|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |
|  | | |
| 方圆ok1 | | | |
|  | **IoM 解决方案**  **V100R001C10**  **华为IoT Agent Lite API参考(C)** | | 附件1-16K |
|  | |
| **文档版本** | **01** |
| **发布日期** | **2016-09-22** |
| **华为技术有限公司** | |

|  |
| --- |
| 版权所有 © 华为技术有限公司2016。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  附件3-版权声明页图和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | <http://www.huawei.com> |
| 客户服务邮箱： | [support@huawei.com](mailto:support@huawei.com) |
| 客户服务电话： | 4008302118 |

目 录

[1 前言 1](#_Toc462337223)

[2 开发者必读 2](#_Toc462337224)

[3 接口列表 3](#_Toc462337225)

[3.1 广播机制介绍 3](#_Toc462337226)

[3.2 直连设备接入 4](#_Toc462337227)

[3.3 网关管理非直连设备 12](#_Toc462337228)

[3.4 设备服务数据上报 14](#_Toc462337229)

[3.5 设备命令接收 16](#_Toc462337230)

[3.6 Json组件使用说明 17](#_Toc462337231)

[4 常用数据结构定义 21](#_Toc462337232)

[5 数据类型定义 27](#_Toc462337233)

# 前言

导读

本文档系统化描述华为Agent Lite对外开放的能力全集、集成原理和集成参考样例等信息，帮助集成开发者快速而准确的掌握集成方法从而高效实现特定的业务需求。本文档主要包含如下几个部分：

| 章节 | 说明 | 备注 |
| --- | --- | --- |
| [开发者必读](#_ZH-CN_TOPIC_0035448157) | 主要是介绍华为Agent Lite开放能力全景图、集成原理、集成流程、集成中关键知识点和常用术语等。确保集成开发者能快速掌握集成的基本知识和技能。 | [开发者必读](#_ZH-CN_TOPIC_0035448157)章节是必看的。 |
| [接口列表](#_ZH-CN_TOPIC_0035448144) | 主要是描述能力开放的接口集合，详细介绍每个接口的功能、输入参数、输出参数、和消息样例等信息。 | 本章节中[广播机制介绍](#_ZH-CN_TOPIC_0035448135)是必看的，其他小节根据使用情况查询选看。 |
| [常用数据结构定义](#_ZH-CN_TOPIC_0035448119) | 主要是给出主要的数据结构、枚举的定义。 | - |
| [数据类型定义](#_ZH-CN_TOPIC_0035448136) | 统一使用的数据格式定义。 | - |

修改记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

| 文档版本 | 发布日期 | 修改说明 | 修改影响 |
| --- | --- | --- | --- |
| 01 | 2016/05/30 | 第一次发布 |  |
|  |  |  |  |

# 开发者必读

1. 概述

华为IoT Agent Lite在智慧家庭、工业物联网、车联网等领域为智能设备提供了标准接入华为IoT联接管理平台的能力。主要面向IPC、轻量级网关、工业网关、车机等计算能力较强的终端/网关设备。

2. 接口全景图

集成开发者通过全景图了解到：Agent Lite提供了哪些功能？每个功能包含哪些接口？接口之间的逻辑关系如何？从而更快速的找到正确的接口来实现具体业务。

| **功能** | **接口** | **说明** |
| --- | --- | --- |
| **直连设备接入** | IOTA\_Init | 初始化Agent Lite模块资源 |
| IOTA\_Destroy | 释放Agent Lite模块资源 |
| IOTA\_SetConfig | 相关参数配置 |
| IOTA\_Bind | 设备绑定 |
| IOTA\_Login | 设备登录 |
| IOTA\_DeviceStatusUpdate | 设备状态更新 |
| **网关管理非直连设备** | IOTA\_HubDeviceAdd | 设备添加 |
| IOTA\_HubDeviceRemove | 设备删除 |
| **设备服务数据上报** | IOTA\_ServiceDataReport | 设备服务数据上报 |
| **设备命令接收** | IOTA\_TOPIC\_SERVICE\_COMMAND\_  RECEIVE/{deviceId} | 设备服务命令接收 |

# 接口列表

Agent Lite对外提供的接口主要包括广播机制、直连设备的接入、非直连设备的添加和删除、设备数据的上报、设备命令接收以及Json接口。

[3.1 广播机制介绍](#_ZH-CN_TOPIC_0035448135)

[3.2 直连设备接入](#_ZH-CN_TOPIC_0035447975)

[3.3 网关管理非直连设备](#_ZH-CN_TOPIC_0035448138)

[3.4 设备服务数据上报](#_ZH-CN_TOPIC_0035448153)

[3.5 设备命令接收](#_ZH-CN_TOPIC_0035447966)

[3.6 Json组件使用说明](#_ZH-CN_TOPIC_0035448147)

## 广播机制介绍

Agent Lite提供了一套广播机制给第三方开发者，用来接收Agent Lite上报的消息。

订阅广播：

HW\_BroadCastReg(HW\_CHAR \*pcTopic ,PFN\_HW\_BROADCAST\_RECV pfnReceiver)

广播接收处理函数原型：

(\*PFN\_HW\_BROADCAST\_RECV)(HW\_UINT uiCookie, HW\_MSG \*pstMsg);

此处uiCookie对应于接口中传入的uiCookie，用来匹配业务的请求与响应；如接口中无uiCookie参数，或传入的是无效值，则广播中该参数无意义。

取消订阅广播：

HW\_BroadCastUnreg(HW\_CHAR \*pcTopic, PFN\_HW\_BROADCAST\_RECV pfnReceiver);

从pstMsg获取数据的函数：

获取字符串数据：

HW\_MsgGetStr(HW\_MSG pstMsg, HW\_UINT uiTag)

获取无符号整形数据：

HW\_MsgGetUint(HW\_MSG pstMsg, HW\_UINT uiTag, HW\_UINT uiDefault)

获取字节数组数据：

HW\_MsgGetByteArray(HW\_MSG pstMsg, HW\_UINT uiTag)

## 直连设备接入

第三方开发者获得Agent Lite后首先需要将设备接入到IoT连接管理平台中。

直连设备：通过设备绑定、登录流程直接接入IoT平台的设备。

非直连设备：通过网关设备接入IoT平台的设备。

1. 初始化Agent Lite资源

接口功能

初始化Agent Lite资源。

接口描述

HW\_INT IOTA\_Init(HW\_CHAR \*pcWorkPath, HW\_CHAR \*pcLogPath);

参数说明

| **字段** | **必选/可选** | **类型** | **描述** |
| --- | --- | --- | --- |
| pcWorkPath | 必选 | String | Agent Lite工作路径，用于存放Agent Lite的配置文件与生产的临时文件。 |
| pcLogPath | 可选 | String | 日志路径（若日志路径为空则日志写在工作路径中） |

接口返回值

参见[函数标准返回值](#section129861814143216)

示例

// 开发者调用该接口初始化Agent Lite资源   
IOTA\_Init(“/usr/data”,HW\_NULL)

2. 释放Agent Lite资源

接口功能

调用此函数，Agent Lite会释放申请的所有动态资源（内存、线程等等）。

接口描述

IOTA\_VOID IOTA\_Destroy();

接口返回值

参见[函数标准返回值](#section129861814143216)

示例

// 开发者调用该接口销毁Agent Lite资源   
IOTA\_Destroy();

3. 参数设置

接口功能

配置Agent Lite相关参数。

接口描述

HW\_INT IOTA\_ConfigSetStr(HW\_INT iItem, HW\_CHAR \*pValue)   
HW\_INT IOTA\_ConfigSetUint(HW\_INT iItem, HW\_UINT uiValue)

参数说明

| **字段** | **必选/可选** | **类型** | **描述** |
| --- | --- | --- | --- |
| iItem | 必选 | 枚举 | 设备登陆所需的配置项（详见EN\_IOTA\_CFG\_TYPE说明） |
| pValue | 必选 | String/int | 设置的值 |

接口返回值

参见[函数标准返回值](#section129861814143216)

示例

// 开发者调用该接口设置参数   
IOTA\_ConfigSetStr (EN\_IOTA\_CONFIG\_IOCM\_ADDR, “10.0.0.1”);   
IOTA\_ConfigSetUint(EN\_IOTA\_CFG\_IOCM\_PORT, 8943);

4. 设备绑定

接口功能

设备第一次接入IoT联接管理平台时需要进行绑定操作，上层应用通过调用该接口传入设备序列号或者MAC地址以及设备信息来绑定到IoT联接管理平台。在绑定前需要调用IOTA\_SetConfig接口设置绑定服务器IP与端口（IoCM服务器地址与端口，Agent Lite会配置默认端口8943）。

说明

设备绑定是指设备第一次接入IoT平台的过程，需要开发者先在IoT平台注册直连设备，之后在设备上发起绑定操作，将设备绑定到IoT平台上。如果未在IoT平台注册该设备，则绑定操作会失败，Agent Lite将会等待一段时间继续尝试。

必要参数配置:

EN\_IOTA\_CFG\_IOCM\_ADDR

EN\_IOTA\_CFG\_IOCM\_PORT

接口描述

HW\_INT IOTA\_Bind(HW\_CHAR \*pcVerifyCode, ST\_IOTA\_DEVICE\_INFO \*pstInfo)

参数说明

| **字段** | **必选/可选** | **类型** | **描述** |
| --- | --- | --- | --- |
| pcVerifyCode | 必选 | String | 设备绑定验证码 |
| pstInfo | 必选 | ST\_IOTA\_DEVICE\_INFO结构体 | 设备信息 |

接口返回值

参见[函数标准返回值](#section129861814143216)

说明

此返回值是调用接口的同步返回结果，返回0只是说明接口调用成功，并不说明绑定成功，绑定成功需要收到IOTA\_TOPIC\_BIND\_RSP广播。



该设备未绑定时，该接口会自动尝试重新绑定，如果绑定失败，则30秒后继续进行重试，如果重试超过5次（总计尝试超过6次），则返回失败，不再进行重试。如果想要重新发起绑定，建议让用户重启设备。开发者无需进行重试，如果需要停止重试，调用Agent Lite销毁接口即可。同时请勿短时间内多次调用该接口。

当前绑定流程的重试策略为，如果绑定失败，则30秒后继续进行重试，如果重试超过5次（总计尝试超过6次），则返回失败，不再进行重试。如果想要重新发起绑定，建议让用户重启设备。

返回结果

| **广播名称** | **广播参数** | **成员** | **描述** |
| --- | --- | --- | --- |
| IOTA\_TOPIC\_BIND\_RSP | HW\_MSG对象 | EN\_IOTA\_BIND\_IE\_TYPE | 绑定返回结果（详见枚举EN\_IOTA\_BIND\_IE\_TYPE） |

注意事项

如果接入设备是网关设备，则DeviceType填写为“Gateway”,如果为传感器设备直连，则填写为对应的设备类型。

示例

//开发者调用该接口进行设备绑定   
ST\_HW\_DEVICE\_INFO stDeviceInfo   
stDeviceInfo.pcNodeId = “SN Number”;   
stDeviceInfo.pcManufacturerId = “Huawei”;   
stDeviceInfo.pcDeviceType = “Gateway”;   
stDeviceInfo.pcModel = “HW\_GW101”;   
stDeviceInfo.pcProtocolType = “HuaweiM2M”;   
   
IOTA\_Bind(“SN Number”, &stDeviceInfo)

当设备成功绑定之后，Agent Lite会返回给UI如下几个参数，需要UI进行持久化存储，设备登录前需要提前进行配置

//注册广播接收处理函数   
HW\_BroadCastReg(“IOTA\_TOPIC\_BIND\_RSP”, Device\_RegResultHandler);

// 开发者注册该函数处理绑定结果   
HW\_INT Device\_RegResultHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
HW\_CHAR \*pcDeviceId;   
HW\_CHAR \*pcDeviceSecret;   
HW\_CHAR \*pcAppId;   
HW\_CHAR \*pcIoCMServerAddr;   
HW\_UINT uiIoCMServerPort;   
HW\_CHAR \*pcMqttServerAddr;   
HW\_UINT uiMqttServerPort;   
   
If (HW\_SUCCESS != HW\_MsgGetUint(pstMsg, EN\_IOTA\_BIND\_IE\_RESULT, 0))   
{   
Return 0;   
}   
   
pcDeviceId = HW\_MsgGetStr(pstMsg, EN\_IOTA\_BIND\_IE\_DEVICEID);   
pcDeviceSecret = HW\_MsgGetStr(pstMsg, EN\_IOTA\_BIND\_IE\_DEVICESECRET);   
pcAppId = HW\_MsgGetStr(pstMsg, EN\_IOTA\_BIND\_IE\_APPID);   
pcIoCMServerAddr = HW\_MsgGetStr(pstMsg, EN\_IOTA\_ BIND\_IE\_IOCM\_ADDR );   
uiIoCMServerPort = HW\_MsgGetUint(pstMsg, EN\_IOTA\_BIND\_IE\_IOCM\_PORT, 0);   
pcMqttServerAddr = HW\_MsgGetStr(pstMsg, EN\_IOTA\_ BIND\_IE\_IOCM\_ADDR );   
uiMqttServerPort = HW\_MsgGetUint(pstMsg, EN\_IOTA\_BIND\_IE\_IOCM\_PORT, 0);   
   
Config\_save(“DeviceId”,pcDeviceId);   
Config\_save(“DeviceSecret”,pcDeviceSecret);   
Config\_save(“AppId”,pcAppId);   
Config\_save(“IoCMAddr”,pcIoCMServerAddr);   
Config\_save(“IoCMPort”,pcIoCMServerPort);   
Config\_save(“MqttAddr”,pcMqttServerAddr);   
Config\_save(“MqttPort”,pcMqttServerPort);   
   
return 0;   
}

5. 设备解绑定

接口功能

注册设备解绑定接收广播来接收处理平台下发的直连设备解绑定命令，开发者收到该广播后需要删除直连设备的配置信息并且释放所有资源，下一次重启后需要重新进行绑定。

接口描述

IOTA\_TOPIC\_CMD\_UNBIND\_RECEIVE

参数说明

无

返回结果

无

示例:

// 开发者注册该函数进行解绑定直连设备的处理   
HW\_INT Gateway\_UnbindRecvtHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
// Delete Config file，free resources.   
return 0;   
}

//初始化时进行注册该函数   
HW\_BroadCastReg(“IOTA\_TOPIC\_CMD\_UNBIND\_RECEIVE”, Gateway\_UnbindRecvtHandler);

6. 设备登录

接口功能

设备在第一次绑定后，或者在设备重启后需要进行登录的流程。

接口描述

HW\_INT IOTA\_Login()

接口返回值

参见[函数标准返回值](#section129861814143216)

说明

此返回值是调用接口的同步返回结果，返回0只是说明接口调用成功，并不说明登陆成功，登陆成功需要收到IOTA\_TOPIC\_CONNECTED\_NTY广播。登录前通过参数配置接口（IOTA\_SetConfig）传入所需的登录信息。

必要参数配置:

EN\_IOTA\_CFG\_DEVICEID

EN\_IOTA\_CFG\_DEVICESECRET

EN\_IOTA\_CFG\_APPID

EN\_IOTA\_CFG\_IOCM\_ADDR

EN\_IOTA\_CFG\_IOCM\_PORT

EN\_IOTA\_CFG\_MQTT\_ADDR

EN\_IOTA\_CFG\_MQTT\_PORT

之后调用登录接口函数进行登录：

HW\_INT IOTA\_Login();

返回结果

| **广播名称** | **广播参数** | **成员** | **描述** |
| --- | --- | --- | --- |
| IOTA\_TOPIC\_CONNECTED\_NTY | HW\_MSG对象 | 无 | 登录成功或重连成功 |
| IOTA\_TOPIC\_DISCONNECT\_NTY | HW\_MSG对象 | EN\_IOTA\_LGN\_IE\_TYPE | 登陆失败或连接断开 |

示例：

Config\_Get(“DeviceId”,pcDeviceId);   
Config\_Get(“DeviceSecret”,pcDeviceSecret);   
Config\_Get(“AppId”,pcAppId);   
Config\_Get(“HAAddr”,pcHAServerAddr);   
Config\_Get(“LVSAddr”,pcLVSServerAddr);   
   
IOTA\_SetConfig(EN\_IOTA\_CFG\_DEVICEID, pcDeviceId);   
IOTA\_SetConfig(EN\_IOTA\_CFG\_DEVICESECRET, pcDeviceSecret);   
IOTA\_SetConfig(EN\_IOTA\_CFG\_APPID, pcAppId);   
IOTA\_SetConfig(EN\_IOTA\_CFG\_HA\_ADDR, pcHAServerAddr);   
IOTA\_SetConfig(EN\_IOTA\_CFG\_LVS\_ADDR, pcLVSServerAddr);   
   
IOTA\_Login();

然后等待Agent Lite的连接状态广播

需要提前实现连接状态通知广播接收处理函数，建议：

1. 对于网关设备，在连接成功的处理函数中需要进行非直连设备状态上报的处理，并且将缓存的所有上报数据进行上报。
2. 在连接断开的处理函数中记录设备断开状态，之后如果有数据上报，需要进行缓存，等到连接成功后再进行上报。

// 开发者注册该函数进行连接成功后的处理   
HW\_INT Device\_ConnectedHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
//update device states   
//send buffer data   
   
return 0;   
}

// 开发者注册该函数进行连接失败后的处理   
HW\_INT Device\_DisconnectHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
//stop reporting data   
return 0;   
}

//绑定广播接收处理函数   
HW\_BroadCastReg(“IOTA\_TOPIC\_CONNECTED\_NTY”, Device\_ConnectedHandler);   
HW\_BroadCastReg(“IOTA\_TOPIC\_DISCONNECT\_NTY”, Device\_DisconnectHandler);

设备登录后，表示该设备已经成功的连接到IoT联接管理平台。

连接成功后，如果因为网络或服务器原因导致连接断开，Agent Lite会自动尝试重新连接，并将实时状态通过这两个广播上报给第三方应用。

7. 设备状态更新

接口功能

通过该接口更新设备的状态信息，包括直连设备与所管理的非直连设备。设备离线上线均可通过该接口刷新设备状态信息。

说明

直连设备状态通过设备的登录状态进行管理，当直连设备连接断开则表示设备离线，当直连设备连接或重连成功，则表示设备上线，无需通过该接口进行刷新。故建议开发者使用该接口刷新非直连设备的状态。

接口描述

HW\_INT IOTA\_DeviceStatusUpdate(HW\_UINT uiCookie, HW\_CHAR \*pcDeviceId, HW\_CHAR pcStatus, HW\_CHAR pcStatusDetail)

参数说明

| **字段** | **必选/可选** | **类型** | **描述** | |
| --- | --- | --- | --- | --- |
| uiCookie | 可选 | HW\_UINT | Cookie有效值为1-65535 | |
| pcDeviceId | 必选 | HW\_CHAR | 设备Id | |
| pcStatus | 必选 | HW\_CHAR | 设备状态 | 在线: ONLINE |
| 离线: OFFLINE |
| pcStatusDetail | 必选 | HW\_CHAR | 设备状态详细信息 | 无：NONE |
| 配置等待：CONFIGURATION\_PENDING |
| 通信错误：COMMUNICATION\_ERROR |
| 配置错误：  CONFIGURATION\_ERROR |
| 桥接器离线  BRIDGE\_OFFLINE |
| 固件升级  FIRMWARE\_UPDATING |
| 循环任务  DUTY\_CYCLE |
| 未激活  NOT\_ACTIVE |

接口返回值

参见[函数标准返回值](#section129861814143216)

返回结果

| **广播名称** | **广播参数** | **成员** | **描述** |
| --- | --- | --- | --- |
| IOTA\_TOPIC\_DEVUPDATE\_RSP/{deviceId} | HW\_MSG对象 | 无 | 设备状态更新结果 |

示例：

HW\_CHAR \*pcDeviceId = stDevice.pcDeviceId;   
   
IOTA\_DeviceStatusUpdate(0, pcDeviceId, “ONLINE” , “NONE”);

然后等待命令执行结果

// 开发者注册该函数进行状态更新后的处理   
HW\_INT Device\_StatusUpdateHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
HW\_CHAR pcCmdContent；   
pcCmdContent = HW\_MsgGetStr(pstMsg, EN\_IOTA\_DEVUPDATE\_IE\_RESULT);   
pcCmdContent = HW\_MsgGetStr(pstMsg, EN\_IOTA\_DEVUPDATE\_IE\_DEVICEID);   
   
return 0;   
}

//绑定广播接收处理函数   
HW\_BroadCastReg(“IOTA\_TOPIC\_DEVUPDATE\_RSP”, Device\_StatusUpdateHandler);

## 网关管理非直连设备

当开发设备为网关设备时，设备需要管理所有非直连设备（传感器设备）的接入与删除，并且记录这些设备ID与对应设备的映射关系。

1.设备添加

接口功能

当有新设备接入网关后，通过调用设备添加接口将非直连设备接入IoT联接管理平台，并且获得平台分配的唯一设备逻辑ID。

接口描述

HW\_INT IOTA\_HubDeviceAdd(HW\_UINT uiCookie, ST\_IOTA\_DEVICE\_INFO \*pstDeviceInfo);

参数说明

| **字段** | **必选/可选** | **类型** | **描述** |
| --- | --- | --- | --- |
| uiCookie | 可选 | HW\_UINT | Cookie有效值为1-65535 |
| pstDeviceInfo | 必选 | ST\_IOTA\_DEVICE\_INFO结构体 | 设备信息 |

接口返回值

参见[函数标准返回值](#section129861814143216)

返回结果

| **广播名称** | **广播参数** | **成员** | **描述** |
| --- | --- | --- | --- |
| IOTA\_TOPIC\_HUB\_ADDDEV\_RSP | HW\_MSG对象 | EN\_IOTA\_HUB\_IE\_TYPE | 返回设备添加结果，如果添加成功则返回设备ID |

示例

// 开发者调用该接口进行设备添加   
ST\_IOTA\_DEVICE\_INFO stDeviceInfo   
stDeviceInfo.pcNodeId = “SN Number”;   
stDeviceInfo.pcManufacturerId = “Huawei”;   
stDeviceInfo.pcDeviceType = “Camera”;   
stDeviceInfo.pcModel = “HW\_CAM101”;   
stDeviceInfo.pcProtocolType = “ONVIF”;

IOTA\_HubDeviceAdd(29011, &stDeviceInfo)

结果处理：

// 开发者注册该函数进行设备添加后的处理   
HW\_INT Device\_AddResultHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
uiResult = HW\_MsgGetUint(pstMsg, EN\_IOTA\_HUB\_IE\_RESULT);   
if (EN\_IOTA\_HUB\_RESULT\_SUCCESS != uiResult)   
{   
// retry with uiCookie   
return 0;   
}   
   
return 0;   
}

//绑定广播接收处理函数   
HW\_BroadCastReg(“IOTA\_TOPIC\_HUB\_ADDDEV\_RSP”, Device\_AddResultHandler);

2.设备删除

接口功能

当有新设备需要从网关移除时，通过调用设备删除接口将非直连设备从IoT联接管理平台删除。

接口描述

HW\_INT IOTA\_HubDeviceRemove(HW\_UINT uiCookie, HW\_CHAR \*pcDeviceId);

参数说明

| **字段** | **必选/可选** | **类型** | **描述** |
| --- | --- | --- | --- |
| uiCookie | 可选 | HW\_UINT | Cookie有效值1-65535 |
| pcDeviceId | 必选 | String | 设备Id |

接口返回值

参见[函数标准返回值](#section129861814143216)

返回结果

| **广播名称** | **广播参数** | **成员** | **描述** |
| --- | --- | --- | --- |
| IOTA\_TOPIC\_HUB\_RMVDEV\_RSP | HW\_MSG对象 | EN\_IOTA\_HUB\_IE\_TYPE | 删除结果 |

示例

// 开发者调用该接口进行设备删除   
HW\_CHAR \*pcDeviceId = stDevice.pcDeviceId;   
   
IOTA\_HubDeviceRemove(HW\_NULL, pcDeviceId);   
结果处理：   
HW\_INT Device\_RemoveResultHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
uiResult = HW\_MsgGetUint (pstMsg, EN\_IOTA\_HUB\_IE\_RESULT);   
if (EN\_IOTA\_HUB\_RESULT\_SUCCESS != uiResult)   
{   
// retry with uiCookie   
return 0;   
}   
   
return 0;   
}   
   
HW\_BroadCastReg(“IOTA\_TOPIC\_HUB\_RMVDEV\_RSP”, Device\_RemovResultHandler);

## 设备服务数据上报

接口功能

当非直连设备上报数据到网关时，网关需要调用设备服务数据上报接口将数据上报到IoT联接管理平台。

接口描述

HW\_INTIOTA\_ServiceDataReport(HW\_UINT uiCookie, HW\_CHAR \*pcRequstId,   
HW\_CHAR \*pcDeviceId, HW\_CHAR \*pcServiceId, HW\_CHAR \*pcServiceProperties);

参数说明

| **字段** | **必选/可选** | **类型** | **描述** |
| --- | --- | --- | --- |
| uiCookie | 可选 | unsign int | Cookie有效值1-65535 |
| pcRequstId | 条件必选 | String | 请求ID，用来匹配之前平台下发的服务命令。当该次数据上报为此前某一次命令请求的响应时，需要填写此次命令请求的请求ID。 |
| pcDeviceId | 必选 | String | 设备ID |
| pcServiceId | 必选 | String | 服务ID |
| pcServiceProperties | 必选 | String | 服务属性 |

接口返回值

参见[函数标准返回值](#section129861814143216)

返回结果

| **广播名称** | **广播参数** | **成员** | **描述** |
| --- | --- | --- | --- |
| IOTA\_TOPIC\_DATATRANS\_REPORT\_RSP/{deviceId} | HW\_MSG对象 | EN\_IOTA\_DATAREPORT\_IE\_TYPE | 数据上报结果 |

示例

用户根据Profile格式使用Json组件拼装服务属性的内容(pcServiceProperties);

HW\_UINT \*uiLen;   
   
IOTA\_ServiceDataReport(1211, NULL, "xxxx\_xxxx\_xxxx\_xxxx"   
, "DoorWindow", “{\“status\”:\“OPEN\”}”);

数据上报结果接收

//开发者注册该函数进行设备服务数据上报后的处理   
HW\_INT Device\_DataReportResultHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
uiResult = HW\_MsgGetUint(pstMsg, EN\_IOTA\_DATATRANS\_IE\_RESULT);   
if (HW\_SUCCESS != uiResult)   
{   
// retry with uiCookie   
return 0;   
}   
   
return 0;   
}

//在设备添加成功后立即注册服务数据上报结果接收广播   
HW\_BroadCastReg(“IOTA\_TOPIC\_SERVICE\_REPORT\_RET/XXXX\_XXXX\_XXXX\_XXXX”,   
Device\_AddResultHandler);

## 设备命令接收

接口功能

注册设备命令接收广播来接收处理平台下发的控制命令。

接口描述

IOTA\_TOPIC\_SERVICE\_COMMAND\_RECEIVE/{deviceId}

参数说明

EN\_IOTA\_DATATRANS\_IE\_TYPE

返回结果

无

示例

// 开发者注册该函数进行命令接收的处理   
HW\_INT Switch\_CommandRecvtHandler(HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
HW\_CHAR \*pcMethod, \*pcServiceId, \*pcCmdContent, \*pcDeviceId;   
   
pcDeviceId = HW\_MsgGetStr(pstMsg, EN\_IOTA\_DATATRANS\_IE\_DEVICEID);   
pcServiceId = HW\_MsgGetStr(pstMsg, EN\_IOTA\_DATATRANS\_IE\_SERVICEID);   
pcMethod = HW\_MsgGetStr(pstMsg, EN\_IOTA\_DATATRANS\_IE\_METHOD);   
pcCmdContent = HW\_MsgGetStr(pstMsg, EN\_IOTA\_DATATRANS\_IE\_CMDCONTENT);   
   
if (strcmp(pcServiceId, “switch”))   
{   
//根据Proflie定义的命令参数，使用Json组件解析pcCmdContent   
//Send command to Switch   
}   
   
return 0;   
}

//在设备添加成功后立即注册设备命令接收广播   
HW\_BroadCastReg(“IOTA\_TOPIC\_SERVICE\_CMD\_RECEIVE/XXXX\_XXXX\_XXXX\_XXXX”,   
Device\_AddResultHandler);

开发者需要在设备添加成功后注册该设备的命令接收广播，广播主题为

“ IOTA\_TOPIC\_SERVICE\_CMD\_RECEIVE/设备ID”，Agent Lite收到平台发往给设备的命令后会直接广播给该设备注册的广播处理函数。如果开发者不需要按设备进行分发，直接使用主题名即可即可，即” IOTA\_TOPIC\_SERVICE\_CMD\_RECEIVE”。

## Json组件使用说明

该组件为Agent Lite提供给开发者的工具组件，如果开发者无法进行Json格式的编码和解码，则可以使用该组件进行编码和解码。主要用于上报数据组装与下发命令解析。

1. Json编码

使用Json组件进行编码的流程。

创建Json编码对象。

HW\_JSONOBJ HW\_JsonObjCreate()

获取Json对象根节点。

HW\_JSON HW\_JsonGetJson(HW\_JSONOBJ hjson)

往Json对象中添加键值对：

添加pcVal为字符串的Json键值对。

HW\_INT HW\_JsonAddStr(HW\_JSON \*pstJson, HW\_CHAR \*pcKey, HW\_CHAR \*pcVal)

添加uiVal为整数的Json键值对。

HW\_INT HW\_JsonAddUint(HW\_JSON \*pstJson, HW\_CHAR \*pcKey, HW\_UINT uiVal)

添加bVal为bool的Json键值对。

HW\_INT HW\_JsonAddBool(HW\_JSON \*pstJson, HW\_CHAR \*pcKey, HW\_BOOL bVal)

添加值为Json的Json键值对，获取到的为子Json对象。

HW\_JSON HW\_JsonAddJson(HW\_JSON \*pstJson, HW\_CHAR \*pcKey)

添加值为Json数组Json键值对，获取到的为子Json数组对象

HW\_JSON\_ARRAY HW\_JsonAddJsonArray(HW\_JSON \*pstJson, HW\_CHAR \*pcKey)

往Json数组中添加键值对：

添加pcVal为字符串的Json键值对。

HW\_INT HW\_JsonArrayAddStr(HW\_JSON\_ARRAY \*pstArray, HW\_CHAR \*pcKey, HW\_CHAR \*pcVal)

添加uiVal为整数的Json键值对。

HW\_INT HW\_JsonArrayAddUint(HW\_JSON\_ARRAY \*pstArray, HW\_CHAR \*pcKey, HW\_UINT uiVal)

添加bVal为bool的Json键值对。

HW\_INT HW\_JsonArrayAddBool(HW\_JSON\_ARRAY \*pstArray, HW\_CHAR \*pcKey, HW\_BOOL bVal)

添加pucValue为Json的Json值，获取到的为子Json对象。

HW\_JSON HW\_JsonArrayAddJson(HW\_JSON\_ARRAY \*pstArray)

添加pucValue为Json数组Json键值对，获取到的为子Json数组对象

HW\_JSON\_ARRAY HW\_JsonArrayAddJsonArray(HW\_JSON\_ARRAY \*pstArray, HW\_CHAR \*pcKey)

获取Json字符串

HW\_CHAR HW\_JsonEncodeStr(HW\_JSONOBJ hJson);

删除Json对象

HW\_VOID HW\_JsonObjDelete(HW\_JSONOBJ \*phJson);

**Json编码示例：**

待解析Json格式：

{   
"temperature":22   
"otherInfo":   
{   
“batteryLevel”:”low”   
}   
}   
/\*变量定义\*/   
HW\_JSONOBJ jsonObj;   
HW\_JSON rootjson;   
HW\_JSON json;   
HW\_CHAR \*pcJsonStr;   
   
/\*创建Json编码对象\*/   
hJsonObj = HW\_JsonObjCreate();   
   
/\*获取跟节点Json对象\*/   
rootjson = HW\_JsonGetJson(hJsonObj);   
   
/\*往根节点中添加键值对\*/   
HW\_JsonAddUint(rootjson, "temperature", 22);   
   
/\*从根节点中获取子Json对象\*/   
json = HW\_JsonAddJson(rootjson, "otherInfo");   
   
/\*在子Json中添加键值对\*/   
HW\_JsonAddStr(json, " batteryLevel", "low");   
   
   
/\*获取Json字符串\*/   
pcJsonStr = HW\_JsonEncodeStr(hjsonObj);   
   
/\*删除之前创建的Json编码对象，释放资源\*/   
HW\_JsonObjDelete(&hJsonObj);

2. Json解码

使用Json组件进行解码的流程

创建Json解析对象。

HW\_JSONOBJ HW\_JsonDecodeCreate(HW\_CHAR \*pucStr, HW\_BOOL bStrCpy)

获取Json解析对象中的Json数据部分。

HW\_JSON HW\_JsonGetJson(HW\_JSONOBJ hJson)

获取Json数据中与pucKey对应的字符串。

HW\_CHAR \*HW\_JsonGetStr(HW\_JSON pstJson, HW\_CHAR \*pucKey)

获取Json数据中与pucKey对应的无符号整型。

HW\_UINT HW\_JsonGetUint(HW\_JSON pstJson, HW\_CHAR \*pucKey, HW\_UINT uiDft)

获取Json数据中与pucKey对应的Boolean值。

HW\_BOOL HW\_JsonGetBool(HW\_JSON pstJson, HW\_CHAR \*pucKey, HW\_BOOL bDft)

获取Json数据中与pucKey对应的数组。

HW\_UJSON\_ARRAY HW\_JsonGetArray(HW\_JSON pstJson, HW\_CHAR \*pucKey)

获取Json数组的长度。

HW\_UINT HW\_JsonArrayGetCount(HW\_UJSON\_ARRAY pstArray)

获取Json数组中序号为uiIndex项的Json数据。

HW\_JSON HW\_JsonArrayGetJson(HW\_UJSON\_ARRAY pstArray, HW\_UINT uiIndex)

获取Json数组中序号为uiIndex项的无符号整形。

HW\_UINT HW\_JsonArrayGetUint(HW\_UJSON\_ARRAY pstArray, HW\_UINT uiIndex, HW\_UINT uiDft)

获取Json数组中序号为uiIndex项的Boolean值。

HW\_UINT HW\_JsonArrayGetBool(HW\_UJSON\_ARRAY pstArray, HW\_UINT uiIndex, HW\_BOOL bDft)

获取Json数组中序号为uiIndex项的字符串。

HW\_CHAR \*HW\_JsonArrayGetStr(HW\_UJSON\_ARRAY pstArray, HW\_UINT uiIndex)

获取Json数组中序号为uiIndex项的子数组。

HW\_UJSON\_ARRAY HW\_JsonArrayGetArray(HW\_UJSON\_ARRAY pstArray, HW\_UINT uiIndex)

删除之前创建的Json解析对象。

HW\_VOID HW\_JsonObjDelete(HW\_JSONOBJ \*phJson)

**Json解析示例：**

待解析Json格式：

{   
"action": "notify",   
"type": "userstate",   
"userstateinfo":   
[   
{ "num": "11111 ", "state": "idle"},   
{ "num": "11111", "state": "ringing"}   
]   
}   
/\*变量定义\*/   
HW\_JSONOBJ jsonObj;   
HW\_JSON json;   
HW\_UJSON\_ARRAY jsonArray;   
HW\_CHAR \*action;   
HW\_CHAR \*type;   
HW\_UINT count;   
HW\_UINT index;   
   
/\*创建Json解析对象\*/   
jsonObj = HW\_JsonDecodeCreate(jsonStr, HW\_TRUE);   
   
/\*获取Json解析对象中的Json数据部分\*/   
json = HW\_JsonGetJson(jsonObj);   
   
/\*获取Json数据中与"action"对应的字符串\*/   
action = HW\_JsonGetStr(json, "action");   
   
/\*获取Json数据中与"type"对应的字符串\*/   
type = HW\_JsonGetStr(json, "type");   
   
/\*获取Json数据中与"userstateinfo"对应的Json数组\*/   
jsonArray = HW\_JsonGetArray(json, "userstateinfo");   
   
/\*获取数组jsonArray的长度\*/   
count = HW\_JsonArrayGetCount(jsonArray);   
for (index = 0; index < count; index++)   
{   
/\*获取数组jsonArray中序号为index项的Json数据\*/   
HW\_JSON jsonItem = HW\_JsonArrayGetJson(jsonArray, index);   
   
/\*获取jsonItem中与" num "对应的字符串\*/   
HW\_CHAR \*num = HW\_JsonGetStr(jsonItem, "num");   
   
/\*获取jsonItem中与" state "对应的字符串\*/   
HW\_CHAR \*state = HW\_JsonGetStr(jsonItem, "state");   
......   
}   
   
/\*删除之前创建的Json解析对象，释放资源\*/   
HW\_JsonObjDelete(jsonObj);

# 常用数据结构定义

1. ST\_IOTA\_DEVICE\_INFO 结构体说明

| **字段** | **必选/可选** | **类型** | **描述** |
| --- | --- | --- | --- |
| pcNodeId | 必选 | String | 关键参数，对接平台的网关下设备唯一标识，设备填写，平台用于判重 |
| pcName | 可选 | String | 设备名称 |
| pcDescription | 可选 | String | 设备描述 |
| pcManufacturerId | 必选 | String | 厂商ID |
| pcManufacturerName | 可选 | String | 厂商名 |
| pcMac | 可选 | String | 设备MAC地址 |
| pcLocation | 可选 | String | 设备的位置 |
| pcDeviceType | 必选 | String | 设备类型 |
| pcModel | 必选 | String | 型号  z-wave: ProductType + ProductId  16 进制： XXXX-XXXX 补0对齐  如：001A-0A12  其他协议再定 |
| pcSwVersion | 可选 | String | 软件版本， Z-Wave 主次版本号  主版本号.次版本号 如：1.1 |
| pcFwVersion | 可选 | String | 固件版本 |
| pcHwVersion | 可选 | String | 硬件版本 |
| pcProtocolType | 必选 | String | 协议类型 Z-Wave |
| pcBridgeId | 可选 | String | 表示设备通过哪个Bridge接入平台 |
| pcStatus | 可选 | String | 表示设备是否在线，  ONLINE 在线  OFFLINE 离线 |
| statusDetail | 可选 | String | 状态详情，如果pcStatus不为空，则该参数必选。  参数值：  无：  NONE  配置等待：  CONFIGURATION\_PENDING  通信错误：  COMMUNICATION\_ERROR  配置错误：  CONFIGURATION\_ERROR  桥接器离线  BRIDGE\_OFFLINE  固件升级  FIRMWARE\_UPDATING  循环任务  DUTY\_CYCLE  未激活  NOT\_ACTIVE |
| pcMute | 可选 | String | 表示设备是否被屏蔽：  TRUE  FALSE |

2. EN\_IOTA\_BIND\_IE\_TYPE 消息信元枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** |
| --- | --- | --- | --- |
| EN\_IOTA\_BIND\_IE\_RESULT | 0 | EN\_IOTA\_BIND\_RESULT\_TYPE | 绑定结果 |
| EN\_IOTA\_BIND\_IE\_DEVICEID | 1 | String | 平台分配的逻辑设备ID |
| EN\_IOTA\_BIND\_IE\_DEVICESECRET | 2 | String | 设备接入的鉴权秘钥 |
| EN\_IOTA\_BIND\_IE\_APPID | 3 | String | 开发者应用ID |
| EN\_IOTA\_BIND\_IE\_IOCM\_ADDR | 4 | String | IoCM服务器地址 |
| EN\_IOTA\_BIND\_IE\_IOCM\_PORT | 5 | unsigned int | IoCM服务器端口 |
| EN\_IOTA\_BIND\_IE\_MQTT\_ADDR | 6 | String | MQTT服务器地址 |
| EN\_IOTA\_BIND\_IE\_MQTT\_PORT | 7 | unsigned int | MQTT服务器端口 |
| EN\_IOTA\_BIND\_IE\_IODM\_ADDR | 8 | String | IoDM服务器地址 |
| EN\_IOTA\_BIND\_IE\_IODM\_PORT | 9 | unsigned int | IoDM服务器端口 |

3. EN\_IOTA\_BIND\_RESULT\_TYPE参数枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** |
| --- | --- | --- | --- |
| EN\_IOTA\_BIND\_RESULT\_SUCCESS | 0 | NA | 绑定成功 |
| EN\_IOTA\_BIND\_RESULT\_DEV\_NOT\_BIND | 1 | NA | 未扫码 |
| EN\_IOTA\_BIND\_RESULT\_VERIFYCODE\_EXPIRED | 2 | NA | 验证码过期 |
| EN\_IOTA\_BIND\_RESULT\_FAILED | 255 | NA | 其余失败 |

4. EN\_IOTA\_LGN\_IE\_TYPE 消息信元枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** |
| --- | --- | --- | --- |
| EN\_IOTA\_LGN\_IE\_REASON | 0 | EN\_IOTA\_LGN\_REASON\_TYPE | 登录失败原因 |

5. EN\_IOTA\_LGN\_REASON\_TYPE 参数枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** |
| --- | --- | --- | --- |
| EN\_IOTA\_LGN\_REASON\_NULL | 0 | NA | 无原因 |
| EN\_IOTA\_LGN\_REASON\_CONNCET\_ERR | 1 | NA | 连接失败 |
| EN\_IOTA\_LGN\_REASON\_SERVER\_BUSY | 2 | NA | 服务器忙 |
| EN\_IOTA\_LGN\_REASON\_AUTH\_FAILED | 3 | NA | 鉴权失败、开发者需要停止重新尝试登陆。 |
| EN\_IOTA\_LGN\_REASON\_NET\_UNAVAILABLE | 5 | NA | 网络不可用 |
| EN\_IOTA\_LGN\_REASON\_DEVICE\_NOEXIST | 12 | NA | 设备不存在、开发者需要停止重新尝试登陆。 |
| EN\_IOTA\_LGN\_REASON\_DEVICE\_RMVED | 13 | NA | 设备已删除、开发者需要停止重新尝试登陆。 |
| EN\_IOTA\_LGN\_REASON\_UNKNOWN | 255 | NA | 未知原因 |

6. EN\_IOTA\_CFG\_TYPE 参数枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** |
| --- | --- | --- | --- |
| EN\_IOTA\_CFG\_DEVICEID | 0 | String | 平台分配的逻辑设备ID |
| EN\_IOTA\_CFG\_DEVICESECRET | 1 | String | 设备接入的鉴权秘钥 |
| EN\_IOTA\_CFG\_APPID | 2 | String | 开发者应用ID |
| EN\_IOTA\_CFG\_IOCM\_ADDR | 3 | String | IoCM服务器地址 |
| EN\_IOTA\_CFG\_IOCM\_PORT | 4 | unsigned int | IoCM服务器端口 |
| EN\_IOTA\_CFG\_MQTT\_ADDR | 5 | String | MQTT服务器地址 |
| EN\_IOTA\_CFG\_MQTT\_PORT | **6** | unsigned int | MQTT服务器端口 |
| EN\_IOTA\_CFG\_IODM\_ADDR | **7** | String | IoDM服务器地址 |
| EN\_IOTA\_CFG\_IODM\_PORT | **8** | unsigned int | IoDM服务器端口 |

7. EN\_IOTA\_HUB\_IE\_TYPE 消息信元枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** |
| --- | --- | --- | --- |
| EN\_IOTA\_HUB\_IE\_RESULT | 0 | EN\_IOTA\_HUB\_RESULT\_TYPE | 添加/删除执行结果 |
| EN\_IOTA\_HUB\_IE\_DEVICEID | 1 | String | 添加成功后分配的设备ID |

8. EN\_IOTA\_HUB\_RESULT\_TYPE 参数枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** |
| --- | --- | --- | --- |
| EN\_IOTA\_HUB\_RESULT\_SUCCESS | 0 | NA | 添加/删除执行成功 |
| EN\_IOTA\_HUB\_RESULT\_DEVICE\_EXIST | 1 | NA | 设备已存在 |
| EN\_IOTA\_HUB\_RESULT\_DEVICE\_NOTEXIST | 2 | NA | 设备不存在 |
| EN\_IOTA\_HUB\_RESULT\_DEVICE\_FAILED | 255 | NA | 执行失败 |

9. EN\_IOTA\_DATATRANS\_IE\_TYPE 消息信元枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** | |
| --- | --- | --- | --- | --- |
| EN\_IOTA\_DATATRANS\_IE\_RESULT | 0 | unsigned int | 命令执行返回结果 | 成功：0 |
| 失败：1 |
| EN\_IOTA\_DATATRANS\_IE\_DEVICEID | 1 | String | 设备ID | |
| EN\_IOTA\_DATATRANS\_IE\_REQUESTID | 2 | String | 请求ID | |
| EN\_IOTA\_DATATRANS\_IE\_SERVICEID | 3 | String | 服务ID | |
| EN\_IOTA\_DATATRANS\_IE\_METHOD | 4 | String | 服务方法 | |
| EN\_IOTA\_DATATRANS\_IE\_CMDCONTENT | 5 | String | 命令内容，以Json格式进行拼装的服务命令参数，开发者根据对应服务命令的定义进行Json解析，从而获取命令参数值。 | |

10. EN\_IOTA\_DEVUPDATE\_IE\_TYPE 消息信元枚举说明

| **枚举项** | **枚举值** | **类型** | **描述** | |
| --- | --- | --- | --- | --- |
| EN\_IOTA\_DEVUPDATE\_IE\_RESULT | 0 | unsigned int | 命令执行返回结果 | 成功：0 |
| 失败：1 |
| EN\_IOTA\_DEVUPDATE\_IE\_DEVICEID | **1** | **String** | 设备ID | |

# 数据类型定义

1. 常用数据类型

| 类型名称 | **类型原型** |
| --- | --- |
| HW\_INT | int |
| HW\_UINT | unsigned int |
| HW\_CHAR | char |
| HW\_UCHAR | unsigned char |
| HW\_BOOL | int |
| HW\_ULONG | unsigned long |
| HW\_USHORT | unsigned short |
| HW\_MSG | void\* |
| HW\_VOID | void |
| HW\_NULL | 0 |

2. 函数标准返回值

| 返回值名称 | **值** | **类型原型** |
| --- | --- | --- |
| HW\_OK | 0 | 执行成功 |
| HW\_ERR | 1 | 执行错误 |
| HW\_ERR\_PTR | 2 | 错误的指针 |
| HW\_ERR\_ID | 3 | 错误的ID |
| HW\_ERR\_PARA | 4 | 错误的参数 |
| HW\_ERR\_KEY | 5 | 错误的KEY |
| HW\_ERR\_NOMEM | 6 | 内存不足 |
| HW\_ERR\_MAGIC | 7 | 保留 |
| HW\_ERR\_OVERFLOW | 8 | 存在溢出 |
| HW\_ERR\_GVAR | 9 | 保留 |
| HW\_ERR\_POOL | 10 | 保留 |
| HW\_ERR\_NO\_MUTEX | 11 | 未加锁 |
| HW\_ERR\_PID | 12 | 保留 |
| HW\_ERR\_FILEOPEN | 13 | 文件打开失败 |
| HW\_ERR\_FD | 14 | 错误的文件描述符 |
| HW\_ERR\_SOCKET | 15 | SOCKET异常 |
| HW\_ERR\_NOTSUPPORT | 16 | 不支持 |
| HW\_ERR\_NOTLOAD | 17 | 未加载 |
| HW\_ERR\_ENCODE | 18 | 编码错误 |
| HW\_ERR\_DECODE | 19 | 解码错误 |
| HW\_ERR\_CALLBACK | 22 | 错误的回调函数 |
| HW\_ERR\_STATE | 23 | 错误的状态 |
| HW\_ERR\_OVERTIMES | 24 | 重试超过次数 |
| HW\_ERR\_ENDOVER | 25 | 保留 |
| HW\_ERR\_ENDLINE | 26 | 保留 |
| HW\_ERR\_NUMBER | 27 | 错误的数字 |
| HW\_ERR\_NOMATCH | 28 | 不匹配 |
| HW\_ERR\_NOSTART | 29 | 未开始 |
| HW\_ERR\_NOEND | 30 | 未结束 |
| HW\_ERR\_OVERLAP | 31 | 保留 |
| HW\_ERR\_DROP | 32 | 丢弃 |
| HW\_ERR\_NODATA | 33 | 无数据 |
| HW\_ERR\_CRC\_CHK | 34 | CRC校验失败 |
| HW\_ERR\_AUTH | 35 | 鉴权失败 |
| HW\_ERR\_LENGTH | 36 | 长度错误 |
| HW\_ERR\_NOTALLOW | 37 | 不被允许的操作 |
| HW\_ERR\_TOKEN | 38 | 凭据错误 |
| HW\_ERR\_NOTIPV4 | 39 | 不支持IPV4 |
| HW\_ERR\_NOTIPV6 | 40 | 不支持IPV6 |
| HW\_ERR\_IELOST | 41 | 保留 |
| HW\_ERR\_IELOST1 | 42 | 保留 |
| HW\_ERR\_IELOST2 | 43 | 保留 |
| HW\_ERR\_AUDIO | 44 | 保留 |
| HW\_ERR\_VIDEO | 45 | 保留 |
| HW\_ERR\_MD5 | 46 | 保留 |
| HW\_ERR\_MD5\_HA1 | 47 | 保留 |
| HW\_ERR\_MD5\_RES | 48 | 保留 |
| HW\_ERR\_DIALOG | 49 | 错误的对话 |
| HW\_ERR\_OBJ | 50 | 错误的对象 |