AC51 项目

—WiFi 控制平台(内网)Android 开发说明

版本：2.6

2016-04-19

声 明

1） 本项目所参考、使用技术必须全部来源于公知技术信息，或自主创新设计。

2） 本项目不得使用任何未经授权的第三方知识产权的技术信息。

3） 如个人使用未经授权的第三方知识产权的技术信息，造成的经济损失和法律后果由个人承担。

目 录

版本历史

日 期	版本号	注释	作者
2015-01-09	V1.0	建立初始版本	慧玲
2015-03-20	V2.0	增加广域网控制并优化	慧玲
2015-06-25	V2.1	添加设备名称	慧玲
2015-07-13	V2.2	完善搜索设备控制	慧玲
2015-08-31	V2.3	增加命令/数据打包和一对多	海波
2015-10-16	V2.4	增加喜玛拉雅功能	卓成
2015-11-03	V2.5	整理文档资料	海波、卓成
2016-04-19	V2.6	修复 android 6.0 以上不兼容问题	卓成

1 文档目的

本文档用于定义 AC51 系列 WIFI 产品，在进行网络控制数据传输时，遵从的数据序列格式（即传输协议）。为开发者提供方便的平台，通过以下的 API 可以编辑不同的设备控制 Android 产品。

2 接入流程

2.1 依赖的第三方 jar 包及 aar 包：

volley.jar ： 安卓开源网络异步下载库
ormlite-android-4.48.jar ： ORM 安卓开源开发包

ormlite-core-4.48.jar ： ORM 源码开源开发包
PS：使用喜玛拉雅 FM 功能，还需要增加喜玛拉雅官方提供的 jar 包和第三方 jar 包，分别为以下 jar 包：
gson-2.2.4.jar
jsoup-1.8.3.jar
litepal_xm.jar
okhttp-2.4.0.jar
okhttp-urlconnection-2.2.0.jar
okio-1.4.0.jar
TingPhoneOpenSDK_1.0.0.4.jar
viewinjected.jar

该工程的运行环境为 android studio ， 使用的开发包是 aar 格式：
jl_libwifi.arr ： Wifi 控制平台（内网）开发包

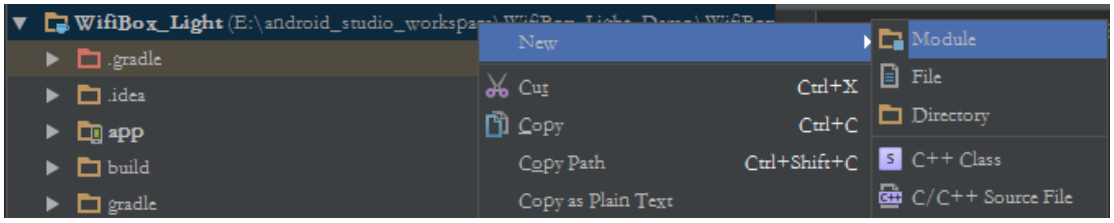
2.2 配置应用工程的 AndroidManifest.xml

2.2.1 添加权限

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission
android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<!-- cache功能需要读写外部存储器 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission
android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<!-- android 6.0 以上使用 wifi 需要申请的权限 -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"/>
```

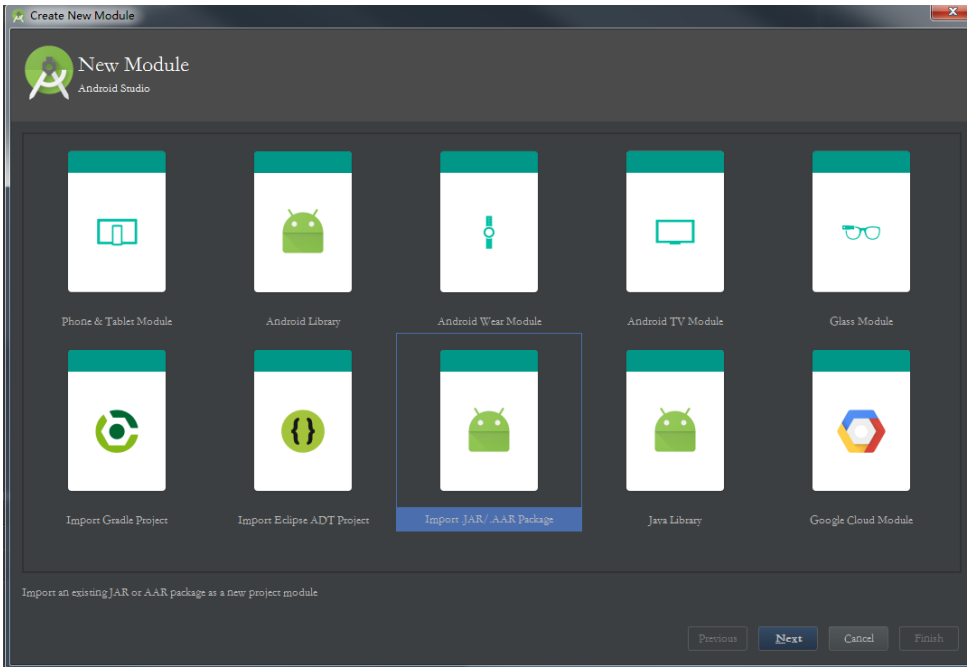
2.3 aar 开发包导入步骤

2.3.1 创建 Module 模式



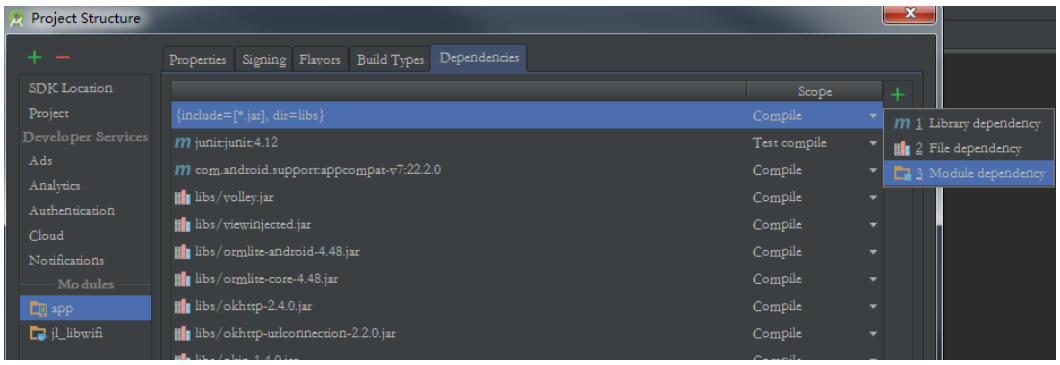
右击主工程，选择 New，选择 Module。

2.3.2 选择导入 JAR/AAR 包



选择导入 JAR/AAR 包，选择 aar 包的路径。Aar 包的路径是\WifiBox_Light 开发资料\libs\jl_libwifi.aar，项目名称可以修改，默认为 jl_libwifi，点击 Finish，完成导入 aar 包。

2.3.1 关联 AAR 包



右击主工程，选择 Open Module Settings，如上图操作，关联对应的 aar 包。选择 jl_libwifi 的 Module 项目，点击 Ok，即可关联 aar 包成功。

3 开发说明

开发 WiFi 控制平台（内网）的安卓程序，需要导入本平台的开发库：jl_demowifi_lib.jar。

3.1 设备控制开发说明

3.1.1 实例化 DeviceControl 对象

调用接口：

```
Public DeviceControl(Context c, Handler devHandler, long HeartBeatTimes)
```

接口说明：

用于构造 DeviceControl 类的对象，方便调用其中的函数与设备通讯，建议在 Application 开始时就调用该构造函数。

参数说明：

c 上下文对象

devHandler 用于数据接收，更新应用程序 UI 的 Handler 对象

HeartBeatTimes 心跳包触发的时间间隔 （可忽略，若忽略，则默认为 3 秒）

示例代码：

```
mControl = new DeviceControl(context, mHandler, 7000L);
```

3.1.2 搜索设备控制

调用接口：

```
public void startSearchDevices()
```

接口说明：

通过收发 UDP 广播搜索该网络下的设备。

参数说明：

无

示例代码：

```
mControl = new DeviceControl(context, mHandler, 7000L);
mControl.startSearchDevices();
```

3.1.3 停止搜索设备控制

调用接口：

```
public void stopSearchDevices()
```

接口说明：

停止搜索该网络下的设备。

接口参数：

无

示例代码：

```
mControl. stopSearchDevices ();
```

提示：虽然停止了搜索设备，但设备会每 30 秒广播 UDP 数据通知应用程序更新设备的状态。

3.1.4 连接设备控制

调用接口：

```
public void addDevice(int version, String username ,String uuid,
String ip, int port)
```

接口说明：

用于发送数据到设备，请求连接设备。

接口参数：

version	版本号
username	用户名
uuid	设备 id
ip	设备 ip
port	设备 port

示例代码：

```
mControl.addDevice(version, username, uuid, ip, port);
```

3.1.5 激活设备控制

调用接口：

```
public void sendActKeyCmd(int version ,String username,String
uuid, String init_codekey)
```

接口说明：

用于激活已连接的设备。

接口参数：

version	版本号
username	用户名
uuid	设备 id
init_codekey	设备密码

示例代码：

```
mControl.sendActKeyCmd(version,        username,        uuid,
init_codekey);
```

3.1.6 发送命令控制

调用接口：

```
public void sendCmdBuf(int version ,String username ,String uuid ,int devClass ,int num ,short devCmd ,byte[] data ,boolean bufNoEncode)
```

接口说明：

用于与设备通讯，发送命令数据。

参数说明：

version	版本号
username	用户名
uuid	设备 id
devClass	设备类别
num	设备数量
devCmd	设备命令
data	发送数据
bufNoEncode	数据是否需要转码 （可忽略，默认为不需要转码）

devCmd 参数定义：

```
/** Net radio play **/
int NET_RADIO_PLAY = 0x09;
/** Net radio pause **/
int NET_RADIO_STOP = 0X0B;
/** set device name **/
short SET_DEVICE_NAME = 0x0C;
 /** get device name **/
short GET_DEVICE_NAME = 0x0D;
```

示例代码：

```
mControl.sendCmdBuf(version, username, uuid, devClass, num, devCmd, data, bufNoEncode);
```

3.1.7 数据接收，通过 Handler 方式更新 UI

参数说明：

```
int KEY_STATUS_MSG = 0x1005;
```

说明：接收小机激活以及修改密码状态，1 为激活成功，2 为修改密码成功

```
int ERROR_INFO = 0x2005;
```

说明：错误信息，以下为错误类型及注释

```
/** -6, Please and open your device Wifi */
int ERR_INFO_OPEN_WIFI = -6;
/** -5, please login ! */
int ERR_INFO_LOGIN = -5;
/** -4, not exist this devices */
int ERR_INFO_DEV_NOT_EXIST = -4;
/** -3, can not find user */
int ERR_INFO_USER_NOT_FOUND = -3;
/** -2, send error */
int ERR_INFO_SEND_DATA = -2;
/** -1, Packets error */
int ERR_INFO_PACKETS = -1;
/** 0, Error for send CMD/buf */
int ERR_INFO_CMD_BUF = 0;
/** 12, Device had connected*/
int ERR_INFO_DEV_CONNECTED = 12;
/** 16, Packaged data format error */
int ERR_INFO_DATA_FORMAT = 16;
/** 18, Error message returned from the server */
int ERR_INFO_SERVER_MSG = 18;
/** 22, Protocol error */
int WAN_ERR_INFO_ERROR_TYPE = 22;
/** 23, This device does not exist on the server */
int WAN_ERR_INFO_DEV_NOT_EXIST = 23;
/** 24, User name is not “guest” */
int WAN_ERR_INFO_LOGIN_NAME = 24;
```

```
int DEVS_MSG = 0x1002;
```

说明：接收所有在线设备返回的数据，这里指的是该局域网下所有在线设备的基本信息。每 30 秒设备自动发送一次，用于更新搜索设备列表信息，保持设备活性。

设备的基本信息定义：

```
String DB_uuid          设备的 uuid
int DB_version          设备的版本号
String DB_ip            设备的ip地址
int DB_port             设备的端口号
int DB_Tag              设备标识
int[] DB_classbuf       设备类
```

```
int[] DB_numbuf          设备类的个数
String DB_uuid32         设备32位的uuid
int DB_devclass          设备的类型
int DB_DevNum            设备的数量
short DB_DevCMD          设备的命令
byte[] devData           设备数据
String DB_pwd            设备密码
boolean spConfigure      一键配置标记位 true为成功，false为失败
String DB_deviceName     设备名
boolean devApSp          设备模式,AP/SP
```

参数说明：以上参数均提供 gett 和 sett 的函数来设置和获取数据，DB_Tag > 1 时，设备为复合设备。

```
int CONN_STATUS_MSG = 0x1004
```

说明：接收设备连接状态参数，从属 msg.what，注意，LAN 才有设备 ID，WAN 没有。

状态参数定义：

```
/** 局域网连接成功*/
int LAN_CONN_SUCCESS = 3;
/** 局域网连接失败*/
int LAN_CONN_FAILURE = 4;
```

通过连接状态可以判断设备是否激活成功，是否在线。

```
int DEV_CMD_MSG = 0x1003;
```

说明：接收激活设备返回的信息，这里指的是已激活的设备命令返回的信息。

返回数据的参数：

```
//命令标记位
private final String BUF = "tcp_app_buf";
private final String CMD = "tcp_app_cmd";
private final String DEV_NUM = "tcp_num";
private final String DEV_CLASS = "tcp_class";
private final String T_UUID = "tcp_uuid";

Bundle b2 = (Bundle) msg.obj;
byte[]buf = b2.getByteArray(BUF);  //返回数据
int dev_cmd = b2.getShort(CMD);     //返回的命令
```

```
int devClass = b2.getInt(DEV_CLASS); //设备类
int num = b2.getInt(DEV_NUM);       //设备序号
String returnUuid = b2.getString(T_UUID); //设备 uuid
```

注意事项：devClass 值为 65533 时，说明是固件升级返回的信息。

dev_cmd 命令定义：

```
//固件升级开始
int NET_UPDATE_FIRMWARE = 0x12;
//请求固件升级
int NET_UPDATE_FIRMWARE_TRANSFER_REQUEST = 0x13;
//固件升级校验成功
int NET_UPDATE_FIRMWARE_TRANSFER = 0x14;
//固件升级结果
int NET_UPDATE_FIRMWARE_RESULT = 0x15;
//固件升级停止
int NET_UPDATE_STOP = 0x16;
```

固件升级错误定义：

```
int NET_UPDATE_FIRMWARE_RESULT_OK =0x0;  //固件升级成功
int NET_UPDATE_NO_MEM = 0x1;            //设备内存不足
int NET_UPDATE_HEAD_FLAG_ERR = 0x2     //头部标记错误
int NET_UPDATE_HEAD_CRC_ERR = 0x3;     //头部CRC校验错误
int NET_UPDATE_KEY_CRC_ERR = 0x4;      //尾部CRC校验错误
int NET_UPDATE_DATA_CRC_ERR = 0x5;     //数据CRC校验错误
int NET_UPDATE_DATA_CHECK_ERR = 0x6;   //数据格式错误
int NET_UPDATE_FILE_SEEK_ERR = 0x7;    //搜索文件错误
int NET_UPDATE_READ_DATA_ERR = 0x8;    //读取数据错误
int NET_UPDATE_READ_SPI_DATA_ERR = 0x9; //读取SPI数据错误
int NET_UPDATE_OPEN_UPFILE_ERR = 0xA; //打开文件错误
int NET_UPDATE_FIND_UPFILE_ERR = 0xB; //搜索文件错误
int NET_UPDATE_NO_SUPP_FILE = 0xC;     //不支持文件错误
int NET_UPDATE_ADDR_ERR = 0xD;          //读写地址错误
int NET_UPDATE_UBOOT_CHANGE_ERR = 0xE //UBOOT转换错误
int NET_UPDATE_PARAM_ERR = 0xF;         //PARAM错误
int NET_UPDATE_DATA_TOO_MANY = 0x10;   //数据太多
int NET_UPDATE_CALLBACK_ERR = 0x11;    //回调错误
int NET_UPDATE_MSG_TO_APP_ERR = 0x12; //发送消息到APP错误
int NET_UPDATE_WIFI_CFG_NO_SPACE = 0X80; //Wifi的CFG缺少空间
int NET_UPDATE_WIFI_CFG_NO_CHANGE = 0X81; //Wifi的CFG没有改变
int NET_UPDATE_FILE_THE_SAME_VERSION = 0X82; //相同版本错误
int NET_UPDATE_DATA_OVER_LIMINT = 0X83;  //数据超出范围
int NET_UPDATE_KEY_ERR = 0X84;          //设备 Key 值错误
```

3.1.8 删除指定的设备

调用接口：

```
public void deleteDevice(int  version ,String username,String uuid)
```

接口说明：

删除指定的已激活设备。断开连接并在设备列表删除。

参数说明：

version	设备版本号
username	用户名
uuid	设备 uuid

示例代码：

```
mControl.deleteDevice(version, username, uuid);
```

3.1.9 断开指定的连接设备

调用接口：

```
public void disconnDevice(int  version ,String username,String uuid)
```

接口说明：

断开指定的已连接设备的通讯，并不在设备列表中删除。

参数说明：

version	设备版本号
username	用户名
uuid	设备 uuid

示例代码：

```
mControl.disconnDevice(version, username, uuid);
```

3.1.10 删除所有设备

接口调用：

```
public void deleteAllDevices(int version,String username)
```

接口说明：

断开连接并删除所有设备。该用户存储的所有设备信息。

参数说明：

version	设备版本号
username	用户名

示例代码：

```
mControl.deleteAllDevices(version, username);
```

3.1.11 断开所有设备连接

调用接口：

```
public void disconnAllDevices()
```

接口说明：

断开所有连接的设备。

参数说明：

无

示例代码：

```
Control.disconnAllDevices();
```

3.2 一键配置

1、onCreate()下添加如下代码

```
//使用默认端口

mTaskManager = new TaskManager(getApplicationContext());
```

2、一键配置的收发数据

2.1 发送一键配置数据：

```
// 发送,会循环发送，直到调用停止函数stop
send(String SSID, String password)

//发现配置成功，请调用停止发送函数
stop()
```

3.3 控制命令的解析和封装

前面部分所使用到的收/发命令、收/发数据操作都需要经过解析或封装，解析出来的信息用于更新 APP 的状态和数据，封装 APP 的数据用于响应或控制设备。下面的 API 都可以通过生成 WifiBoxHub 对象来使用:

例如 mWifiBoxHub = new WifiBoxHub(this.getApplicationContext());

---

public void setMsgHandler(Handler handler)

当设备端有状态或数据更新时会通过传进去的参数 handler 发出来。因此，需要更新 UI 的类都可以传 Handler 的对象进去。

参数：handler 用于传递消息。

返回：无

---

public void tryToSendControlCmd(short cmd, byte[] param)

向设备发送命令。

参数：

cmd：设备命令，相关命令查看 Flags 类。

param：配合 cmd 命令设置音量或模式

返回：无

---

public List<PlayModeInfo> getPlayModeInfoList()

获取当前播放模式标识和模式名称列表

参数：无

返回：模式标识和模式名称列表

---

public byte getDevicePlayMode()

获取设备当前所在模式。
参数：无
返回：模式标识

---

public byte getDevicePlayStatus()

获取设备当前所处状态，有忙碌、暂停、播放等状态，更多请查看 `Flags` 类。
参数：无
返回：设备当前状态

---

public String getPlayModeName(int mode)

获取设备当前所在模式的名称。
参数：模式标识
返回：模式名称

---

public byte getDevicePlayVolume()

获取设备的当前播放音量。
参数：无
返回：设备音量

---

public void tryToGetRtcTime()

获取设备的 RTC 时间，数据将通过 handler 传送回来。
参数：无
返回：无

---

public void tryToSetRTC(RtcInfo info)

设置设备的 RTC 时间。
参数：info 为设置的时间信息
返回：无

---

public void tryToGetRtcAlarm()

获取设备的 RTC 模式下的闹钟信息，数据将通过 handler 传送回来。
参数：无
返回：无

---

public List<Map<String, Integer>> getRadioStationInfoList()

获取收音模式的电台序号和频率的列表。
参数：无
返回：电台序号和频率的列表

---

public short getRadioPlayingFrequency()

获取设备收音模式的当前播放频率。
参数：无
返回：频率

---

public ID3v2Info getPlayingId3Info()

获取正在播放的歌曲的 ID3 信息
参数：无
返回：歌曲的 ID3 信息

---

public byte getMusicUsbSdMode();

获取当前浏览处于USB还是SD的标识。
参数：无
返回：USB 或 SD 的标识

---

public byte getMusicCycleMode();

获取音乐模式下当前播放模式，更多请查看Flags类定义。
参数：无
返回：播放模式标识

---

public void createBrowserBond();

说明：创建搜索浏览绑定
参数：无
返回：无

---

public String getShownCompletePath(int id)

根据ID获取设备当前所处的完整路径。
参数：id 为对应的路径唯一标识
返回：设备当前所处的完整路径

---

public boolean requestRemoteUpdateFileNames()

请求设备刷新文件列表数据
参数：无
返回：请求是否成功响应

---

public void releaseBrowserBond()

释放搜索浏览绑定
参数：无
返回：无

FilePathItem 参数说明：

```
int   mId;          //当前浏览的簇号
int   mParentId;    //上一级浏览的簇号
boolean   mIsPath; //类型标记，0为文件，1为文件夹；
byte      mNameCode;//编码格式，0为unicode,1为GBK
int       mFileNumber;//文件序号
String    mShowName; //显示名
short     mNameCrc;  //名字CRC校验码
short  mSubItemsCrc; //尾部CRC校验码
byte[]    mRawDataBuf; //数据
boolean   mIsLongName; //长名字标记
boolean   mIsJustVisit;//浏览过的标记，未浏览过，向设备请求数据，已
//浏览，向数据库请求数据。
```

---

public FilePathItem getDefaultPathItem()

获取设备根目录的文件目录信息。
参数：无
返回：根目录的文件目录信息。

---

public List<FilePathItem> getFileNamesList(int parent_id)

根据路径标识获取指定目录下的文件信息列表。
参数：parent_id 路径标识
返回：指定目录下的文件信息列表

---

public FilePathItem getFilePathItem(int id)

根据簇号标识获取指定的文件信息。
参数：id 簇号标识
返回：指定簇号对应的文件信息

public int tryToPlayFile(int id)

请求设备播放指定的音乐文件。

参数：id 为音乐的唯一标识

返回：成功能返回 READ_DIR_STATUS_OK；失败则返回 READ_DIR_STATUS_ERROR

public int tryToGetFileNamesList(int path_id, boolean force)

请求返回上一级的目录文件列表信息。

参数：

path_id：上一级浏览的簇号

force： 为 false 时，从数据库读取数据，为 true 时，从设备获取文件列表

返回：成功能返回 READ_DIR_STATUS_OK；失败则返回 READ_DIR_STATUS_ERROR；请求需要时间则返回 READ_DIR_STATUS_WAIT。

public byte[] getCSWFromCBW(byte[] buf)

仅当手机端激活多个设备时，用于回复非当前设备的请求，可通过 sendCmdBuf()发送出去。

参数：buf 为接收到的数据包。

返回：封装好的 CSW 数据。

public void onDataReceived(byte[] buf)

用于解析设备发过来的数据包，主要用于ID3的数据解析。

参数：buf 为设备发过来的数据包

返回：无

public interface OnPackagedListener

- void onCBWPackaged(byte[] buf)

  当数据封装好时，回调此方法。可通过 sendCmdBuf()发出去。

  参数：buf 为封装好的数据包

  返回：无

- void onCSWPackaged(byte [] buf)

  当数据封装好时，回调此方法。可通过 sendCmdBuf()发出去。

  参数：buf 为封装好的数据包

  返回：无

说明：与设备交互的数据都需要先封装好才能发出去。

3.4 一对多开发说明

要开发手机端“一对多”设备控制应用，就要用到“一对多”的接口或方法。很简单，只需生成**MultideviceHub**对象，就可使用里面的方法去实现一对多功能。

需要注意的是：

1）设备的IP是唯一区分设备的标识，也即方法参数中使用的**key**。

2）那些接口的函数都是在线程中回调，故不要在回调函数中做UI上的操作。

3.4.1 可实现的接口

OnCompletionListener

- void onCompletion();
  说明：播放完后，回调此方法。
  参数：无
  返回：无

OnConnectStatusListener

- void onDisconnect(String key);
  客户端断开连接时，回调此方法。
  参数：key 是客户端的 IP
  返回：无

OnPlayStatusListener

- void onStart();
  客户端开始播放时，回调此方法。
  参数：无
  返回：无
- void onPlay();
  客户端暂停再播放时，回调此方法。
  参数：无
  返回：无
- void onPause();
  客户端播放再暂停时，回调此方法。
  参数：无
  返回：无
- void onStop();

客户端停止播放时，回调此方法。
参数：无
返回：无

OnAudioInfoListener

- void onGetInfo(AudioInfo audioInfo);
  传回客户端的音频信息对象，包含有 key 和音频音量值。
  参数：audioInfo 为音频对象
  返回：无

OnPlayPositionListener

- void onPosition(int position);
  当前播放的位置，回调此方法。
  参数：无
  返回：无

3.4.2 可使用的方法

---

public int getClientNumber()

获取客户端当前(播放和未播放的)数量。
参数：无
返回：当前连接的客户端数

---

public void release()

释放使用的资源。
参数：无
返回：无

---

public void closeClient(String key)

关闭指定客户端。

参数：key 为客户端 IP。
返回：无

---

public void try2Seek(int time)

从给定的时间开始播放。
参数：time 为歌曲的某一时间。
返回：无

---

public void try2Play(PlayInfo playInfo)

播放歌曲。
参数：playInfo 为播放所需要信息的对象。
返回：无

---

public void try2PlayOrPause()

设置客户端播放或暂停。
参数：无
返回：无

---

public PlayStatus getClientStatus()

获取当前正在播放客户端的状态
参数：无
返回：播放状态

---

public int getLastTimePosition()

获取上一次的播放位置。
参数：无
返回：上一次播放位置

---

public Hashtable<String, String> getCurrentClients()

获取当前所有连接的客户端 IP。

参数：无
返回：所有已选择播放的客户端 IP 集合

---

public void setCurrentClients(Hashtable<String,String> list)

赋予所有当前播放的客户端 IP 集合。
参数：list 为客户端 IP 集合。
返回：无

---

public void try2GetVolume(String key)

获取对应正播放客户端音量，结果在 OnAudioInfoListener 中的 onGetInfo()取得。
参数：key 为客户端 IP。
返回：无

---

public void try2SetVolume(String key, int volume)

设置对应正在播放的客户端音量。
参数：key 为客户端 IP，volume 为要设置的音量值。
返回：无

---

public void search()

搜索在线客户端，搜索到的设备需要 getAllKeys()来区分。
参数：无
返回：无

---

public List<String> getAllKeys()

获取所有播放和未播放的客户端 IP。
参数：无
返回：所有客户端 IP 的集合

---

public String getClientName(String key)

获取客户端的名称。
参数：客户端的 IP
返回：客户端名称

最后，简单说明设备播放这一过程的步骤：
1）首先搜索设备search()
2）选择要播放的设备并把这些设备赋予setCurrentClients()
3）开始播放try2Play()，播放成功会在OnPlayStatusListener回调函数onStar()，操作暂停、播放、停止成功，也会执行对应的回调函数。
4）播放完后，会执行OnCompletionListener的onCompletion()函数，若要继续播放，可执行 3）步骤。

3.5 喜玛拉雅功能实现

喜玛拉雅功能主要是体现 Wifi 设备的网络资源播放功能。该功能需要与喜玛拉雅公司合作，取得其服务器接口协议，根据接口协议获取喜玛拉雅资源数据，这里不详细说明。重点说明的是喜玛拉雅功能与小机通讯协议。需要创建 XmlySocketHelper 对象，调用其中的 perpare 方法创建 Socket。

需要注意的是，
1） 设备的 IP 是唯一区分设备的标识；
2） 端口号默认是 8081；

通讯数据内容定义：

```
public class XMLYMsg{
    private int id;          //类别id
    private int playStatus; //播放状态
    private int albumId;    //专辑id
    private int urlPos;     //声音排列序号
    private int urlId;      //声音链接id
    private int playTime;   //播放时间（以秒为单位）
    private String singUrl; //声音链接 url（用于声音点播协议）
}
```

3.5.1 可实现的接口

```
public void setOnSyncInfoListener(OnSyncInfoListener
listener)
```

- `void onSyncInfo(byte protol, XMLYMsg xmly)`

  说明：通知上层更新 UI
  参数：protol 为通讯协议类型，XMLYMsg 为通讯数据内容
  返回：无

3.5.2 可使用的方法

```
public void prepare()
```

说明：创建与设备通讯 Socket
参数：默认参数为设备的 IP，端口号：8081
返回：无

```
private byte[] packageData(byte protocol, byte cmd,
XMLYMsg xmlyMsg)
```

说明：按协议格式组装数据报文
参数：protocol 为通讯协议类型，cmd 为通讯操作命令,XMLYMsg 为通讯数据内容
返回：通讯数据报文

```
public void sendData(byte protocol, byte cmd, XMLYMsg
mXmlyMsg)
```

说明：用于往设备发送命令数据报文
参数：protocol 为通讯协议类型，cmd 为通讯操作命令,XMLYMsg 为通讯数据内容
返回：无

```
private void parseResult(byte[] data)
```

说明：解析接收到的数据报文
参数：data 为接收的数据报文
返回：无

```
public void release()
```

说明：关闭通讯 Socket
参数：无
返回：无

简单说明一下，喜玛拉雅通讯的步骤：

1) 首先通过 prepare()创建 Socket。
2) 获取喜玛拉雅资源，然后用 packageData()组装数据报文，通过 sendData()方法把命令数据报文发往设备处理。
3) 通过 parseResult()解析接收的数据报文，然后通过接口的 onSyncInfo()通知上层更新 UI。