A-Link 手机端 SDK 使用说明文档

系统名称	A-Link
文档提交日期	2019/11/12

Amap

1 开发环境

本文档是向 Android 应用添加 A-Link 手机端(Server 端)SDK 的快速入门说明。

1.1 下载并安装 Android Studio

从 Google 下载安装 Android Studio(https://developer.android.com/studio)。

1.2 创建项目

按以下步骤新建一个 Empty Activity 的应用项目。

启动 Android Studio。如果您看到 Welcome to Android Studio 对话框，请选择 Start a new Android Studio project，否则，请点击 Android Studio 菜单栏中的 File，然后点击 New->New Project，按提示输入您的应用名称、公司域和项目位置。然后点击 Next。

选择您的应用所需的机型。 如果您不能确定自己的需要，只需选择 Phone and Tablet。然后点击 Next。

在“Add an activity to Mobile”对话框中选择 Empty Activity。然后点击 Next。

按提示输入 Activity 名称、布局名称和标题。使用默认值即可。然后点击 Finish。

1.3 导入 SDK 的 AAR 包

按照以下步骤导入高德提供的 A-Link Server SDK AAR 包。

将 aar 包复制到 lib 目录下，然后配置 build.gradle 文件，加入如下内容：

```
repositories {
   flatDir {
        dirs 'libs'
    }
}
compile(name:'alinkserver_1.0.0', ext:'aar')
```

修改完成后再次编译 Project，操作成功后可以在扩展包下看到被引用的 AAR 包文件。

2 SDK 接口调用时序

SDK 接口调用说明如下：

1. 调用 onCreate()方法初始化 SDK。

2. 调用 registerVideoListener()方法把 VideoListener 注册到 SDK，SDK 在和车机建立连接后调用 onConfigure()方法通知上层应用录制视频的参数，SDK 调用 onStart()方法通知上层应用开始录制视频，上层应用录制视频时持续调用 sendVideoData()方法把录制的视频数据发送给 SDK。停止录制时 SDK 会调用 onStop()方法通知上层应用，上层应用应该停止录制视频，不再调用 sendVideoData()方法给 SDK 发送视频数据。

3. 调用 registerAudioListener()方法把 AudioListener 注册到 SDK，SDK 在和车机建立连接后调用 onStart()方法通知上层应用开始录制音频，上层应用录制 PCM 音频时持续调用 sendAudioData()方法把录制的 PCM 音频数据发送给 SDK。停止录制时 SDK 会调用 onStop()方法通知上层应用，上层应用应该停止录制音频，不再调用 sendAudioData()方法给 SDK 发送音频数据。

4. 调用 registerStatusListener()方法把 StatusListener 注册到 SDK，之后当连接状态发生变化时 SDK 会调用 notifyDeviceStatus()方法将变更后的状态通知上层应用。上层应用也可以调用 getConnectionStatus()方法主动获取当前连接状态。

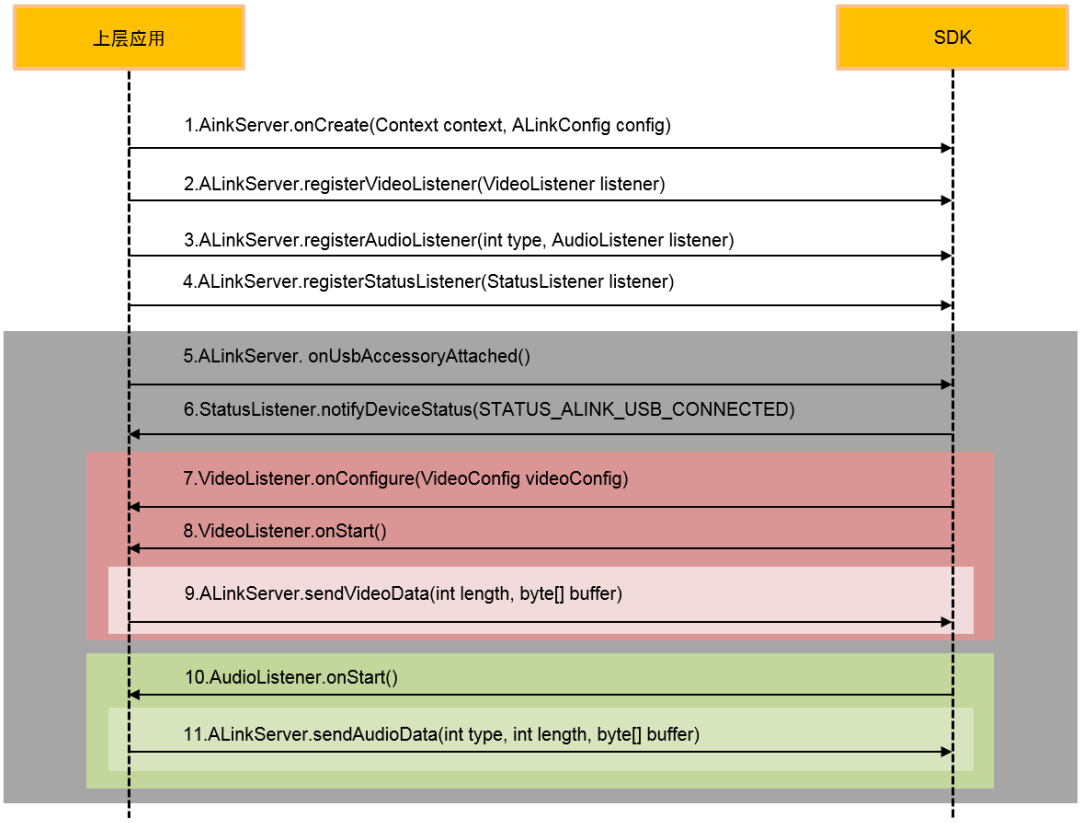
5. 调用 registerEventListener()方法把 EventListener 注册到 SDK，事件通过 json 串返回给上层应用。

6. 当手机以 USB 配件模式连接其他设备时，上层应用必须调用 onUsbAccessoryAttached()方法通知 SDK。

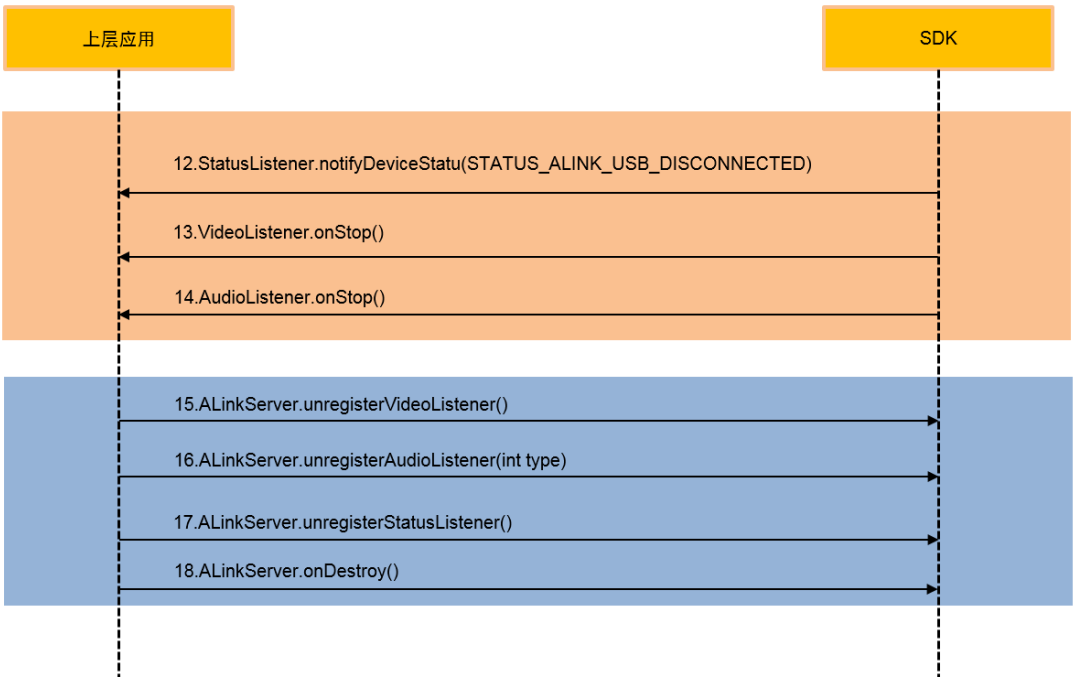
7. 停止使用 SDK 时，上层应用需要先调用 unregisterVideoListener()、unregisterAudioListener()、unregisterStatusListener()方法取消注册的 listener，最后调用 onDestroy()方法释放 SDK 资源。

SDK 接口调用时序图如下：

应用和SDK交互时序图(一)



应用和SDK交互时序图(二)



3. SDK 接口使用说明

3.1 SDK 初始化

设置 A-Link Server 配置信息：

```
private AlinkConfig getALinkConfig() {
    AlinkConfig config = new AlinkConfig();

    Point point = getPhoneSize();
    config.screenWidth = point.x;
    config.screenHeight = point.y;

    config.useTestAudio = false;
    config.useTestVideo = false;

    return config;
}
```

初始化 A-Link Server SDK：

```
public class MainActivity extends Activity {

    private ALinkServer mALinkServer = new ALinkServer();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /**
         * 获取ALink Server配置信息
         */
        ALinkConfig config = getALinkConfig();

        /**
         * ALink Server SDK初始化
         */
        mALinkServer.onCreate(this, config);
```

3.2 视频采集

视频采集类需要实现 VideoListener 接口，调用 registerVideoListener()方法把 listener 注册到 SDK，来监听视频编码的数据规格(SDK 调用 onConfigure()方法通知视频录制的参数)、开始录制视频通知(SDK 调用 onStart()方法通知开始录制视频)和结束录制视频通知(SDK 调用 onStop()方法停止录制视频)。

开始录制视频后上层应用持续调用 sendVideoData()方法将采集的视频数据发送给 SDK，每次调用需要是完整的视频帧数据。

停止使用 listener 需要调用 unregisterVideoListener()方法取消注册。

使用示例如下：
1.视频录制控制类实现 VideoListener 接口中的 onConfigure，onStart 和 onStop 方法，在开始录制视频后按照 onConfigure 中请求的视频参数制作 H264 数据并调用 sendVideoData()方法把视频数据发送给 SDK。

```java
/**
 * 视频录制类。
 *
 * @author ALink
 */
public class VideoCapture implements VideoListener {
    private ALinkServer mALinkServer = null;
    private VideoConfig mVideoConfig;
    private VideoRecordThread mVideoRecordThread = null;

    public VideoCapture(ALinkServer aLinkServer) { mALinkServer = aLinkServer; }

    /**
     * 将SDK请求的视频参数通知上层应用。
     *
     * @param config 视频录制的参数，包含分辨率的宽、高，密度以及帧率。
     */
    @Override
    public void onConfigure(VideoConfig config) {
        /**
         * 此处记录SDK请求的H264格式视频的参数。
         */
        mVideoConfig = config;
    }

    /**
     * 通知开始录制视频。
     */
    @Override
    public void onStart() {
        /**
         * 按照onConfigure中请求的参数制作H.264格式的数据。
         * 示例以创建线程的方式采集视频数据，并把视频数据发送给SDK。
         */
        mVideoRecordThread = new VideoRecordThread(mALinkServer, mVideoConfig);
        mVideoRecordThread.start();
    }

    /**
     * 通知停止录制视频。
     */
    @Override
    public void onStop() {
        /**
         * 结束录制视频，停止调用sendVideoData方法。
         */
        mVideoRecordThread.stopThread();
    }

    /**
     * 视频录制线程。
     */
    private static class VideoRecordThread extends Thread {
        ALinkServer mALinkServer = null;
        private AtomicBoolean mRunning = new AtomicBoolean(true);

        public VideoRecordThread(ALinkServer aLinkServer, VideoConfig videoConfig) { mALinkServer = aLinkServer; }
```

```
        /**
         * 此处用MediaCodec和VirtualDisplay录制H264格式的屏幕画面，伪代码描述这段实现。
         */
        @Override
        public void run() {
            创建video/avc类型的MediaCodec编码器;
            设置MediaCodec编码器的参数;
            创建MediaCodec编码器的输入源Surface;
            创建VirtualDisplay并作为MediaCodec编码器的输入源;
            MediaCodec编码器开始编码;

            while (mRunning.get()) {
                获取MediaCodec输出区的缓冲的索引;
                从输出区取出视频数据到buffer中;
                调用mALinkServer的sendVideoData方法把视频数据发送给SDK;
            }
            释放资源;
        }

        public void stopThread() {
            mRunning.set(false);
        }
    }
}
```

2.注册 VideoListener 到 SDK

```
/**
 * 创建视频录制类对象。
 */
VideoCapture videoCapture = new VideoCapture(mALinkServer);

/**
 * VideoListener对象注册到SDK。
 */
mALinkServer.registerVideoListener(videoCapture);
```

3.停止使用 VideoListener

```
/**
 * 停止使用VideoListener时取消注册。
 */
mALinkServer.unregisterVideoListener();
```

3.3 音频采集

A-Link 协议支持手机端使用三通道(media，navigation，TTS)将手机端 PCM 音频数据传输到车机端，最少需要实现其中一种通道。

media 音频的采样格式相同：16 bit, stereo, 48 kHz。
TTS 和 Navigation 音频的采样格式：16bit，mono，16kHz。

音频采集类需要实现 AudioListener 接口，之后调用 registerAudioListener()方法把 listener 注册到 SDK，注册时需指定录制音频的类型(取值有 AUDIO_TYPE_MEDIA，AUDIO_TYPE _TTS，AUDIO_TYPE _NAVIGATION 三种)，来监听音频录制开始和结束的通知(SDK 调用 onStart()方法通知开始录制音频，调用 onStop()方法通知结束录制音频)。

开始录制音频数据后，上层应用需要持续调用 sendAudioData()方法把采集的音频数据发送给 SDK。

结束使用 listener 需要调用 unregisterAudioListener()方法取消注册。

下面以 media 通道为例：
1.音频采集类实现 AudioListener 接口中的 onStart，onStop 方法，按照 16 bit，stereo，48 kHz 的格式录制 PCM 音频数据。

```
/**
 * 音频录制类。
 *
 * @author ALink
 */
public class AudioCapture implements AudioListener {
    /**
     * 音频采样率48KHz。
     */
    private static final int SAMPLE_RATE = 48000;
    private ALinkServer mALinkServer = null;
    private RecordThread mRecordThread = null;

    public AudioCapture(ALinkServer aLinkServer) { mALinkServer = aLinkServer; }

    /**
     * 重载AudioListener的onStart方法，SDK调用此方法通知上层应用开始录制音频。
     */
    @Override
    public void onStart() {
        /**
         * 开始录制media音频，media音频录制格式：16 bit, stereo, 48 kHz。
         * 示例以创建线程的方式采集音频数据，并把音频数据发送给SDK。
         */
        mRecordThread = new RecordThread(mALinkServer);
        mRecordThread.start();
    }
```

```
    /**
     * 重载AudioListener的onStop方法，SDK调用此方法通知上层应用结束录制音频。
     */
    @Override
    public void onStop() {
        /**
         * 结束录音线程，停止调用sendAudioData方法。
         */
        mRecordThread.stopThread();
    }

    /**
     * 音频采集线程类。
     */
    private static class RecordThread extends Thread {
        ALinkServer mALinkServer = null;
        private AtomicBoolean mRunning = new AtomicBoolean(true);

        public RecordThread(ALinkServer aLinkServer) { mALinkServer = aLinkServer; }

        @Override
        public void run() {
            /**
             * 创建16 bit, stereo, 48 kHz格式的AudioRecord。
             */
            int bufferSize = AudioRecord.getMinBufferSize(SAMPLE_RATE, AudioFormat.CHANNEL_IN_STEREO,
                AudioFormat.ENCODING_PCM_16BIT);
            AudioRecord audioRecord = new AudioRecord(AudioSource.REMOTE_SUBMIX, SAMPLE_RATE,
                AudioFormat.CHANNEL_IN_STEREO, AudioFormat.ENCODING_PCM_16BIT, bufferSize * 2);
            byte[] buffer = new byte[bufferSize];

            /**
             * 开始录制音频数据，调用sendAudioData方法把音频数据发送给SDK。
             */
            audioRecord.startRecording();
            while (mRunning.get()) {
                int readLength = audioRecord.read(buffer, 0, bufferSize);
                if (readLength > 0 && mALinkServer != null) {
                    mALinkServer.sendAudioData(ALinkServer.AUDIO_TYPE_MEDIA, readLength, buffer);
                }
            }

            /**
             * 停止录制音频，释放资源。
             */
            audioRecord.stop();
            audioRecord.release();
        }

        public void stopThread() { mRunning.set(false); }
    }
}
```

2.注册 AudioListener 到 SDK

```
    /**
     * 创建media音频的采集类对象。
     */
    AudioCapture audioCapture = new AudioCapture(mALinkServer);

    /**
     * AudioListener对象注册到SDK，并指明采集数据的类型。
     */
    mALinkServer.registerAudioListener(ALinkServer.AUDIO_TYPE_MEDIA, audioCapture);
```

3.停止使用 AudioListener

```
/**
 * 停止使用AudioListener时取消注册。
 */
mALinkServer.unregisterAudioListener(ALinkServer.AUDIO_TYPE_MEDIA);
```

3.4 状态监听

SDK 提供连接状态监听接口。连接状态监听类需要实现 StatusListener 接口，然后调用 registerStatusListener()方法把 StatusListener 注册到 SDK，之后由 SDK 调用 notifyDeviceStatus()方法通知连接状态的变更。

使用示例如下：
1. 连接状态接收类需要实现 StatusListener 的接口，notifyDeviceStatus 方法通知上层应用当前的连接状态。

```
/**
 * 连接状态接收类。
 *
 * @author ALink
 */
public class StatusReceiver implements StatusListener {
    /**
     * 连接状态变更通知。
     * 当SDK连接状态发生变化时，SDK调用此方法通知上层应用。
     *
     * @param status 当前设备连接状态。
     *         SDK通过USB与车机端连接上，该值为STATUS_ALINK_USB_CONNECTED；
     *         断开USB连接，该值为STATUS_ALINK_USB_DISCONNECTED；
     *         SDK通过WiFi与车机端连接上，该值为STATUS_ALINK_WIFI_CONNECTED；
     *         断开WiFi连接，该值为STATUS_ALINK_WIFI_DISCONNECTED。
     */
    @Override
    public void notifyDeviceStatus(int status) {
    }
}
```

2.注册 StatusListener 到 SDK

```
/**
 * 创建连接状态接收类对象。
 */
StatusReceiver statusReceiver = new StatusReceiver();

/**
 * StatusListener注册到SDK。
 */
mALinkServer.registerStatusListener(statusReceiver);
```

3.停止使用 StatusListener

```
/**
 * 停止使用连接状态监听。
 */
mALinkServer.unregisterStatusListener();
```

3.5 事件监听

SDK 提供事件监听接口。事件监听类需要实现 EventListener 接口, 然后调用 registerEventListener()方法把 EventListener 注册到 SDK。之后当某事件发生时, 由 SDK 将事件封装成 json 串作为 listener 参数回调通知上层应用。目前, sdk 支持的事件包括:

// type=1: 表示 sdk 事件
// client_beta=0: 表示 client 是正式证书; client_beta=1: 表示 client 是测试证书
// server_beta=0: 表示 server 是正式证书; server_beta=1: 表示 server 是测试证书
// 当 client 和 server 有一端是测试证书时, sdk 运行 1 小时会自动断开, 需要插拔 usb 重新建立连接
{"type":1, "client_beta":0, "server_beta":1}

// type=2: 表示 HMI 的按键事件
// key=277: 表示 ASR 开始的键值, key=86: 表示 ASR 停止的键值
// action=11: 表示按下的动作, action=12: 表示抬起的动作
{"type":2, "key":277, "action":11}
{"type":2, "key":277, "action":12}
{"type":2, "key":86, "action":11}
{"type":2, "key":86, "action":12}

3.6 USB 连接通知

上层应用监听系统的 USB_ACCESSORY_ATTACHED 消息，当手机以 USB 配件模式连接其他设备时必须要调用 onUsbAccessoryAttached()方法通知 SDK。

1.在 AndroidManifest.xml 里添加 USB attach 监听，其中的 xml/accessory_filter 在 AAR 包中定义：

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <meta-data
        android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
        android:resource="@xml/accessory_filter" />
</activity>
```

2.在 AudioListener、VideoListener、StatusListener 注册到 SDK 后，查看 USB 是否连接并通知 SDK。

```
/**
 * 在SDK的listener注册完成之后，检测USB连接并通知SDK。
 */
Intent intent = getIntent();
if ("android.hardware.usb.action.USB_ACCESSORY_ATTACHED".equals(intent.getAction())) {
    mALinkServer.onUsbAccessoryAttached();
}
```

3.在 onNewInten 方法监听 USB_ACCESSORY_ATTACHED 消息，通知 SDK。

```
@Override
protected void onNewIntent(Intent intent) {
    super.onNewIntent(intent);
    if ("android.hardware.usb.action.USB_ACCESSORY_ATTACHED".equals(intent.getAction())) {
        mALinkServer.onUsbAccessoryAttached();
    }
}
```

3.7 获取 SDK 版本号

获取 SDK 的版本信息。

```
/**
 * 获取SDK版本号。
 */
String version = mALinkServer.getVersion();
```

3.8 获取当前连接状态

获取当前的连接状态，返回值有 STATUS_ALINK_NONE_CONNECTED(未连接)，STATUS_ALINK_USB_CONNECTED(USB 模式连接)，STATUS_ALINK_USB_DISCONNECTED(USB 模式连接断开)，STATUS_ALINK_WIFI_CONNECTED(WiFi 模式连接)，STATUS_ALINK_WIFI_DISCONNECTED(WiFi 模式连接断开)五种状态。

```
/**
 * 获取当前连接状态。
 */
int status = mALinkServer.getConnectionStatus();
```

3.9 结束 SDK 使用

调用 ALinkServer 的 onCreate 方法之后，结束 SDK 使用时需要调用 onDestroy 方法释放资源：

```
/**
 * 销毁ALinkServer资源，结束调用。
 */
mALinkServer.onDestroy();
```