# CarAudioManager 返回值说明

## Result 类常量

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `Result.SUCCESS` | 0 | 调用接口成功，设置成功 |
| `Result.ERROR` | -1 | 调用接口失败，设置失败 |
| `Result.NO_PERMISSION` | -2 | 没有权限 |
| `Result.NOT_SUPPORT` | -3 | 不支持该功能 |
| `Result.INVALID_PARAMS` | -4 | 参数错误 |
| `Result.ERROR_REMOTE` | -5 | 远程调用失败 |

## 使用说明

1. 所有接口调用后都应检查返回值是否为 `Result.SUCCESS` (0)
2. 当返回值为负数时，表示操作失败，可根据具体错误码进行相应处理
3. 常见错误情况：
    - `NO_PERMISSION`: 检查是否申请了正确的权限
    - `NOT_SUPPORT`: 当前设备不支持该功能
    - `INVALID_PARAMS`: 检查传入参数是否合法
    - `ERROR_REMOTE`: 服务通信异常，可尝试重试

# 音量组ID常量说明

## 音量组ID定义

| 常量名称 | 值 | 说明 |
| --- | --- | --- |
| `VOLUME_GROUP_ID_MUSIC` | 1 | 媒体音量组 |
| `VOLUME_GROUP_ID_NAVIGATION` | 2 | 导航音量组 |
| `VOLUME_GROUP_ID_PHONE` | 3 | 电话音量组 |
| `VOLUME_GROUP_ID_VOICE_TTS` | 4 | TTS语音音量组 |
| `VOLUME_GROUP_EBCALL` | 6 | ECALL/BCALL紧急呼叫音量组 |
| `VOLUME_GROUP_ID_HEADREST_LEFT` | 9 | 左头枕音量组 |
| `VOLUME_GROUP_ID_HEADREST_RIGHT` | 10 | 右头枕音量组 |
| `VOLUME_GROUP_ID_RING` | 11 | 铃声音量组 |
| `VOLUME_GROUP_OUTMEDIA` | 12 | 外放媒体音量组 |
| `VOLUME_GROUP_OCC` | 13 | 车外喊话音量组 |

| 常量名称 | 值 | 说明 |
| --- | --- | --- |
| VOLUME_GROUP_AMBIENC | 14 | 背景氛围音音量组 |
| VOLUME_GROUP_AUTO_TTS | 15 | 辅助驾驶语音音量组 |
| AUDIO_GROUP_ID_UNKNOWN | 0 | 未知音量组 |

# CarAudioManager 权限说明

## 音频控制权限

| 常量 | 值 | 说明 |
| --- | --- | --- |
| PERMISSION_CAR_CONTROL_AUDIO_VOLUME | "android.car.permission.CAR_CONTROL_AUDIO_VOLUME" | 控制车载音频音量的权限 |

# 音频焦点常量说明

## 音频焦点请求结果

| 常量 | 值 | 说明 |
| --- | --- | --- |
| AUDIOFOCUS_REQUEST_FAILED | 0 | 音频焦点请求失败 |
| AUDIOFOCUS_REQUEST_GRANTED | 1 | 音频焦点请求成功 |
| AUDIOFOCUS_REQUEST_DELAYED | 2 | 音频焦点请求被延迟授予 |

## 音频焦点状态

| 常量 | 值 | 说明 |
| --- | --- | --- |
| AUDIOFOCUS_NONE | 0 | 无音频焦点状态 |
| AUDIOFOCUS_GAIN | 1 | 获得音频焦点(持续时间未知) |
| AUDIOFOCUS_GAIN_TRANSIENT | 2 | 获得临时音频焦点(短时间) |
| AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK | 3 | 获得临时音频焦点(允许其他应用降低音量继续播放) |

| 常量 | 值 | 说明 |
| --- | --- | --- |
| AUDIOFOCUS_GAIN_TRANSIENT_EXCLUSIVE | 4 | 获得独占临时音频焦点(其他应用必须停止播放) |

### 音频焦点丢失状态

| 常量 | 值 | 说明 |
| --- | --- | --- |
| AUDIOFOCUS_LOSS | -1 | 永久丢失音频焦点 |
| AUDIOFOCUS_LOSS_TRANSIENT | -2 | 临时丢失音频焦点 |
| AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK | -3 | 临时丢失音频焦点(可以降低音量继续播放) |

## 音量调节常量说明

### 音量调节操作类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| ADJUST_RAISE | 1 | 增加音量 |
| ADJUST_LOWER | -1 | 降低音量 |
| ADJUST_SAME | 0 | 保持当前音量不变(可用于显示音量提示) |
| ADJUST_MUTE | -100 | 静音 |
| ADJUST_UNMUTE | 100 | 取消静音 |
| ADJUST_TOGGLE_MUTE | 101 | 切换静音状态 |
| ADJUST_TO_MAXIMUM | 102 | 将音量调至最大值 |
| ADJUST_TO_MINIMUM | 103 | 将音量调至最小值 |

## 音量调节标志位说明

### 音量调节标志位常量

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| FLAG_SHOW_UI | 1 << 0 | 显示当前音量的Toast提示 | 音量调节时显示UI提示 |
| FLAG_FROM_KEY | 1 << 1 | 由硬件按键触发的音量调节 | 物理按键音量控制 |
| FLAG_PLAY_SOUND | 1 << 2 | 音量变化时播放提示音 | 音量反馈音效 |
| FLAG_FROM_VOICE_COMMAND | 1 << 3 | 由语音命令触发的音量调节 | 语音控制音量 |
| FLAG_MAX_VOLUME | 1 << 4 | 调节至最大音量(已废弃) | 建议使用ADJUST_TO_MAXIMUM |
| FLAG_MIN_VOLUME | 1 << 5 | 调节至最小音量(已废弃) | 建议使用ADJUST_TO_MINIMUM |
| FLAG_ONLY_MEDIA_AND_NAVI | 1 << 6 | 仅调节媒体和导航音量 | 特定场景音量控制 |

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| FLAG_POWER_MUTE | 1 << 7 | 仅电源管理可调用的静音 | 系统级静音控制 |

# 音频焦点标志位说明

## 音频焦点标志位常量

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIOFOCUS_FLAG_DELAY_OK | 1 << 0 | 允许延迟获取音频焦点 | 当系统繁忙时可延迟获取焦点 |
| AUDIOFOCUS_FLAG_PAUSES_ON_DUCKABLE_LOSS | 1 << 1 | 在可降低音量时暂停播放 | 避免在降低音量时继续播放 |
| AUDIOFOCUS_FLAG_LOCK | 1 << 2 | 锁定音频焦点 | 防止其他应用抢占焦点 |
| AUDIOFOCUS_FLAG_LAST_MODE | 1 << 3 | 保留上次音频模式 | 恢复焦点时使用上次模式 |
| AUDIOFOCUS_FLAG_NOT_RESTORE | 1 << 4 | 不恢复之前的音频焦点 | 放弃焦点时不恢复之前的焦点状态 |

# 音频参数与模式常量说明

## 音频滤波器类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| FILTER_TYPE_TREBLE | 1 | 高音滤波器 |
| FILTER_TYPE_MIDDLE | 2 | 中音滤波器 |
| FILTER_TYPE_BASS | 3 | 低音滤波器 |

## 音频Duck策略

| 常量 | 值 | 说明 |
| --- | --- | --- |
| AUDIO_DUCKABLE_POLICY_DUCK_IN | 1 | 音频Duck进入策略 |
| AUDIO_DUCKABLE_POLICY_DUCK_OUT | 2 | 音频Duck退出策略 |

## 音频参数常量

| 常量 | 值 | 说明 | 备注 |
| --- | --- | --- | --- |
| PARAM_SURROUND_MODE | 1001 | 环绕声模式 | @hide |
| PARAM_SOUND_FX | 1002 | 音效设置 | @hide |

| 常量 | 值 | 说明 | 备注 |
| --- | --- | --- | --- |
| PARAM_SVC_MODE | 1003 | SVC模式 | @hide |
| MEDIA_STATUS_WHEN_REVERSING | 1004 | 倒车时媒体状态 | @hide |
| BEST_LISTENING_PHONEME | 1005 | 最佳听音位置 | @hide |
| REAR_MUTE | 1006 | 后座静音 | @hide |
| EXTERNAL_AMP | 1007 | 外置功放 | @hide |

### 已废弃常量

| 常量 | 值 | 说明 | 替代方案 |
| --- | --- | --- | --- |
| SVC_CLOSE | 0 | SVC关闭模式 | 已废弃 |
| SVC_NORMAL | 2 | SVC普通模式 | 已废弃 |
| SVC_WEAK | 1 | SVC弱模式 | 已废弃 |
| SVC_STRONG | 3 | SVC强模式 | 已废弃 |
| MEDIA_KEEP | 0 | 倒车保持媒体播放 | 已废弃 |
| MEDIA_MUTE | 1 | 倒车静音媒体 | 已废弃 |

# 音频状态与参数常量说明

## CarPlay Duck相关参数

| 常量 | 类型 | 说明 | 备注 |
| --- | --- | --- | --- |
| DUCK_SOURCE | String | CarPlay Duck源标识 | @hide |
| DUCK_SOURCE_ATTENUATIONVALUE | String | CarPlay Duck衰减值参数 | @hide |
| UNDUCK_SOURCE | String | CarPlay取消Duck源标识 | @hide |

## 音频状态常量

| 常量 | 类型 | 值 | 说明 |
| --- | --- | --- | --- |
| MUTE | boolean | true | 静音状态(麦克风、铃声、收音机) |
| UNMUTE | boolean | false | 非静音状态(麦克风、铃声、收音机) |
| OFF | boolean | false | 关闭状态 |
| ON | boolean | true | 开启状态 |

## 结果字符串常量

| 常量 | 类型 | 值 | 说明 |
| --- | --- | --- | --- |

| 常量 | 类型 | 值 | 说明 |
| --- | --- | --- | --- |
| RESULT_TRUE | String | "true" | 表示操作成功的字符串结果 |
| RESULT_FALSE | String | "false" | 表示操作失败的字符串结果 |
| ERROR_PARAMETER | String | "error_parameter" | 参数错误标识字符串 |

### 音频区域常量

| 常量 | 值 | 说明 |
| --- | --- | --- |
| ZONE_ID_MAIN | 0 | 主音频区域 |
| ZONE_ID_SECOND | 10 | 第二音频区域 |

## CarAudioAttributes 音频用途常量说明

### 音频用途常量定义

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_INVALID | -1 | 无效音频用途 | 错误处理 |
| AUDIO_USAGE_UNKNOWN | 0 | 未知用途 | 默认值 |
| AUDIO_USAGE_USB_MUSIC | 1 | USB音乐播放 | USB设备音乐播放 |
| AUDIO_USAGE_USB_MOVIE | 2 | USB电影播放 | USB设备视频播放 |
| AUDIO_USAGE_RADIO | 3 | 收音机 | 车载收音机播放 |
| AUDIO_USAGE_BT_MUSIC | 4 | 蓝牙音乐(A2DP) | 蓝牙音乐播放 |
| AUDIO_USAGE_CARLIFE_MEDIA | 5 | Carlife媒体 | 百度Carlife媒体播放 |
| AUDIO_USAGE_CARPLAY_MEDIA | 6 | Carplay媒体 | Apple Carplay媒体播放 |
| AUDIO_USAGE_TX_MEDIA | 7 | TX媒体 | 腾讯系媒体播放 |
| AUDIO_USAGE_TX_MEDIA_OTHER | 8 | TX其他媒体 | 腾讯系其他媒体 |
| AUDIO_USAGE_3RD | 9 | 第三方应用 | 第三方应用音频 |
| AUDIO_USAGE_AUTO_NAVI | 10 | 自动地图导航 | 车载导航语音 |
| AUDIO_USAGE_TX_NAVI | 11 | 腾讯导航 | 腾讯地图导航语音 |
| AUDIO_USAGE_CARLIFE_NAVI | 12 | Carlife导航 | 百度Carlife导航语音 |
| AUDIO_USAGE_CARPLAY_NAVI | 13 | Carplay导航 | Apple Carplay导航语音 |
| AUDIO_USAGE_RINGTONE | 14 | 蓝牙来电铃声 | 蓝牙电话铃声 |
| AUDIO_USAGE_BLUETOOTH_CALL | 15 | 蓝牙通话 | 蓝牙电话通话 |
| AUDIO_USAGE_ECALL | 16 | 紧急呼叫 | 车载紧急呼叫系统 |

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_BCALL | 17 | Bcall | 特定品牌车载电话 |
| AUDIO_USAGE_CARPLAY_PHONE_RINGING | 18 | Carplay电话铃声 | Apple Carplay来电铃声 |

## 通话与语音交互类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_CARPLAY_TEL | 19 | Carplay电话 | Apple Carplay通话 |
| AUDIO_USAGE_CARLIFE_TEL | 20 | Carlife电话 | 百度Carlife通话 |
| AUDIO_USAGE_VOIP | 21 | VoIP通话 | 网络语音通话 |
| AUDIO_USAGE_VOIP_RING | 22 | VoIP来电铃声 | 网络电话铃声 |
| AUDIO_USAGE_TX_VR_TTS | 23 | 腾讯VR语音 | 腾讯VR语音交互 |
| AUDIO_USAGE_TX_VR | 24 | 腾讯VR | 腾讯VR场景 |
| AUDIO_USAGE_CARLIFE_VR | 25 | Carlife VR | 百度Carlife VR场景 |
| AUDIO_USAGE_CARPLAY_VR | 26 | Carplay VR | Apple Carplay VR场景 |

## 系统提示音类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_MESSAGE | 27 | 消息提示 | 短信/通知提示音 |
| AUDIO_USAGE_NOTIFICATION | 28 | 通知提示 | 系统通知音 |
| AUDIO_USAGE_LOCKSCREEN | 29 | 锁屏音效 | 锁屏/解锁音效 |
| AUDIO_USAGE_DISCLAIMER | 30 | 免责声明 | 法律声明提示 |
| AUDIO_USAGE_ACC_OFF | 31 | ACC关闭提示 | 车辆熄火提示 |
| AUDIO_USAGE_BOOT_ANIMATION | 32 | 启动动画 | 系统启动音效 |

## 车辆功能类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_RADAR | 33 | 雷达提示 | 倒车雷达提示音 |
| AUDIO_USAGE_READY | 34 | 准备就绪 | 车辆准备就绪提示 |
| AUDIO_USAGE_CLUSTER_BOOT | 35 | 仪表盘启动 | 仪表盘启动音效 |
| AUDIO_USAGE_CLUSTER_SHUTDOWN | 36 | 仪表盘关闭 | 仪表盘关闭音效 |
| AUDIO_USAGE_SYSTEM_SOUND | 37 | 系统音效 | 系统界面操作音 |
| AUDIO_USAGE_DOW_CLUSTER_WARNING | 38 | 仪表盘警告 | 仪表盘警告提示音 |

## 驾驶辅助类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_REVERSE | 39 | 倒车提示 | 倒车场景提示音 |
| AUDIO_USAGE_REVERSE_NOT_MIX | 40 | 倒车独占提示 | 倒车独占音频通道 |
| AUDIO_USAGE_ADAS | 41 | ADAS提示 | 驾驶辅助系统提示 |
| AUDIO_USAGE_ADAS_NOT_MIX | 42 | ADAS独占提示 | 驾驶辅助独占音频通道 |

## 媒体与设备类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_USB1_MUSIC | 43 | USB1音乐 | 第一个USB接口音乐播放 |
| AUDIO_USAGE_USB2_MUSIC | 44 | USB2音乐 | 第二个USB接口音乐播放 |
| AUDIO_USAGE_RADIO_AM | 45 | AM收音机 | AM广播播放 |
| AUDIO_USAGE_RADIO_FM | 46 | FM收音机 | FM广播播放 |
| AUDIO_USAGE_REVERSE_DUCK | 47 | 倒车媒体降音 | 倒车时媒体音量自动降低 |
| AUDIO_USAGE_CHANGBA_MEDIA | 48 | 唱吧媒体 | 车载KTV应用媒体播放 |
| AUDIO_USAGE_PSE_MEDIA | 49 | PSE媒体 | 特定媒体源播放 |
| AUDIO_USAGE_OTA | 50 | OTA升级 | 系统升级时音频播放 |

## 系统功能类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_RESTORE | 51 | 恢复设置 | 系统设置恢复时音频 |
| AUDIO_USAGE_ICC | 52 | 前后放大器 | 前后声场放大器控制 |
| AUDIO_USAGE_ADAS_DUCK | 53 | ADAS降音 | ADAS提示时媒体降音 |
| AUDIO_USAGE_CARPLAY_ENHANCE_VR | 54 | Carplay增强VR | Carplay VR增强模式 |
| AUDIO_USAGE_APA | 55 | APA提示 | 自动泊车系统提示音 |

## 特殊场景类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_TRANSIENT_MEDIA | 56 | 临时媒体源 | 临时媒体播放场景 |
| AUDIO_USAGE_TRANSIENT_ATMOSPHERE | 57 | 座舱氛围音 | 车内氛围音效 |
| AUDIO_USAGE_OCC | 58 | 车外喊话 | 车外语音播报系统 |
| AUDIO_USAGE_OUT_MEDIA | 59 | 媒体外放 | 车外媒体播放 |

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_GAME_PROJECTION | 60 | 游戏投屏 | 车载游戏投屏音频 |

## 安全提示类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_AVAS | 61 | AVAS提示音 | 低速行人警示音 |
| AUDIO_USAGE_NATURE_SOUND | 62 | 自然之声 | 环境模拟音效 |
| AUDIO_USAGE_REAR_VIDEO | 63 | 后排音视频 | 后排娱乐系统音频 |
| AUDIO_USAGE_MEDIA_712 | 64 | 7.1.2声道 | 高端音响系统播放 |
| AUDIO_USAGE_OCC_TTS | 66 | 快捷外放 | 快速车外语音播报 |
| AUDIO_USAGE_AUTO_TTS | 67 | 行车智驾 | 自动驾驶语音提示 |
| AUDIO_USAGE_EXIT_CHANGEBA_MEDIA | 68 | 唱吧退出 | KTV应用退出音效 |

## 智能互联类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_HICAR_MEDIA | 69 | HiCar媒体 | 华为HiCar媒体播放 |
| AUDIO_USAGE_HICAR_NAVI | 70 | HiCar导航 | 华为HiCar导航语音 |
| AUDIO_USAGE_HICAR_TEL | 71 | HiCar电话 | 华为HiCar通话功能 |
| AUDIO_USAGE_HICAR_VR | 72 | HiCar VR | 华为HiCar虚拟现实场景 |
| AUDIO_USAGE_CARLINK_MEDIA | 73 | CarLink媒体 | 车联易媒体播放 |
| AUDIO_USAGE_CARLINK_NAVI | 74 | CarLink导航 | 车联易导航语音 |
| AUDIO_USAGE_CARLINK_TEL | 75 | CarLink电话 | 车联易通话功能 |
| AUDIO_USAGE_CARLINK_VR | 76 | CarLink VR | 车联易虚拟现实场景 |
| AUDIO_USAGE_ANDROIDAUTO_MEDIA | 77 | AndroidAuto媒体 | Google AndroidAuto媒体播放 |
| AUDIO_USAGE_ANDROIDAUTO_NAVI | 78 | AndroidAuto导航 | Google AndroidAuto导航语音 |
| AUDIO_USAGE_ANDROIDAUTO_TEL | 79 | AndroidAuto电话 | Google AndroidAuto通话功能 |
| AUDIO_USAGE_ANDROIDAUTO_VR | 80 | AndroidAuto VR | Google AndroidAuto虚拟现实场景 |

## 广播与媒体类

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_RADIO_DAB | 81 | DAB数字广播 | 数字音频广播播放 |
| AUDIO_USAGE_RADIO_DAB_TTS | 82 | DAB广播TTS | 数字广播文本转语音 |
| AUDIO_USAGE_RADIO_RDS_TTS | 83 | RDS广播TTS | 无线数据系统文本转语音 |

| 常量 | 值 | 说明 | 应用场景 |
| --- | --- | --- | --- |
| AUDIO_USAGE_QQ_MEDIA | 84 | QQ音乐 | 腾讯QQ音乐播放 |
| AUDIO_USAGE_MEDIA_SPEECH | 85 | 媒体语音 | 音频书籍/播客播放 |

# API说明

# CarAudioManager.getInstance() 方法说明

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public static CarAudioManager getInstance(Context context)` |
| 函数作用 | 获取CarAudioManager的单例实例 |
| 参数说明 | `Context context` - 应用上下文对象。 |
| 返回值 | `CarAudioManager` - CarAudioManager的单例实例 |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(this);
```

# 焦点相关

# CarAudioManager.requestAudioFocus() 方法说明

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int requestAudioFocus(CarAudioFocusRequest afr)` |
| 函数作用 | 请求音频焦点控制权 |
| 参数说明 | `CarAudioFocusRequest afr` - 封装音频焦点请求信息的对象，包含音频属性、焦点类型等 |
| 返回值 | `int` - 返回请求结果：<br>`AUDIOFOCUS_REQUEST_GRANTED`(1): 成功获取焦点<br>`AUDIOFOCUS_REQUEST_FAILED`(0): 获取焦点失败 |

## 示例代码

```
// 创建音频属性
CarAudioAttributes attributes = new CarAudioAttributes.Builder()
        .setUsage(CarAudioAttributes.AUDIO_USAGE_MEDIA)
        .build();
```

```
// 构建焦点请求
CarAudioFocusRequest request = new
CarAudioFocusRequest.Builder(CarAudioManager.AUDIOFOCUS_GAIN)
        .setAudioAttributes(attributes)
        .setOnAudioFocusChangeListener(new OnAudioFocusChangeListener() {
            @Override
            public void onAudioFocusChange(int focusChange) {
                // 处理焦点变化事件
            }
        })
        .build();

// 请求音频焦点
int result = carAudioManager.requestAudioFocus(request);

if (result == CarAudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    // 获取焦点成功，开始播放音频
} else {
    // 获取焦点失败处理
}
```

# CarAudioManager.getZoneIdByUserId() 方法说明

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getZoneIdByUserId(int userId)` |
| 函数作用 | 根据用户ID获取对应的音频区域ID |
| 参数说明 | `int userId` - 调用者的用户ID |
| 返回值 | `int` - 返回区域ID：<br>`ZONE_ID_MAIN`(0): 主区域<br>`ZONE_ID_SECOND`(10): 第二区域<br>`Result.NOT_SUPPORT`(-3): 不支持 |

## 示例代码

```
// 获取当前用户ID（示例值）
int currentUserId = UserHandle.myUserId();

// 获取用户对应的音频区域
int zoneId = carAudioManager.getZoneIdByUserId(currentUserId);
```

# CarAudioManager.requestAudioFocusByZoneId() 方法说明

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int requestAudioFocusByZoneId(CarAudioFocusRequest afr, @ZoneIdDef int zoneId)` |
| 函数作用 | 在指定音频区域请求音频焦点控制权 |
| 参数说明 | `CarAudioFocusRequest afr` - 封装音频焦点请求信息的对象<br>`@ZoneIdDef int zoneId` - 目标音频区域ID |
| 返回值 | `int` - 返回请求结果:<br>`AUDIOFOCUS_REQUEST_GRANTED`(1): 成功获取焦点<br>`AUDIOFOCUS_REQUEST_FAILED`(0): 获取焦点失败 |

## 示例代码

```
// 创建音频属性
CarAudioAttributes attributes = new CarAudioAttributes.Builder()
        .setUsage(CarAudioAttributes.AUDIO_USAGE_MEDIA)
        .build();

// 构建焦点请求
CarAudioFocusRequest request = new
CarAudioFocusRequest.Builder(CarAudioManager.AUDIOFOCUS_GAIN)
        .setAudioAttributes(attributes)
        .setOnAudioFocusChangeListener(new OnAudioFocusChangeListener() {
            @Override
            public void onAudioFocusChange(int focusChange) {
                // 处理焦点变化事件
            }
        })
        .build();

// 请求第二区域的音频焦点
int zoneId = CarAudioManager.ZONE_ID_SECOND;
int result = carAudioManager.requestAudioFocusByZoneId(request, zoneId);

if (result == CarAudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    // 获取焦点成功，开始播放音频
    Log.d(TAG, "Audio focus granted in zone: " + zoneId);
} else {
    // 获取焦点失败处理
    Log.w(TAG, "Failed to get audio focus in zone: " + zoneId);
}
```

# CarAudioManager.abandonAudioFocus() 方法说明

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int abandonAudioFocus(CarAudioFocusRequest afr)` |

| 项目 | 说明 |
| --- | --- |
| 函数作用 | 释放音频焦点控制权 |
| 参数说明 | `CarAudioFocusRequest afr` - 之前请求焦点时使用的音频焦点请求对象 |
| 返回值 | `int` - 返回释放结果：<br>`AUDIOFOCUS_REQUEST_GRANTED`(1): 成功释放<br>`AUDIOFOCUS_REQUEST_FAILED`(0): 释放失败 |

## 示例代码

```
// 创建音频属性（与请求焦点时一致）
CarAudioAttributes attributes = new CarAudioAttributes.Builder()
        .setUsage(CarAudioAttributes.AUDIO_USAGE_MEDIA)
        .build();

// 构建焦点请求（与请求焦点时一致）
CarAudioFocusRequest request = new
CarAudioFocusRequest.Builder(CarAudioManager.AUDIOFOCUS_GAIN)
        .setAudioAttributes(attributes)
        .setOnAudioFocusChangeListener(new OnAudioFocusChangeListener() {
            @Override
            public void onAudioFocusChange(int focusChange) {
                // 处理焦点变化事件
            }
        })
        .build();

// 释放音频焦点
int result = carAudioManager.abandonAudioFocus(request);

if (result == CarAudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    Log.d(TAG, "音频焦点释放成功");
} else {
    Log.w(TAG, "音频焦点释放失败");
}
```

# CarAudioManager.abandonAudioFocusByZoneId() 方法说明

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int abandonAudioFocusByZoneId(CarAudioFocusRequest afr, @ZoneIdDef int zoneId)` |
| 函数作用 | 在指定音频区域释放音频焦点控制权 |

| 项目 | 说明 |
| --- | --- |
| 参数说明 | `CarAudioFocusRequest afr` - 之前请求焦点时使用的音频焦点请求对象<br>`@ZoneIdDef int zoneId` - 目标音频区域ID |
| 返回值 | `int` - 返回释放结果:<br>`AUDIOFOCUS_REQUEST_GRANTED`(1): 成功释放<br>`AUDIOFOCUS_REQUEST_FAILED`(0): 释放失败 |

## 示例代码

```
// 创建音频属性（与请求焦点时一致）
CarAudioAttributes attributes = new CarAudioAttributes.Builder()
        .setUsage(CarAudioAttributes.AUDIO_USAGE_MEDIA)
        .build();

// 构建焦点请求（与请求焦点时一致）
CarAudioFocusRequest request = new
CarAudioFocusRequest.Builder(CarAudioManager.AUDIOFOCUS_GAIN)
        .setAudioAttributes(attributes)
        .setOnAudioFocusChangeListener(new OnAudioFocusChangeListener() {
            @Override
            public void onAudioFocusChange(int focusChange) {
                // 处理焦点变化事件
            }
        })
        .build();

// 释放第二区域的音频焦点
int zoneId = CarAudioManager.ZONE_ID_SECOND;
int result = carAudioManager.abandonAudioFocusByZoneId(request, zoneId);

if (result == CarAudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    Log.d(TAG, "音频焦点释放成功，区域ID: " + zoneId);
} else {
    Log.w(TAG, "音频焦点释放失败，区域ID: " + zoneId);
}
```

# CarAudioManager.bindUserIdForSecondStack() 方法说明

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void bindUserIdForSecondStack(int userId)` |
| 函数作用 | 将用户ID绑定到第二音频堆栈(zone) |
| 参数说明 | `int userId` - 要绑定的用户ID ,没有提供unbind函数如果需要解绑可以传入 UserHandle.USER_SYSTEM |

| 项目 | 说明 |
| --- | --- |
| 返回值 | 无 |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 将第二用户ID绑定到第二音频堆栈
int secondUserId = UserHandle.MIN_SECONDARY_USER_ID; // 通常为10
audioManager.bindUserIdForSecondStack(secondUserId);

// 解绑用户ID
audioManager.bindUserIdForSecondStack(UserHandle.USER_SYSTEM);
```

# CarAudioManager 音频源监听方法说明

## registerAudioSourceChangeListener() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void registerAudioSourceChangeListener(OnAudioSourceChangeListener listener)` |
| 函数作用 | 注册音频源变化监听器 |
| 参数说明 | `OnAudioSourceChangeListener listener` - 音频源变化监听器接口 |
| 返回值 | 无 |

## unregisterAudioSourceChangeListener() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void unregisterAudioSourceChangeListener(OnAudioSourceChangeListener listener)` |
| 函数作用 | 注销音频源变化监听器 |
| 参数说明 | `OnAudioSourceChangeListener listener` - 要注销的监听器接口 |
| 返回值 | 无 |

## 示例代码

```
// 创建音频源变化监听器
OnAudioSourceChangeListener sourceListener = new OnAudioSourceChangeListener() {
    @Override
    public void onAudioSourceGrant(int usage, int result) {
        // 处理音频源获取事件
    }

    @Override
    public void onAudioSourceLoss(int usage, int result) {
        // 处理音频源丢失事件
    }
};

// 注册监听器
CarAudioManager.getInstance(context).registerAudioSourceChangeListener(sourceListene
r);

// 使用完毕后注销监听器
CarAudioManager.getInstance(context).unregisterAudioSourceChangeListener(sourceListe
ner);
```

# CarAudioManager 音频焦点策略管理

## getFocusInfoForLastSource() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public CarAudioFocusInfo getFocusInfoForLastSource()` |
| 函数作用 | 获取最后一次音频源的焦点信息 |
| 参数说明 | 无 |
| 返回值 | `CarAudioFocusInfo` - 包含最后一次音频源焦点信息的对象 |

## getFocusInfosForActive() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public CarAudioFocusInfo[] getFocusInfosForActive()` |
| 函数作用 | 获取当前所有活跃音频源的焦点信息列表 |
| 参数说明 | 无 |
| 返回值 | `CarAudioFocusInfo[]` - 活跃音频源焦点信息数组 |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);
```

```
// 获取最后一次音频源焦点信息
CarAudioFocusInfo lastFocusInfo = audioManager.getFocusInfoForLastSource();
int lastUsage = lastFocusInfo.getAttributes().getUsage();
String lastPackage = lastFocusInfo.getPackageName();

// 获取当前所有活跃音频源焦点信息
CarAudioFocusInfo[] activeFocusInfos = audioManager.getFocusInfosForActive();
for (CarAudioFocusInfo info : activeFocusInfos) {
    int usage = info.getAttributes().getUsage();
    String pkg = info.getPackageName();
    // 处理每个活跃音频源信息
}
```

# 音量相关

# CarAudioManager 音量控制方法说明

## setGroupVolume() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setGroupVolume(int groupId, int index, int flags)` |
| 函数作用 | 设置指定音量组的音量值(带标志位) |
| 参数说明 | `groupId` - 音量组ID<br>`index` - 要设置的音量值<br>`flags` - 控制标志(如显示UI) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT/成功状态) |

## setGroupVolume() 简化版方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setGroupVolume(int groupId, int index)` |
| 函数作用 | 设置指定音量组的音量值(不带标志位) |
| 参数说明 | `groupId` - 音量组ID<br>`index` - 要设置的音量值 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT/成功状态) |

## 注意 gropId参数、flags参数在CarAudioManager常量说明文档有介绍

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 设置音量组1的音量为5(不显示UI)
int result = audioManager.setGroupVolume(CarAudioManger.VOLUME_GROUP_ID_MUSIC, 5);
if (result == Result.NOT_SUPPORT) {
    Log.e(TAG, "音量设置不支持");
}

// 设置音量组2的音量为8并显示UI
result = audioManager.setGroupVolume(CarAudioManger.VOLUME_GROUP_ID_MUSIC, 8,
CarAudioManager.FLAG_SHOW_UI);
if (result == Result.NOT_SUPPORT) {
    Log.e(TAG, "音量设置不支持");
}
```

# setGroupVolumeByZoneId() 方法说明

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setGroupVolumeByZoneId(int groupId, int index, int flags, @ZoneIdDef int zoneId)` |
| 函数作用 | 根据区域ID设置指定音量组的音量值 |
| 参数说明 | `groupId` - 音量组ID<br>`index` - 要设置的音量值<br>`flags` - 控制标志(如显示UI)<br>`zoneId` - 区域ID(主区/副区) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT/成功状态) |

## getGroupMaxVolume() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getGroupMaxVolume(int groupId)` |
| 函数作用 | 获取指定音量组的最大音量值 |
| 参数说明 | `groupId` - 要查询的音量组ID |
| 返回值 | `int` - 音量组的最大音量值(Result.NOT_SUPPORT表示不支持) |

## getGroupMinVolume() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getGroupMinVolume(int groupId)` |
| 函数作用 | 获取指定音量组的最小音量值 |

| 项目 | 说明 |
| --- | --- |
| 参数说明 | `groupId` - 要查询的音量组ID |
| 返回值 | `int` - 音量组的最小音量值(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 获取音量组1在主区的最大音量
int maxVol = audioManager.getGroupMaxVolume(CarAudioManger.VOLUME_GROUP_ID_MUSIC);

// 设置主区(zoneId=0)音量组1的音量为最大值并显示UI
int result = audioManager.setGroupVolumeByZoneId(
    CarAudioManger.VOLUME_GROUP_ID_MUSIC,
    maxVol,
    CarAudioManager.FLAG_SHOW_UI,
    CarAudioManager.ZONE_ID_MAIN
);

if (result == Result.NOT_SUPPORT) {
    Log.e(TAG, "音量设置不支持");
} else {
    Log.d(TAG, "音量设置成功");
}
```

## getGroupVolume() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getGroupVolume(int groupId)` |
| 函数作用 | 获取指定音量组的当前音量值 |
| 参数说明 | `groupId` - 要查询的音量组ID |
| 返回值 | `int` - 当前音量值(Result.NOT_SUPPORT表示不支持) |

## getGroupVolumeByZoneId() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getGroupVolumeByZoneId(int groupId, @ZoneIdDef int zoneId)` |
| 函数作用 | 根据区域ID获取指定音量组的当前音量值 |
| 参数说明 | `groupId` - 音量组ID<br>`zoneId` - 区域ID(主区0或副区10) |
| 返回值 | `int` - 当前音量值(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 获取主区(zoneId=0)VOLUME_GROUP_ID_MUSIC的当前音量
int currentVol =
audioManager.getGroupVolumeByZoneId(CarAudioManger.VOLUME_GROUP_ID_MUSIC,
CarAudioManager.ZONE_ID_MAIN);
if (currentVol == Result.NOT_SUPPORT) {
    Log.e(TAG, "音量获取不支持");
} else {
    Log.d(TAG, "主区音量组1当前音量: " + currentVol);
}

// 获取默认区域VOLUME_GROUP_ID_MUSIC的当前音量
int defaultVol = audioManager.getGroupVolume(CarAudioManger.VOLUME_GROUP_ID_MUSIC);
if (defaultVol != Result.NOT_SUPPORT) {
    // 设置音量增加5个单位
    int newVol = Math.min(defaultVol + 5,

audioManager.getGroupMaxVolume(CarAudioManger.VOLUME_GROUP_ID_MUSIC));
    audioManager.setGroupVolume(CarAudioManger.VOLUME_GROUP_ID_MUSIC, newVol);
}
```

## getVolumeGroupCount() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getVolumeGroupCount()` |
| 函数作用 | 获取系统中可用的音量组数量 |
| 参数说明 | 无 |
| 返回值 | `int` - 音量组数量(Result.NOT_SUPPORT表示不支持) |

## getVolumeGroupIdForUsage() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getVolumeGroupIdForUsage(int usage)` |
| 函数作用 | 根据音频用途获取对应的音量组ID |
| 参数说明 | `usage` - 音频用途(如媒体、导航等) |
| 返回值 | `int` - 音量组ID(Result.NOT_SUPPORT表示不支持) |

## getUsagesForVolumeGroupId() 方法

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getUsagesForVolumeGroupId(int groupId)` |
| 函数作用 | 获取指定音量组支持的所有音频用途 |
| 参数说明 | `groupId` - 音量组ID |
| 返回值 | `int[]` - 音频用途数组(null表示不支持) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 示例1：获取音量组数量并遍历
int groupCount = audioManager.getVolumeGroupCount();
if (groupCount != Result.NOT_SUPPORT) {
    for (int i = 0; i < groupCount; i++) {
        int[] usages = audioManager.getUsagesForVolumeGroupId(i);
        if (usages != null) {
            Log.d(TAG, "音量组" + i + "支持的用途: " + Arrays.toString(usages));
        }
    }
}

// 示例2：根据音频用途获取音量组ID
int musicGroupId =
audioManager.getVolumeGroupIdForUsage(AudioAttributes.USAGE_MEDIA);
if (musicGroupId != Result.NOT_SUPPORT) {
    Log.d(TAG, "媒体音频所属音量组ID: " + musicGroupId);

    // 获取该音量组的当前音量
    int currentVol = audioManager.getGroupVolume(musicGroupId);
    Log.d(TAG, "当前媒体音量: " + currentVol);
}
```

## setGroupMute() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setGroupMute(int groupId, boolean mute)` |
| 函数作用 | 设置指定音量组的静音状态 |
| 参数说明 | `groupId` - 音量组ID<br>`mute` - true表示静音，false表示取消静音 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## setGroupMute() 方法(带flags参数)

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setGroupMute(int groupId, boolean mute, int flags)` |
| 函数作用 | 设置指定音量组的静音状态(带控制标志) |
| 参数说明 | `groupId` - 音量组ID<br>`mute` - true表示静音，false表示取消静音<br>`flags` - 控制标志(如FLAG_SHOW_UI) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## isGroupMute() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean isGroupMute(int groupId)` |
| 函数作用 | 检查指定音量组是否处于静音状态 |
| 参数说明 | `groupId` - 音量组ID |
| 返回值 | `boolean` - true表示静音，false表示未静音 |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 设置音量组1静音
int result = audioManager.setGroupMute(CarAudioManger.VOLUME_GROUP_ID_MUSIC, true);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "音量组1静音设置成功");

    // 检查静音状态
    boolean isMuted =
audioManager.isGroupMute(CarAudioManger.VOLUME_GROUP_ID_MUSIC);
    Log.d(TAG, "音量组1当前静音状态: " + isMuted);

    // 带UI标志取消静音
    audioManager.setGroupMute(CarAudioManger.VOLUME_GROUP_ID_MUSIC, false,
CarAudioManager.FLAG_SHOW_UI);
}
```

## adjustVolume() 方法 (两个参数)

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void adjustVolume(int adjustment, int flags)` |
| 函数作用 | 调整当前活跃音量组音量(默认步长) |

| 项目 | 说明 |
| --- | --- |
| 参数说明 | `adjustment` - 调整类型(ADJUST_LOWER/RAISE等)<br>`flags` - 控制标志(FLAG_SHOW_UI等) |
| 返回值 | 无 |

## adjustVolume() 方法 (三个参数)

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void adjustVolume(int adjustment, int flags, int step)` |
| 函数作用 | 调整当前活跃音量组音量(自定义步长) |
| 参数说明 | `adjustment` - 调整类型<br>`flags` - 控制标志<br>`step` - 调整步长(必须>0) |
| 返回值 | 无 |

## 注意 adjustment参数、flags参数在CarAudioManager常量说明文档有介绍

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 简单音量增加(使用默认步长)
audioManager.adjustVolume(CarAudioManager.ADJUST_RAISE,
CarAudioManager.FLAG_PLAY_SOUND);

// 精确音量控制(自定义步长为5)
audioManager.adjustVolume(
    CarAudioManager.ADJUST_LOWER,
    CarAudioManager.FLAG_FROM_KEY,
    5
);

// 静音操作
audioManager.adjustVolume(CarAudioManager.ADJUST_MUTE, 0);
```

## setVolumeGain()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setVolumeGain(float gain)` |
| 函数作用 | 设置音量增益(如根据车速调整音量) |
| 参数说明 | `gain` - 要设置的音量增益值 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

| 项目 | 说明 |
| --- | --- |
| 权限要求 | 仅限内部类使用(@hide) |

## setVolumeAdaptiveOn()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setVolumeAdaptiveOn(boolean on)` |
| 函数作用 | 设置音量自适应开关状态 |
| 参数说明 | `on` - 音量自适应开关状态(true:开启, false:关闭) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

# 示例代码

```
// 设置音量增益为1.5倍
int result = audioManager.setVolumeGain(1.5f);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "音量增益设置成功");
}

// 开启音量自适应功能
int adaptiveResult = audioManager.setVolumeAdaptiveOn(true);
if (adaptiveResult != Result.NOT_SUPPORT) {
    Log.d(TAG, "音量自适应设置成功");
}
```

## isVolumeAdaptiveOn()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean isVolumeAdaptiveOn()` |
| 函数作用 | 获取音量自适应开关状态 |
| 参数说明 | 无 |
| 返回值 | `boolean` - 音量自适应开关状态(true:开启, false:关闭) |

## setVolumeAdaptiveOn()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setVolumeAdaptiveOn(boolean on)` |
| 函数作用 | 设置音量自适应开关状态 |
| 参数说明 | `on` - 音量自适应开关状态(true:开启, false:关闭) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 检查音量自适应是否开启
boolean isAdaptiveOn = audioManager.isVolumeAdaptiveOn();
Log.d(TAG, "音量自适应状态: " + isAdaptiveOn);

// 开启音量自适应功能
int result = audioManager.setVolumeAdaptiveOn(true);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "音量自适应设置成功");
}
```

## setVolumeWithSpeedOpen()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setVolumeWithSpeedOpen(boolean open)` |
| 函数作用 | 设置车速音量补偿功能开关状态 |
| 参数说明 | `open` - 开关状态(true:开启, false:关闭) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 注意事项 | 关闭时会自动重置音量增益为0 |

## isVolumeWithSpeedOpen()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean isVolumeWithSpeedOpen()` |
| 函数作用 | 获取车速音量补偿功能当前状态 |
| 参数说明 | 无 |
| 返回值 | `boolean` - 当前开关状态(true:开启, false:关闭) |

## 示例代码

```
// 检查车速音量补偿功能是否开启
boolean isSpeedVolumeOn = audioManager.isVolumeWithSpeedOpen();
Log.d(TAG, "车速音量补偿状态: " + isSpeedVolumeOn);

// 开启车速音量补偿功能
int result = audioManager.setVolumeWithSpeedOpen(true);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "车速音量补偿设置成功");
}

// 关闭车速音量补偿功能
audioManager.setVolumeWithSpeedOpen(false);
```

registerVolumeChangeListener()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void registerVolumeChangeListener(OnVolumeChangeListener listener)` |
| 函数作用 | 注册全局音量变化监听器 |
| 参数说明 | `listener` - 音量变化监听器实例 |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |
| 注意事项 | 1. 首次注册时会绑定服务端回调<br>2. 支持多监听器注册 |

registerVolumeChangeListenerByZoneId()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void registerVolumeChangeListenerByZoneId(OnVolumeChangeListener listener, @ZoneIdDef int zoneId)` |
| 函数作用 | 按音频区域注册音量变化监听器 |
| 参数说明 | `listener` - 音量变化监听器实例<br>`zoneId` - 音频区域ID(`ZONE_ID_MAIN`或`ZONE_ID_SECOND`) |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |
| 注意事项 | 1. 支持按区域独立监听<br>2. 每个区域维护独立的监听器列表 |

unregisterVolumeChangeListener()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void unregisterVolumeChangeListener(OnVolumeChangeListener listener)` |
| 函数作用 | 注销全局音量变化监听器 |
| 参数说明 | `listener` - 要注销的监听器实例 |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |
| 注意事项 | 1. 当没有监听器时会自动解绑服务端回调<br>2. 线程安全操作 |

unregisterVolumeChangeListenerByZoneId()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void unregisterVolumeChangeListenerByZoneId(OnVolumeChangeListener listener, @ZoneIdDef int zoneId)` |

| 项目 | 说明 |
| --- | --- |
| 函数作用 | 按音频区域注销音量变化监听器 |
| 参数说明 | `listener` - 要注销的监听器实例<br>`zoneId` - 音频区域ID |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |
| 注意事项 | 1. 仅移除指定区域的监听器<br>2. 当区域无监听器时会解绑该区域回调 |

## 示例代码

```
// 创建监听器实例
OnVolumeChangeListener listener = new OnVolumeChangeListener() {
    @Override
    public void onVolumeChanged(int zoneId, int groupId, int flags) {
        Log.d(TAG, "音量变化: zone=" + zoneId + " group=" + groupId);
    }

    @Override
    public void onMuteChanged(int zoneId, int groupId, int flags) {
        Log.d(TAG, "静音状态变化: zone=" + zoneId + " group=" + groupId);
    }
};

// 注册全局监听
audioManager.registerVolumeChangeListener(listener);

// 注册主区域监听
audioManager.registerVolumeChangeListenerByZoneId(listener,
CarAudioManager.ZONE_ID_MAIN);

// 注销监听
audioManager.unregisterVolumeChangeListener(listener);
audioManager.unregisterVolumeChangeListenerByZoneId(listener,
CarAudioManager.ZONE_ID_MAIN);
```

## setMasterMute()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setMasterMute(boolean mute)` |
| 函数作用 | 设置功放静音状态 |
| 参数说明 | `mute` - 静音状态(true:静音, false:取消静音) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

| 项目 | 说明 |
| --- | --- |
| 权限要求 | 需要PERMISSION_CAR_CONTROL_AUDIO_VOLUME权限 |
| 注意事项 | 1. 会立即生效<br>2. 状态变更会保存到系统设置 |

### getMasterMute()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public boolean getMasterMute() |
| 函数作用 | 获取功放当前静音状态 |
| 返回值 | boolean - 当前静音状态(true:静音, false:取消静音) |
| 权限要求 | 需要PERMISSION_CAR_CONTROL_AUDIO_VOLUME权限 |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 获取当前静音状态
boolean isMuted = audioManager.getMasterMute();
Log.d(TAG, "当前功放静音状态: " + isMuted);

// 设置功放静音
int result = audioManager.setMasterMute(true);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "功放静音设置成功");
}

// 取消功放静音
audioManager.setMasterMute(false);
```

## 仪表音量等级控制方法

ClusterVolumeLevel 枚举类

| 常量 | 值 | 说明 |
| --- | --- | --- |
| HIGH | 0 | 高音量等级 |
| MID | 1 | 中音量等级 |
| LOW | 2 | 低音量等级 |

setClusterVolumeLevel()

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setClusterVolumeLevel(int level)` |
| 函数作用 | 设置仪表音量等级 |
| 参数说明 | `level` - 音量等级(使用`ClusterVolumeLevel`枚举值) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 注意事项 | 1. 会立即生效<br>2. 需要检查配置是否支持此功能 |

getClusterVolumeLevel()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getClusterVolumeLevel()` |
| 函数作用 | 获取当前仪表音量等级 |
| 返回值 | `int` - 当前音量等级(ClusterVolumeLevel枚举值) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 设置仪表音量为中等
int result =
audioManager.setClusterVolumeLevel(CarAudioManager.ClusterVolumeLevel.MID);
if (result != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "仪表音量设置成功");
}

// 获取当前仪表音量等级
int currentLevel = audioManager.getClusterVolumeLevel();
String levelStr = "";
switch(currentLevel) {
    case CarAudioManager.ClusterVolumeLevel.HIGH:
        levelStr = "高"; break;
    case CarAudioManager.ClusterVolumeLevel.MID:
        levelStr = "中"; break;
    case CarAudioManager.ClusterVolumeLevel.LOW:
        levelStr = "低"; break;
}
Log.d(TAG, "当前仪表音量等级: " + levelStr);
```

## 音量渐变控制接口

fadeInVolume() 音量渐入

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int fadeInVolume(int groupId, int level, int time)` |
| 函数作用 | 实现音量渐入效果 |
| 参数说明 | `groupId` - 音量组ID<br>`level` - 目标音量大小<br>`time` - 渐入时长(毫秒) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

### fadeOutVolume() 音量渐出

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int fadeOutVolume(int groupId, int level, int time)` |
| 函数作用 | 实现音量渐出效果 |
| 参数说明 | `groupId` - 音量组ID<br>`level` - 目标音量大小<br>`time` - 渐出时长(毫秒) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 音量渐入示例 - 组1在500毫秒内渐入到音量级别15
int result = audioManager.fadeInVolume(1, 15, 500);
if(result >= 0) {
    Log.d(TAG, "音量渐入操作成功");
}

// 音量渐出示例 - 组2在300毫秒内渐出到音量级别5
result = audioManager.fadeOutVolume(2, 5, 300);
if(result >= 0) {
    Log.d(TAG, "音量渐出操作成功");
}

# 声场平衡控制方法说明
# CarAudioManager 声场平衡控制方法说明

## setFadeTowardFront() 方法

| 项目 | 说明 |
|------|------|
| `方法签名` | `public int setFadeTowardFront(int value)` |
| `函数作用` | 设置前后声场平衡值 |
| `参数说明` | `value` - 平衡值(范围从getFadeTowardFrontMin()到
getFadeTowardFrontMax()) |
| `返回值` | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
```

## getFadeTowardFront() 方法

| 项目 | 说明 |
|------|------|
| `方法签名` | `public int getFadeTowardFront()` |
| `函数作用` | 获取当前前后声场平衡值 |
| `参数说明` | 无 |
| `返回值` | `int` - 当前平衡值(Result.NOT_SUPPORT表示不支持) |

## getFadeTowardFrontMax() 方法

| 项目 | 说明 |
|------|------|
| `方法签名` | `public int getFadeTowardFrontMax()` |
| `函数作用` | 获取前后声场平衡最大值 |
| `参数说明` | 无 |
| `返回值` | `int` - 最大平衡值(Result.NOT_SUPPORT表示不支持) |

## getFadeTowardFrontMin() 方法

| 项目 | 说明 |
|------|------|
| `方法签名` | `public int getFadeTowardFrontMin()` |
| `函数作用` | 获取前后声场平衡最小值 |
| `参数说明` | 无 |
| `返回值` | `int` - 最小平衡值(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```java
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 获取声场平衡范围
int minFade = audioManager.getFadeTowardFrontMin();
int maxFade = audioManager.getFadeTowardFrontMax();
Log.d(TAG, "声场平衡范围: " + minFade + " 到 " + maxFade);

// 获取当前声场平衡值
int currentFade = audioManager.getFadeTowardFront();
Log.d(TAG, "当前声场平衡值: " + currentFade);

// 设置声场完全向前偏移
int result = audioManager.setFadeTowardFront(maxFade);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "声场设置成功");
}

// 重置为平衡状态
audioManager.setFadeTowardFront(0);
```

# CarAudioManager 左右声道平衡控制方法说明

## setBalanceTowardRight() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setBalanceTowardRight(int value)` |
| 函数作用 | 设置左右声道平衡值 |
| 参数说明 | `value` - 平衡值(范围从getBalanceTowardRightMin()到getBalanceTowardRightMax()) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## getBalanceTowardRight() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getBalanceTowardRight()` |
| 函数作用 | 获取当前左右声道平衡值 |
| 参数说明 | 无 |
| 返回值 | `int` - 当前平衡值(Result.NOT_SUPPORT表示不支持) |

## getBalanceTowardRightMax() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getBalanceTowardRightMax()` |
| 函数作用 | 获取左右声道平衡最大值 |
| 参数说明 | 无 |
| 返回值 | `int` - 最大平衡值(Result.NOT_SUPPORT表示不支持) |

## getBalanceTowardRightMin() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getBalanceTowardRightMin()` |
| 函数作用 | 获取左右声道平衡最小值 |
| 参数说明 | 无 |
| 返回值 | `int` - 最小平衡值(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);
```

```
// 获取声道平衡范围
int minBalance = audioManager.getBalanceTowardRightMin();
int maxBalance = audioManager.getBalanceTowardRightMax();
Log.d(TAG, "声道平衡范围: " + minBalance + " 到 " + maxBalance);

// 获取当前声道平衡值
int currentBalance = audioManager.getBalanceTowardRight();
Log.d(TAG, "当前声道平衡值: " + currentBalance);

// 设置声道向右偏移50%
int targetValue = (int)(maxBalance * 0.5);
int result = audioManager.setBalanceTowardRight(targetValue);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "声道平衡设置成功");
}

// 重置为平衡状态
audioManager.setBalanceTowardRight(0);
```

# CarAudioManager 音频规避策略方法说明

# CarAudioManager EQ方法说明

## setPresetEqualizer() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setPresetEqualizer(int type)` |
| 函数作用 | 设置预设均衡器类型 |
| 参数说明 | `type` - 均衡器类型 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 设置均衡器示例
// 设置预设均衡器为"流行"模式
int result = mCarAudioManager.setPresetEqualizer(
    CarAudioManager.EQUALIZER_PRESET_POP
);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "均衡器设置成功");
}

// 获取当前均衡器设置
```

```
int currentEq = mCarAudioManager.getPresetEqualizer();
Log.d(TAG, "当前均衡器模式: " + currentEq);
```

# CarAudioManager 音效相关方法说明

## setDuckAudioPolicy() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void setDuckAudioPolicy(int audioUsage, int policy, float volume, int durationMs)` |
| 函数作用 | 设置音频闪避策略 |
| 参数说明 | `audioUsage` - 音频用途类型<br>`policy` - 闪避策略(DUCK_IN/DUCK_OUT)<br>`volume` - 最终衰减分贝值<br>`durationMs` - 淡入淡出持续时间(毫秒) |
| 返回值 | 无 |

## setPresetEqualizer() 方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setPresetEqualizer(int type)` |
| 函数作用 | 设置预设均衡器类型 |
| 参数说明 | `type` - 均衡器类型 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

|

## 示例代码

```
// 设置音频闪避策略示例
// 对导航音频设置闪入策略，最终衰减到-10dB，过渡时间500ms
mCarAudioManager.setDuckAudioPolicy(
    AudioAttributes.USAGE_ASSISTANCE_NAVIGATION_GUIDANCE,
    CarAudioManager.AUDIO_DUCKABLE_POLICY_DUCK_IN,
    -10.0f,
    500
);

// 设置均衡器示例
// 设置预设均衡器为"流行"模式
int result = mCarAudioManager.setPresetEqualizer(
    CarAudioManager.EQUALIZER_PRESET_POP
);
```

```
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "均衡器设置成功");
}

// 获取当前均衡器设置
int currentEq = mCarAudioManager.getPresetEqualizer();
Log.d(TAG, "当前均衡器模式: " + currentEq);
```

### getPresetEqualizer()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getPresetEqualizer()` |
| 函数作用 | 获取当前预设均衡器类型 |
| 参数说明 | 无 |
| 返回值 | `int` - 当前均衡器类型值(Result.NOT_SUPPORT表示不支持) |

#### 示例代码

```
// 获取当前均衡器设置
int currentEq = audioManager.getPresetEqualizer();
if (currentEq == Result.NOT_SUPPORT) {
    Log.e(TAG, "获取均衡器不支持");
} else {
    Log.d(TAG, "当前均衡器模式: " + currentEq);
}
```

## CarAudioManager 均衡器相关接口文档

### getPresetEqualizerCount()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getPresetEqualizerCount()` |
| 函数作用 | 获取系统支持的均衡器预设类型数量 |
| 参数说明 | 无 |
| 返回值 | `int` - 支持的均衡器类型总数(Result.NOT_SUPPORT表示不支持) |

#### 示例代码

```
// 获取均衡器类型数量
int eqCount = audioManager.getPresetEqualizerCount();
if (eqCount == Result.NOT_SUPPORT) {
    Log.e(TAG, "获取均衡器数量不支持");
```

```
} else {
    Log.d(TAG, "系统支持" + eqCount + "种均衡器模式");
}
```

## setEqualizerSubbandLevel()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setEqualizerSubbandLevel(int band, int level)` |
| 函数作用 | 设置均衡器指定子频段的增益级别 |
| 参数说明 | `band` - 要设置的子频段索引(从0开始)<br>`level` - 要设置的增益级别 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT/成功状态) |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 设置第3个子频段的增益为5
int result = audioManager.setEqualizerSubbandLevel(2, 5);
if (result == Result.NOT_SUPPORT) {
    Log.e(TAG, "均衡器子频段设置不支持");
} else {
    Log.d(TAG, "子频段增益设置成功");
}
```

## getEqualizerSubbandLevel()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getEqualizerSubbandLevel(int band)` |
| 函数作用 | 获取均衡器指定子频段的当前增益级别 |
| 参数说明 | `band` - 要查询的子频段索引(从0开始) |
| 返回值 | `int` - 当前增益级别(Result.NOT_SUPPORT表示不支持) |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 获取第3个子频段的当前增益
int level = audioManager.getEqualizerSubbandLevel(2);
if (level == Result.NOT_SUPPORT) {
    Log.e(TAG, "均衡器子频段获取不支持");
} else {
    Log.d(TAG, "第3个子频段当前增益: " + level);
}
```

## getEqualizerSubbandCount()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getEqualizerSubbandCount()` |
| 函数作用 | 获取均衡器支持的子频段总数 |
| 参数说明 | 无 |
| 返回值 | `int` - 子频段数量(Result.NOT_SUPPORT表示不支持) |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 获取均衡器支持的频段数量
int bandCount = audioManager.getEqualizerSubbandCount();
if (bandCount == Result.NOT_SUPPORT) {
    Log.e(TAG, "均衡器功能不支持");
} else {
    Log.d(TAG, "均衡器支持" + bandCount + "个子频段");
}
```

## getEqualizerSubbandLevelMax()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getEqualizerSubbandLevelMax(int band)` |
| 函数作用 | 获取均衡器指定子频段的最大增益值 |
| 参数说明 | `band` - 要查询的子频段索引(从0开始) |
| 返回值 | `int` - 最大增益值(Result.NOT_SUPPORT表示不支持) |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 获取第2个子频段的最大增益值
int maxLevel = audioManager.getEqualizerSubbandLevelMax(1);
if (maxLevel == Result.NOT_SUPPORT) {
    Log.e(TAG, "获取子频段最大增益不支持");
} else {
    Log.d(TAG, "第2个子频段最大增益: " + maxLevel);
}
```

## getEqualizerSubbandLevelMin()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getEqualizerSubbandLevelMin(int band)` |
| 函数作用 | 获取均衡器指定子频段的最小增益值 |
| 参数说明 | `band` - 要查询的子频段索引(从0开始) |
| 返回值 | `int` - 最小增益值(Result.NOT_SUPPORT表示不支持) |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 获取第2个子频段的最小增益值
int maxLevel = audioManager.getEqualizerSubbandLevelMin(1);
if (maxLevel == Result.NOT_SUPPORT) {
    Log.e(TAG, "获取子频段最大增益不支持");
} else {
    Log.d(TAG, "第2个子频段最大增益: " + maxLevel);
}
```

## getEqualizerSubbandFreqMax()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getEqualizerSubbandFreqMax(int band)` |
| 函数作用 | 获取均衡器指定子频段的最大频率值 |
| 参数说明 | `band` - 要查询的子频段索引(从0开始) |
| 返回值 | `int` - 最大频率值(Hz)，Result.NOT_SUPPORT表示不支持 |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 获取第2个子频段的最大频率
int maxFreq = audioManager.getEqualizerSubbandFreqMax(1);
if (maxFreq == Result.NOT_SUPPORT) {
    Log.e(TAG, "获取子频段最大频率不支持");
} else {
    Log.d(TAG, "第2个子频段最大频率: " + maxFreq + "Hz");
}
```

## getEqualizerSubbandFreqMin()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getEqualizerSubbandFreqMin(int band)` |

| 项目 | 说明 |
| --- | --- |
| 函数作用 | 获取均衡器指定子频段的最小频率值 |
| 参数说明 | band - 要查询的子频段索引(从0开始) |
| 返回值 | int - 最小频率值(Hz)，Result.NOT_SUPPORT表示不支持 |
| 权限要求 | 需要PERMISSION_CAR_CONTROL_AUDIO_VOLUME权限 |

### 示例代码

```
// 获取第2个子频段的最小频率
int minFreq = audioManager.getEqualizerSubbandFreqMin(1);
if (minFreq == Result.NOT_SUPPORT) {
    Log.e(TAG, "获取子频段最小频率不支持");
} else {
    Log.d(TAG, "第2个子频段最小频率: " + minFreq + "Hz");
}
```

## getEqualizerSubbandCenterFreq()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public int getEqualizerSubbandCenterFreq(int band) |
| 函数作用 | 获取均衡器指定子频段的中心频率值 |
| 参数说明 | band - 要查询的子频段索引(从0开始) |
| 返回值 | int - 中心频率值(Hz)，Result.NOT_SUPPORT表示不支持 |
| 权限要求 | 需要PERMISSION_CAR_CONTROL_AUDIO_VOLUME权限 |

### 示例代码

```
// 获取第3个子频段的中心频率
int centerFreq = audioManager.getEqualizerSubbandCenterFreq(2);
if (centerFreq == Result.NOT_SUPPORT) {
    Log.e(TAG, "获取子频段中心频率不支持");
} else {
    Log.d(TAG, "第3个子频段中心频率: " + centerFreq + "Hz");
}
```

# CarAudioManager BMT音效控制接口文档

## setBmtLevel()

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setBmtLevel(int filter, int level)` |
| 函数作用 | 设置BMT音效(低音/中音/高音)的增益级别 |
| 参数说明 | `filter` - 音效类型(参见CarAudioConstant中的常量定义)<br>`level` - 增益级别(-10~10) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 设置低音增益为5
int result = audioManager.setBmtLevel(
    CarAudioConstant.TONE_BOOST_TYPE_TREBLE,
    5
);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "BMT音效设置成功");
}
```

## getBmtLevel()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getBmtLevel(int filter)` |
| 函数作用 | 获取指定BMT音效的当前增益级别 |
| 参数说明 | `filter` - 音效类型(参见CarAudioConstant中的常量定义) |
| 返回值 | `int` - 当前增益级别(-10~10)，Result.NOT_SUPPORT表示不支持 |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 获取当前高音增益
int trebleLevel = audioManager.getBmtLevel(
    CarAudioConstant.TONE_BOOST_TYPE_TREBLE
);
if (trebleLevel != Result.NOT_SUPPORT) {
    Log.d(TAG, "当前高音增益: " + trebleLevel);
}
```

## getBmtLevelMax()

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getBmtLevelMax(int filter)` |
| 函数作用 | 获取指定BMT音效的最大增益级别 |
| 参数说明 | `filter` - 音效类型(参见CarAudioConstant中的常量定义) |
| 返回值 | `int` - 最大增益级别(通常为10)，Result.NOT_SUPPORT表示不支持 |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

## getBmtLevelMin()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getBmtLevelMix(int filter)` |
| 函数作用 | 获取指定BMT音效的最小增益级别 |
| 参数说明 | `filter` - 音效类型(参见CarAudioConstant中的常量定义) |
| 返回值 | `int` - 最大增益级别(通常为10)，Result.NOT_SUPPORT表示不支持 |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_VOLUME`权限 |

### 示例代码

```
// 获取低音最大增益
int bassMax = audioManager.getBmtLevelMax(
    CarAudioConstant.TONE_BOOST_TYPE_TREBLE
);
if (bassMax != Result.NOT_SUPPORT) {
    Log.d(TAG, "低音最大增益: " + bassMax);
}
```

## setParameter()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setParameter(String key, int value)` |
| 函数作用 | 设置单个音频参数 |
| 参数说明 | `key` - 参数键名<br>`value` - 参数值 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## setParameters()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setParameters(String keyValuePairs)` |

| 项目 | 说明 |
| --- | --- |
| 函数作用 | 批量设置音频参数 |
| 参数说明 | `keyValuePairs` - 参数键值对列表，格式为key1=value1;key2=value2;... |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## getParameters()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public String getParameters(String keys)` |
| 函数作用 | 从音频硬件获取多个参数值 |
| 参数说明 | `keys` - 要查询的参数键名列表 |
| 返回值 | `String` - 参数键值对列表，格式为key1=value1;key2=value2;... |
| 权限要求 | 无特殊权限要求 |

```
# 3D环绕音
## set3DSurroundMode()

| 项目 | 说明 |
|------|------|
| `方法签名` | `public int set3DSurroundMode(int mode)` |
| `函数作用` | 设置3D环绕音效模式(用于内置功放) |
| `参数说明` | `mode` - 环绕模式:<br>`VALUE_MODE_OFF`(0):关闭
<br>`VALUE_MODE_DRIVER`(1):驾驶员模式<br>`VALUE_MODE_ALL_PASSENGERS`(2):全乘客模式 |
| `返回值` | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## get3DSurroundMode()

| 项目 | 说明 |
|------|------|
| `方法签名` | `public int get3DSurroundMode()` |
| `函数作用` | 获取当前3D环绕音效模式 |
| `参数说明` | 无 |
| `返回值` | `int` - 当前环绕模式:<br>`VALUE_MODE_OFF`(0):关闭
<br>`VALUE_MODE_DRIVER`(1):驾驶员模式<br>`VALUE_MODE_ALL_PASSENGERS`(2):全乘客模式
<br>Result.NOT_SUPPORT表示不支持 |

## 示例代码

```java
// 设置3D环绕音效为全乘客模式
int result =
audioManager.set3DSurroundMode(CarAudioManager.VALUE_MODE_ALL_PASSENGERS);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "3D环绕音效设置成功");
}

// 获取当前3D环绕音效模式
```
```

```
int currentMode = audioManager.get3DSurroundMode();
if (currentMode != Result.NOT_SUPPORT) {
    String modeName = "";
    switch(currentMode) {
        case CarAudioManager.VALUE_MODE_OFF:
            modeName = "关闭";
            break;
        case CarAudioManager.VALUE_MODE_DRIVER:
            modeName = "驾驶员模式";
            break;
        case CarAudioManager.VALUE_MODE_ALL_PASSENGERS:
            modeName = "全乘客模式";
            break;
    }
    Log.d(TAG, "当前3D环绕音效模式: " + modeName);
}
```

# CarAudioManager Trd音效

## setTrdSoundEQMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setTrdSoundEQMode(int mode)` |
| 函数作用 | 设置第三方音效模式(已废弃) |
| 参数说明 | `mode` - 音效模式:<br>`VALUE_MODE_NATURE`(0):自然<br>`VALUE_MODE_BRIGHT_VOICE`(1):明亮人声<br>`VALUE_MODE_OVER_WEIGHT`(2):超重低音<br>`VALUE_MODE_REST`(3):休息模式 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 废弃说明 | 该方法已标记为@Deprecated |

## getTrdSoundEQMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getTrdSoundEQMode()` |
| 函数作用 | 获取当前第三方音效模式(已废弃) |
| 参数说明 | 无 |

| 项目 | 说明 |
| --- | --- |
| 返回值 | `int` - 当前音效模式:<br>`VALUE_MODE_NATURE`(0):自然<br>`VALUE_MODE_BRIGHT_VOICE`(1):明亮人声<br>`VALUE_MODE_OVER_WEIGHT`(2):超重低音<br>`VALUE_MODE_REST`(3):休息模式<br>Result.NOT_SUPPORT表示不支持 |
| 废弃说明 | 该方法已标记为@Deprecated |

## 示例代码

```
// 设置第三方音效为超重低音模式
int result = audioManager.setTrdSoundEQMode(CarAudioManager.VALUE_MODE_OVER_WEIGHT);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "第三方音效设置成功");
}

// 获取当前第三方音效模式
int currentMode = audioManager.getTrdSoundEQMode();
if (currentMode != Result.NOT_SUPPORT) {
    String modeName = "";
    switch(currentMode) {
        case CarAudioManager.VALUE_MODE_NATURE:
            modeName = "自然模式";
            break;
        case CarAudioManager.VALUE_MODE_BRIGHT_VOICE:
            modeName = "明亮人声";
            break;
        case CarAudioManager.VALUE_MODE_OVER_WEIGHT:
            modeName = "超重低音";
            break;
        case CarAudioManager.VALUE_MODE_REST:
            modeName = "休息模式";
            break;
    }
    Log.d(TAG, "当前第三方音效模式: " + modeName);
}
```

# CarAudioManager - 最佳听音位置设置功能

## setBestListeningPhoneme()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setBestListeningPhoneme(int mode)` |
| 函数作用 | 设置最佳听音位置(用于外置功放) |

| 项目 | 说明 |
| --- | --- |
| 参数说明 | `mode` - 听音位置模式:<br>`BEST_LISTENING_PHONEME_DRIVER`(0):驾驶员模式<br>`BEST_LISTENING_PHONEME_ALL`(1):全车模式<br>`BEST_LISTENING_PHONEME_VIP`(2):VIP模式 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 废弃说明 | 该方法已标记为@Deprecated |

## getBestListeningPhoneme()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getBestListeningPhoneme()` |
| 函数作用 | 获取当前最佳听音位置模式(用于内置功放) |
| 参数说明 | 无 |
| 返回值 | `int` - 当前听音位置模式:<br>`BEST_LISTENING_PHONEME_DRIVER`(0):驾驶员模式<br>`BEST_LISTENING_PHONEME_ALL`(1):全车模式<br>`BEST_LISTENING_PHONEME_VIP`(2):VIP模式<br>Result.NOT_SUPPORT表示不支持 |
| 废弃说明 | 该方法已标记为@Deprecated |

## 示例代码

```
// 设置最佳听音位置为VIP模式
int result =
audioManager.setBestListeningPhoneme(CarAudioManager.BEST_LISTENING_PHONEME_VIP);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "最佳听音位置设置成功");
}

// 获取当前最佳听音位置模式
int currentMode = audioManager.getBestListeningPhoneme();
if (currentMode != Result.NOT_SUPPORT) {
    String modeName = "";
    switch(currentMode) {
        case CarAudioManager.BEST_LISTENING_PHONEME_DRIVER:
            modeName = "驾驶员模式";
            break;
        case CarAudioManager.BEST_LISTENING_PHONEME_ALL:
            modeName = "全车模式";
            break;
        case CarAudioManager.BEST_LISTENING_PHONEME_VIP:
            modeName = "VIP模式";
            break;
    }
```

```
    Log.d(TAG, "当前最佳听音位置模式: " + modeName);
}
```

# CarAudioManager - 倒车媒体状态控制功能

## setMediaStatusWhenReversing()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setMediaStatusWhenReversing(int mode)` |
| 函数作用 | 设置倒车时的媒体状态 |
| 参数说明 | `mode` - 媒体状态模式:<br>`MEDIA_STATUS_WHEN_REVERSING_KEEP`(0):保持原状<br>`MEDIA_STATUS_WHEN_REVERSING_MUTE`(1):静音<br>`MEDIA_STATUS_WHEN_REVERSING_DUCK`(2):降低音量 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## getMediaStatusWhenReversing()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getMediaStatusWhenReversing()` |
| 函数作用 | 获取当前倒车时的媒体状态 |
| 参数说明 | 无 |
| 返回值 | `int` - 当前媒体状态模式:<br>`MEDIA_STATUS_WHEN_REVERSING_KEEP`(0):保持原状<br>`MEDIA_STATUS_WHEN_REVERSING_MUTE`(1):静音<br>`MEDIA_STATUS_WHEN_REVERSING_DUCK`(2):降低音量<br>Result.NOT_SUPPORT表示不支持 |

## 示例代码

```
// 设置倒车时媒体状态为降低音量
int result =
audioManager.setMediaStatusWhenReversing(CarAudioManager.MEDIA_STATUS_WHEN_REVERSING
_DUCK);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "倒车媒体状态设置成功");
}

// 获取当前倒车媒体状态
int currentMode = audioManager.getMediaStatusWhenReversing();
if (currentMode != Result.NOT_SUPPORT) {
    String modeName = "";
    switch(currentMode) {
```

```
            case CarAudioManager.MEDIA_STATUS_WHEN_REVERSING_KEEP:
                modeName = "保持原状";
                break;
            case CarAudioManager.MEDIA_STATUS_WHEN_REVERSING_MUTE:
                modeName = "静音";
                break;
            case CarAudioManager.MEDIA_STATUS_WHEN_REVERSING_DUCK:
                modeName = "降低音量";
                break;
        }
        Log.d(TAG, "当前倒车媒体状态: " + modeName);
    }
```

## setDtsEnabled()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setDtsEnabled(boolean enabled)` |
| 函数作用 | 设置DTS音效开关状态 |
| 参数说明 | `enabled` - 开关状态(true:开启, false:关闭) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 注意事项 | 1. 需要检查配置是否支持DTS功能<br>2. 状态变更会触发监听回调 |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_EFFECT`权限 |

## getDtsEnabled()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean getDtsEnabled()` |
| 函数作用 | 获取DTS音效当前状态 |
| 参数说明 | 无 |
| 返回值 | `boolean` - 当前开关状态(true:开启, false:关闭) |
| 权限要求 | 需要`PERMISSION_CAR_CONTROL_AUDIO_EFFECT`权限 |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

    // 获取当前状态
    boolean isDtsEnabled = audioManager.getDtsEnabled();
    Log.d(TAG, "当前DTS状态: " + isDtsEnabled);

    // 开启DTS音效
```

```
int result = audioManager.setDtsEnabled(true);
if (result != Result.NOT_SUPPORT) {
    Log.d(TAG, "DTS音效开启成功");
}

// 关闭DTS音效
audioManager.setDtsEnabled(false);
```

## 模拟引擎音效控制方法

### setEngineSoundEffectEnable()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setEngineSoundEffectEnable(boolean enable)` |
| 函数作用 | 设置模拟引擎音效开关状态 |
| 参数说明 | `enable` - 音效开关状态(true:开启, false:关闭) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 注意事项 | 1. 会立即生效<br>2. 需要检查配置是否支持此功能 |

### getEngineSoundEffectEnable()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean getEngineSoundEffectEnable()` |
| 函数作用 | 获取模拟引擎音效当前状态 |
| 返回值 | `boolean` - 当前音效开关状态(true:开启, false:关闭) |

## 主动降噪控制方法

### setANCEnable()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setANCEnable(boolean enable)` |
| 函数作用 | 设置整车主动降噪开关状态 |
| 参数说明 | `enable` - 降噪开关状态(true:开启, false:关闭) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 注意事项 | 1. 会立即生效<br>2. 需要检查配置是否支持此功能 |

### getANCEnable()

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean getANCEnable()` |
| 函数作用 | 获取整车主动降噪当前状态 |
| 返回值 | `boolean` - 当前降噪开关状态(true:开启, false:关闭) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 模拟引擎音效控制
int engineResult = audioManager.setEngineSoundEffectEnable(true);
if (engineResult != CarAudioManager.Result.NOT_SUPPORT) {
    boolean isEngineEffectOn = audioManager.getEngineSoundEffectEnable();
    Log.d(TAG, "模拟引擎音效状态: " + isEngineEffectOn);
}

// 主动降噪控制
int ancResult = audioManager.setANCEnable(true);
if (ancResult != CarAudioManager.Result.NOT_SUPPORT) {
    boolean isANCOn = audioManager.getANCEnable();
    Log.d(TAG, "主动降噪状态: " + isANCOn);
}
```

## 前排扩音模式控制方法

### ICCMode 枚举类

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭扩音模式 |
| `LOW` | 1 | 低档扩音模式 |
| `MID` | 2 | 中档扩音模式 |
| `HIGH` | 3 | 高档扩音模式 |

### setICCMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setICCMode(int mode)` |
| 函数作用 | 设置前排扩音模式 |
| 参数说明 | `mode` - 扩音模式(使用`ICCMode`枚举值) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

| 项目 | 说明 |
| --- | --- |
| 注意事项 | 1. 会立即生效<br>2. 需要检查配置是否支持此功能 |

getICCMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getICCMode()` |
| 函数作用 | 获取当前前排扩音模式 |
| 返回值 | `int` - 当前扩音模式(ICCMode枚举值) |

## 已废弃方法

```
@Deprecated
public boolean getICCEnable() // 已废弃，请使用getICCMode()

@Deprecated
public int setICCEnable(boolean enable) // 已废弃，请使用setICCMode()
```

## 头枕模式控制方法

HeadrestMode 枚举类

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `NONE` | -1 | 无头枕模式 |
| `OFF` | 0 | 关闭头枕模式 |
| `ONLY_HEAD` | 1 | 仅头枕扬声器工作 |
| `ALL_SPEAK` | 2 | 所有扬声器工作 |

setHeadrestMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setHeadrestMode(int mode)` |
| 函数作用 | 设置头枕模式状态 |
| 参数说明 | `mode` - 头枕模式(使用`HeadrestMode`枚举值) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 注意事项 | 1. 会立即生效<br>2. 需要检查配置是否支持此功能 |

getHeadrestMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getHeadrestMode()` |
| 函数作用 | 获取当前头枕模式 |
| 返回值 | `int` - 当前头枕模式(HeadrestMode枚举值) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 设置头枕模式为仅头枕扬声器工作
int result = audioManager.setHeadrestMode(CarAudioManager.HeadrestMode.ONLY_HEAD);
if (result != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "头枕模式设置成功");
}

// 获取当前头枕模式
int currentMode = audioManager.getHeadrestMode();
String modeStr = "";
switch(currentMode) {
    case CarAudioManager.HeadrestMode.NONE:
        modeStr = "无模式"; break;
    case CarAudioManager.HeadrestMode.OFF:
        modeStr = "关闭"; break;
    case CarAudioManager.HeadrestMode.ONLY_HEAD:
        modeStr = "仅头枕"; break;
    case CarAudioManager.HeadrestMode.ALL_SPEAK:
        modeStr = "全部扬声器"; break;
}
Log.d(TAG, "当前头枕模式: " + modeStr);
```

## 外置功放音效模式控制方法 (已废弃)

### ExtSoundMode 枚举类

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭外置功放音效 |
| `YAMAHA` | 1 | Yamaha音效模式 |
| `ADIGO` | 2 | Adigo音效模式 |

### setExtSoundMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setExtSoundMode(int mode)` |
| 函数作用 | 设置外置功放音效模式 |

| 项目 | 说明 |
| --- | --- |
| 参数说明 | `mode` - 音效模式(使用`ExtSoundMode`枚举值) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 状态 | 已废弃(@Deprecated) |

getExtSoundMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getExtSoundMode()` |
| 函数作用 | 获取当前外置功放音效模式 |
| 返回值 | `int` - 当前音效模式(ExtSoundMode枚举值) |
| 状态 | 已废弃(@Deprecated) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 设置外置功放音效模式为Yamaha
int result = audioManager.setExtSoundMode(CarAudioManager.ExtSoundMode.YAMAHA);
if (result != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "外置功放音效模式设置成功");
}

// 获取当前外置功放音效模式
int currentMode = audioManager.getExtSoundMode();
String modeStr = "";
switch(currentMode) {
    case CarAudioManager.ExtSoundMode.OFF:
        modeStr = "关闭"; break;
    case CarAudioManager.ExtSoundMode.YAMAHA:
        modeStr = "Yamaha模式"; break;
    case CarAudioManager.ExtSoundMode.ADIGO:
        modeStr = "Adigo模式"; break;
}
Log.d(TAG, "当前外置功放音效模式: " + modeStr);
```

## Adigo虚拟场景音效模式控制方法 (已废弃)

AdigoVirtualLiveMode 枚举类

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭虚拟场景音效 |
| `XINGHAI_LIVE` | 1 | 星海现场模式 |

| 常量 | 值 | 说明 |
| --- | --- | --- |
| CONCERT | 2 | 音乐会模式 |

setAdigoVirtualLiveMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public int setAdigoVirtualLiveMode(int mode) |
| 函数作用 | 设置Adigo虚拟场景音效模式 |
| 参数说明 | mode - 音效模式(使用AdigoVirtualLiveMode枚举值) |
| 返回值 | int - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 状态 | 已废弃(@Deprecated) |

getAdigoVirtualLiveMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public int getAdigoVirtualLiveMode() |
| 函数作用 | 获取当前Adigo虚拟场景音效模式 |
| 返回值 | int - 当前音效模式(AdigoVirtualLiveMode枚举值) |
| 状态 | 已废弃(@Deprecated) |

# 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 设置Adigo虚拟场景音效模式为音乐会模式
int result =
audioManager.setAdigoVirtualLiveMode(CarAudioManager.AdigoVirtualLiveMode.CONCERT);
if (result != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "Adigo虚拟场景音效模式设置成功");
}

// 获取当前Adigo虚拟场景音效模式
int currentMode = audioManager.getAdigoVirtualLiveMode();
String modeStr = "";
switch(currentMode) {
    case CarAudioManager.AdigoVirtualLiveMode.OFF:
        modeStr = "关闭"; break;
    case CarAudioManager.AdigoVirtualLiveMode.XINGHAI_LIVE:
        modeStr = "星海现场"; break;
    case CarAudioManager.AdigoVirtualLiveMode.CONCERT:
        modeStr = "音乐会"; break;
}
Log.d(TAG, "当前Adigo虚拟场景音效模式: " + modeStr);
```

# Yamaha音效模式控制方法 (已废弃)

## YamahaSoundMode 枚举类

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭Yamaha音效 |
| `SPRING` | 1 | 春季音效模式 |
| `SUMMER` | 2 | 夏季音效模式 |
| `AUTUMN` | 3 | 秋季音效模式 |
| `WINTER` | 4 | 冬季音效模式 |

## setYamahaSoundMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setYamahaSoundMode(int mode)` |
| 函数作用 | 设置Yamaha音效模式 |
| 参数说明 | `mode` - 音效模式(使用`YamahaSoundMode`枚举值) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 状态 | 已废弃(@Deprecated) |

## getYamahaSoundMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getYamahaSoundMode()` |
| 函数作用 | 获取当前Yamaha音效模式 |
| 返回值 | `int` - 当前音效模式(YamahaSoundMode枚举值) |
| 状态 | 已废弃(@Deprecated) |

# 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 设置Yamaha音效模式为冬季模式
int result =
audioManager.setYamahaSoundMode(CarAudioManager.YamahaSoundMode.WINTER);
if (result != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "Yamaha音效模式设置成功");
}

// 获取当前Yamaha音效模式
int currentMode = audioManager.getYamahaSoundMode();
```

```
String modeStr = "";
switch(currentMode) {
    case CarAudioManager.YamahaSoundMode.OFF:
        modeStr = "关闭"; break;
    case CarAudioManager.YamahaSoundMode.SPRING:
        modeStr = "春季"; break;
    case CarAudioManager.YamahaSoundMode.SUMMER:
        modeStr = "夏季"; break;
    case CarAudioManager.YamahaSoundMode.AUTUMN:
        modeStr = "秋季"; break;
    case CarAudioManager.YamahaSoundMode.WINTER:
        modeStr = "冬季"; break;
}
Log.d(TAG, "当前Yamaha音效模式: " + modeStr);
```

## 外置功放环绕模式控制方法

### enableExtSurroundMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int enableExtSurroundMode(boolean enable)` |
| 函数作用 | 设置外置功放环绕模式开关状态 |
| 参数说明 | `enable` - true:开启环绕模式, false:关闭环绕模式 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

### isExtSurroundModeEnabled()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean isExtSurroundModeEnabled()` |
| 函数作用 | 获取当前外置功放环绕模式状态 |
| 返回值 | `boolean` - true:环绕模式已开启, false:环绕模式已关闭 |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 开启外置功放环绕模式
int result = audioManager.enableExtSurroundMode(true);
if (result != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "外置功放环绕模式设置成功");
}

// 获取当前外置功放环绕模式状态
boolean isEnabled = audioManager.isExtSurroundModeEnabled();
Log.d(TAG, "当前外置功放环绕模式状态: " + (isEnabled ? "已开启" : "已关闭"));
```

## 音频上下文管理方法

### getAllContexts()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getAllContexts()` |
| 函数作用 | 获取所有音频上下文的编号 |
| 返回值 | `int[]` - 包含所有音频上下文编号的数组 |

### getAllContextsName()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public String[] getAllContextsName()` |
| 函数作用 | 获取所有音频上下文的名称 |
| 返回值 | `String[]` - 包含所有音频上下文名称的数组 |

### getContextsByAOSPUsage()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getContextsByAOSPUsage(int AOSPUsage)` |
| 函数作用 | 根据AOSP音频用途获取匹配的音频上下文编号 |
| 参数说明 | `AOSPUsage` - AOSP定义的音频用途值 |
| 返回值 | `int[]` - 匹配指定用途的音频上下文编号数组 |

### getContextsByBus()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getContextsByBus(int busNo)` |
| 函数作用 | 根据总线号获取关联的音频上下文编号 |
| 参数说明 | `busNo` - 音频总线编号 |
| 返回值 | `int[]` - 关联到指定总线的音频上下文编号数组 |

### getUsagesByBus()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getUsagesByBus(int busNo)` |
| 函数作用 | 根据总线号获取关联的音频用途 |
| 参数说明 | `busNo` - 音频总线编号 |

| 项目 | 说明 |
| --- | --- |
| 返回值 | `int[]` - 关联到指定总线的音频用途数组 |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 获取所有音频上下文
int[] allContexts = audioManager.getAllContexts();
String[] allContextNames = audioManager.getAllContextsName();

// 根据AOSP用途获取上下文
int[] mediaContexts =
audioManager.getContextsByAOSPUsage(AudioAttributes.USAGE_MEDIA);

// 根据总线号获取上下文和用途
int busNo = 1; // 示例总线号
int[] busContexts = audioManager.getContextsByBus(busNo);
int[] busUsages = audioManager.getUsagesByBus(busNo);

// 打印结果
Log.d(TAG, "所有音频上下文数量: " + allContexts.length);
Log.d(TAG, "总线" + busNo + "关联的用途数量: " + busUsages.length);
```

## 音频设备信息获取方法

### getDeviceIdForUsage()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getDeviceIdForUsage(int usage)` |
| 函数作用 | 根据音频用途获取对应的设备ID |
| 参数说明 | `usage` - 音频用途值(如`CarAudioAttributes.AUDIO_USAGE_USB_MUSIC`) |
| 返回值 | `int` - 设备ID(Result.NOT_SUPPORT表示不支持) |

### getOutputDeviceForUsage()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public AudioDeviceInfo getOutputDeviceForUsage(int zoneId, int usage)` |
| 函数作用 | 获取指定音频区域和用途的输出设备信息 |
| 参数说明 | `zoneId` - 音频区域ID<br>`usage` - 音频用途值(如`AudioAttributes.USAGE_MEDIA`) |
| 返回值 | `AudioDeviceInfo` - 音频设备信息(找不到返回null) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 获取USB音乐用途的设备ID
int deviceId =
audioManager.getDeviceIdForUsage(CarAudioAttributes.AUDIO_USAGE_USB_MUSIC);
if (deviceId != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "USB音乐设备ID: " + deviceId);
}

// 获取主区域媒体用途的输出设备信息
AudioDeviceInfo mediaDevice = audioManager.getOutputDeviceForUsage(
    CarAudioManager.ZONE_ID_MAIN,
    AudioAttributes.USAGE_MEDIA
);
if (mediaDevice != null) {
    Log.d(TAG, "媒体输出设备: " + mediaDevice.getProductName());
}
```

## 音频设备地址解析与总线ID获取方法

### parseDeviceAddress()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public static int parseDeviceAddress(String str)` |
| 函数作用 | 从设备地址字符串中解析出总线编号 |
| 参数说明 | `str` - 包含设备地址的字符串(格式如"bus0_xxx") |
| 返回值 | `int` - 解析出的总线编号(解析失败返回-1) |

### getBusByZoneId()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getBusByZoneId(int zoneId)` |
| 函数作用 | 根据音频区域ID获取关联的总线ID数组 |
| 参数说明 | `zoneId` - 音频区域ID |
| 返回值 | `int[]` - 关联到指定区域的总线ID数组 |

## 示例代码

```
// 解析设备地址获取总线编号
String deviceAddress = "bus2_output";
int busNumber = CarAudioManager.parseDeviceAddress(deviceAddress);
```

```
if (busNumber != -1) {
    Log.d(TAG, "设备总线编号: " + busNumber);
}

// 获取主音频区域关联的总线ID
int[] mainZoneBusIds = audioManager.getBusByZoneId(CarAudioManager.ZONE_ID_MAIN);
Log.d(TAG, "主区域总线数量: " + mainZoneBusIds.length);
for (int busId : mainZoneBusIds) {
    Log.d(TAG, "总线ID: " + busId);
}
```

## 麦克风预处理控制方法

### setPreprocessMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setPreprocessMode(int mode, int audioSource)` |
| 函数作用 | 设置麦克风预处理模式(如ECNR回声消除降噪) |
| 参数说明 | `mode` - 预处理模式值<br>`audioSource` - 音频源类型 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

### setMicrophoneMute()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setMicrophoneMute(boolean mute, int audioSource)` |
| 函数作用 | 设置麦克风静音状态 |
| 参数说明 | `mute` - true:静音, false:取消静音<br>`audioSource` - 音频源类型 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 获取CarAudioManager实例
CarAudioManager audioManager = CarAudioManager.getInstance(context);

// 设置麦克风预处理模式(ECNR模式)
int result = audioManager.setPreprocessMode(
    CarAudioManager.PREPROCESS_MODE_ECNR,
    AudioSource.VOICE_COMMUNICATION
);
if (result != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "麦克风预处理模式设置成功");
}
```

```
// 设置麦克风静音
int muteResult = audioManager.setMicrophoneMute(
    true,
    AudioSource.VOICE_COMMUNICATION
);
if (muteResult != CarAudioManager.Result.NOT_SUPPORT) {
    Log.d(TAG, "麦克风静音设置成功");
}
```

## 音频设置与事件监听管理

### SettingsParam 设置参数常量

| 常量 | 值 | 说明 |
| --- | --- | --- |
| NONE | 0 | 无设置 |
| ICC_MODE | 1 | 前排扩音模式 |
| HEADREST_MODE | 2 | 头枕模式 |

### 监听器注册与注销方法

**registerSettingsChangeListener()**

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public void registerSettingsChangeListener(OnSettingsChangeListener listener) |
| 函数作用 | 注册设置变更监听器 |
| 参数说明 | listener - 设置变更监听器实例 |
| 注意事项 | 内部类专用，需要PERMISSION_CAR_CONTROL_AUDIO_SETTINGS权限 |

**unregisterSettingsChangeListener()**

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public void unregisterSettingsChangeListener(OnSettingsChangeListener listener) |
| 函数作用 | 注销设置变更监听器 |
| 参数说明 | listener - 要注销的监听器实例 |

**registerEventListener()**

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public void registerEventListener(OnEventChangeListener listener) |

| 项目 | 说明 |
| --- | --- |
| 函数作用 | 注册事件变更监听器 |
| 参数说明 | `listener` - 事件变更监听器实例 |

**unregisterEventChangeListener()**

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public void unregisterEventChangeListener(OnEventChangeListener listener)` |
| 函数作用 | 注销事件变更监听器 |
| 参数说明 | `listener` - 要注销的监听器实例 |

## 示例代码

```
// 创建监听器实例
OnSettingsChangeListener settingsListener = new OnSettingsChangeListener() {
    @Override
    public void onSettingsEnabledChanged(int param, boolean enabled) {
        Log.d(TAG, "设置启用状态变更: param=" + param + ", enabled=" + enabled);
    }

    @Override
    public void onSettingsValueChanged(int param, int value) {
        Log.d(TAG, "设置值变更: param=" + param + ", value=" + value);
    }
};

// 注册设置监听器
audioManager.registerSettingsChangeListener(settingsListener);

// 创建事件监听器实例
OnEventChangeListener eventListener = new OnEventChangeListener() {
    @Override
    public void onReceiveEvent(int event, int state) {
        Log.d(TAG, "收到事件: event=" + event + ", state=" + state);
    }
};

// 注册事件监听器
audioManager.registerEventListener(eventListener);

// 使用后注销监听器
audioManager.unregisterSettingsChangeListener(settingsListener);
audioManager.unregisterEventChangeListener(eventListener);
```

## YAMAHA音效与场景设置

YAMAHAEffect 音效模式

| 常量 | 值 | 说明 |
| --- | --- | --- |
| BRIGHT_WILLOWS | 0 | 柳暗花明 |
| ORAINY_RAILING | 1 | 凭栏听雨 |
| ZEN_GARDEN | 2 | 禅房花木 |
| BABBLING_BROOK | 3 | 小桥流水 |

### EffectScence 音效场景

| 常量 | 值 | 说明 |
| --- | --- | --- |
| DTS_OFF | 0 | DTS关闭 |
| DTS_3D | 1 | DTS 3D模式 |
| ADIGO_3D | 2 | ADIGO 3D模式 |
| ADIGO_VRLIVE | 3 | ADIGO虚拟现场模式 |
| ADIGO_FAM_CINEMA | 4 | ADIGO家庭影院模式 |
| YAMAHA | 5 | YAMAHA音效模式 |

### SweetArea 最佳听音位

| 常量 | 值 | 说明 |
| --- | --- | --- |
| DRIVER | 0 | 主驾模式 |
| FRONT | 1 | 前排 |
| ALL | 2 | 全部位置 |
| REAR | 3 | 后排 |
| PASSENGER | 4 | 副驾驶 |

### ZoneIdDef 音频区域

| 常量 | 值 | 说明 |
| --- | --- | --- |
| ZONE_ID_MAIN | - | 主区域 |
| ZONE_ID_SECOND | - | 次区域 |

## 音效设置方法

### setSoundEffect() 基础方法

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public int setSoundEffect(@EffectScence int scence, int mode, @SweetArea int area) |

| 项目 | 说明 |
| --- | --- |
| 函数作用 | 设置音效(自动保存到系统) |
| 参数说明 | `scence` - 音效场景<br>`mode` - 音效模式(根据场景而定)<br>`area` - 最佳听音位 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

### setSoundEffect() 带保存选项

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setSoundEffect(@EffectScence int scence, int mode, @SweetArea int area, boolean save)` |
| 函数作用 | 设置音效并指定是否保存 |
| 参数说明 | `scence` - 音效场景<br>`mode` - 音效模式<br>`area` - 最佳听音位<br>`save` - 是否保存到系统 |
| 返回值 | `int` - 操作结果 |

### setSoundEffect() 音效共创版

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setSoundEffect(@EffectScence int scence, int mode, @SweetArea int area, boolean save, CarAudioSoundCoreation soundCoreationInfo)` |
| 函数作用 | 设置音效(支持音效共创参数) |
| 参数说明 | `scence` - 音效场景<br>`mode` - 音效模式<br>`area` - 最佳听音位<br>`save` - 是否保存<br>`soundCoreationInfo` - 音效共创参数 |
| 异常 | `IllegalArgumentException` - 参数不合法时抛出 |

| 项目 | 说明 |
| --- | --- |
| 返回值 | `int` - 操作结果 |

## 示例代码

```
// 基础音效设置
int result = audioManager.setSoundEffect(
    CarAudioManager.EffectScence.YAMAHA,
    CarAudioManager.YAMAHAEffect.BRIGHT_WILLOWS,
    CarAudioManager.SweetArea.DRIVER
);

// 临时音效设置(不保存)
int tempResult = audioManager.setSoundEffect(
    CarAudioManager.EffectScence.ADIGO_FAM_CINEMA,
    AdigoFamCinemaMode.ON,
    CarAudioManager.SweetArea.ALL,
    false
);

// 音效共创设置
CarAudioSoundCoreation creationInfo = new CarAudioSoundCoreation();
// 设置共创参数...
int creationResult = audioManager.setSoundEffect(
    CarAudioManager.EffectScence.YAMAHA,
    YAMAHAEffect.ZEN_GARDEN,
    CarAudioManager.SweetArea.ALL,
    true,
    creationInfo
);
```

getSoundEffect()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getSoundEffect()` |
| 函数作用 | 获取当前系统音效设置(由底层区分内置/外置音效) |
| 返回值 | `int[]` - 音效信息数组，包含以下内容:<br>`[0]` = 音效场景(`EffectScence`)，小于0时见`Result`<br>`[1]` = 音效模式<br>`[2]` = 最佳听音位(`SweetArea`)<br>`[3]` = 预留字段 |

## 示例代码

```
// 获取当前音效设置
int[] soundEffect = audioManager.getSoundEffect();

// 解析返回结果
if(soundEffect[0] >= 0) {
    int scene = soundEffect[0]; // 音效场景
    int mode = soundEffect[1];  // 音效模式
    int area = soundEffect[2];  // 最佳听音位

    Log.d(TAG, "当前音效设置: 场景=" + scene + ", 模式=" + mode + ", 听音位=" + area);
} else {
    Log.e(TAG, "获取音效失败，错误码=" + soundEffect[0]);
}
```

## 声场设置方法

### SoundFiledType 声场类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| SEATS_POSITATION | 0 | 水平位置 |
| PLANE_POSITION | 1 | 抬升位置 |
| WIDTH | 2 | 声场宽度 |
| DEPTH | 3 | 声场纵深 |
| HEIGHT | 4 | 天空高度 |

### SeatsPositionValue 水平位置参数

| 常量 | 值 | 说明 |
| --- | --- | --- |
| DREIVER_SEAT | 0 | 主驾 |
| FRONT_SEAT | 1 | 副驾 |
| FRONT_CENTER | 2 | 前排中央 |
| SECOND_ROW_CENTER | 3 | 二排中央 |
| ALL_SEATS | 4 | 全车中心 |
| CLOSE | 5 | 关 |
| ON | 6 | 开 |

### PlanePositionValue 抬升位置参数

| 常量 | 值 | 说明 |
| --- | --- | --- |
| OFF | 0 | 关 |
| ON | 1 | 开 |

| 常量 | 值 | 说明 |
| --- | --- | --- |
| LOW | 2 | 低 |
| MIDDLE | 3 | 中 |
| HIGHT | 4 | 高 |

setSoundFiled()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public int setSoundFiled(@SoundFiledType int type, int value) |
| 函数作用 | 设置声场参数 |
| 参数说明 | type - 声场类型<br>value - 对应类型的参数值 |
| 返回值 | int - 操作结果(Result.NOT_SUPPORT表示不支持) |

getSoundFiled()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public int getSoundFiled(@SoundFiledType int type) |
| 函数作用 | 获取声场参数 |
| 参数说明 | type - 声场类型 |
| 返回值 | int - 当前参数值(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 设置水平位置为主驾
int result = audioManager.setSoundFiled(
    CarAudioManager.SoundFiledType.SEATS_POSITATION,
    CarAudioManager.SeatsPositionValue.DREIVER_SEAT
);

// 获取当前声场宽度设置
int width = audioManager.getSoundFiled(CarAudioManager.SoundFiledType.WIDTH);
Log.d(TAG, "当前声场宽度设置: " + width);
```

## 空间音效设置方法

SpaceType 空间类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| SPACE_REVERBERATION | 0 | 模拟空间(预设使用) |

| 常量 | 值 | 说明 |
| --- | --- | --- |
| USER_REVERBERATION | 1 | 自定义空间(用户可自定义尺寸) |

SpaceValue 预设空间类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| OFF | 0 | 关闭 |
| ON | 1 | 开启 |
| LISTENING_ROOM | 2 | 听音室 |
| UNDERGROUND | 3 | 地下通道 |
| GRAND_THEATRE | 4 | 大剧院 |

setSpaceInfo()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setSpaceInfo(boolean isCustomized, int value)` |
| 函数作用 | 设置空间音效参数(区分自定义/预设) |
| 参数说明 | `isCustomized` - 是否自定义<br>`value` - 空间参数值 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

setSpaceSize()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setSpaceSize(@SpaceType int type, int value)` |
| 函数作用 | 设置指定类型的空间音效参数 |
| 参数说明 | `type` - 空间类型<br>`value` - 参数值 |
| 返回值 | `int` - 操作结果 |

getSpaceSize()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getSpaceSize(@SpaceType int type)` |
| 函数作用 | 获取指定类型的空间音效参数 |
| 参数说明 | `type` - 空间类型 |
| 返回值 | `int` - 当前参数值(Result.NOT_SUPPORT表示不支持) |

# 示例代码

```
// 设置预设空间类型为听音室
int result = audioManager.setSpaceInfo(false, SpaceValue.LISTENING_ROOM);

// 设置自定义空间参数
int customResult = audioManager.setSpaceSize(SpaceType.USER_REVERBERATION, 5);

// 获取当前空间设置
int spaceSize = audioManager.getSpaceSize(SpaceType.SPACE_REVERBERATION);
Log.d(TAG, "当前空间设置: " + spaceSize);
```

## 空间信息获取方法

getSpaceInfo()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getSpaceInfo()` |
| 函数作用 | 获取当前空间音效的设置信息 |
| 返回值 | `int[]` - 包含两个元素的数组:<br>`[0]` - 是否自定义(1表示自定义，0表示预设)<br>`[1]` - 当前空间参数值 |
| 异常 | 可能抛出`RuntimeException` - 当远程调用失败时 |

## 示例代码

```
// 获取当前空间设置信息
int[] spaceInfo = audioManager.getSpaceInfo();

if(spaceInfo[0] == 1) {
    Log.d(TAG, "当前使用自定义空间设置，参数值: " + spaceInfo[1]);
} else {
    Log.d(TAG, "当前使用预设空间设置，参数值: " + spaceInfo[1]);
}
```

## 音效共创管理方法

saveSoundCoreation()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int saveSoundCoreation(int id, String keypairs)` |
| 函数作用 | 保存音效共创的参数配置 |
| 参数说明 | `id` - 音效ID(1-20)<br>`keypairs` - 音效参数键值对字符串 |

| 项目 | 说明 |
| --- | --- |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 参数格式 | `"id=1;seatsMode=3;planePosition=2;unlimitedWidth=25"` |

### getSoundCoreation()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public String getSoundCoreation(int id)` |
| 函数作用 | 获取指定ID的音效参数配置 |
| 参数说明 | `id` - 音效ID(0表示获取默认值) |
| 返回值 | `String` - 音效参数键值对字符串 |

## 示例代码

```
// 保存音效配置
String params = "id=1;seatsMode=3;planePosition=2;unlimitedWidth=25";
int saveResult = audioManager.saveSoundCoreation(1, params);

// 获取音效配置
String savedParams = audioManager.getSoundCoreation(1);
Log.d(TAG, "获取的音效配置: " + savedParams);
```

## 背景氛围音效设置方法

### AmbienceMode 背景氛围音效模式

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭 |
| `STORMY_NIGHT` | 1 | 风雨夜归人 |
| `WALKING_SEASIDE` | 2 | 海边漫步 |
| `WISP_BONFIRE` | 3 | 一缕篝火 |

### setAmbienceMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setAmbienceMode(@AmbienceMode int mode)` |
| 函数作用 | 设置背景氛围音效模式 |
| 参数说明 | `mode` - 音效模式(使用AmbienceMode中的常量) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

### getAmbienceMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getAmbienceMode()` |
| 函数作用 | 获取当前背景氛围音效模式 |
| 返回值 | `int` - 当前音效模式(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 设置背景氛围音效为"海边漫步"
int result = audioManager.setAmbienceMode(AmbienceMode.WALKING_SEASIDE);

// 获取当前背景氛围音效模式
int currentMode = audioManager.getAmbienceMode();
if(currentMode == AmbienceMode.WALKING_SEASIDE) {
    Log.d(TAG, "当前背景氛围音效: 海边漫步");
}
```

## 车外喊话功能接口

### OCCVoiceType 变声类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭 |
| `DEFAULT` | 0 | 同OFF |
| `CARTOON` | 1 | 卡通音效 |
| `UNCLE` | 2 | 大叔音效 |
| `LOLITA` | 3 | 萝莉音效 |
| `ROBOT` | 4 | 机器人音效 |
| `BABY` | 5 | 娃娃音效 |

### setOCCStateAndMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setOCCStateAndMode(int state, @OCCVoiceType int mode)` |
| 函数作用 | 设置车外喊话开关状态和变声模式 |
| 参数说明 | `state` - 开关状态(0:关闭, 1:开启)<br>`mode` - 变声模式(使用OCCVoiceType中的常量) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

getOCCStateAndMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int[] getOCCStateAndMode()` |
| 函数作用 | 获取当前车外喊话状态和变声模式 |
| 返回值 | `int[]` - 包含两个元素的数组:<br>`[0]` - 当前开关状态(0:关闭, 非0:开启)<br>`[1]` - 当前变声模式 |

## 示例代码

```
// 开启车外喊话并设置变声模式为机器人
int result = audioManager.setOCCStateAndMode(1, OCCVoiceType.ROBOT);

// 获取当前车外喊话状态
int[] occState = audioManager.getOCCStateAndMode();
if(occState[0] != 0) {
    Log.d(TAG, "车外喊话已开启，当前变声模式: " + occState[1]);
} else {
    Log.d(TAG, "车外喊话已关闭");
}
```

## 媒体外放功能接口

OutMediaMode 媒体外放模式

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭外放 |
| `INCAR` | 1 | 车内媒体外放 |
| `OUTCAR` | 2 | 车外媒体外放 |

setOutMediaMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setOutMediaMode(@OutMediaMode int mode)` |
| 函数作用 | 设置媒体外放模式 |
| 参数说明 | `mode` - 外放模式(使用OutMediaMode中的常量) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

getOutMediaMode()

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getOutMediaMode()` |
| 函数作用 | 获取当前媒体外放模式 |
| 返回值 | `int` - 当前外放模式(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 设置媒体外放为车外模式
int result = audioManager.setOutMediaMode(OutMediaMode.OUTCAR);

// 获取当前媒体外放模式
int currentMode = audioManager.getOutMediaMode();
if(currentMode == OutMediaMode.OUTCAR) {
    Log.d(TAG, "当前媒体外放模式: 车外");
}
```

## AVAS低速行人提示音功能接口

### AVASStyle 提示音风格

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭提示音 |
| `STYLE_1` | 1 | 风格1 |
| `STYLE_2` | 2 | 风格2 |
| `STYLE_3` | 3 | 风格3 |

### enableAVAS()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int enableAVAS(boolean enable)` |
| 函数作用 | 设置AVAS开关状态 |
| 参数说明 | `enable` - true:开启 false:关闭 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |
| 特殊说明 | 内部会调用setAVAS()方法，开启时会使用上次记录的样式 |

### setAVAS()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setAVAS(@AVASStyle int style)` |
| 函数作用 | 设置AVAS提示音风格 |

| 项目 | 说明 |
| --- | --- |
| 参数说明 | `style` - 提示音风格(使用AVASStyle中的常量) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

getAVAS()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getAVAS()` |
| 函数作用 | 获取当前AVAS状态 |
| 返回值 | `int` - 当前状态(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 开启AVAS提示音(使用上次记录的样式)
int result = audioManager.enableAVAS(true);

// 设置AVAS提示音为风格2
int setResult = audioManager.setAVAS(AVASStyle.STYLE_2);

// 获取当前AVAS状态
int currentStyle = audioManager.getAVAS();
if(currentStyle == AVASStyle.STYLE_2) {
    Log.d(TAG, "当前AVAS风格: 风格2");
}
```

## ESEStyle声浪风格功能接口

ESEStyle 声浪风格类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `OFF` | 0 | 关闭声浪风格 |
| `STYLE_1` | 1 | 风格1 |
| `STYLE_2` | 2 | 风格2 |
| `STYLE_3` | 3 | 风格3 |

setESEStyle()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setESEStyle(@ESEStyle int style)` |
| 函数作用 | 设置ESEStyle声浪风格 |
| 参数说明 | `style` - 声浪风格(使用ESEStyle中的常量) |

| 项目 | 说明 |
| --- | --- |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

getESEStyle()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getESEStyle()` |
| 函数作用 | 获取当前ESEStyle声浪风格 |
| 返回值 | `int` - 当前声浪风格(Result.NOT_SUPPORT表示不支持) |

setESEEnable()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setESEEnable(boolean enable)` |
| 函数作用 | 设置引擎声效开关 |
| 参数说明 | `enable` - true:开启 false:关闭 |
| 返回值 | `int` - 操作结果 |

getESEEnable()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean getESEEnable()` |
| 函数作用 | 获取引擎声效开关状态 |
| 返回值 | `boolean` - 当前开关状态 |

## 示例代码

```
// 设置声浪风格为风格2
int result = audioManager.setESEStyle(ESEStyle.STYLE_2);

// 获取当前声浪风格
int currentStyle = audioManager.getESEStyle();
if(currentStyle == ESEStyle.STYLE_2) {
    Log.d(TAG, "当前声浪风格: 风格2");
}

// 开启引擎声效
audioManager.setESEEnable(true);

// 检查引擎声效是否开启
boolean isEnabled = audioManager.getESEEnable();
```

# 引擎声效和媒体类型接口

CarHalEvent 引擎声效事件类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| MediaBus | 0 | 媒体总线事件 |

MediaBusState 媒体类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| NONE | 0 | 无媒体 |
| STEREO | 1 | 立体声 |
| MEDIA_712 | 2 | 7.1.2媒体 |
| MEDIA_714 | 3 | 7.1.4媒体 |
| THIRD_PARTY | 4 | 第三方媒体 |
| REAR_MEDIA | 5 | 后排媒体 |
| PROJECTION_SCREEN | 6 | 投影屏幕 |
| GAME | 7 | 游戏 |
| NATURE_SOUND | 8 | 自然声音 |
| MEDIA_51 | 9 | 5.1媒体 |

getCarEventState()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public int getCarEventState(@CarHalEvent int event) |
| 函数作用 | 获取当前媒体事件状态 |
| 参数说明 | event - 事件类型(使用CarHalEvent中的常量) |
| 返回值 | int - 当前媒体状态(Result.NOT_SUPPORT表示不支持) |

# 示例代码

```
// 获取当前媒体事件状态
int mediaState = audioManager.getCarEventState(CarHalEvent.MediaBus);
if(mediaState == MediaBusState.STEREO) {
    Log.d(TAG, "当前媒体状态: 立体声");
}
```

# 外部AudioBus接口

PatchType 外部AudioBus类型

| 常量 | 值 | 说明 |
| --- | --- | --- |
| TYPE_GAME_BOX | 200 | 游戏盒子 |
| TYPE_EBCALL | 201 | 紧急呼叫 |
| TYPE_BTCALL | 202 | 蓝牙通话 |
| TYPE_OCC | 203 | 车外喊话 |

getSourceAndSink()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public String[] getSourceAndSink(@PatchType int type) |
| 函数作用 | 获取外部AudioBus对应的source和sink信息 |
| 参数说明 | type - 外部AudioBus类型(使用PatchType中的常量) |
| 返回值 | String[] - 返回数组，[0]=source名称，[1]=sink名称 |

## 示例代码

```
// 获取游戏盒子的AudioBus信息
String[] busInfo = audioManager.getSourceAndSink(PatchType.TYPE_GAME_BOX);
Log.d(TAG, "游戏盒子AudioBus - source: " + busInfo[0] + ", sink: " + busInfo[1]);

// 获取蓝牙通话的AudioBus信息
busInfo = audioManager.getSourceAndSink(PatchType.TYPE_BTCALL);
Log.d(TAG, "蓝牙通话AudioBus - source: " + busInfo[0] + ", sink: " + busInfo[1]);
```

## 音频属性设置接口

getAmbientAudioAttribute() 氛围音播放参数

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public static AudioAttributes.Builder getAmbientAudioAttribute() |
| 函数作用 | 获取氛围音播放的音频属性构建器 |
| 参数说明 | 无 |
| 返回值 | AudioAttributes.Builder - 音频属性构建器，已设置usage为100(USAGE_SY_ATMOSPHERE) |

getTTSAudioAttribute() TTS/OCC/APA播放参数

| 项目 | 说明 |
| --- | --- |
| 方法签名 | public static AudioAttributes.Builder getTTSAudioAttribute() |
| 函数作用 | 获取TTS/OCC/APA播放的音频属性构建器 |

| 项目 | 说明 |
| --- | --- |
| 参数说明 | 无 |
| 返回值 | `AudioAttributes.Builder` - 音频属性构建器，已设置usage为101(USAGE_SY_OCC_TTS) |

## 示例代码

```
// 创建氛围音播放属性
AudioAttributes ambientAttrs = CarAudioManager.getAmbientAudioAttribute()
    .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
    .build();

// 创建TTS播放属性
AudioAttributes ttsAttrs = CarAudioManager.getTTSAudioAttribute()
    .setContentType(AudioAttributes.CONTENT_TYPE_SPEECH)
    .build();
```

## 音频通路管理接口

### createGameBoxAudioPatch() 游戏投屏通路

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public static AudioPatch createGameBoxAudioPatch(AudioPatch audioPatch)` |
| 函数作用 | 创建游戏投屏音频通路 |
| 参数说明 | `audioPatch` - 音频通路对象 |
| 返回值 | `AudioPatch` - 创建成功的音频通路对象 |

### createRadioAudioPatch() Radio通路

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public static AudioPatch createRadioAudioPatch(AudioPatch audioPatch)` |
| 函数作用 | 创建Radio音频通路 |
| 参数说明 | `audioPatch` - 音频通路对象 |
| 返回值 | `AudioPatch` - 创建成功的音频通路对象 |

### createBtCallAudioPatch() 蓝牙通话通路

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public static AudioPatch createBtCallAudioPatch(AudioPatch audioPatch)` |
| 函数作用 | 创建蓝牙通话音频通路 |
| 参数说明 | `audioPatch` - 音频通路对象 |

| 项目 | 说明 |
| --- | --- |
| 返回值 | `AudioPatch` - 创建成功的音频通路对象 |

releaseAudioPatch() 释放音频通路

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public static int releaseAudioPatch(AudioPatch audioPatch)` |
| 函数作用 | 释放音频通路资源 |
| 参数说明 | `audioPatch` - 要释放的音频通路对象 |
| 返回值 | `int` - 操作结果 |

## 示例代码

```
// 创建游戏投屏音频通路
AudioPatch gamePatch = CarAudioManager.createGameBoxAudioPatch(null);
if(gamePatch != null) {
    Log.d(TAG, "游戏投屏通路创建成功");
}

// 创建蓝牙通话音频通路
AudioPatch btPatch = CarAudioManager.createBtCallAudioPatch(null);
if(btPatch != null) {
    Log.d(TAG, "蓝牙通话通路创建成功");
}

// 释放音频通路
int result = CarAudioManager.releaseAudioPatch(gamePatch);
if(result == AudioManager.SUCCESS) {
    Log.d(TAG, "音频通路释放成功");
}
```

## VR扬声器位置控制接口

VRSpeakerLocation VR扬声器位置定义

| 常量 | 值 | 说明 |
| --- | --- | --- |
| `FRONT_LEFT` | 1 << 0 | 左前扬声器 |
| `FRONT_RIGHT` | 1 << 1 | 右前扬声器 |
| `REAR_LEFT` | 1 << 2 | 左后扬声器 |
| `REAR_RIGHT` | 1 << 3 | 右后扬声器 |

setVRSpeakerLocation()

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setVRSpeakerLocation(@VRSpeakerLocation int location)` |
| 函数作用 | 设置VR音频的扬声器输出位置 |
| 参数说明 | `location` - 扬声器位置(使用VRSpeakerLocation中的常量或组合) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## 示例代码

```
// 设置VR音频输出到左前和右前扬声器
int result = audioManager.setVRSpeakerLocation(
    VRSpeakerLocation.FRONT_LEFT | VRSpeakerLocation.FRONT_RIGHT);
if(result >= 0) {
    Log.d(TAG, "VR扬声器位置设置成功");
}

// 设置VR音频输出到所有扬声器
result = audioManager.setVRSpeakerLocation(
    VRSpeakerLocation.FRONT_LEFT | VRSpeakerLocation.FRONT_RIGHT |
    VRSpeakerLocation.REAR_LEFT | VRSpeakerLocation.REAR_RIGHT);
if(result >= 0) {
    Log.d(TAG, "VR扬声器全位置设置成功");
}
```

## 音效共创功能接口

### setEnableSoundCoreation() 开启/关闭音效共创

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setEnableSoundCoreation(boolean enableSoundCoreation)` |
| 函数作用 | 开启或关闭音效共创功能 |
| 参数说明 | `enableSoundCoreation` - true开启，false关闭 |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

### isSoundCoreation() 检查音效共创状态

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public boolean isSoundCoreation()` |
| 函数作用 | 检查当前是否使用音效共创参数 |
| 返回值 | `boolean` - true表示使用音效共创参数，false表示使用普通音效 |

### OutTtsMode TTS输出模式定义

| 常量 | 值 | 说明 |
| --- | --- | --- |
| OFF | 0 | 关闭 |
| INCAR | 1 | 车内输出 |
| OUTCAR | 2 | 车外输出 |

## 示例代码

```
// 开启音效共创
int result = audioManager.setEnableSoundCoreation(true);
if(result >= 0) {
    Log.d(TAG, "音效共创开启成功");
}

// 检查音效共创状态
boolean isEnabled = audioManager.isSoundCoreation();
Log.d(TAG, "当前音效共创状态: " + isEnabled);

// 设置TTS输出模式为车内
audioManager.setOutTtsMode(OutTtsMode.INCAR);
```

## TTS模式控制接口

### OutTtsMode TTS输出模式定义

| 常量 | 值 | 说明 |
| --- | --- | --- |
| OFF | 0 | 关闭TTS输出 |
| INCAR | 1 | 车内TTS输出 |
| OUTCAR | 2 | 车外TTS输出 |

### setOutTtsMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int setOutTtsMode(@OutTtsMode int mode)` |
| 函数作用 | 设置TTS输出模式 |
| 参数说明 | `mode` - TTS输出模式(使用OutTtsMode中的常量) |
| 返回值 | `int` - 操作结果(Result.NOT_SUPPORT表示不支持) |

## 节能模式音量控制接口

### saveGroupVolumeBySaverMode()

| 项目 | 说明 |
| --- | --- |

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int saveGroupVolumeBySaverMode(int groupId, int index, boolean isHeadreset)` |
| 函数作用 | 设置节能模式下的头枕/非头枕音量 |
| 参数说明 | `groupId` - 音量组ID<br>`index` - 音量值<br>`isHeadreset` - 是否为头枕模式 |
| 返回值 | `int` - 操作结果 |

getGroupVolumeBySaverMode()

| 项目 | 说明 |
| --- | --- |
| 方法签名 | `public int getGroupVolumeBySaverMode(int groupId, boolean isHeadreset)` |
| 函数作用 | 获取节能模式下的头枕/非头枕音量 |
| 参数说明 | `groupId` - 音量组ID<br>`isHeadreset` - 是否为头枕模式 |
| 返回值 | `int` - 音量值或错误码 |

## 示例代码

```
// 设置TTS为车内输出模式
int result = audioManager.setOutTtsMode(OutTtsMode.INCAR);
if(result >= 0) {
    Log.d(TAG, "TTS模式设置成功");
}

// 设置节能模式下头枕音量
audioManager.saveGroupVolumeBySaverMode(1, 10, true);

// 获取节能模式下头枕音量
int volume = audioManager.getGroupVolumeBySaverMode(1, true);
Log.d(TAG, "当前头枕音量: " + volume);
```

# 预留函数

```
    public int setParamEnable(int param, boolean enable)
    public boolean isParamEnabled(int param, boolean def)
    public int setParamValue(int param, int value)
    public int getParamValue(int param, int def)
    void setDuckAudioPolicy(int audioUsage, int policy, float volume, int
durationMs)
```

```
    public int setVolumeLimitingEnable(boolean enable)
    public boolean isVolumeLimitingEnabled()
    public int getGroupLimitingVolume(int groupId)
    public int enableAudioChime(int group, boolean enable)
    public boolean isAudioChimeEnabled(int group)
    public int playPresetAudioChime(int group)
    public int playAudioChime(int group, int chime)
    public int setPresetAudioChimeForGroup(int group, int chime)
    public int getPresetAudioChimeForGroup(int group)
    public int[] getAudioChimeListForGroup(int group)
```