# TODO

<https://blog.csdn.net/deng0zhaotai/article/details/45058935>

<https://www.cnblogs.com/zzb-Dream-90Time/p/7766306.html>

<https://blog.csdn.net/wlia/article/details/46865783>

https://blog.csdn.net/xiashaohua/article/details/53842337

深入理解AudioService

<https://blog.csdn.net/Innost/article/details/47419025>

设置的声音值怎么传递给audoService用的？

耳机插拔

<https://blog.csdn.net/lsn946803746/article/details/82787795>

# API

AudioManager简介： AudioManager类提供了访问音量和振铃器mode控制。使用Context.getSystemService（Context.AUDIO\_SERVICE）来得到这个类的一个实例。

公有方法：

## 方法：adjustVolume(int direction, int flags)

作用：控制手机音量,调大或者调小一个单位,根据第一个参数进行判断 。

参数：AudioManager.ADJUST\_LOWER,可调小一个单位; AudioManager.ADJUST\_RAISE,可调大一个单位

3.2.

## 方法：adjustStreamVolume(int streamType, int direction, int flags)：

作用：同上,不过可以选择调节的声音类型

参数1：streamType参数,指定声音类型,有下述几种声音类型:

STREAM\_ALARM：手机闹铃 STREAM\_MUSIC：手机音乐

STREAM\_RING：电话铃声 STREAM\_SYSTEAM：手机系统

STREAM\_DTMF：音调 STREAM\_NOTIFICATION：系统提示

STREAM\_VOICE\_CALL:语音电话

参数2：上面那个一样,调大或调小音量的。

参数3：可选的标志位,比如AudioManager.FLAG\_SHOW\_UI,显示进度条,AudioManager.PLAY\_SOUND:播放声音。

3.3.

## 方法：setStreamVolume(int streamType, int index, intflags)

作用：直接设置音量大小。

## 声音模式

3.4.

方法：getMode( )

作用：返回当前的音频模式。

3.5.

方法：setMode( )

作用：设置声音模式 。

返回值：有下述几种模式:

MODE\_NORMAL(普通), MODE\_RINGTONE(铃声)

MODE\_IN\_CALL(打电话)，MODE\_IN\_COMMUNICATION(通话)

3.6.

方法：getRingerMode( )

作用：返回当前的铃声模式。

## 铃声模式

3.7.

方法：setRingerMode(int streamType)

作用：设置铃声模式 。

返回值：有下述几种模式:

如RINGER\_MODE\_NORMAL（普通）

RINGER\_MODE\_SILENT（静音）

RINGER\_MODE\_VIBRATE（震动）

## 手机的当前音量

方法：getStreamVolume(int streamType)

作用：获得手机的当前音量,最大值为7,最小值为0,当设置为0的时候,会自动调整为震动模式

3.9.

方法：getStreamMaxVolume(int streamType)

作用：获得手机某个声音类型的最大音量值。

3.10.

方法：setStreamMute(int streamType,boolean state)

作用：将手机某个声音类型设置为静音。

3.11.

方法：setSpeakerphoneOn(boolean on)

作用：设置是否打开扩音器。

3.12.

方法：setMicrophoneMute(boolean on)

作用：设置是否让麦克风静音。

3.13.

方法：isMicrophoneMute()

作用：判断麦克风是否静音或是否打开。

## 方法：isMusicActive()

作用：判断是否有音乐处于活跃状态。

3.15.

方法：isWiredHeadsetOn()

作用：判断是否插入了耳机。

3.16.

方法：abandonAudioFocus(AudioManager.OnAudioFocusChangeListenerl)

作用：放弃音频的焦点。

3.17.

## adjustSuggestedStreamVolume

方法：adjustSuggestedStreamVolume(int,int suggestedStreamType intflags)

作用：调整最相关的流的音量，或者给定的回退流。

3.18.

方法：getParameters(String keys)

作用：给音频硬件设置一个varaible数量的参数值。

3.19.

方法：getVibrateSetting(int vibrateType)

作用：返回是否该用户的振动设置为振动类型。

3.20.

方法：isBluetoothA2dpOn()

作用：检查是否A2DP蓝牙耳机音频路由是打开或关闭。

3.21.

方法：isBluetoothScoAvailableOffCall()

作用：显示当前平台是否支持使用SCO的关闭调用用例。

3.22.

方法：isBluetoothScoOn()

作用：检查通信是否使用蓝牙SCO。

3.23.

方法：loadSoundEffects()

作用：加载声音效果。

3.24.

方法：playSoundEffect((int effectType, float volume)

作用：播放声音效果。

3.25.

方法：egisterMediaButtonEventReceiver(ComponentName eventReceiver)

作用：注册一个组件MEDIA\_BUTTON意图的唯一接收机。

3.26.

方法：requestAudioFocus(AudioManager.OnAudioFocusChangeListener l,int streamType,int durationHint)

作用：请求音频的焦点。

3.27.

方法：setBluetoothScoOn(boolean on)

作用：要求使用蓝牙SCO耳机进行通讯。

3.28.

方法：startBluetoothSco/stopBluetoothSco()

作用：启动/停止蓝牙SCO音频连接。

3.29.

方法：unloadSoundEffects()

作用：卸载音效。

---------------------

作者：Penny-听海

来源：CSDN

原文：https://blog.csdn.net/weixin\_37730482/article/details/80567141

版权声明：本文为博主原创文章，转载请附上博文链接！

本文为博主原创文章，转载请附上博文链接！

附上博文链接！

## **adjustVolume**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Public Methods** | | | | | | | | | | | |
| int | abandonAudioFocus(AudioManager.OnAudioFocusChangeListenerl)  放弃音频的焦点。 |  |  |  |  |  |  |  |  |  |  |
| void | adjustStreamVolume(int streamType, int direction, int flags)  调整手机指定类型的声音。 |  |  |  |  |  |  |  |  |  |  |
| void | adjustSuggestedStreamVolume(int,int suggestedStreamType intflags)  调整最相关的流的音量，或者给定的回退流。 |  |  |  |  |  |  |  |  |  |  |
| void | adjustVolume(int,intflags)  调整最相关的数据流的音量。 |  |  |  |  |  |  |  |  |  |  |
| int | getMode()  返回当前音频模式。 |  |  |  |  |  |  |  |  |  |  |
| String | getParameters(String keys)  给音频硬件设置一个varaible数量的参数值 |  |  |  |  |  |  |  |  |  |  |
| int | getRingerMode()  返回当前的铃声模式。 |  |  |  |  |  |  |  |  |  |  |
| int | getRouting(int mode)  这个方法是不赞成的。不直接查询音频路由,使用isSpeakerphoneOn(), isBluetoothA2dpOn isBluetoothScoOn(),()和isWiredHeadsetOn()方法。 |  |  |  |  |  |  |  |  |  |  |
| int | getStreamMaxVolume(int streamType)  返回特定流的最大音量指数。 |  |  |  |  |  |  |  |  |  |  |
| int | getStreamVolume(int streamType)  返回特定数据流的当前音量索引。 |  |  |  |  |  |  |  |  |  |  |
| int | getVibrateSetting(int vibrateType)  返回是否该用户的振动设置为振动类型。 |  |  |  |  |  |  |  |  |  |  |
| boolean | isBluetoothA2dpOn()  检查是否A2DP蓝牙耳机音频路由是打开或关闭。 |  |  |  |  |  |  |  |  |  |  |
| boolean | isBluetoothScoAvailableOffCall()  显示当前平台是否支持使用SCO的关闭调用用例。 |  |  |  |  |  |  |  |  |  |  |
| boolean | isBluetoothScoOn()  检查通信是否使用蓝牙SCO。 |  |  |  |  |  |  |  |  |  |  |
| boolean | isMicrophoneMute()  检查是否打开或关闭麦克风静音。 |  |  |  |  |  |  |  |  |  |  |
| boolean | isMusicActive()  检查是否有音乐处于活跃状态。 |  |  |  |  |  |  |  |  |  |  |
| boolean | isSpeakerphoneOn()  检查是否打开或关闭扬声器。 |  |  |  |  |  |  |  |  |  |  |
| boolean | isWiredHeadsetOn()  检查是否插入耳机 |  |  |  |  |  |  |  |  |  |  |
| void | loadSoundEffects()  加载声音效果。 |  |  |  |  |  |  |  |  |  |  |
| void | playSoundEffect((int effectType, float volume)  播放声音效果（按键点击，盖开启/关闭...） |  |  |  |  |  |  |  |  |  |  |
| void | playSoundEffect(int effectType)  播放声音效果（按键点击，盖开启/关闭...） |  |  |  |  |  |  |  |  |  |  |
| void | registerMediaButtonEventReceiver(ComponentName eventReceiver)  注册一个组件MEDIA\_BUTTON意图的唯一接收机。 |  |  |  |  |  |  |  |  |  |  |
| int | requestAudioFocus(AudioManager.OnAudioFocusChangeListener l,int streamType,int durationHint)  请求音频的焦点。 |  |  |  |  |  |  |  |  |  |  |
| void | setBluetoothA2dpOn(boolean on)  这个方法是不赞成的。不要使用。 |  |  |  |  |  |  |  |  |  |  |
| void | setBluetoothScoOn(boolean on)  要求使用蓝牙SCO耳机进行通讯。 |  |  |  |  |  |  |  |  |  |  |
| void | setMicrophoneMute(boolean on)  设置是否让麦克风静音。 |  |  |  |  |  |  |  |  |  |  |
| void | setMode(int mode)  设置声音模式。 |  |  |  |  |  |  |  |  |  |  |
| void | setParameters(String keyValuePairs)  设置一个音频硬件数量可变的参数值。 |  |  |  |  |  |  |  |  |  |  |
| void | setRingerMode(int ringerMode)  设置手机电话铃声的模式。 |  |  |  |  |  |  |  |  |  |  |
| void | setRouting(int mode, int routes, int mask)  这个方法是不赞成的。不设置音频路由直接使用setSpeakerphoneOn(), setBluetoothScoOn()方法。 |  |  |  |  |  |  |  |  |  |  |
| void | setSpeakerphoneOn(boolean on)  设置扬声器打开或关闭。 |  |  |  |  |  |  |  |  |  |  |
| void | setStreamMute(int streamType,booleanstate)  将手机的指定类型的声音调整为静音。 |  |  |  |  |  |  |  |  |  |  |
| void | setStreamSolo(int streamType,boolean state)  独奏或来取消特定流。 |  |  |  |  |  |  |  |  |  |  |
| void | setStreamVolume (int streamType, int index, int flags)  直接设置手机的指定类型的音量值。 |  |  |  |  |  |  |  |  |  |  |
| void | setVibrateSetting (int vibrateType, int vibrateSetting)  设置设置时应该振动的振动类型。 |  |  |  |  |  |  |  |  |  |  |
| void | setWiredHeadsetOn(boolean on)  这个方法是不赞成的。不要使用。 |  |  |  |  |  |  |  |  |  |  |
| boolean | shouldVibrate(int vibrateType)  返回特定类型是否应该根据用户震动设置和当前的振铃器 mode。 |  |  |  |  |  |  |  |  |  |  |
| void | startBluetoothSco()  启动蓝牙SCO音频连接。 |  |  |  |  |  |  |  |  |  |  |
| void | stopBluetoothSco()  停止蓝牙SCO音频连接。 |  |  |  |  |  |  |  |  |  |  |
| void | unloadSoundEffects()  卸载音效。 |  |  |  |  |  |  |  |  |  |  |
| void | unregisterMediaButtonEventReceiver(ComponentName eventReceiver)  注销MEDIA\_BUTTON意图的接收机。 |  |  |  |  |  |  |  |  |  |  |

## adjustStreamVolume

常用方法说明：

void  adjustStreamVolume(int streamType,int direction, int flags)

调整手机指定类型的声音。其中第一个参数streamType指定声音类型，该参数可接受如下几个值。

l  int    STREAM\_ALARM：手机闹铃的声音。

l  int    STREAM\_DTMF：DTMF音调的声音。

l  int    STREAM\_MUSIC：手机音乐的声音。

l  int    STREAM\_NOTIFICATION：系统提示的声音。

l  int    STREAM\_RING         The：电话铃声的声音。

l  int    STREAM\_SYSTEM：手机系统的声音。

l  int    STREAM\_VOICE\_CALL：语音电话的声音。

第二个参数指定对声音进行增大、还是减小；第三个参数是调整声音时的标志，例如指定FLAG\_SHOW\_UI，则指定调整声音时显示音量进度条。

void setMicrophoneMute(booleanon)

设置是否让麦克风静音。设置为true将麦克风静音;false关闭静音

void  setMode(intmode)

设置声音模式。可设置的值有 NORMAL,RINGTONE, 和IN\_CALL。

void setRingerMode(intringerMode)

设置手机电话铃声的模式。可支持如下几个属性值。

l  int    RINGER\_MODE\_NORMAL：正常的手机铃声。

l  int    RINGER\_MODE\_SILENT：手机铃声静音。

l  int    RINGER\_MODE\_VIBRATE：手机震动。

void setSpeakerphoneOn(booleanon)

设置扬声器打开或关闭。设置为true开启免提通话;false关闭免提。

void setStreamMute(intstreamType,booleanstate)

将手机的指定类型的声音调整为静音。其中streamType参数与adjustStreamVolume方法中第一个参数的意义相同。

void setStreamVolume (int streamType, int index, int flags)

直接设置手机的指定类型的音量值。其中streamType参数与adjustStreamVolume方法中第一个参数的意义相同。

# playSoundEffectVolume

## 音效事件起源

Android 平台所有view类型控件的touch，遥控器按键等事件在系统音效开启时，都可以触发按键音。

## 音效事件流程分析

从touch事件为例，整个流程如下图所示

如图中所示，在touch事件触发后，onTouchEvent会被调用，接着performClick就会处理Click事件，

### View.performClick

public boolean performClick() {

sendAccessibilityEvent(AccessibilityEvent.TYPE\_VIEW\_CLICKED);

ListenerInfo li = mListenerInfo;

if (li != null && li.mOnClickListener != null) {

playSoundEffect(SoundEffectConstants.CLICK);

li.mOnClickListener.onClick(this);

return true;

}

return false;

}

performClick

在里面主要做了两件事，调用监听器的onClick回调和playSoundEffect，监听器的回调是在setOnClickListener中注册下来，下面主要分析playSoundEffect

public void playSoundEffect(int soundConstant) {

if (mAttachInfo == null || mAttachInfo.mRootCallbacks == null || !isSoundEffectsEnabled()) {

return;

}

mAttachInfo.mRootCallbacks.playSoundEffect(soundConstant);

}

利用成员变量mAttachInfo的mRootCallbacks回调调用另外一个音效处理函数playSoundEffect，这里找到mAttachInfo和mRootCallbacks的源头就很重要，其实每个View控件

在创建后，都会绑定到对应窗口Window,下面看一下函数dispatchAttachedToWindow

void dispatchAttachedToWindow(AttachInfo info, int visibility) {

*//System.out.println("Attached! " + this);*

mAttachInfo = info;

if (mOverlay != null) {

mOverlay.getOverlayView().dispatchAttachedToWindow(info, visibility);

}

mWindowAttachCount++;

*// We will need to evaluate the drawable state at least once.*

mPrivateFlags |= PFLAG\_DRAWABLE\_STATE\_DIRTY;

if (mFloatingTreeObserver != null) {

info.mTreeObserver.merge(mFloatingTreeObserver);

mFloatingTreeObserver = null;

}

if ((mPrivateFlags&PFLAG\_SCROLL\_CONTAINER) != 0) {

mAttachInfo.mScrollContainers.add(this);

mPrivateFlags |= PFLAG\_SCROLL\_CONTAINER\_ADDED;

}

performCollectViewAttributes(mAttachInfo, visibility);

onAttachedToWindow();

ListenerInfo li = mListenerInfo;

final CopyOnWriteArrayList<OnAttachStateChangeListener> listeners =

li != null ? li.mOnAttachStateChangeListeners : null;

if (listeners != null && listeners.size() > 0) {

*// NOTE: because of the use of CopyOnWriteArrayList, we \*must\* use an iterator to*

*// perform the dispatching. The iterator is a safe guard against listeners that*

*// could mutate the list by calling the various add/remove methods. This prevents*

*// the array from being modified while we iterate it.*

for (OnAttachStateChangeListener listener : listeners) {

listener.onViewAttachedToWindow(this);

}

}

int vis = info.mWindowVisibility;

if (vis != GONE) {

onWindowVisibilityChanged(vis);

}

if ((mPrivateFlags&PFLAG\_DRAWABLE\_STATE\_DIRTY) != 0) {

*// If nobody has evaluated the drawable state yet, then do it now.*

refreshDrawableState();

}

needGlobalAttributesUpdate(false);

}

函数开始处有mAttachInfo = info，就给mAttachInfo赋值了，继续向上索引，ViewRootImpl.performTraversals中调用host.dispatchAttachedToWindow(attachInfo, 0);

将ViewRootImpl.mAttachInfo设置下去，而此成员在ViewRootImpl构造函数中赋值。

public ViewRootImpl(Context context, Display display) {

mContext = context;

mWindowSession = WindowManagerGlobal.getWindowSession();

mDisplay = display;

mBasePackageName = context.getBasePackageName();

mDisplayAdjustments = display.getDisplayAdjustments();

mThread = Thread.currentThread();

mLocation = new WindowLeaked(null);

mLocation.fillInStackTrace();

mWidth = -1;

mHeight = -1;

mDirty = new Rect();

mTempRect = new Rect();

mVisRect = new Rect();

mWinFrame = new Rect();

mWindow = new W(this);

mTargetSdkVersion = context.getApplicationInfo().targetSdkVersion;

mViewVisibility = View.GONE;

mTransparentRegion = new Region();

mPreviousTransparentRegion = new Region();

mFirst = true; *// true for the first time the view is added*

mAdded = false;

mAccessibilityManager = AccessibilityManager.getInstance(context);

mAccessibilityInteractionConnectionManager =

new AccessibilityInteractionConnectionManager();

mAccessibilityManager.addAccessibilityStateChangeListener(

mAccessibilityInteractionConnectionManager);

mAttachInfo = new View.AttachInfo(mWindowSession, mWindow, display, this, mHandler, this);

mViewConfiguration = ViewConfiguration.get(context);

mDensity = context.getResources().getDisplayMetrics().densityDpi;

mNoncompatDensity = context.getResources().getDisplayMetrics().noncompatDensityDpi;

mFallbackEventHandler = PolicyManager.makeNewFallbackEventHandler(context);

mChoreographer = Choreographer.getInstance();

PowerManager powerManager = (PowerManager) context.getSystemService(Context.POWER\_SERVICE);

mAttachInfo.mScreenOn = powerManager.isScreenOn();

loadSystemProperties();

}

mAttachInfo = new View.AttachInfo(mWindowSession, mWindow, display, this, mHandler, this);

再看AttachInfo构造函数，如下：

*/\*\**

*\* Creates a new set of attachment information with the specified*

*\* events handler and thread.*

*\**

*\* @param handler the events handler the view must use*

*\*/*

AttachInfo(IWindowSession session, IWindow window, Display display,

ViewRootImpl viewRootImpl, Handler handler, Callbacks effectPlayer) {

mSession = session;

mWindow = window;

mWindowToken = window.asBinder();

mDisplay = display;

mViewRootImpl = viewRootImpl;

mHandler = handler;

mRootCallbacks = effectPlayer;

}

mRootCallbacks 其实就是ViewRootImp的实例，这样View.playSoundEffect就进入了ViewRootImp.playSoundEffect.

### ViewRootImp.playSoundEffect.

## 音频系统接管音效控制

从ViewRootImp的函数playSoundEffect

public void playSoundEffect(int effectId) {

checkThread();

if (mMediaDisabled) {

return;

}

try {

final AudioManager audioManager = getAudioManager();

switch (effectId) {

case SoundEffectConstants.CLICK:

audioManager.playSoundEffect(AudioManager.FX\_KEY\_CLICK);

return;

case SoundEffectConstants.NAVIGATION\_DOWN:

audioManager.playSoundEffect(AudioManager.FX\_FOCUS\_NAVIGATION\_DOWN);

return;

case SoundEffectConstants.NAVIGATION\_LEFT:

audioManager.playSoundEffect(AudioManager.FX\_FOCUS\_NAVIGATION\_LEFT);

return;

case SoundEffectConstants.NAVIGATION\_RIGHT:

audioManager.playSoundEffect(AudioManager.FX\_FOCUS\_NAVIGATION\_RIGHT);

return;

case SoundEffectConstants.NAVIGATION\_UP:

audioManager.playSoundEffect(AudioManager.FX\_FOCUS\_NAVIGATION\_UP);

return;

default:

throw new IllegalArgumentException("unknown effect id " + effectId +

" not defined in " + SoundEffectConstants.class.getCanonicalName());

}

} catch (IllegalStateException e) {

*// Exception thrown by getAudioManager() when mView is null*

Log.e(TAG, "FATAL EXCEPTION when attempting to play sound effect: " + e);

e.printStackTrace();

}

}

AudioManager接管音效控制权，接着AudioService的onPlaySoundEffect会被调用

## AudioManager.playSoundEffect

**public void** playSoundEffect(**int** effectType, **int** userId) {  
 **if** (effectType < 0 || effectType >= NUM\_SOUND\_EFFECTS) {  
 **return**;  
 }  
  
 **if** (!querySoundEffectsEnabled(userId)) {  
 **return**;  
 }  
  
 IAudioService service = getService();  
 **try** {  
 service.playSoundEffect(effectType);  
 } **catch** (RemoteException e) {  
 **throw** e.rethrowFromSystemServer();  
 }  
}

## AudioService.

sendMsg(mAudioHandler, MSG\_PLAY\_SOUND\_EFFECT, SENDMSG\_QUEUE,  
 effectType, (**int**) (volume \* 1000), **null**, 0);

## sendMsg

**rivate static void** sendMsg(Handler handler, **int** msg,  
 **int** existingMsgPolicy, **int** arg1, **int** arg2, Object obj, **int** delay) {  
  
 **if** (existingMsgPolicy == SENDMSG\_REPLACE) {  
 handler.removeMessages(msg);  
 } **else if** (existingMsgPolicy == SENDMSG\_NOOP && handler.hasMessages(msg)) {  
 **return**;  
 }  
 **synchronized** (mLastDeviceConnectMsgTime) {  
 **long** time = SystemClock.uptimeMillis() + delay;  
 handler.sendMessageAtTime(handler.obtainMessage(msg, arg1, arg2, obj), time);  
 **if** (msg == MSG\_SET\_WIRED\_DEVICE\_CONNECTION\_STATE ||  
 msg == MSG\_SET\_A2DP\_SRC\_CONNECTION\_STATE ||  
 msg == MSG\_SET\_A2DP\_SINK\_CONNECTION\_STATE) {  
 mLastDeviceConnectMsgTime = time;  
 }  
 }  
}

## case MSG\_PLAY\_SOUND\_EFFECT: onPlaySoundEffect(msg.arg1, msg.arg2);

## AudioService.onPlaySoundEffect

private void onPlaySoundEffect(int effectType, int volume) {

synchronized (mSoundEffectsLock) {

onLoadSoundEffects();

if (mSoundPool == null) {

return;

}

float volFloat;

*// use default if volume is not specified by caller*

if (volume < 0) {

volFloat = (float)Math.pow(10, (float)sSoundEffectVolumeDb/20);

} else {

volFloat = (float) volume / 1000.0f;

}

if (SOUND\_EFFECT\_FILES\_MAP[effectType][1] > 0) {

mSoundPool.play(SOUND\_EFFECT\_FILES\_MAP[effectType][1],

volFloat, volFloat, 0, 0, 1.0f);

} else {

MediaPlayer mediaPlayer = new MediaPlayer();

try {

String filePath = Environment.getRootDirectory() + SOUND\_EFFECTS\_PATH +

SOUND\_EFFECT\_FILES.get(SOUND\_EFFECT\_FILES\_MAP[effectType][0]);

mediaPlayer.setDataSource(filePath);

mediaPlayer.setAudioStreamType(AudioSystem.STREAM\_SYSTEM);

mediaPlayer.prepare();

mediaPlayer.setVolume(volFloat);

mediaPlayer.setOnCompletionListener(new OnCompletionListener() {

public void onCompletion(MediaPlayer mp) {

cleanupPlayer(mp);

}

});

mediaPlayer.setOnErrorListener(new OnErrorListener() {

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

cleanupPlayer(mp);

return true;

}

});

mediaPlayer.start();

} catch (IOException ex) {

Log.w(TAG, "MediaPlayer IOException: "+ex);

} catch (IllegalArgumentException ex) {

Log.w(TAG, "MediaPlayer IllegalArgumentException: "+ex);

} catch (IllegalStateException ex) {

Log.w(TAG, "MediaPlayer IllegalStateException: "+ex);

}

}

}

}

sSoundEffectVolumeDb 是音效使用的音量大小值，mSoundPool.play完成音效的播放功能。

**private boolean** onLoadSoundEffects() {  
 **int** status;  
  
 **synchronized** (mSoundEffectsLock) {  
 **if** (!mSystemReady) {  
 Log.w(TAG, **"onLoadSoundEffects() called before boot complete"**);  
 **return false**;  
 }  
  
 **if** (mSoundPool != **null**) {  
 **return true**;  
 }

音频系统音效全局控制

*/\*\**

*\* Plays a sound effect (Key clicks, lid open/close...)*

*\* @param effectType The type of sound effect. One of*

*\* {@link #FX\_KEY\_CLICK},*

*\* {@link #FX\_FOCUS\_NAVIGATION\_UP},*

*\* {@link #FX\_FOCUS\_NAVIGATION\_DOWN},*

*\* {@link #FX\_FOCUS\_NAVIGATION\_LEFT},*

*\* {@link #FX\_FOCUS\_NAVIGATION\_RIGHT},*

*\* {@link #FX\_KEYPRESS\_STANDARD},*

*\* {@link #FX\_KEYPRESS\_SPACEBAR},*

*\* {@link #FX\_KEYPRESS\_DELETE},*

*\* {@link #FX\_KEYPRESS\_RETURN},*

*\* {@link #FX\_KEYPRESS\_INVALID},*

*\* NOTE: This version uses the UI settings to determine*

*\* whether sounds are heard or not.*

*\*/*

public void playSoundEffect(int effectType) {

if (effectType < 0 || effectType >= NUM\_SOUND\_EFFECTS) {

return;

}

if (!querySoundEffectsEnabled()) {

return;

}

*//Ctv Patch Begin*

effectTypeForHandler = effectType;

userHandler.removeCallbacks(queueRun);

userHandler.postDelayed(queueRun, 20);

*/\*IAudioService service = getService();*

*try {*

*service.playSoundEffect(effectType);*

*} catch (RemoteException e) {*

*Log.e(TAG, "Dead object in playSoundEffect"+e);*

*}\*/*

}

querySoundEffectsEnabled来查询系统是否设置开机全局按键音，而系统开启关闭按键也是通过Settings.System.SOUND\_EFFECTS\_ENABLED的设置来完成控制。

Settings.System.putInt(mContext.getContentResolver(), Settings.System.SOUND\_EFFECTS\_ENABLED, 0/1) 关闭/开启

FLAG\_PLAY\_SOUND

**public void** playSoundEffectVolume(**int** effectType, **float** volume) {  
 **if** (effectType >= AudioManager.NUM\_SOUND\_EFFECTS || effectType < 0) {  
 Log.w(TAG, **"AudioService effectType value "** + effectType + **" out of range"**);  
 **return**;  
 }  
  
 sendMsg(mAudioHandler, MSG\_PLAY\_SOUND\_EFFECT, SENDMSG\_QUEUE,  
 effectType, (**int**) (volume \* 1000), **null**, 0);  
}

## SoundPool

**public class** SoundPool {  
 **static** { System.loadLibrary(**"soundpool"**); }

**public final int** play(**int** soundID, **float** leftVolume, **float** rightVolume,  
 **int** priority, **int** loop, **float** rate) {  
 **if** (isRestricted()) {  
 leftVolume = rightVolume = 0;  
 }  
 **return** \_play(soundID, leftVolume, rightVolume, priority, loop, rate);  
}

**private native final int** \_play(**int** soundID, **float** leftVolume, **float** rightVolume,  
 **int** priority, **int** loop, **float** rate);

### android\_media\_SoundPool\_play

**static** jint  
android\_media\_SoundPool\_play(JNIEnv \*env, jobject thiz, jint sampleID,  
 jfloat leftVolume, jfloat rightVolume, jint priority, jint loop,  
 jfloat rate)  
{  
 ALOGV(**"android\_media\_SoundPool\_play\n"**);  
 SoundPool \*ap = MusterSoundPool(env, thiz);  
 **if** (ap == NULL) **return** 0;  
 **return** (jint) ap->play(sampleID, leftVolume, rightVolume, priority, loop, rate);  
}

### SoundPool::play

**int** SoundPool::play(**int** sampleID, **float** leftVolume, **float** rightVolume,  
 **int** priority, **int** loop, **float** rate)  
{  
 ALOGV(**"play sampleID=%d, leftVolume=%f, rightVolume=%f, priority=%d, loop=%d, rate=%f"**,  
 sampleID, leftVolume, rightVolume, priority, loop, rate);  
 SoundChannel\* channel;  
 **int** channelID;  
  
 Mutex::Autolock lock(&mLock);  
  
 **if** (mQuit) {  
 **return** 0;  
 }  
 *// is sample ready?* sp<Sample> sample(findSample\_l(sampleID));  
 **if** ((sample == 0) || (sample->state() != Sample::READY)) {  
 ALOGW(**" sample %d not READY"**, sampleID);  
 **return** 0;  
 }  
  
 dump();  
  
 *// allocate a channel* channel = allocateChannel\_l(priority, sampleID);  
  
 *// no channel allocated - return 0* **if** (!channel) {  
 ALOGV(**"No channel allocated"**);  
 **return** 0;  
 }  
  
 channelID = ++mNextChannelID;  
  
 ALOGV(**"play channel %p state = %d"**, channel, channel->state());  
 channel->play(sample, channelID, leftVolume, rightVolume, priority, loop, rate);  
 **return** channelID;  
}

### SoundChannel::play

*// call with sound pool lock held***void** SoundChannel::play(**const** sp<Sample>& sample, **int** nextChannelID, **float** leftVolume,  
 **float** rightVolume, **int** priority, **int** loop, **float** rate)

*// if not idle, this voice is being stolen***if** (mState != IDLE) {  
 ALOGV(**"channel %d stolen - event queued for channel %d"**, channelID(), nextChannelID);  
 mNextEvent.set(sample, nextChannelID, leftVolume, rightVolume, priority, loop, rate);  
 stop\_l();  
 **return**;  
}

newTrack = **new** AudioTrack(streamType, sampleRate, sample->format(),  
 channelMask, sample->getIMemory(), AUDIO\_OUTPUT\_FLAG\_FAST, callback, userData,  
 0 */\*default notification frames\*/*, AUDIO\_SESSION\_ALLOCATE,  
 AudioTrack::TRANSFER\_DEFAULT,  
 NULL */\*offloadInfo\*/*, -1 */\*uid\*/*, -1 */\*pid\*/*, mSoundPool->attributes());

mAudioTrack->start();

### AudioTrack

**static void**android\_media\_AudioTrack\_start(JNIEnv \*env, jobject thiz)  
{  
 sp<AudioTrack> lpTrack = getAudioTrack(env, thiz);  
 **if** (lpTrack == NULL) {  
 jniThrowException(env, **"java/lang/IllegalStateException"**,  
 **"Unable to retrieve AudioTrack pointer for start()"**);  
 **return**;  
 }  
  
 lpTrack->start();  
}

## ref

<http://www.voidcn.com/article/p-zgqoifcu-bhx.html>

# 定制默认声音

Sd

系统代码frameworks/base/media/java/android/media/AudioService.java的开头定义了两个数组，一个MAX\_STREAM\_VOLUME 这里面定义了各种声音的最大值（**最大值不是100**，所以需要AudioManager.getStreamMaxVolume(type)来获取各个音量的最大值），然后进行设置。

还定义了一个数组DEFAULT\_STREAM\_VOLUME 这里面则和MAX\_STREAM\_VOLUME里定义的顺序是一样，表明了各种声音的默认的大小。此块代码如下：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 | /\*\* @hide Maximum volume index values for audio streams \*/  private static int[] MAX\_STREAM\_VOLUME = new int[] {  5, // STREAM\_VOICE\_CALL  7, // STREAM\_SYSTEMX\_STREAM\_VOLUMEMAX\_STREAM\_VOLUMEMAX\_STREAM\_VOLUME  7, // STREAM\_RING  15, // STREAM\_MUSIC  7, // STREAM\_ALARM  7, // STREAM\_NOTIFICATION  15, // STREAM\_BLUETOOTH\_SCO  7, // STREAM\_SYSTEM\_ENFORCED  15, // STREAM\_DTMF  15 // STREAM\_TTS  };   private static int[] DEFAULT\_STREAM\_VOLUME = new int[] {  4, // STREAM\_VOICE\_CALL  7, // STREAM\_SYSTEM  5, // STREAM\_RING  11, // STREAM\_MUSIC  6, // STREAM\_ALARM  5, // STREAM\_NOTIFICATION  7, // STREAM\_BLUETOOTH\_SCO  7, // STREAM\_SYSTEM\_ENFORCED  11, // STREAM\_DTMF  11 // STREAM\_TTS  }; |

如果我们需要修改默认的通知声音，则可以将STREAM\_NOTIFICATION 前面的数值 5 给为 0即可，这样默认声音就为0 了。

具体的代码在frameworks/base/packages/SettingsProvider/src/com/android/providers/settings/DatabaseHelper.java 其中有个方法loadVolumeLevels(db) 此方法则是加载所有默认声音大小

SQLiteStatement stmt = **null**;  
**try** {  
 stmt = db.compileStatement(**"INSERT OR IGNORE INTO system(name,value)"** + **" VALUES(?,?);"**);  
  
 loadSetting(stmt, Settings.System.VOLUME\_MUSIC,  
 AudioSystem.getDefaultStreamVolume(AudioManager.STREAM\_MUSIC));  
 loadSetting(stmt, Settings.System.VOLUME\_RING,  
 AudioSystem.getDefaultStreamVolume(AudioManager.STREAM\_RING));  
 loadSetting(stmt, Settings.System.VOLUME\_SYSTEM,  
 AudioSystem.getDefaultStreamVolume(AudioManager.STREAM\_SYSTEM));  
 loadSetting(  
 stmt,  
 Settings.System.VOLUME\_VOICE,  
 AudioSystem.getDefaultStreamVolume(AudioManager.STREAM\_VOICE\_CALL));  
 loadSetting(stmt, Settings.System.VOLUME\_ALARM,  
 AudioSystem.getDefaultStreamVolume(AudioManager.STREAM\_ALARM));  
 loadSetting(  
 stmt,  
 Settings.System.VOLUME\_NOTIFICATION,  
 AudioSystem.getDefaultStreamVolume(AudioManager.STREAM\_NOTIFICATION));  
 loadSetting(  
 stmt,  
 Settings.System.VOLUME\_BLUETOOTH\_SCO,  
 AudioSystem.getDefaultStreamVolume(AudioManager.STREAM\_BLUETOOTH\_SCO));

# 情景分析

## adjustSuggestedStreamVolume

### 吊起来

**static** IAudioService getAudioService() {  
 IAudioService audioService = IAudioService.Stub.asInterface(  
 ServiceManager.checkService(Context.AUDIO\_SERVICE));  
 **if** (audioService == **null**) {  
 Log.w(TAG, **"Unable to find IAudioService interface."**);  
 }  
 **return** audioService;  
}

KeyEvent.KEYCODE\_VOLUME\_UP:

getAudioService().adjustSuggestedStreamVolume(AudioManager.ADJUST\_RAISE,  
 AudioManager.USE\_DEFAULT\_STREAM\_TYPE, flags, pkgName, TAG);

### AudioService.adjustSuggestedStreamVolume

[AudioService.java-->AudioService.adjustSuggestedStreamVolume()]

public void adjustSuggestedStreamVolume(intdirection, int suggestedStreamType,int flags) {格式要调整好

int streamType;

// ①从这一小段代码中，可以看出在 AudioService中还有地方可以强行改变音量键控制的流类型

    if(mVolumeControlStream != -1) {

        streamType = mVolumeControlStream;

} else {

     // ②通过getActiveStreamType()函数获取要控制的流类型

     // 这里根据建议的流类型与AudioService的实际情况，返回一个值

       streamType = getActiveStreamType(suggestedStreamType);

    }

// ③这个啰嗦的if判断的目的，就是只有在特定的流类型下,并且没有处于锁屏状态时才会播放声音

    if((streamType != STREAM\_REMOTE\_MUSIC) &&

           (flags & AudioManager.FLAG\_PLAY\_SOUND) != 0 &&

           ((mStreamVolumeAlias[streamType] != AudioSystem.STREAM\_RING)

            || (mKeyguardManager != null &&mKeyguardManager.isKeyguardLocked()))) {

        flags&= ~AudioManager.FLAG\_PLAY\_SOUND;

    }

    if(streamType == STREAM\_REMOTE\_MUSIC) {

…… //我们不讨论远程播放的情况

} else {

    // ④调用adjustStreamVolume

       adjustStreamVolume(streamType, direction, flags);

    }

}

注意初看着段代码时，可能有读者会对下面这句话感到疑惑：

VolumeStreamState streamState =mStreamStates[mStreamVolumeAlias[streamType]];

其实这是为了满足所谓的“将铃声音量用作通知音量”这种需求。这样就需要实现在两个有这个需求的流A与B之间建立起一个A→B映射。当我们对A流进行音量操作时，实际上是在操作B流。其实笔者个人认为这个功能对用户体验的提升并不大，但是却给AudioService的实现增加了不小的复杂度。直观上来想，我们可能想使用一个HashMap解决这个问题，键是源流类型，值目标流类型。而Android使用了一个更简单那但是却不是那么好理解的方法来完成这件事。AudioService用一个名为mStreamVolumeAlias的整形数组来描述这个映射关系。

如果想要实现“以铃声音量用作音乐音量”，只需要修改相应位置的值为STREAM\_RING即可，就像下面这样：

mStreamVolumeAlias[AudioSystem.STREAM\_MUSIC] =AudioSystem.STREAM\_RING;

之后，因为需求要求对A流进行音量操作时，实际上是在操作B流，所以就不难理解为什么在很多和流相关的函数里都会先做这样的一个转换：

streamType = mStreamVolumeAlias[streamType];

其具体的工作方式就留给读者进行思考了。在本章的分析过程中，大可忽略这种转换，这并不影响我们对音量控制原理的理解。

#### 确定要调整音量的流类型。

##### getActiveStreamType

这个函数简单来说，做三件事：

·   确定要调整音量的流类型。

·   在某些情况下屏蔽FLAG\_PLAY\_SOUND。

·   调用adjustStreamVolume()。

关于这个函数仍然有几点需要说明一下。它刚开始的时候有一个判断，条件是一个名为mVolumeControlStream的整型变量是否等于-1，从这块代码来看，mVolumeControlStream比参数传入的suggestedStreamType厉害多了，只要它不是-1，那么要调整音量的流类型就是它。那这么厉害的控制手段，是做什么用的呢？其实，mVolumeControlStream是VolumePanel通过forceVolumeControlStream()函数设置的。什么是VolumePanel呢？就是我们按下音量键后的那个音量条提示框了。VolumePanel在显示时会调用forceVolumeControlStream强制后续的音量键操作固定为促使它显示的那个流类型。并在它关闭时取消这个强制设置，即置mVolumeControlStream为-1。这个我们在后面分析VolumePanel时会看到。

接下来我们继续看一下adjustStreamVolume()的实现。

---------------------

作者：阿拉神农

来源：CSDN

原文：https://blog.csdn.net/Innost/article/details/47419025

版权声明：本文为博主原创文章，转载请附上博文链接！