科大讯飞股份有限公司

IFLYTEK CO.,LTD.

科大讯飞 AIUI 集成指南

目 录

1. 概述

AIUI 客户端 Android 版本 SDK 由两部分组成：

1) AIUIService.apk，即 AIUI Service 服务包。安装在本地并随第三方 app 绑定启动，以 Android 的 Service 服务组件形式提供 AIUI 的各项服务，充当 AIUI 在终端上的代理；
2) AIUIServiceKit.jar ，即 AIUI Service 服务对应的开发套件包。由第三方 app 集成，通过 Binder 的 IPC 机制与 AIUI Service 绑定，调用其提供的功能。

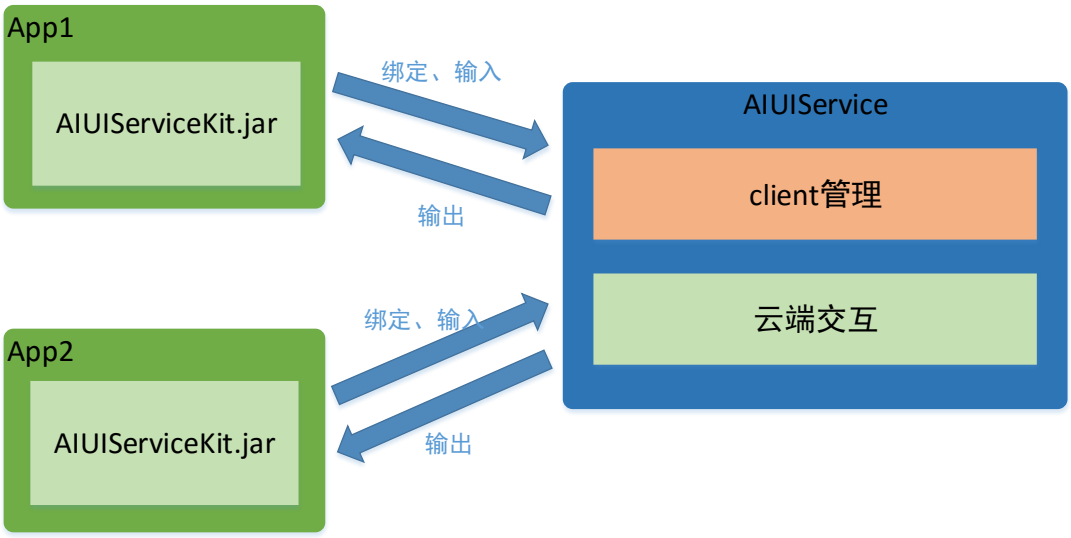

图-1 AIUI SDK 架构

AIUIService 与第三方 app 为一对多的关系，允许多个 app 集成 AIUIServiceKit.jar 来绑定和使用服务。需要注意的是，开发者需要处理好这些 app 的关系，保证某一时刻只有一个 app 向 AIUIService 进行输入，其他 app 只被动地接收并处理输出结果。

注：AIUIService 输出给每个 app 的结果都是一致的。

2. 预备工作

2.1. 导入 SDK

将开发工具包中 public/bin 目录下的 AIUIService.4.5.xxxx.xxxx.apk（以下简称 AIUIService.apk，“4.5.xxxx.xxxx”为版本号）安装到设备上，然后将 public/libs 目录下的 AIUIServiceKit.jar 复制到 Android 工程 libs 目录（如果工程无 libs 目录，请自行创建），即完成SDK导入。其中AIUIService.apk会随第三方app绑定启动，AIUIServiceKit.jar通过binder IPC 机制与之绑定并使用服务。

2.2. 权限说明

AIUI 已经在 AIUIService.apk 中添加 AIUI 服务所需的各项权限，开发者如需其他权限，请在工程的 AndroidManifest.xml 文件中按需求添加。

2.3. 混淆说明

如需在打包或者生成 APK 的时候进行混淆，请在 proguard.cfg 中添加以下代码，以防止 SDK 代码被混淆：

```
-keep class com.iflytek.**{*;}
-keepattributes Signature
```

3. SDK 集成

3.1. 参数设置

AIUI Service 采用 ContentProvider 组件的方式进行参数设置，即需要开发者自行创建一个 ContentProvider（可以创建在自己的应用中，也可以在其他应用中），AIUI Service 通过固定的 URI 访问该 ContentProvider 来获取参数设置。

参数设置的示例如下：

```
/* AIUI 参数设置 */
{
    /* 语音云平台登录参数 */
    "login":{
        "appid ":"xxxxxxxx",
        "key ":"xxxxxxxx"
    },

    /* 全局设置 */
    "global":{
        "scene":"main"
    },

    /* 交互参数 */
    "interact":{
        "interact_timeout":"60000",
        "result_timeout":"5000"
    },

    /* 业务相关参数 */
    // 语音合成参数
    "tts":{
        "res_type":"assets",
        "res_path":"tts/common.jet;tts/xiaoai_neutral.jet",
        "voice_name":"xiaoai"
    },

    // 唤醒参数
    "ivw":{
        "res_type":"assets",
        "res_path":"ivw/ivw_resource.jet"
    },
```

```
        /* 业务流程相关参数 *
        /* 硬件参数设置 */
        // alsa 录音参数
        "alsa":{
            "sound_card":"3",
            "card_sample_rate":"96000"
        },

        /* 日志设置 */
        "log":{
            "debug_log":"1",
            "save_datalog":"1",
            "datalog_path":"",
            "datalog_siz":1024
        },
}
```

整个参数配置存储在一个 JSON 字符串中，按类型组织，最外层的 key 即为类型名称，各字段的详细说明参见 4.1. AIUI 参数字段说明。

提供参数设置的 ContentProvider 示例如下：

```
// 提供参数设置的provider
public class AIUIConfigProvider extends ContentProvider {

    // 重写call方法
    @Override
    public Bundle call(String method, String arg, Bundle extras) {
        if ("readAIUICfg".equals(method)) {
            // 若是读取参数设置则将参数设置放在bundle对象的config字段返回
            Bundle bundle = new Bundle();

            String config = readAIUICfg();
            bundle.putString("config", config);

            return bundle;
        }

        return super.call(method, arg, extras);
    }
    // 读取参数设置，需要开发者自己实现，返回String格式的设置
    private String readAIUICfg() {
        // 从sdcard或其实地方得到参数设置config

        return config;
    }

}
```

最后，在 AndroidManifest.xml 中声明 ContentProvider：

```
<!-- AIUI配置provider -->
<provider
    android:name="XXX.AIUIConfigProvider"
    android:authorities="com.iflytek.aiui.cfg.provider"
    android:exported="true"
    android:multiprocess="false">
</provider>
```

该 provider 的 authorities 一定要声明为“com.iflytek.aiui.cfg.provider”，AIUI Service 会通过该 URI 来访问 provider。

注：在启动 AIUI Service 之前，要确保集成参数 provider 的应用已经安装。

3.2. 接口调用

AIUI 接口简单易用，调用示例如下：

```
// 1.首先创建AIUIListener监听器对象，在onEvent回调中处理AIUI服务抛出的各种事件。
private AIUIListener mAIUIListener = new AIUIListener() {

    @Override
    public void onEvent(AIUIEvent event) {
        switch (event.eventType) {
            case AIUIConstant.EVENT_BIND_SUCCESS:  // 绑定成功事件
            {…} break;
            case AIUIConstant.EVENT_WAKEUP:     // 唤醒事件
            {…} break;
            case AIUIConstant.EVENT_SLEEP:      // 休眠事件
            {…} break;
            case AIUIConstant.EVENT_RESULT:     // 结果事件
            {…} break;
            case AIUIConstant.EVENT_ERROR:      // 错误事件
            {…} break;
            case AIUIConstant.EVENT_VAD:        // VAD事件
            {…} break;

            default:
                break;
        }
    }
};

// 2.创建AIUIAgent的对象，传入应用上下文、AIUIListener监听器。
AIUIAgent mAIUIAgent = AIUIAgent.createAgent(MainActivity.this, mAIUIListener);

// 3.不断发送消息对象，以msgType字段区分类型，可携带参数和数据。
AIUIMessage message = new AIUIMessage(AIUIConstant.CMD_WRITE,
                                      0, 0, params, data);
mAIUIAgent.sendMessage(message);

// 4.当不再使用AIUI服务时，销毁AIUIAgent对象。
mAIUIAgent.destroy();
mAIUIAgent = null;
```

外部通过 AIUIMessage 消息载体向 AIUI Service 输入各种信息，如音频数据、控制指令等。AIUIMessage 中成员变量 msgType 字段标识消息类型，arg1、arg2 为两个扩展参数，params 为业务参数，data 为携带的输入数据。当 msgType 取值不同时，各字段有着不同的含义，具体说明见附录 4.2. AIUIMessage 类型说明。

3.3. 事件处理

AIUIEvent 事件是 AIUI 服务各种输出的载体，其中成员变量 eventType 字段标识事件类型，arg1、arg2 为两个扩展参数，data 为携带的输出数据，info 字段为数据的描述字段。当 eventType 取值不同时，各字段有着不同含义，具体说明见附录 4.3. AIUIEvent 类型说明。开发者需要重点关注以下几种事件：

1) 唤醒事件

用户通过说出唤醒词（如“叮咚叮咚”）成功唤醒 AIUI 时，SDK 会抛出 EVENT_WAKEUP 类型的唤醒事件，其中 info 字段为 JSON 格式的唤醒结果，有唤醒的角度、麦克风编号、得分等字段，具体格式参见 4.4.1 唤醒结果。

2) 结果事件

AIUI 服务从云端得到听写或语义理解结果时，会通过 EVENT_RESULT 类型的 AIUIEvent 事件抛出，其中 data 字段存储结果数据，info 字段为一个描述结果的 JSON 字符串。data 是一个 Bundle 对象（键的取值存储在 info 中，值为结果的二进制数据），结果解析即为从 info 中获取键值，然后从 data 中取出结果的过程。具体结果格式和解析方法参见 4.4.2. 听写结果、4.4.3. 语义结果。

3) 错误事件

出现错误时，AIUI 会抛出 EVENT_ERROR 类型的事件，arg1 字段为错误代码，info 字段为错误描述。当出现错误时，建议开发者先给相应的友好提示（如对 10120 提示“网络有点问题，请待会再试”），然后将 AIUI 重置到未唤醒状态（发送 EVENT_RESET_WAKEUP 类开的消息）以结束本次交互，以保证良好的用户体验。常见的错误码参见 4.5. 错误列表。

4. 附录

4.1. AIUI 参数字段说明

参数类型		参数名称	
login	语音云登录参数	appid	在讯飞开放平台上注册的 8 位应用唯一标识。
		key	appid 校验串。
global	全局参数设置	scene	用户定制的场景参数，不同的场景可对应不同的云端处理流程。
interact	人机交互参数	interact_timeout	交互超时（单位：ms），即唤醒之后，如果在这段时间内无有效交互（无有效结果返回），则重新进入待唤醒状态，取值：[10000,180000)，默认为 1min。
		result_timeout	结果超时（单位：ms），即检测到语音后端点后一段时间内无结果返回则抛出超时错误，默认值：5000。
tts	语音合成参数	engine_type	引擎类型，取值：local（本地），cloud（云端）。
		res_type	资源类型，取值：assets 资源（apk 工程的 assets 文件），res 资源（apk 工程的 res 文件），path 资源（sdcard 文件）。使用 ivw 唤醒时必须设置。
		res_path	资源路径，以“；”隔开，前面为合成共用资源，后面为发音人模型资源。
		voice_name	发音人名称。
ivw	语音唤醒参数	res_type	资源类型，取值：assets 资源（apk

			工程的 assets 文件），res 资源（apk 工程的 res 文件），path 资源（sdcard 文件）。使用 ivw 唤醒时必须设置。
		res_path	唤醒资源文件路径，必须与 res_type 匹配。
alsa	alsa（Advanced Linux Sound Architecture）录音参数	sound_card	声卡设备号，请根据实际情况设置，在使用麦克风阵列时必须设置正确的设备号。
		card_sample_rate	声卡采样率，请根据实际情况设置，在使用麦克风阵列时必须设置正确的采样率。
log	日志相关参数	debug_log	Debug 日志开关，取值：1（打开），0（关闭），默认值：0。日志打开时会向 logcat 打印调试日志。
		save_datalog	是否保存数据日志，取值：1（打开），0（关闭），默认值：0。打开之后，会将所有上传到云端的音频和云端返回的结果保存到本地。
		datalog_path	数据日志的保存路径，当不设置或者为空值时，使用默认值：“/sdcard/AIUI/data/”。
		datalog_size	数据日志的大小限制（单位：MB），取值：[-1，+∞)。默认值：-1（表示无大小限制）。注意：设置成-1 可能会造成 SD 卡被日志写满，从来导致 AIUI Service 性能下降，影响体验效果。

对于有默认值的参数，若参数中不存在对应的 key-value 设置，即表示取默认值。若某类型参数全部使用默认值，可以将该类型从配置文件中删除。

4.2. AIUIMessage 类型说明

msgType（消息类型）	取值	说明
CMD_STATE	1	查询服务状态。
CMD_RESET	4	重置 AIUI 服务的状态。服务会立即停止并重新启动，进入到待唤醒状态。
CMD_START	5	启动 AIUI 服务。当 AIUI 服务停止后，然后此命令启动服务。
CMD_STOP	6	停止 AIUI 服务。服务停止之后，将不响应外部输入。
CMD_RESET_WAKEUP	8	重置唤醒状态。AIUI 服务重置为待唤醒状态。
CMD_SET_BEAM	9	设置麦克风阵列的拾音波束。用 arg1 携带拾音波束号。
CMD_SET_PARAMS	10	设置参数配置。用 params 携带参数设置 JSON 字符串，具体格式参照 aiui.cfg 文件，配置在服务重置后生效。
CMD_SEND_LOG	12	发送应用日志到云端，可以帮助分析应用问题。需要将 JSON 格式的字符串放在 params 字段中携带。

4.3. AIUIEvent 类型说明

eventType（事件类型）	取值	说明
EVENT_RESULT	1	结果事件。data 字段携带结果数据，info 字段为描述数据的 JSON 字符串。
EVENT_ERROR	2	出错事件。arg1 字段为错误码，info 字段为错误描述信息。
EVENT_STATE	3	服务状态事件。arg1 字段取值为 STATE_IDLE（空闲状态）、STATE_READY（就绪状态）、STATE_WORKING（工作状态）状态之一。

EVENT_WAKEUP	4	唤醒事件。info 字段为唤醒结果 JSON 字符串，具体格式见 4.4.1. 唤醒结果。
EVENT_SLEEP	5	休眠事件。当出现交互超时，服务会进入休眠状态（待唤醒），抛出该事件。
EVENT_VAD	6	VAD 事件。当检测到输入音频的前端点后，会抛出该事件，用 arg1 标识前后端点或者音量信息：0（前端点）、1（音量）、2（后端点）。当 arg1 取值为 1 时，arg2 为音量大小。

4.4. 结果格式示例

4.4.1. 唤醒结果

```
{
    "power": 12342435436,       // 唤醒能量值
    "beam":3,                   // 拾音波束号，唤醒成功后阵列将在该波束方向上拾音
    "angle":180,                // 唤醒角度
    "channel":5,                // 唤醒声道，即麦克风编号，表示该声道的音频质量最好
    "CMScore":132               // 声道对应的唤醒得分
}
```

4.4.2. 听写结果

听写结果描述 JSON 字符串示例：

```
{
    "data": [{
        "params": {
            "sub": "iat"      // 标识这是听写结果
        },
        "content": [{
            "dte": "utf8",    // 结果编码
            "dtf": "json",    // 结果格式
            "cnt_id": "0"     // 听写结果在 data 中对应的键
        }]
    }]
}
```

听写结果（result）示例：

```
{
“text”: {"sn":1,"ls":true,"bg":0,"ed":0,"ws":[
      {"bg":0,"cw":[{"w":"北京","sc":0}]},
      {"bg":0,"cw":[{"w":"的","sc":0}]},
      {"bg":0,"cw":[{"w":"天气","sc":0}]},
      {"bg":0,"cw":[{"w":"怎么样","sc":0}]},
      {"bg":0,"cw":[{"w":"。","sc":0}]}]}
}
```

听写结果字段说明：

JSON 字段	英文全称	类型	说明
sn	sentence	number	第几句
ls	last sentence	boolean	是否最后一句
bg	begin	number	开始
ed	end	number	结束
ws	words	array	词
cw	chinese word	array	中文分词
w	word	string	单字
sc	score	number	分数

4.4.3. 语义结果

语义结果描述 JSON 字符串示例：

```
{
    "data": [{
        "params": {
            "sub": "nlp"    // 标识这是语义结果
        },
        "content": [{
            "dte": "utf8",    // 结果编码
            "dtf": "json",    // 结果格式
            "cnt_id": "0"    // 语义结果在 data 中对应的键
        }]
    }]
}
```

结果解析示例：

```
private AIUIListener mAIUIListener = new AIUIListener() {

    @Override
    public void onEvent(AIUIEvent event) {
        switch (event.eventType) {
        case AIUIConstant.EVENT_RESULT:
        {
            try {
                JSONObject bizParamJson = new JSONObject(event.info);
                JSONObject data = bizParamJson.getJSONArray("data")
                                              .getJSONObject(0);
                JSONObject params = data.getJSONObject("params");
                JSONObject content = data.getJSONArray("content").getJSONObject(0);

                if (content.has("cnt_id")) {
                    // 获取结果数据在data中的key
                    String cnt_id = content.getString("cnt_id");
                    // 通过key从data中获取结果
                    String resultStr = new String(event.data.getByteArray(cnt_id),
                        "utf-8");
                    // 通过key从params中获取sub字段
                    String sub = params.getString("sub");

                    if ("nlp".equals(sub)) {
                        // 语义结果格式请参考《Open Semantic Platform API
Documents》文档
                    } else if ("iat".equals(sub)) {
                        // 听写结果处理
                    }
            } catch (JSONException e) {
                e.printStackTrace();
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
    }
```

4.5. 错误列表

错误码	取值	含义
MSP_ERROR_INVALID_PARA	10106	参数名称错误
MSP_ERROR_INVALID_PARA_VALUE	10107	参数取值错误
MSP_ERROR_NOT_FOUND	10116	云端无对应的 scene 场景参数
MSP_ERROR_NO_RESPONSE_DATA	10120	结果等待超时
MSP_ERROR_LMOD_RUNTIME_EXCEPTION	16005	MSC 内部错误
ERROR_NO_NETWORK	20001	无有效的网络连接
ERROR_NETWORK_TIMEOUT	20002	网络连接超时
ERROR_NET_EXPECTION	20003	网络连接发生异常
ERROR_INVALID_RESULT	20004	无有效的结果
ERROR_NO_MATCH	20005	无匹配结果
ERROR_AUDIO_RECORD	20006	录音失败
ERROR_COMPONENT_NOT_INSTALLED	21001	没有安装服务组件
ERROR_SERVICE_BINDER_DIED	21020	与服务的绑定已消亡
ERROR_LOCAL_NO_INIT	22001	本地引擎未初始化
ERROR_UNKNOWN	20999	未知错误