }  
 }  
 }  
 }  
 //初始化录音文件  
 private void initAudios(){  
 //初始化资源文件  
 audios = *GetAutioFileName*(Environment.*getExternalStorageDirectory*() + "/test");  
 }  
  
 //播放录音文件  
 private void playRecords(){  
 player = new MediaPlayer();//无参的构造函数用于加载资源  
  
 //只是在第一次播放时或有新的录音问价加入时刷新列表  
 if (playFlag == 0){  
 initAudios();  
 }  
  
 try{  
 if (!audios.isEmpty()){  
 player.setDataSource(Environment.*getExternalStorageDirectory*() + "/test/"+audios.get(0));  
 player.prepare();  
 player.start(); //开始或恢复,阻塞式方法？  
 play.setEnabled(false);//防止同时播放两个录音  
 audios.remove(0);  
 playFlag++;  
 }else {  
 Toast.*makeText*(this,"录音已经播放完了！",Toast.*LENGTH\_SHORT*).show();  
 }  
  
 }catch(Exception e){  
 e.printStackTrace();  
 }  
  
 //对播放完成的监听  
 player.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {  
 @Override  
 public void onCompletion(MediaPlayer mediaPlayer) {  
 play.setEnabled(true);  
 continuePaly.setEnabled(false);  
 pause.setEnabled(false);  
 }  
 });  
  
 }  
  
 //暂停播放录音  
 private void pauseRecords(){  
 player.pause();  
 continuePaly.setEnabled(true);  
 }  
  
 //继续播放  
 private void continueRecords(){  
 player.start();  
 }