AIUI 模块串口开发指南

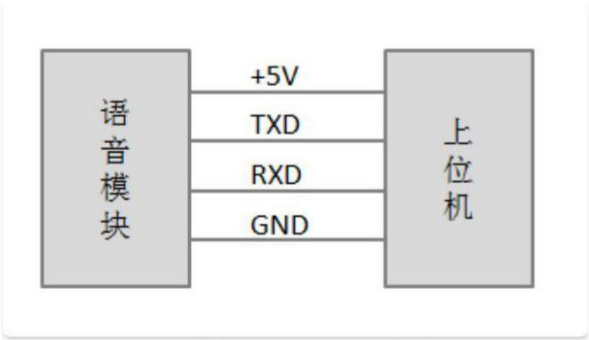
1.概述

上位机和 AIUI 语音模块之间通过串口进行数据交互。

上位机可以通过串口向 AIUI 语音模块发送握手消息同步状态，发送 WIFI 配置信息，发送 AIUI 配置消息，发送控制消息控制 AIUI 状态。

2.通信协议

2.1 通讯接口



通讯接口引脚图

2.2 通信参数

波特率：115200

数据位：8

停止位：1

奇偶校验：无

流控：无

2.3 消息格式

字节	值	含义说明
0	0xA5	同步头
1	0x01	用户 ID
2	0xXX	消息类型
3~4	0xXXXX	消息数据长度
5~6	0xXXXX	消息 ID
7~n	消息数据	消息数据
n+1	0xXX	内部校验码

注 消息类型和消息数据的具体定义请参照下面通信数据格式定义。

2.4 通信规则

1. 双向通信，进行一问一答式通信；
2. 不分主从机，双方都可向对方发送请求信息；
3. 双方发送握手信号，确定与对方串口通信是否正常；在未接收到响应时，可以每隔 100ms 发送一次握手请求信号；
4. 接收方接到请求后需在 50ms 内发送响应；
5. 发送方发送完成后若超过 300ms 未得到响应则判定为超时进行重发，重发次数为 3 次。

3.通信格式

通信格式目前定义了握手消息，AIUI 配置消息，WIFI 配置消息，AIUI 消息，主控消息，确认消息。下面是分类详述：

分类	同步头	用户ID	消息类型	消息长度	消息ID	消息数据						校检码
字节	0	1	2	3~4	5~6	7	8	9	10			11
握手请求	0xA5	0x01	0x01	*见附注*	*见附注*	0xA5	0x00	0x00	0x00			*见附注*
字节	0	1	2	3~4	5~6	7	8	9	10	11~m	m+1~n	n+1
WIFI配置	0xA5	0x01	0x02	同上	同上	状态	加密方式	SSID长度	password长度	SSID	password	同上
字节	0	1	2	3~4	5~6	7~m						m+1
AIUI配置	0xA5	0x01	0x03	同上	同上	AIUI 配置						同上
字节	0	1	2	3~4	5~6	7~m						m+1
AIUI消息	0xA5	0x01	0x04	同上	同上	AIUI 消息（gzip 压缩）						同上
字节	0	1	2	3~4	5~6	7~m						m+1
主控消息	0xA5	0x01	0x05	同上	同上	主控消息						同上
字节	0	1	2	3~4	5~6	7	8	9	10			11
确认消息	0xA5	0x01	0xff	同上	同上	0xA5	0x00	0x00	0x00			同上

消息长度：

*数据格式中 3~4 字节为**消息数据**长度，消息长度编码为小端模式，即第 3 字节存储低字节，第 4 字节存储高字节。*

如握手请求消息中消息数据有 4 字节，则消息长度为 4，编码到 3~4 字节就是 0x04 0x00

消息 ID：

数据格式中 5~6 字节为消息 ID，与消息长度类似，也是小端模式编码。

可以使用消息 ID 过滤因超时重发导致的重复消息。两字节长度的消息 ID 取值 0-65535，所以在实际使用中需要循环使用，具体实现可以参考 Android 平台上 的源码实现。

校检码：

数据格式中每种数据类型的最后一个字节都是校检码，用于检验串口传输的正确性。其计算方式为除校检码字节外所有字节求和取反并加 1。公式如下：

```
checkcode = ~sum(字节 0+字节 2+...+字节 n) + 1
```

编码格式：

消息中字符串类型的数据编码格式为 UTF-8。

AIUI 消息格式中的数据采用 GZIP 压缩格式，压缩前的编码格式也是 UTF-8 格式。

3.1 确认消息

确认消息是一个特殊的消息类型，它是对其他类型消息的确认，它的消息 ID 与其要确认的消息的 ID 相同。

如一个 AIUI 消息的消息 ID 为 0x9527，那对应的确认消息的消息 ID 也应该是 0x9527。

3.2 WIFI 配置

WIFI 配置结果中，状态取值：

- 0：从机当前与路由连接

加密方式分为三类：

- 0：OPEN
- 1：WEP
- 2：WPA

3.3 AIUI 配置

AIUI 配置的格式为 JSON。支持配置 appid，key，场景，是否启动 AIUIProductDemo（解析 AIUI 结果，进行播报的 APP）。

配置 appid，key，场景等示例如下：

```
{
    "type": "aiui_cfg",
    "content": {
        "appid": "appid",
        "key": "key", "scene":
        "main", "launch_demo":
        false
    }
}
```

注：配置项字段取值参见《AIUI 集成指南》中 AIUI 参数设置部分说明

3.4 主控消息

主控消息的格式为 JSON。
主控消息根据内部字段 type 的不同，有不同的控制功能。

- type 为 aiui_msg，发送 AIUI 控制消息

```
{

    "type": "aiui_msg",
    "content": {
        "msg_type": 8, //CMD_RESET_WAKEUP 重置 AIU 唤醒状态
        "arg1": 0,
        "arg2": 0,
        "params": ""
    }
}
```

注：各个字段取值具体说明参见《AIUI 集成指南》中 AIUIMessage 类型说明部分

- type 为 voice，控制 AIUI 声音播放

```
{
    "type": "voice",
    "content": {
        "enable_voice":true/false   // 是否禁止 AIUI 声音播放
    }
}
```

- type 为 status，通过 query 字段查询不同状态（目前仅支持查询 WIFI 状态）

```
{
    "type": "status",
    "content": {
        "query": "wifi" // 查询 AIUI WIFI 状态信息
    }
}
```

- type 为 save_audio，控制 AIUI 保存原始音频，通过 save_len 指定保存音频时长，单位为秒。

```
{
    "type": "save_audio",
    "content": {
        "save_len": 10
    }
}
```

- type 为 tts，发送文本让 AIUI 开始合成播放或者停止合成播放
    - 开始合成命令

```
{
    "type": "tts",
    "content": {
        "action": "start", //开始合成
        "text":"xxx" //需要合成播放的文本
    }
}
```

- 停止合成命令

```
{
    "type": "tts",
    "content": {
        "action": "stop",  //停止合成
    }
}
```

3.5 AIUI 消息

AIUI 消息原始内容格式为 JSON，但是为了传输的效率，实际内容采用了 GZIP 压缩格式。

AIUI 消息根据 type 的不同，解析不同的数据。

1. type 为 wifi_status，代表 WIFI 状态查询返回，示例如下：

```
{
    "type": "wifi_status",
    "content": {
        "connected": true/false,  //AIUI WIFI 查询状态信息
        "ssid": "connected_ssid"  //当 connected 为 true 时，此字段表示当前连接的 wifi 名称
    }
}
```

2. type 为 aiui_event，代表为 AIUI 语音结果返回，总体结构示例：

```
{
    "type": "aiui_event",
    "content": {
        "eventType":1,  //事件类型
        "arg1":0,       //参数 1
        "arg2":0,       //参数 2
        "info":{},      //描述信息
        "result":{}     //结果
    }
}
```

注：具体字段参考《AIUI 集成指南》中 AIUIEvent 说明部分

3. type 为 tts_event，表示合成事件

- 合成开始事件

```
{
    "type": "tts _event",
    "content": {
        "eventType": 0, // 合成开始事件
    }
}
```

- 合成结束事件

```
{
    "type": "tts _event",
    "content": {
        "eventType": 1, //合成结束事件
        "error": ttsErrorCode //当发生错误时 error 字段代表合成错误码
    }
}
```

4.Android 平台实现

根据上述的通信协议和通信格式，在 Android 平台进行了实现以供开发参考。
如果上位机平台是 Android 平台，可以直接集成使用。

4.1 权限

请确保应用有读写对应串口设备的权限（即对/dev/下的串口设备文件有 rw 权限）。

4.2 调用流程

调用主要接口类是 UARTAgent。

在程序首次初始化的地方调用静态方法 createAgent 创建 UARTAgent 实例，传入 EventListener 参数用于接收串口事件，后面调用创建的 UARTAget 实例的 sendMessage 方法发送串口消息，在程序结束前调用 UARTAgent 实例的 destroy 方法释放资源。

4.3 接口说明

- 创建 UARTAgent:

```
UARTAgent createAgent(String device, int speed, EventListener listener)
```

  - *device 串口设备名（如/dev/ttyS2）*
  - *speed 串口速率 (串口速率一般为 115200)*
  - *listener 串口事件监听器*

- 发送串口消息

```
boolean sendMessage(MsgPacket reqPacket)
```

  - *reqPacket 串口数据包*

- 销毁串口，释放资源

```
void destroy()
```

4.4 串口数据包

实现根据通信数据格式对串口数据类型进行了封装，MsgPacket 是所有类型串口数据包的父类。

实现包装的串口数据类型如下：

- *AIUIPacket AIUI 消息*
- *AIUIConfPacket AIUI 配置*
- *ControlPacket 主控消息*
- *WIFIConfPacket WIFI 配置*

注：各个类型中字段的定义参见附录

实现中在 PacketBuilder 类中提供了静态方法构造上面列举的数据包：

- 构造 AIUI 配置请求数据包

```
MsgPacket obtainAIUIConfPacket(String appid, String key, String sence, boolean launchDemo)
```

- 构造保存音频请求数据包

```
MsgPacket obtainAIUIAudioRecordCmdPacket()
```

- 构造获取 WIFI 状态请求数据包

```
MsgPacket obtainWIFIStatusReqPacket()
```

- 构造声音控制请求数据包

```
MsgPacket obtainVoiceCtrPacket(boolean enable)
```

- 构造 AIUI 控制请求数据包

```
MsgPacket obtainAIUICtrPacket(int msgType, int arg1, int arg2, String params)
```

- 构造 WIFI 配置结果数据包

```
MsgPacket obtainWIFIConfPacket(WIFIStatus status, EncryptMethod type, String ssid, String pas
swd)
```

4.5 串口事件

在上面创建 UARTAgent 实例中，传入的 EventListener 类型的参数 listener 用于监听接收串口事件。

EventListener 的类型定义如下：

```
interface EventListener {
    void onEvent(UARTEvent event);
}
```

接口方法 onEvent 中 UARTEvent 即为串口事件，定义如下：

```
class UARTEvent {
    int eventType; //串口事件类型
    Object data; //串口事件数据
}
```

在 eventType 取不同的事件类型是，串口事件数据也有不同的数据类型。

定义的串口事件类型和对应事件数据如下：

事件说明	事件类型	数据类型	数据说明
初始化成功	EVENT_INIT_SUCCESS	无	无
初始化失败	EVENT_INIT_FAILED	无	无
串口消息	EVENT_MSG	MsgPacket	接收到的数据包
消息发送失败	EVENT_SEND_FAILED	MsgPacket	发送失败的数据包

*注 具体事件处理参见代码示例

4.6 代码示例

下面代码示例了上位机集成使用的通用流程：

```
mAgent = UARTAgent.createAgent("/dev/ttyS2", 115200, new EventListener() {

    @Override
    public void onEvent(UARTEvent event) {
        switch (event.eventType) {
            case UARTConstant.EVENT_INIT_SUCCESS:               //处理初始化成功事件
                Log.d(TAG, "Init UART Success");
                break;
            case UARTConstant.EVENT_INIT_FAILED:                //处理初始化失败事件
                Log.d(TAG, "Init UART Failed");
                break;
            case UARTConstant.EVENT_MSG:                        //消息回调事件
                MsgPacket recvPacket = (MsgPacket) event.data;
                processPacket(recvPacket);
                break;
            case UARTConstant.EVENT_SEND_FAILED:                //消息发送失败事件
                MsgPacket sendPacket = (MsgPacket) event.data;
                mAgent.sendMessage(sendPacket);
                break;
            default:
                break;
        }
    }
});
//消息处理
private void processPacket(MsgPacket packet)
    { switch (packet.getMsgType()) {
        case MsgPacket.AIUI_PACKET_TYPE:
            Log.d(TAG, "recv aiui result" + new String(((AIUIPacket) packet).content));
            break;
        default:
            break;
    }
}

//发送 AIUI 配置信息
if(intent.hasExtra("aiui_conf")){
    mAgent.sendMessage(PacketBuilder.obtainAIUIConfPacket("appid", "key", "main", false));
}
```

Ⅰ.附录

所有串口数据包的父类都是 *MsgPacket*，方法 getMsgType()返回数据包类型。

MsgPacket 中定义的数据包类型如下：

- *MsgPacket.WIFI_CONF_TYPE*
- *MsgPacket.AIUI_CONF_TYPE*
- *MsgPacket.AIUI_PACKET_TYPE*
- *MsgPacket.CTR_PACKET_TYPE*

各个数据包类型成员定义：

- **AIUI 消息**

```
class AIUIPacket {          //AIUI 语音识别数据
    String content
}
```

- **AIUI 配置**

```
class AIUIConfPacket {
    String config            //AIUI 配置信息
}
```

- **WIFI 配置**

```
class WIFIConfPacket {
    WIFIStatus status;        //WIFI 连接状态
    EncryptMethod encrypt;    //WIFI 加密方式
    String ssid;              //WIFI 名称
    String passwd;            //WIFI 密码
}
```

- **主控消息**

```
class ControlPacket {
    String controlCMD         //主控控制消息
}
```