API 手册

(V2.0)

历史更新记录：

Update V1.02:

1 增加 airkiss 功能。

2 增加设备离线监听函数。

3 增加设备在线状态获取函数。

4 删除 smartconfig.jar 包(旧版本 esptouch 用新加的 airkiss 功能代替，集成在 libJifan.so)

5，删除 JifanScanhelper 类（smartcofig.jar 的使用方法类）

Update V2.0:

1 优化设备入网控制接口。

2 优化设备控制接口。

3 优化设备登录接口。

4 优化设备文件读取接口。

5 优化设备情景文件读取接口。

6 优化设备报警记录读取接口。

一、设备管理（DeviceHelper 类）：

1.1，接口

```
public  interface  Listener{
void DeviceJoinIndicates(byte[] addr);
void NewPortIndicates(byte[] addr, JSONObject describe);
void ReceiveDeviceInfo(byte[] addr, JSONObject devInfo, int state);
void ReceivePortList(byte[] addr, byte[] ports, int state);
void ReceivePortDescribe(byte[] addr, JSONObject describe, int state);
void CompleteDevice(byte[] addr, JSONObject devInfo, byte[] ports, ArrayList<JSONObject> descList);
void ReceiveLqi(byte[] addr, byte port, byte lqi, int state);
void ReceiveBeacon(byte[] addr, int state);
void SendStatusCB(byte[] addr,int seq, int port, int aID,int cmd, int option, int state);
}
```

a)　新设备加入通知

```
void DeviceJoinIndicates(byte[] addr);
```

@addr:设备地址。

@return:none;

b) 新子设备加入通知

```
void NewPortIndicates(byte[] addr, JSONObject describe);
```

@addr:设备地址。

@describe:子设备描述（端口描述）。

@return:none;

c) 接收到设备信息回调

```
void ReceiveDeviceInfo(byte[] addr, JSONObject devInfo, int state);
```

@addr:设备地址。

@ devInfo:设备描述。

@state:状态标志，0 成功，其它失败。

@return:none;

d) 接收到设备端口列表（子设备列表）

```
void ReceivePortList(byte[] addr, byte[] ports, int state);
```

@addr:设备地址。

@ ports:端口列表

@state:状态标志，0 成功，其它失败。

@return:none;

e) 接收到端口描述（子设备描述）

```
void ReceivePortDescribe(byte[] addr, JSONObject describe, int state);
```

@addr:设备地址。

@ describe:端口描述。

@state:状态标志，0 成功，其它失败。

@return:none;

f) 设备信息获取完成回调

```
void CompleteDevice(byte[] addr, JSONObject devInfo, byte[] ports, ArrayList<JSONObject> descList);
```

@addr:设备地址。

@ devInfo:设备描述。

@ ports:端口列表

@ descList:端口描述表。

@return:none;

g) 接收设备信号质量

```
void ReceiveLqi(byte[] addr, byte port, byte lqi, int state);
```

@addr:设备地址。

@ port:端口号。0 表示根设备，其它表示其子设备

@lqi:信号质量 0~255。

@state:状态标志，0 成功，其它失败。

@return:none;

h) 接收到设备信标回调

```
void ReceiveBeacon(byte[] addr, int state);
```

@addr:设备地址。

@state:状态标志，0 成功，其它失败。

i) 操作状态

```
void SendStatusCB(byte[] addr,int seq, int port, int aID,int cmd, int option, int state);
```

@addr:设备地址。

@seq:发送数据序列号。

@port:发送的目标端口。

@aID:操作属性。

@cmd:操作命令。

@option:选项。

@state:状态标志，0 成功，其它失败。

1.2，设置监听回调方法

void int setSectionListener(onSectionListener myListener)设置监听函数。

@ myListener:监听接口。可以同时多次设置不同的接口，消息会复制到每个监听的接口中。

@return:返回 0 成功，其它失败。

a) 新设备加入通知：

void DeviceJoinIndicates(**byte**[] addr)当设置多个 listener 时，所有注册的 listener 接口都能接收到新设备加入通知。并且对新加入的设备依次执行获取设备信息，设备端口，设备端口描述的操作，也就是说新设备加入会依次调用 ReceiveDeviceInfo(),RecievePortList(), ReceiveDescribe(),Completer()方法。

@addr:新设备地址。

@return: none。

b) 子设备加入通知：

void NewPortIndicates(**byte**[] addr, String describe)调用设备允许入网 reqDevEnableJoin 后,新的子设备加入通知从此接口接收。

@addr:新的子设备加入的设备地址。

@describe:加入的新端口描述。json 格式。

@return: none。

c) 接收设备信息：

void ReceiveDeviceInfo(**byte[]** addr, JSONObject devInfo, **int** state)

@addr:消息来源地址。

@devInfo:设备信息描述。json 格式。

@state:状态标志。0 成功，其它失败。

@return: none。

d) 接收设备端口列表：

void ReceivePortList(**byte[]** addr, **byte[]** ports, **int** state);

@addr: 消息来源地址。

@points:端口列表，每个字节表示一个端口。

@state:状态标志。0 成功，其它失败。

@return: none。

e) 接收设备端口描述：

void ReceivePortDescribe (**byte[]** addr,**String** describe, **int** state);

@addr: 消息来源地址。

@describe:端点描述，json 格式。

@state: 状态标志。0 成功，其它失败。

@return: none。

f) 设备的描述信息接收完整：

void CompleteDevice(**byte**[] addr, JSONObject devInfo, **byte**[] ports, ArrayList<JSONObject> descList);

@addr: 消息来源地址。

@devInfo:设备信息描述。json 格式。

@ports:端口列表。

@descList:端口描述表集。

@return: none。

g) 通信信号质量：

```
void  ReceiveLqi(byte[] addr, byte port, byte lqi, int state);
```

接收设备端口的通信信号质量。

@消息来源地址。

@port:端口信号质量。

@lqi:信号质量强度。

@state:状态标志，0 成功，其它失败。

@return: none。

h) 接收设备信标：

```
void  ReceiveBeacon(byte[] addr, int state);
```

@消息来源地址。

@state:状态标志，0 成功，其它失败。

i) 发送信息状态回调：

```
void  SendStatusCB(byte[] addr, int aID, int state);
```

@addr:发送目标设备地址。

@aID:attributeID.

@state:状态标志，0 成功，其它失败。

@return: none。

1.3，移除监听回调方法

```
public static int removeSectionListener(onSectionListener myListener)
```

@myListener:设置时的监听接口的对象，完全一致才能正确移除。

@return:返回 0 成功，其它 失败。

1.4，移除所有监听回调方法

```
static int resetSectionListener()
```

@return:返回 0 成功，其它 失败。

1.5，添加关联设备

第一次运程程序时调用，并自动获取设备相关信息缓存在本地。

```
public static void  AttachDevice(byte[] addr,byte[] adminKey,byte[] comKey, byte[] guestKey);(。
```

a) @addr:设备地址。

b) @adminkey:管理员密钥，可为 null。

c) @commkey:成员密钥，可为 null。

d) @guestkey:访客密钥，可为 null。

e) @return: none。

1.6，移除关联设备。

```
public static int RemoveDevice(byte[] addr);
```

a) addr:设备地址。

b) @return:返回 0 成功。其它失败。

1.7，移除所有关联设备。

```
public static int RemoveAllDevice()
```

a) @return:返回 0 成功。其它失败。

1.8，Wifi 配置入网

```
public static int wifiSmartConfigStart(String ssid, String psw, boolean hideSsid, int timeout)
```

向设备发送无线网络信息。

a) @ssid:无线网络名称。

b) @psw:无线网络密钥。

c) @hideSsid:是否是隐藏名称的无线网络，true 隐藏，false 不隐藏。

d) @timeout:持续时间。

e) @return:返回 0 成功，其它失败。

1.9，Wifi 停止配网

```
public static int wifiSmartConfigStop()
```

停止向设备发送无线网络信息。

a) @return:返回 0 成功，其它失败。

1.10，允许添加子设备

```
public static int reqDevEnableJoin(int keyID, byte[] dst,int seq,int duration)
```

a) @keyID:权限，只有管理员权限才能操作。

b) @dst:操作的目标设备地址。

c) @seq:序列号。

d) @duration:持续时间，单位秒。

e) @return:返回 0 成功，其它失败。

1.11，禁止添加子设备

```
public static int reqDevDisableJoin(int keyID, byte[] dst,int seq)
```

a) @keyID:权限，只有管理员权限才能操作。

b) @dst:操作的目标设备地址。

c) @seq:序列号。

d) @return:返回 0 成功，其它失败。

1.12，删除子设备

```
public static int reqRemovePort(int keyID, byte[] dst,int seq, int port)
```

a) @keyID:权限，只有管理员权限才能操作。

b) @dst:操作的目标设备地址。

c) @seq:序列号。

d) @port:要删除的子设备对应的端口号（每个子设备对应唯一一个端口号）。

e) @return:返回 0 成功，其它 失败

1.13，读设备信息

```
public static int reqReadDeviceInfo(int keyID, byte[] dst, int seq)
```

a) @keyID:权限

b) @dst:操作的目标设备地址。

c) @seq:序列号。

d) @return:返回 0 成功，其它 失败

1.14，读端口列表

```
public static int reqReadPorts(int keyID, byte[] dst,int seq)
```

a) @keyID:权限

b) @dst:操作的目标设备地址。

c) @seq:序列号。

d) @return:返回 0 成功，其它 失败

1.15，读端口描述信息

```
public static int reqReadPortDescribe(int keyID, byte[] dst, int seq,int port)
```

a) @keyID:权限

b) @dst:操作的目标设备地址。

c) @seq:序列号。

d) @return:返回 0 成功，其它 失败

1.16，写入设备信息

```
public static int reqWriteDevInfo(int keyID, byte[] dst,int seq, JSONObject info)
```

a) @keyID:权限，只有管理员权限才能操作。

b) @dst:操作的目标设备地址。

c) @seq:序列号。

d) info:设备信息。

e) @return:返回 0 成功，其它 失败

1.17，写入端口信息

```
public static int reqWritePortDescribe(int keyID, byte[] dst, int seq,JSONObject info)
```

a) @keyID:权限，只有管理员权限才能操作。

b) @dst:操作的目标设备地址。

c) @seq:序列号。

d) info:设备信息。

e) @return:返回 0 成功，其它 失败

1.18，发送信标

```
public static int reqSendBeacon(int keyID, byte[] dst,int seq)
```

a) @keyID:权限。

b) @dst:目标设备地址。也可以是广播地址，此时所有的设备都会回应信息。

c) @seq:序列号。

d) @return:返回 0 成功，其它 失败

1.19，获取设备信号质量

```
public static int reqGetLqi(int keyID, byte[] dst,int seq)
```

a) @keyID:权限。

b) @dst:目标设备地址。也可以是广播地址，此时所有的设备都会回应信息。

c) @seq:序列号。

d) @return:返回 0 成功，其它 失败

1.20，获取子设备信号质量

```
public static int reqGetLqi(int keyID, byte[] dst,int seq, int port)
```

a) @keyID:权限

b) @dst:目标设备地址。也可以是广播地址，此时所有的设备都会回应信息。

c) @seq:序列号。

d) @return:返回 0 成功，其它 失败

二、接收与发送(Aps 类)

2.1，接口

```
public  interface  onSectionListener{
   void RecieveCB(int keyID, byte[] src, int seq, int port, int aID, int cmd, int option, byte[] pdata, int len);
   void SendStatusCB(byte[] addr,int seq, int port, int aID,int cmd, int option, int state);
}
```

a) 消息接收回调：

```
void RecieveCB(int keyID, byte[] src, int seq, int port, int aID, int cmd, int option, byte[] pdata, int len);
```

@keyID：权限。

@src:消息来源地址。

@seq:消息序列号。

@port:消息来源端口。

@aID：消息对应用的属性。

@cmd:命令。

@option:属性的选项。

@pdata:收到的数据。

@len:数据长度。

@return:none。

b) 发送状态回调：

```
void SendStatusCB(byte[] addr,int seq, int port, int aID,int cmd, int option, int state);
```

@addr:设备地址。

@seq:发送数据序列号。

@port:发送的目标端口。

@aID:操作属性。

@cmd:操作命令。

@option:选项。

@state:状态标志，0 成功，其它失败。

2.2，设置监听回调方法 1

```
public static int setSectionListener(int aID, onSectionListener listener)
```

a) @aID：属性 ID,每个属性 ID 对应一个监听接口，也可以是多个属性 ID 对应同一个监听接口，可以多次重复调用，相同的 aID 与对象将被最后一次调用设置的覆盖。相同的 aID 不同的接口会外依次收到对应的消息。

b) @listener:监听接口。

c) @return:返回 0 成功，其它失败。

2.3，设置监听回调方法 2

```
public static int setSectionListener(int minID,int maxID, onSectionListener listener)
```

a) @minID：设置监听的最小 aID。

b) @maxID:设置监听的最大 aID.

c) @listener:监听接口。

d) @return:返回 0 成功，其它失败。

2.4，移除监听回调方法 1

```
public static int removeSectionListener(int aID, onSectionListener listener)
```

a) @aID:属性 ID。

b) @listener:移除的接口，要跟设置监听接口对象一致。

c) @return:返回 0 成功，其它失败。

2.5，移除监听回调方法 2

```
public static int removeSectionListener(int minID,int maxID, onSectionListener listener)
```

a) @minID：设置监听的最小 aID。

b) @maxID:设置监听的最大 aID.

c) @listener:监听接口。

d) @return:返回 0 成功，其它失败。

2.6，移除所有监听回调方法

```
public static int resetSectionListener()
```

a) @return:返回 0 成功，其它失败。

2.7，数据发送

```
public static int reqSend(int keyID,byte[] dst,int seq, int port, long aID, int cmd, int option, byte[] pdata, int len)
```

a) @keyID:权限。

b) @dst:目标地址。

c) @seq:序列号。

d) @port:操作的端口。

e) @aID:操作的属性。

f) @cmd:命令。

g) @option:属性选项。

h) @pdata:传输的数据。

i) @len:数据长度。

j) @return:返回 0 成功，其它失败。

三、智能情景(Scene 类)

3.1，接口

```
public interface onSectionListener{
    void ReadStateCB(int keyID, byte[] addr, String name, int state);
    void WriteStateCB(int keyID, byte[] addr, String name, int state);
    void ReadNameCB(int keyID, byte[] addr, String name, int state);
    void RenameCB(int keyID, byte[] addr, String fname, int state);
    void ReadCB(int keyID, byte[] addr, String fname, String scene);
    void WriteCB(int keyID, byte[] addr, String fname, int state);
    void DeleteCB(int keyID, byte[] addr, String fname, int state);
    void Notify(int keyID, byte[] addr, String fname, int state);
}
```

a) 读情景状态回调

```
void ReadStateCB(int keyID, byte[] addr, String name, int state);
```

@keyID:权限。

@addr:消息来源设备地址。

@name:情景名称。

@state:情景状态。

@return:none.

状态值包括如下,下同：

```
public static final byte op_succeed = 0x00;//成功
public static final byte op_faile = 1;//失败
public static final byte op_error = 2;//错误
public static final byte op_invalid= 3;//无效
public static final byte op_permit_denied = 4;//无权限
public static final byte op_repeat = 5;
public static final byte op_no_port = 6;
public static final byte op_no_file = 7;
public static final byte op_read_error = 8;
public static final byte op_write_error = 9;
public static final byte op_vmInvalid = 10;//无效的情景
public static final byte op_vmRun = 11;//运行中
public static final byte op_vmPause = 12;//暂停
public static final byte op_vmStop = 13;//停止
public static final byte op_vmFinished = 14;//情景执行完成
public static final byte op_vmNoTask = 15;//没有此情景
public static final byte op_vmSysTick = 16;
public static final byte op_vmTest = 17;//测试情景
public static final byte op_vmJoined = 18;//加入情景
public static final byte op_vm_param=19;//情景参数。
```

b) 保存情景状态回调：

```
void WriteStateCB(int keyID, byte[] addr, String name, int state);
```

@keyID:权限。

@addr:消息来源设备地址。

@name:情景名称。

@state:情景状态。

@return:none。

c) 读情景名称回调：

```
void ReadNameCB(int keyID, byte[] addr, String name, int state);
```

@keyID:权限。

@addr:消息来源设备地址。

@name:情景名称。

@state:情景状态。

@return:none。

d) 重命名回调：

```
void RenameCB(int keyID, byte[] addr, String fname, int state);
```

@keyID:权限。

@addr:消息来源设备地址。

@fname:情景名称。

@state:操作状态，0 成功，其它失败。

@return:none。

e) 读情景回调：

```
void ReadCB(int keyID, byte[] addr, String fname, String scene);
```

@keyID:权限。

@addr:消息来源设备地址。

@fname:情景名称。

@ scene:情景内容。

@return:none。

f) 保存情景回调：

void WriteCB(int keyID, byte[] addr, String fname, int state);

@keyID:权限。

@addr:消息来源地址。

@fname:情景名称。

@state:保存状态，0 保存成功，其它失败。

@return:none。

g) 删除情景回调：

void DeleteCB(**int** keyID, **byte**[] addr, String fname, **int** state);

@keyID:权限。

@addr:消息来源地址。

@fname:情景名称。

@state:0 删除成功，其它失败。

@return:none。

h) 情景状态变更通知：

void Notify(**int** keyID, **byte**[] addr, String fname, **int** state);

@keyID:权限。

@addr:消息来源地址。

@fname:情景名称。

@state:情景状态。

3.2，设置监听回调方法

public static int setSectionListener(onSectionListener myListener)

@ myListener：回调接口。可多次重复调用。

@return:返回 0 成功，其它失败。

3.3，移除监听回调方法

public static int removeSectionListener(onSectionListener myListener)

@ myListener：设置监听回调的接口。

@return:返回 0 成功，其它失败。

3.4，清除所有监听回调方法

public static int resetSectionListener()

@return:返回 0 成功，其它失败。

3.5，读设备情景

```
public static int reqRead(int keyID,byte[] addr,int seq, String fname)
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@fname:情景名称。当名称为“*.vm”时，将读取设备上所有情景。

@return:操作状态，0 成功，其它失败。

3.6，保存情景

```
public static int reqWrite(int keyID, byte[] addr, int seq, String fname,JSONObject scene)
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@fname:情景名称。

@scene:情景内容。

@return:操作状态，0 成功，其它失败。

3.7，删除情景

```
public static int reqDel(int keyID, byte[] addr,int seq, String fname)
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@fname:情景名称。

@return:操作状态，0 成功，其它失败。

3.8，读取所有情景名称

```
public static int reqReadAllName(int keyID, byte[] addr,int seq)//read all scence name
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@return:操作状态，0 成功，其它失败。

3.9，重命名

```
public static int reqRenameScene(int keyID, byte[] addr, int seq, String oldname, String newname)
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@oldname:旧情景名称。

@newname:新情景名称。

@return:操作状态，0 成。其它失败。

3.10，加入情景

```
public static int reqJoin(int keyID, byte[] addr, int seq, String tsk0, String tsk1)
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@tsk0:被加入的情景名称。

@tsk1:要加入的情景名称。

@return:操作状态，0 成。其它失败。

3.11，测试情景

```
public static int reqRun(int keyID, byte[] addr, int seq, String name, String arg)
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@name:测试的情景名称。

@arg:携带的参数。

@return:操作状态，0 成。其它失败。

3.12，读情景状态

```
public static int reqReadState(int keyID, byte[] addr,int seq, String name)
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@name:情景名称。

@return:操作状态，0 成。其它失败。

3.13，设置情景运行状态

```
public static int reqSetState(int keyID, byte[] addr, int seq, String name, int state)
```

@keyID:权限。

@addr:目标设备。

@seq:消息序列号。

@name:情景名称。

@state:情景状态。

@return:操作状态，0 成。其它失败。

3.20，创建情景执行体

```
public static int creatBody();
```

@return：返回当前创建的执行体引用。

3.21，添加并创建条件语句块

```
public static int addIfBlock(int root, String ifType, String reason)
```

@root:目标结构引用，可以新创建的执行体引用，也可以是其它语句块的引用。

@ifTyep:条件类型，可以是“if”也可以是“read”。

@reason:条件表达式。

@return:返回创建的语句块引用。

3.22，添加 else 语句块

```
public static int addElseBlock(int root)
```

@root:目标结构引用，可以新创建的执行体引用，也可以是其它语句块的引用。

@return:返回创建的语句块引用。

注：else 语句必须在添加 if 语句块之后平级创建，否则无效。

3.23，添加动作语句块

```
public static int addAction(int root, String what, String param)
```

@root:目标结构引用，可以新创建的执行体引用，也可以是其它语句块的引用。

@what:动作名称，可是“do”，“send”，“delay”。

@param:对应的参数，比如”do”对应的参数可以为“i++”。

@return:返回创建的语句块引用。

3.24，把语句块转成文本

```
public static String output(int root)
```

@root:根执行体。

@return:返回文件。

3.25，示例

某情景只能触发 10 次，每次触发执行 5 次循环

```
public static void test(byte[] addr, byte keyID) {
    int body = creatBody();
    //addAction(jbody,"state","auto");
    addAction(body, "do", "x=0");
    int while = addWhileBlock(body, "x < 10");
    int if = addIfBlock(while, "if", "wait(point=04,cmd=08,dtype=01,aID=008010)");
    addAction(if, "do", "k=0");
    int while1 = addWhileBlock(if, "k < 5");
    addAction(while1, "send", "addr:0100010100000019,keyID:00,point:01,cmd:02,dtype:01,aID:008010,dlen:0001,data:01");
    addAction(while1, "delay", "200");
    addAction(while1, "send", "addr:0100010100000019,keyID:00,point:02,cmd:02,dtype:01,aID:008010,dlen:0001,data:01");
    addAction(while1, "delay", "200");
    addAction(while1, "do", "k++");
```

```
    addAction(if, "do", "x++");
    String out = output(body);
    Log.d(Common.TAG_Debug, "Test1: " + out);
    Scene.reqWrite(keyID, addr, Common.getSeq(),"Test1.vm", out.getBytes());
}
```

四、黑白名单(UserTable 类)

4.1，接口

```
public interface onSectionListener{
    void ReadCB(int keyID, byte[] addr, int state, boolean white_list, byte[] userAddr, byte[] portlist);
    void WriteCB(int keyID, byte[] addr, int state);
    void DeleteCB(int keyID, byte[] addr, int state);
    void SendStatusCB(byte[] addr, int state);
}
```

a) 读黑白名单回调接口

```
void ReadCB(int keyID, byte[] addr, int state, boolean white_list, byte[] userAddr, byte[] portlist);
```

@keyID 操作权限。

@addr:源地址。

@white_list:false 黑名单，true 白名单。

@permite：禁止 false，允许 true。

@userAddr:用户地址。

@portlist:用户允许或禁止控制的端口列表。

b) 写黑白名单回调接口

```
void WriteCB(int keyID, byte[] addr, int state);
```

@keyID 操作权限。

@addr:源地址。

@state:状态，0 成功，其它失败

c) 删除黑白名单回调接口

```
void DeleteCB(int keyID, byte[] addr, int state);
```

@keyID 操作权限。

@addr:源地址。

@state:状态，0 成功，其它失败

d) 操作状态回调接口

```
void SendStatusCB(byte[] addr, int state);
```

@addr:源地址。

@state:状态，0 成功，其它失败

4.2，设置监听回调方法

```
public static int setSectionListener(onSectionListener myListener)
```

@myLinstener:监听接口。

@return:返回 00 成功，其它失败。

4.3，移除监听回调方法

```
public static int setSectionListener(onSectionListener myListener)
```

@myLinstener:监听接口。

@return:返回 00 成功，其它失败。

4.4，移除所有监听回调方法

```
public static int resetSectionListener()
```

@return:返回 00 成功，其它失败。

4.5，读取黑白名单

```
public static void reqRead(int keyID, byte[] addr, int seq)
```

@keyID:操作权限。

@add:目标地址。

@seq:消息序列号。

@return: 返回 00 成功，其它失败。

4.6，删除黑白名单

```
public static void reqDelete(int keyID, byte[] addr, int seq)
```

@keyID:操作权限。

@add:目标地址。

@seq:消息序列号。

@return: 返回 00 成功，其它失败。

4.6，创建黑白名单表

```
public static int create(boolean white_list)
```

@white_list:true 为白名单，false 为黑名单。

@return: 返回 00 成功，其它失败。

4.7，添加名单

```
public static int put(byte[] addr,byte[] port_list)
```

@addr:用户地址。

@port_list:用户端口列表

@return:返回 00 成功，其它失败。

4.8，保存黑白名单

```
public static int reqSave (int keyID, byte[] dst, int seq)
```

@keyID:操作权限。

@dst:目标地址。

@seq:序列号。

@return:返回 00 成功，其它失败

五、报警与操作记录(Record 类)

5.1，接口

```
public interface onSectionListener{
    public void ReadCB(int keyID, byte[] src, int seq, int state,String tag, long unix_time, byte[] dev_addr, int port,
                int cmd, int option, long aID, byte[] pdata);
    public void DeleteCB(int keyID, byte[] addr, int seq, int state);
    public void ReadStateCB(int keyID, byte[] src, int seq, int state, int enable);
    public void WriteStateCB(int keyID, byte[] src, int seq, int  state);
    public void SendState(byte[] src, int seq, int state);
}
```

a) 读记录回调接口：

public void ReadCB(int keyID, byte[] src, int seq, int state, long unix_time, byte[] dev_addr, int port, int cmd, int option, long aID, byte[] pdata);

@keyID:操作权限。

@src:消息来源地址。

@seq:消息序列号。

@state:状态标志，00 成功，其它失败。

@tag:记录标签，指示记录设备接收还是发送。

@unix_time：本条记录发生的时间。

@dev_addr:本条记录操作的相关设备地址。

@port:本条记录操作的端口。

@cmd:本条记录操作的命令。

@option:本条记录操作的选项。

@aID:本条记录操作的属性。

@pdata:本条记录操作的数据。

b) 删除记录回调接口

public void DeleteCB(int keyID, byte[] addr, int seq, int state);

@keyID:操作权限。

@addr:设备地址。

@seq:消息序列号。

@state:操作状态，0 成功，其它失败。

@return:none;

c) 读记录使能标志

public void ReadStateCB(int keyID, byte[] src, int seq, int state, int enable);

@keyID:操作权限。

@addr:设备地址。

@seq:消息序列号。

@state:操作状态，0 成功，其它失败。

@enbale:0 禁止记录，1 使能记录。

@return:none;

d) 写记录使能标志

```
public void WriteStateCB(int keyID, byte[] src, int seq, int  state);
```

@keyID:操作权限。

@addr:设备地址。

@seq:消息序列号。

@state:操作状态，0 成功，其它失败。

@return:none;

e) 操作状态回调接口

```
public void SendState(byte[] src, int seq, int state);
```

@src:目标设备地址。

@seq:消息序列号。

@state:操作状态，0 成功，其它失败。

@return:none。

5.2，设置报警记录回调接口

```
public static int setAlarmRecordSectionListener(onSectionListener listener)
```

@listener:监听接口。

@return:返回 0 成功，其它失败。

5.3，移除报警记录回调接口

```
public static int removeAlarmSectionListener(onSectionListener myListener)
```

@listener:监听接口。

@return:返回 0 成功，其它失败。

5.4，移除所有报警记录回调接口

```
public static int resetAlarmRecordSectionListener()
```

@listener:监听接口。

@return:返回 0 成功，其它失败。

5.5，读报警记录

```
public static int reqReadAlarm(byte keyID,byte[] addr,int seq)
```

@keyID:操作权限。

@addr:目标设备地址。

@seq:序列号。

@return:返回 00 成功，其它失败。

5.6，删除报警记录

```
public static int reqDeleteAlarm(byte keyID,byte[] addr,int seq)
```

@keyID:操作权限。

@addr:目标设备地址。

@seq:序列号。

@return:返回 00 成功，其它失败。

5.7，设置历史记录回调接口

```
public static int setHistroryRecordSectionListener(onSectionListener listener)
```

@listener:监听接口。

@return:返回 0 成功，其它失败。

5.8，移除历史记录回调接口

```
public static int removeHistorySectionListener(onSectionListener myListener)
```

@listener:监听接口。

@return:返回 0 成功，其它失败。

5.9，移除所有历史记录回调接口

```
public static int resetHistoryRecordSectionListener()
```

@listener:监听接口。

@return:返回 0 成功，其它失败。

5.10，读历史记录

```
public static int reqReadHistory(byte keyID,byte[] addr,int seq)
```

@keyID:操作权限。

@addr:目标设备地址。

@seq:序列号。

@return:返回 00 成功，其它失败。

5.11，删除历史记录

```
public static int reqDeleteHistory(byte keyID,byte[] addr,int seq)
```

@keyID:操作权限。

@addr:目标设备地址。

@seq:序列号。

@return:返回 00 成功，其它失败。

5.11，读历史记录使能标志

```
public static int reqReadHistoryEnableFlag(byte keyID, byte[] addr, int seq)
```

@keyID:操作权限。

@addr:目标设备地址。

@seq:序列号。

@return:返回 00 成功，其它失败。

5.11，写历史记录使能标志

```
public static int reqWriteHistoryEnableFlag(byte keyID, byte[] addr, int seq, boolean enable)
```

@keyID:操作权限。

@addr:目标设备地址。

@seq:序列号。

@enable:使能记录 true,禁止记录 false。

@return:返回 00 成功，其它失败。

六、设备升级(Upgrade 类)

6.1，接口

```
public interface onSectionListener{
void ReadInfoCB(int keyID, byte[] addr, int state, JSONObject info);
void UpdateStateCB(int keyID, byte[] addr, int state, int percent);
}
```

a) 读设备版本信息

```
void ReadInfoCB(int keyID, byte[] addr, int state, JSONObject info);
```

@keyID：操作权限。

@addr:消息源地址。

@state:状态标志，0 成功，其它失败。

@info:设备版本信息。

@return:none;

b)

```
void UpdateStateCB(int keyID, byte[] addr, int state, int percent);
```

@keyID：操作权限。

@addr:消息源地址。

@state:更新状态标志，0 成功，0x20 停止更新，0x21 开始更新，0x22 更新完成，其它失败。

@ percent:进度百分比 0~100。

@return:none;

6.2，设置监听回调方法

```
public static int setSectionListener(onSectionListener myListener)
```

@myListener:监听接口。

@return:返回 0 成功，其它失败。

6.3，移除监听回调方法

```
public static int removeSectionListener(onSectionListener myListener)
```

@myListener:监听接口。

@return:返回 0 成功，其它失败。

6.4，移除所有监听回调方法

```
public static int resetSectionListener()
```

@return:返回 0 成功，其它失败。

6.5，读取设备版本信息

```
public static int reqReadInfo(byte keyID,byte[] addr)
```

@keyID：操作权限。

@addr:消息源地址。

6.6，请求设备升级

```
public static int reqUpdate(byte keyID,byte[] addr,  String server_ip, int server_port, String server_url)
```

@keyID：操作权限。

@addr:消息源地址。

@server_ip:升级服务 IP 地址。

@server_port:升级服务端口地址。

@server_url:升级服务完 url(包括升级文件名)。

@return:返回 00 成功，01 失败。

七、工厂配置(Factory 类)

7.1、接口

```
public interface onSectionListener{
    void ReadCB(int keyID, byte[] addr, int state, JSONObject info);
    void WriteCB(int keyID, byte[] addr, int state);
}
```

a) 读设备版本信息回调

```
void ReadCB(int keyID, byte[] addr, int state, JSONObject info);
```

@keyID:操作权限。

@addr:设备地址。

@state:操作状态，0 成功，其它失败。

@info:设备版本信息。

@return:none。

b) 写入设备信息回调

```
void WriteCB(int keyID, byte[] addr, int state);
```

@keyID:操作权限。

@addr:设备地址。

@state:操作状态，0 成功，其它失败。

@return:none。

7.2、读工厂配置信息

```
public static int reqReadInfo(int keyID, byte[] addr, int seq)
```

@keyID:操作权限。

@addr:设备地址。

@seq:消息序列号。

@return:返回 00 成功，其它失败。

7.3、创建工厂配置信息

```
public static int creat(String server_domain, String server_ipv4, int server_udp_port, int server_tcp_port, int local_udp_port)
```

@ server_domain:服务器域名地址。

@ server_ipv4:服务器 ipv4 地址。

@ server_udp_port:服务器 udp 服务端口。

@ server_tcp_port:服务器 tcp 服务端口。

@ local_udp_port:局域网通信服务端口。

@return:返回 00 成功，其它失败。

7.4、工厂配置添加额外的数值信息

```
public static int putExtraNum(String name, int num)
```

@name：数值键名。

@num:数值大小。

@return:返回 00 成功，其它失败。

7.5、工厂配置添加额外的字符串信息

```
public static int putExtraString(String name, String str)
```

@name：数值键名。

@str:字符信息。

@return:返回 00 成功，其它失败。

7.6、写入工厂配置信息

```
public static int reqWriteInfo(int keyID, byte[] addr, int seq)
```

@keyID:操作权限。

@addr:设备地址。

@seq:消息序列号。

@info:配置信息。

@return:返回 00 成功，其它失败。