|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |
|  | | |
| 方圆ok1 | | | |
|  | **IoM 解决方案**  **V100R001C10**  **华为IoT Agent Lite集成开发指南(C)** | | 附件1-16K |
|  | |
| **文档版本** | **01** |
| **发布日期** | **2016-09-29** |
| **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](#_Toc462950655)

[2 概述 3](#_Toc462950656)

[3 开发环境 9](#_Toc462950657)

[4 开发准备 11](#_Toc462950658)

[5 网关业务开发指导 14](#_Toc462950659)

[5.1 开发步骤 14](#_Toc462950660)

[5.1.1 将Demo在目标设备上运行起来 15](#_Toc462950661)

[5.1.2 设备初始化 16](#_Toc462950662)

[5.1.3 网关绑定 16](#_Toc462950663)

[5.1.4 网关登录 19](#_Toc462950664)

[5.2 开发样例（Demo库） 20](#_Toc462950665)

[5.3 调测结果 20](#_Toc462950666)

[5.3.1 注册APP账号 20](#_Toc462950667)

[5.3.2 网关上电 23](#_Toc462950668)

[5.3.3 调测 23](#_Toc462950669)

[6 设备业务开发 30](#_Toc462950670)

[7 参考 32](#_Toc462950671)

# 前 言

概述

本文档针对各种设备的集成，从设备的集成条件、集成方法到设备的简单使用，提供了具体的处理过程和步骤。

本文档能指导开发者快速集成设备到华为IoT联接管理平台中，并使用华为HiHo App呈现并添加设备。本文档的适用场景主要有两种：

* 网关开发厂商：开发厂商通过把网关集成到IoT联接管理平台，通关网关能够集成和管理多个非直连设备（比如通过网关添加温度传感器，进行数据上报的操作）。
* 设备开发厂商：例如门磁、 IP摄像头等传感器设备。该种直连设备需要直连云端，绑定并登陆之后即可直接数据上报等操作，无需再进行设备的添加。

读者对象

本文档主要适用于网关和设备厂商的开发人员，开发人员必须熟悉所要集成的设备的功能、掌握相关的接口知识、具备一定的设备知识背景。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

| 符号 | 说明 |
| --- | --- |
|  | 表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。 |
|  | 表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。 |
|  | 表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。 |
|  | 表示能帮助您解决某个问题或节省您的时间。 |
|  | 表示是正文的附加信息，是对正文的强调和补充。 |

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 |
| --- | --- | --- | --- |
| 1 | 2016/05/30 | 第一次发布 |  |
|  |  |  |  |

# 概述

能力概述

华为IoT Agent Lite是华为公司开发的轻量级的物联网设备中间件，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。

通过华为提供的IoT联接管理平台、Agent Lite开放API和HiHo App，开发者通过少量开发就能快速集成网关和智能终端等设备，实现智能设备功能的快速验证。

IoT Agent Lite提供了如下功能：

* 设备的绑定登录
* 设备的添加删除
* 设备的数据上报
* 下发平台控制命令
* 更新设备状态信息

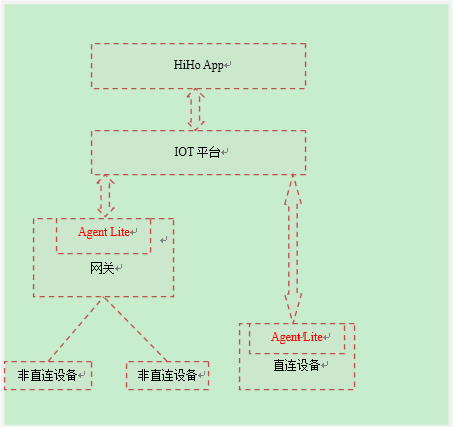
[网关业务开发指导](#_ZH-CN_TOPIC_0035448630)中主要描述了传感器数据上报结果处理、传感器状态更新结果处理和传感器移除结果处理函数。

[设备业务开发](#_ZH-CN_TOPIC_0035448726)指导中主要描述开发者可直接将Demo在目标设备上运行起来，并基于Demo的已有代码做必要的修改，完成所需功能。

如下图所示，左边场景是针对有自己的网关和标准协议(Z-Wave，ZigBee)的传感器的开发厂商，可以通过Agent Lite把网关连接到IoT平台，同时通过网关把传感器集成到平台。

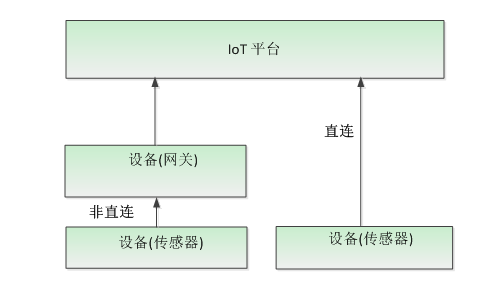
右边场景针对的是没有自己的网关，只有非标准协议的传感器的开发厂商，也可以通过Agent Lite把传感器连接到IoT平台。

不同厂商的开发场景



基本概念

* 服务（Service）：服务是设备的一项完整功能，每种服务对应一种设备能力
* 属性（Property）：一个或多个属性组成一种服务，是服务的具体分化。例如温度服务，有温度（用于表示当前温度）及目标温度（用于调节温度）两个属性。开发者可以根据所集成设备的情况指定使用一种服务下的一个或多个属性。
* 设备分类设备根据逻辑拓扑可以分为两类，直连设备与非直连设备：



- 对于直连设备，可以直接进行绑定与登录。

- 对于非直连设备，需要通过网关设备进行接入。

直连设备又分为网关与传感器设备（例如传感器、网桥）：

- 对于网关设备，具备设备管理能力，能够管理多个非直连设备（添加、删除、服务能力上报、服务数据上报、服务命令分发）。

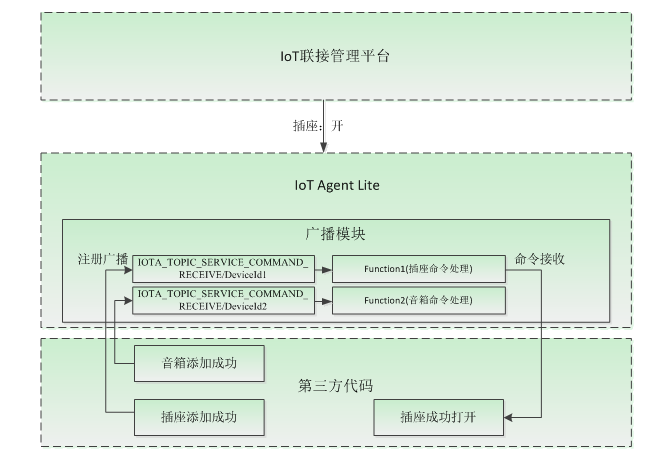
- 对于传感器设备（例如门磁、 IP摄像头、或者抽象的数据源），绑定并登录之后即可直接进行服务能力上报*、*服务数据上报、接收服务命令，无需再进行设备的添加。

* 广播机制

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。广播由广播主题与广播接收处理函数组成。

广播主题包含一些列特定的事件，例如“连接成功”、“命令接收”等；

广播接收处理函数则是触发该事件后，第三方开发者需要进行对应处理的函数，相当于回调函数。其原理如下：



技术优势

华为IoT Agent Lite在技术上主要优势体现在如下几个方面。

技术优势

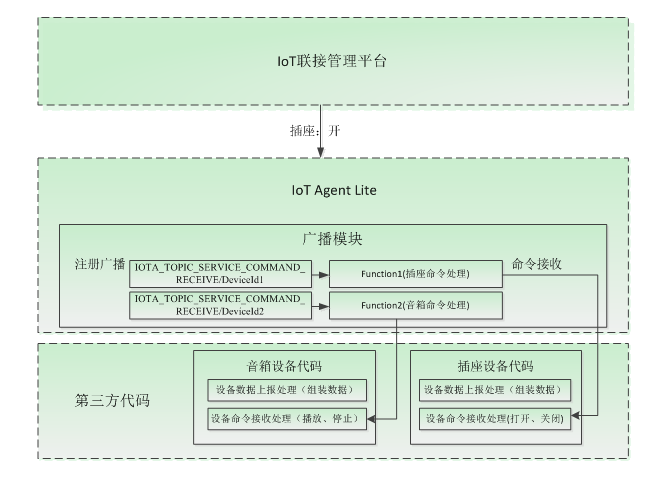
| 优势 | 优势说明 |
| --- | --- |
| 支持多平台多操作系统 | 支持多种操作系统，支持通过交叉编译快速适配各种嵌入式系统。 |
| 少量适配，快速集成 | 第三方开发者只需要理解少量业务接口即可完成接入，无需适配硬件与系统接口。 |
| 支持断网重连 | Agent Lite支持网络断开后自动重新连接。 |
| 支持设备解耦 | Agent Lite支持根据设备号广播控制命令，可以帮助第三方开发者在在开发应用时，实现软件架构上的解耦。 |

说明

Agent Lite支持根据设备号广播控制命令，可以帮助第三方开发者在在开发应用时，实现软件架构上的解耦。例如开发者需要接入了电灯与门磁两类传感器设备，开发者可以分别对两类设备实现设备数据上报、设备数据上报结果接收、设备命令接收

公共的设备管理模块在添加完设备后，注册设备数据上报结果接收与设备命令接收的广播，广播主题为({主题名}/{设备ID})。则在收到设备的数据与命令后，Agent Lite会将对应命令的直接送给该模块注册的广播处理函数，无需再进行分发，从而实现公共管理模块与具体设备模块之间的解耦。

示例：



总体开发流程

开发流程

| 优势 | 优势说明 |
| --- | --- |
| 1.获取软件及文档 | 详情参见[获取软件及文档，检查资源是否齐全](#section62741028412) |
| 2.准备网关或设备 | 详情参见[准备网关或设备，检查开发环境是否满足要求](#section1640019131859) |
| 3.确认是否新增设备描述文件Profile | 详情参见[确认是否需要新增Profile](#section1191616353714) |
| 4.参考样例程序开发 | 详情参见[网关业务开发指导](#_ZH-CN_TOPIC_0035448630)和[设备业务开发](#_ZH-CN_TOPIC_0035448726) |
| 5．使用Agent Lite相关接口进行开发 | 参考《华为IoT Agent Lite API参考(C).docx》 |
| 6.调测结果 | 详情参见[网关业务开发指导，调测结果](#_ZH-CN_TOPIC_0035448753)和[设备业务开发，调测结果](#section3609212182516) |

# 开发环境

开发环境要求

硬件规格要求

| CPU | RAM | FLASH | 电池 |
| --- | --- | --- | --- |
| 暂无 | >4M | >600KB | 带电设备 |

交叉编译支持

| 芯片架构 | 交叉编译链 |
| --- | --- |
| ARM | gcc-linaro-arm-linux-gnueabihf-r[asp](http://www.2cto.com/kf/web/asp/)bian |
| ARM | arm-none-linux-gnueabi |
| ARM | arm-linux-uclibceabi |

说明

如果开发者没有设备，可以直接在X86 Linux系统进行开发。

工程目录结构及文件说明

工程目录结构及文件说明

| 目录结构 | 目录 | 说明 |
| --- | --- | --- |
| Agent Lite  │  ├─linux  │ ├─include  │ ├─demo  │ ├─lib  │ │ ├─platform1  │ │ └─platform2  │ └─test  │ ├─platform1  │ └─platform2  └─workdir  └─conf | include | 存放Agent Lite头文件 |
| lib | 存放Agent Lite编译后的库文件和第三方库文件 |
| demo\_\* | 存放演示工程源文件和头文件 |
| conf | 存放TLS证书文件 |
| test | 用于交叉编译验证的测试程序 |

# 开发准备

获取软件及文档，检查资源是否齐全

* 下载生态开发资源包

生态开发资源包

| 软件及文档 | 内容说明 | 获取方式 |
| --- | --- | --- |
|  | AgentLite开发工具包，其中包含库文件及头文件、开发环境检查工具 |  |
| Agent Lite Demo(C).zip |  |  |
| 华为IoT Agent Lite API参考(C).docx | Agent Lite API文档 |  |
| 华为IoT Agent Lite集成开发指南(C).docx | Agent Lite开发指导 |  |
| 华为IoT设备能力描述文件profile开发指南.docx |  | 需要开发者联系华为IoT客服获取。客服邮箱：<mailto:iotadmin@huawei.com> |
| 华为IoT 树莓派Raspberry Pi系统安装指导.docx |  |  |

准备网关或设备，检查开发环境是否满足要求

* 准备网关或设备

开发者需要根据[开发环境](#_ZH-CN_TOPIC_0035448750)确认待集成的网关或设备（传感器）的开发环境是否满足要求。

* 确认是否支持交叉编译

交叉编译的目的是在某一个平台下能够编译出另外一个平台下运行的程序。为了确保提供的库文件能够在第三方的交叉编译工具下编译出可执行的程序，开发前请确认华为IoT Agent Lite是否已经支持所开发的设备交叉编译，华为会提供每个交叉编译版本的测试程序。

* 如果在开发环境上所有基本用例通过，表示支持该交叉编译工具。
* 如有用例不通过则表示不支持，则需要提交交叉编译工具与寄送开发设备进行联调测试。

交叉编译验证工具mytest使用步骤如下：

进入开发环境上存放mytest的目录

SZX1000032993:/opt/z00367380/iotm1/code/project # cd .linux\_build/   
SZX1000032993:/opt/z00367380/iotm1/code/project/.linux\_build # ls   
CMakeCache.txt Makefile libhwjson.a libhwsys.a libuspbase.a libusphttp.a libuspiodev.a libuspmapi.a libuspmqtt.a libuspsdk.so libuspwatcher.a   
CMakeFiles cmake\_install.cmake libhwmapi.a libiota.a libuspbind.a libuspiocmreg.a libusplogin.a libuspmqta.a libuspprotocol.a libusptls.a mytest

执行./mytest运行mytest程序

SZX1000032993:/opt/z00367380/iotm1/code/project/.linux\_build # ./mytest

如果mytest程序执行成功，则表示支持该交叉编译工具

Case 32006: UHttp\_TestOnRecvSACheck() 151 ...... SetOK   
Case 32006: ........... OK   
Case 42001: UHttp\_TestOnRecvDnsCheck() 311 ...... Check   
Case 42001: UHttp\_TestOnRecvSACheck() 163 ...... SetOK   
Case 42001: ........... OK   
USimulator\_SockUnload   
Test Start <--- ULogin\_TestFunc() ......   
Case 1023: ULogin\_TestOnRecvSALogoutCheck() 70 ...... SetOK   
Case 1023: ........... OK   
Case 1024: ULogin\_TestOnRecvSALoginCheck() 102 ...... SetOK   
Case 1024: ........... OK   
Test OK <--- ULogin\_TestLocal() ......   
======================= Export Mem Leak ======================   
=======================AUTO TEST END SUCC======================

如果mytest程序过程中有失败，则表示不支持该交叉编译工具或者存在适配问题。

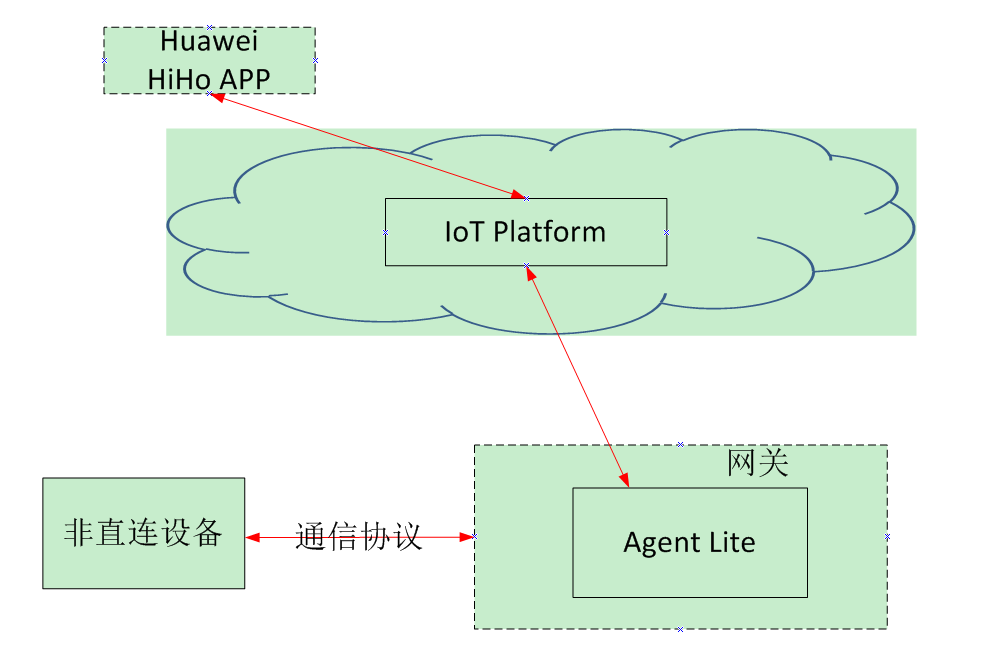
Case 42001: UHttp\_TestOnRecvDnsCheck() 311 ...... Check   
Case 42001: UHttp\_TestOnRecvSACheck() 163 ...... SetOK   
Case 42001: ........... OK   
USimulator\_SockUnload   
Test Start <--- UBind\_TestFunc() ......   
Case 22000: Fail!!---> UBind\_TestServerFunc(234) wait: UBind\_TestServerOnRecvSACheck()...   
Test Start <--- ULogin\_TestFunc() ......   
Case 1023: ULogin\_TestOnRecvSALogoutCheck() 70 ...... SetOK   
Case 1023: ........... OK   
Case 1024: ULogin\_TestOnRecvSALoginCheck() 102 ...... SetOK   
Case 1024: ........... OK   
Test OK <--- ULogin\_TestLocal() ......   
======================= Export Mem Leak ======================   
!!Assert Failed!! Ugp\_TestDestroy(168): 20000 > Ugp\_MemDebugGetAllSize()

确认是否需要新增Profile

本文后续的Demo，要运行起来，Profile已经预置在IoT平台，开发者不需要写作Profile。如果是新增的设备，请参考《华为IoT 设备能力描述文件profile开发指南.docx》判断新增的设备是否需要新增Profile，如果涉及新增Profile，请参考该文档的要求写作Profile。

# 网关业务开发指导

基本概念



[5.1 开发步骤](#_ZH-CN_TOPIC_0035448766)

[5.2 开发样例（Demo库）](#_ZH-CN_TOPIC_0037842505)

[5.3 调测结果](#_ZH-CN_TOPIC_0035448753)

## 开发步骤

开发者可直接将Demo在目标设备上运行起来，并基于Demo的已有代码做必要的修改，完成所需功能。

本文基于Demo介绍绑定网关接口的使用，其他接口的介绍，可参考《华为IoT Agent Lite API参考(C).docx》。

说明

如果开发者需要了解其他接口使用示例，请联系华为IoT客服获取。客服邮箱：iotadmin@huawei.com

### 将Demo在目标设备上运行起来

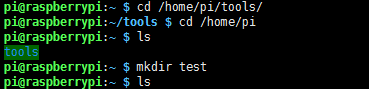
本节基于在树莓派环境下运行Demo介绍。开发者如果使用的是非树莓派，只要硬件环境满足Agent Lite的运行条件，开发方法相同。

说明

操作前需要确保树莓派环境已经搭建好，在xshell中新建新会话，输入环境ip，并输入用户名密码进入树莓派环境。详细参考《华为IoT 树莓派Raspberry Pi系统安装指导.docx》。

交叉编译准备，将Demo程序拷贝到树莓派上。举例说明如下：

1. 新建目录home，在home目录下新建pi目录，使用cd /home/pi进入到工作目录，使用mkdir test在该目录下新建一个目录test，所需要的文件多放在该目录中。具体文件所放置的目录开发者自己定。



1. 上传下载的所有文件，包括所有的.h文件和.so文件，以及开发者所编写的.h文件和.c文件。在下面功能栏中找到新建文件传输，打开，在该界面中输入put命令，然后在开发者本地文件夹中把文件上传。使用ls命令确认文件是否上传成功。



进行交叉编译。

1. 输入“chmod 777 \*”命令修改权限。
2. 输入“.export LD\_LIBRARY\_PATH=./”设置程序共享库位置。
3. 输入“gcc demo.c -L /home/pi/test -lpthread -ldl -lrt -luspsdk -o sdk.out”命令，如果交叉编译成功，会产生sdk.out库文件。

说明

其中luspsdk为文件libuspsdk.so，具体linux交叉编译命令gcc请开发者自行学习。

运行Demo。

启动demo命令为“./ sdk.out”，输入该命令程序就能运行起来了，同时打开APP进行扫码，可以进行绑定、登录和数据上报等流程的验证。

----结束

下面以Demo使用的接口为例，说明Agent Lite API接口使用：

### 设备初始化

在发起业务前，需要先初始化Agent Lite相关资源。

说明

调用API接口IOTA\_Init()，初始化Agent Lite资源，具体API的参数使用参考《华为IoT Agent Lite API参考(C).docx》中的设备初始化接口HW\_INT IOTA\_Init说明。

代码示例：

IOTA\_Init(CONFIG\_PATH,HW\_NULL)；

初始化Agent Lite相关资源，其中：

* CONFIG\_PATH：工作路径。
* HW\_NULL：打印日志路径。

说明

当pcLogPath为HW\_NULL时，打印路径默认为工作路径，开发者也可以自己定义打印日志路径。

### 网关绑定

设备第一次接入IoT联接管理平台时需要进行绑定操作，将设备关联到平台上。开发者通过传入设备序列号以及设备信息，将网关绑定到IoT联接管理平台。绑定操作需要通过IoT联接管理平台北向接口配合操作，通常有两种绑定方式：

* 提前在平台上预制设备的设备序列号，设备初始启动后发起绑定流程，将设备绑定到IoT联接管理平台中。
* 通过APP终端扫描网关序列号生成的二维码图片动态绑定网关，扫码后APP通过命令动态往平台注册设备的序列号，之后重启网关进行绑定。

说明

APP为华为自己开发的HiHo APP，请联系华为技术支撑人员获取HiHo App。

绑定前先调用API接口HW\_INT IOTA\_ConfigSetStr ()与IOTA\_ConfigSetUint()设置绑定服务器IP与端口，接着调用API接口HW\_INT IOTA\_Bind()进行网关绑定。

说明

具体API的参数使用《华为IoT Agent Lite API参考(C).docx》中的参数配置接口HW\_INT IOTA\_SetConfig和设备绑定接口HW\_INT IOTA\_Bind说明。

代码示例：   
DEVICE\_ReadConf();   
if (NULL == g\_stGateWayInfo.pcDeviceID)   
 {   
 while(1)   
 {   
 if(HW\_OK == DEVICE\_BindGateWay())   
 {   
 HW\_LOG\_INF("IOTA\_Register ok");   
 break;   
 }   
   
 HW\_Sleep(1);   
 continue;   
 }   
 }   
 else   
 {   
 //进行登录流程   
 }   
HW\_BroadCastReg(Device\_ServiceCommandReceiveHandler,acBuf);   
HW\_BroadCastReg(Device\_ServiceDataReportResultHandler,acBuf);   
}   
HW\_BroadCastReg(IOTA\_TOPIC\_BIND\_RSP, Device\_RegResultHandler);   
HW\_BroadCastReg(IOTA\_TOPIC\_CMD\_UNBIND\_RECEIVE, Gateway\_UnbindRecvtHandler);   
HW\_BroadCastReg(IOTA\_TOPIC\_CONNECTED\_NTY, Device\_ConnectedHandler);   
HW\_BroadCastReg(IOTA\_TOPIC\_DISCONNECT\_NTY, Device\_DisconnectHandler);   
HW\_BroadCastReg(IOTA\_TOPIC\_HUB\_ADDDEV\_RSP, Device\_AddResultHandler);   
HW\_BroadCastReg(IOTA\_TOPIC\_HUB\_RMVDEV\_RSP, Device\_RemovResultHandler);   
HW\_BroadCastReg(IOTA\_TOPIC\_DEVUPDATE\_RSP, Device\_DevUpDateHandler);   
   
HW\_UINT DEVICE\_BindGateWay()   
{   
ST\_IOTA\_DEVICE\_INFO stDeviceInfo = {0};   
HW\_CHAR szEsnInfo[BUFF\_MAX\_LEN] = {0};   
 //get gw reg info   
 fp = fopen(szGwRegInfoFileName,"r");   
 fseek(fp , 0 , SEEK\_END );   
 file\_size = ftell( fp );   
 pcJsonStr = (char \*)malloc( (file\_size+1) \* sizeof( char ) );   
 fclose(fp);   
   
 jsonObj = HW\_JsonDecodeCreate(pcJsonStr, HW\_TRUE);   
 json = HW\_JsonGetJson(jsonObj);   
 pucEsnFile = HW\_JsonGetStr(json, IOTA\_AR\_ESN\_INI\_FILE);   
 pucPlatformAddr = HW\_JsonGetStr(json, IOTA\_PLATFORM\_ADDR);   
 uiPort = HW\_JsonGetUint(json, IOTA\_PLATFORM\_PORT,8943);   
 stDeviceInfo.pcManufacturerId = HW\_JsonGetStr(json, IOTA\_MANUFACTURE\_ID);   
 stDeviceInfo.pcDeviceType = HW\_JsonGetStr(json, IOTA\_DEVICE\_TYPE);   
 stDeviceInfo.pcModel = HW\_JsonGetStr(json, IOTA\_MODEL);   
 stDeviceInfo.pcProtocolType = HW\_JsonGetStr(json, IOTA\_PROTOCOL\_TYPE);   
 g\_uiLogLevel = HW\_JsonGetUint(json, IOTA\_LOGLEVEL,0);   
 HW\_LogSetLevel(g\_uiLogLevel);   
   
 //get esn   
 fpEsn = fopen(pucEsnFile,"r");   
 fgets(linebuf,sizeof(linebuf),fpEsn);   
 uiLen = strlen(linebuf);   
 uiCnt = 0;   
 memset(szEsnInfo,0,sizeof(szEsnInfo));   
 for(uiLoop = 0; uiLoop < uiLen; uiLoop++)   
 {   
 if('\n' != linebuf[uiLoop])   
 {   
 if(uiCnt >= BUFF\_MAX\_LEN)   
 {   
 HW\_LOG\_ERR("buf is too small.\r\n");   
 fclose(fpEsn);   
 HW\_JsonObjDelete(&jsonObj);   
 free(pcJsonStr);   
 return HW\_ERR;   
 }   
   
 szEsnInfo[uiCnt++] = linebuf[uiLoop];   
 }   
 }   
 fclose(fpEsn);   
   
 stDeviceInfo.pcMac = szEsnInfo;   
 stDeviceInfo.pcNodeId = szEsnInfo;   
 IOTA\_ConfigSetStr (EN\_IOTA\_CFG\_IOCM\_ADDR, pucPlatformAddr);   
 IOTA\_ConfigSetUint(EN\_IOTA\_CFG\_IOCM\_PORT, &uiPort);   
 IOTA\_Bind(stDeviceInfo.pcMac,&stDeviceInfo);   
}

DEVICE\_ReadConf()

| 参数 | 参数说明 |
| --- | --- |
| DEVICE\_ReadConf() | 读取配置文件信息。 |
| if (NULL == g\_stGateWayInfo.pcDeviceID) | 判断设备信息文件是否存在，如果不存在，则进行绑定流程。 |
| DEVICE\_BindGateWay()函数 | * ST\_IOTA\_DEVICE\_INFO stDeviceInfo = {0}为存储设备信息的结构体 * szEsnInfo[BUFF\_MAX\_LEN] = {0}存储mac地址的字符串数组。 * fp = fopen(szGwRegInfoFileName,"r")到HW\_LogSetLevel(g\_uiLogLevel)之间的流程是把配置文件的信息以json形式读取，并存储在结构体数组中 * fpEsn = fopen(pucEsnFile,"r")到fclose(fpEsn)获取根文件esn中的mac地址 * IOTA\_ConfigSetStr、IOTA\_ConfigSetUint进行参数配置，然后调用API接口IOTA\_Bind进行网关的绑定。同时在网关绑定和登录流程成功后，进行一系列广播函数的注册， * HW\_BroadCastReg(Device\_RegResultHandler,IOTA\_TOPIC\_BIND\_RSP)是在第一次绑定成功后触发该广播的，其余的广播是在网关绑定成功后或者设备添加成功后触发的。 |

### 网关登录

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，在网关设备成功登录IoT联接管理平台后，可以进行其它服务操作，比如传感器接入，数据上报等等。如果网关登录成功，那么网关在平台的状态则显示已在线。

* 如果网关第一次绑定成功，会触发广播函数HW\_BroadCastReg(IOTA\_TOPIC\_REG\_RSP, Device\_RegResultHandler)中的广播接收处理函数Device\_RegResultHandler()，在该函数中调用API接口HW\_INT IOTA\_Login()进行网关的登录。
* 如果网关设备不是第一次绑定，比如网关设备绑定后遇到断电重启网关设备的情况：在网关登录前需要通过参数配置接口HW\_INT IOTA\_ConfigSetStr 与IOTA\_ConfigSetUint配置所需的登录信息，接着开发者调用API接口IOTA\_Login()进行网关登录平台的流程。

**代码示例：**

HW\_INT Device\_RegResultHandler(HW\_CHAR \*pucAction,HW\_UINT uiCookie, HW\_MSG pstMsg)   
{   
 uiRegRet = HW\_MsgGetUint(pstMsg,EN\_IOTA\_REG\_IE\_RESULT,EN\_IOTA\_REG\_RESULT\_FAILED);   
 pcDeviceId = HW\_MsgGetStr(pstMsg, EN\_IOTA\_REG\_IE\_DEVICEID);   
 pcDeviceSecret = HW\_MsgGetStr(pstMsg, EN\_IOTA\_REG\_IE\_DEVICESECRET);   
 pcAppId = HW\_MsgGetStr(pstMsg, EN\_IOTA\_REG\_IE\_APPID);   
 pcIoCMServerAddr = HW\_MsgGetStr(pstMsg, EN\_IOTA\_REG\_IE\_IOCM\_ADDR);   
 hJsonObj = HW\_JsonObjCreate();   
 rootjson = HW\_JsonGetJson(hJsonObj);   
 HW\_JsonAddStr(rootjson , IOTA\_CFG\_DEVICEID, pcDeviceId);   
 HW\_JsonAddStr(rootjson , IOTA\_CFG\_DEVICESECRET, pcDeviceSecret);   
 HW\_JsonAddStr(rootjson , IOTA\_CFG\_APPID, pcAppId);   
 HW\_JsonAddStr(rootjson , IOTA\_CFG\_IOCM\_ADDR, pcIoCMServerAddr);   
 HW\_JsonAddUint(rootjson , IOTA\_LOGLEVEL, g\_uiLogLevel);   
 pcJsonStr = HW\_JsonEncodeStr(hJsonObj);   
 sprintf(aszName,"%s/%s",CONFIG\_PATH,GATEWAY\_BIND\_INFO\_FILE);   
 fw = fopen(aszName,"w");   
   
 fputs(pcJsonStr,fw);   
 fclose(fw);   
 HW\_JsonObjDelete(&hJsonObj);   
 DEVICE\_ReadConf();   
 IOTA\_Login();   
 return HW\_OK;   
}

代码解释如下：

* 在第一次绑定成功流程执行后，会触发广播函数HW\_BroadCastReg(Device\_RegResultHandler,IOTA\_TOPIC\_REG\_RSP)。
* 在广播接收处理函数Device\_RegResultHandler函数中，pcDeviceId = HW\_MsgGetStr(pstMsg, EN\_IOTA\_REG\_IE\_DEVICEID)到pcIoCMServerAddr = HW\_MsgGetStr(pstMsg, EN\_IOTA\_REG\_IE\_IOCM\_ADDR)直接获取消息指针中的pcDeviceId等信息。
* hJsonObj = HW\_JsonObjCreate()到HW\_JsonObjDelete(&hJsonObj)之间是创建.json文件，按json的形式保存pcDeviceId等信息。
* DEVICE\_ReadConf()读取这些信息，然后调用API接口IOTA\_Login()进行网关的登录操作。

HW\_INT main()   
{   
 if (NULL == g\_stGateWayInfo.pcDeviceID)   
 {   
 //第一次绑定流程   
 }   
 else   
 {   
 IOTA\_ConfigSetStr (EN\_IOTA\_CFG\_DEVICEID, g\_stGateWayInfo.pcDeviceID);   
 IOTA\_ConfigSetStr (EN\_IOTA\_CFG\_IOCM\_ADDR, g\_stGateWayInfo.pcIOCMAddr);   
 IOTA\_ConfigSetStr (EN\_IOTA\_CFG\_APPID, g\_stGateWayInfo.pcAppID);   
 IOTA\_ConfigSetStr (EN\_IOTA\_CFG\_DEVICESECRET, g\_stGateWayInfo.pcSecret);   
 IOTA\_ConfigSetStr (EN\_IOTA\_CFG\_MQTT\_ADDR, g\_stGateWayInfo.pcIOCMAddr);   
 IOTA\_ConfigSetUint(EN\_IOTA\_CFG\_MQTT\_PORT, g\_stGateWayInfo.pcMqttPort);   
 IOTA\_ConfigSetUint(EN\_IOTA\_CFG\_IOCM\_PORT, g\_stGateWayInfo.pcIOCMPort);   
 HW\_LogSetLevel(g\_uiLogLevel);   
 IOTA\_Login();   
 }   
}

IOTA\_ConfigSetStr (EN\_IOTA\_CFG\_DEVICEID, g\_stGateWayInfo.pcDeviceID)到IOTA\_ConfigSetUint(EN\_IOTA\_CFG\_IOCM\_PORT, g\_stGateWayInfo.pcIOCMPort)之间是设备信息相关参数的配置，然后调用API接口IOTA\_Login()进行登录操作。

## 开发样例（Demo库）

参考Agent Lite Demo(C).zip中的Demo工程。

## 调测结果

网关、设备集成开发完毕后，使用华为提供的HiHo APP进行测试。

### 注册APP账号

下载华为 HiHo App并安装。

说明

目前下载地址暂未确定，请联系华为技术支撑人员获取HiHo App。

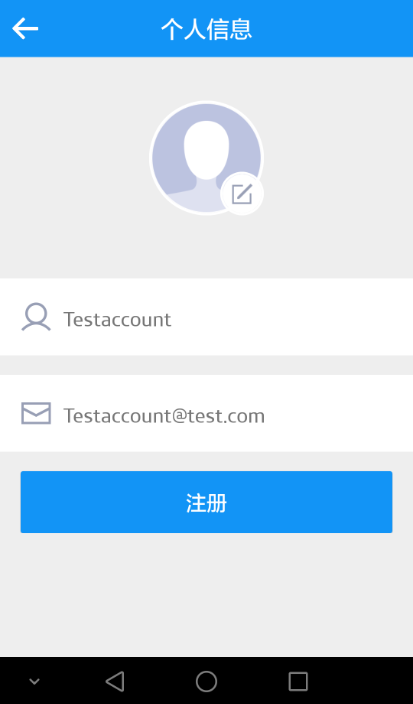
在欢迎页面选择“注册”。



在注册页面填写好相应信息后，单击“下一步”。



在个人信息页面填写好匿称和邮箱，单击“注册”。



----结束

### 网关上电

请开发者按照网关Agent Lite安装指南安装网关并上电，并测试网关的可用性。

### 调测

登录App账号



输入已注册的账号信息，并单击“登录”。



绑定网关。

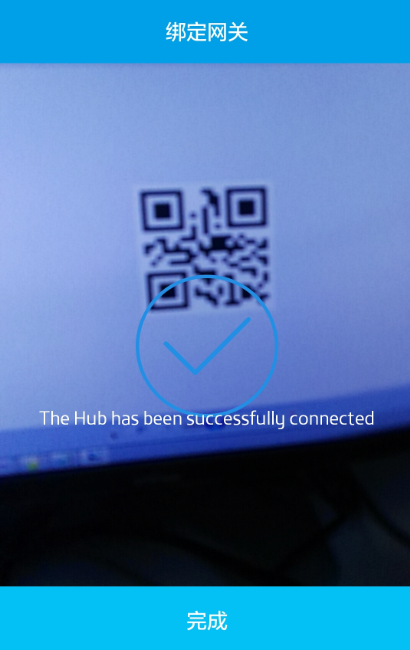
进入设备列表标签页，单击“添加设备”。



进入绑定网关页面后，按照App提示步骤绑定网关。



如果出现如下图片，证明网关已经绑定登录成功。



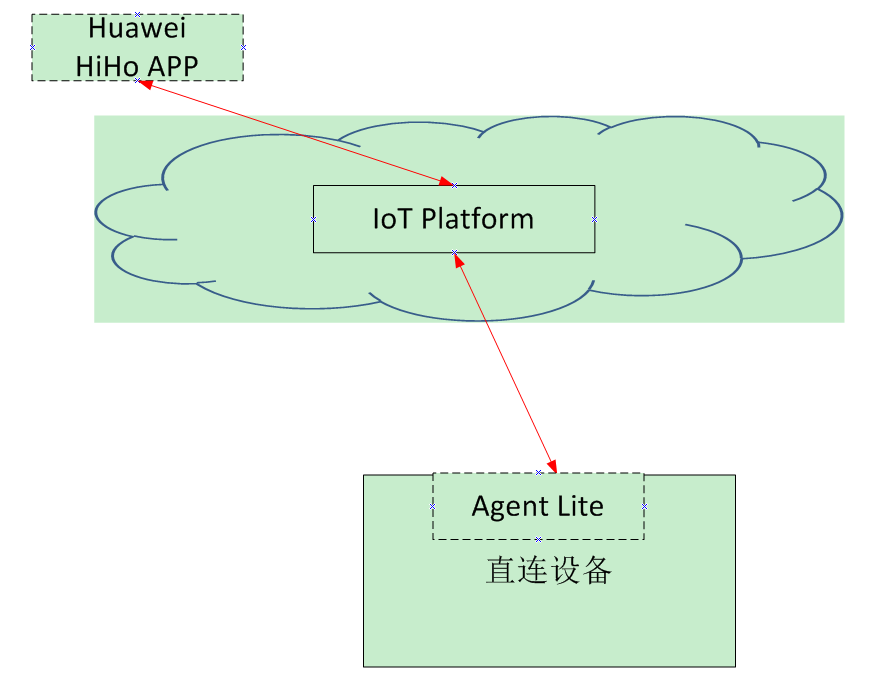
----结束

同时查看打印日志也可以分析出是否绑定、登录成功。出现如下红色部分，证明网关绑定平台成功。

{"verifyCode":\*\*\*\*\*\*\*\*89","deviceInfo":{"nodeId":"21\*\*\*\*789","manufacturerId":"Huawei","deviceType":"Gateway","model":"AR512","protocolType":"HuaweiM2M","mac":"21\*\*\*\*789"}}   
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   
05-19 14:36:10 HTTP INF: UHttpFsm(2)[1:CONNECTING] run SCK msg(1:TCP\_CONN) to [CONNECT] ok   
05-19 14:36:11 HTTP TRC: recv data length 509:   
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   
HTTP/1.1 200 OK   
X-Frame-Options: SAMEORIGIN   
Content-Type: application/json;charset=UTF-8   
Transfer-Encoding: chunked   
Date: Thu, 19 May 2016 06:36:49 GMT   
Server: OpenAS   
Set-Cookie: SESSION\_COOKIE\*; Expires=Thu, 01-Jan-1970 00:00:01 GMT; path=/   
   
fc   
{"deviceId":"1d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*4-fb034cdc0752","appId":"62bd72ae-8d9a-4404-8dd4-52c9e1f94422","secret":\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*b2a","addrHAServer":"183.\*\*.\*\*.67","addrLVSServer":"183.\*\*.\*\*.67",   
出现如下红色部分，证明网关登录平台成功。   
{"appId":"62bd72ae-8d9a-4404-8dd4-52c9e1f94422","deviceId":"1d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*4-fb034cdc0752","secret":\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*b2a","deviceIP":"188.\*\*\*.\*\*\*.60"}   
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   
05-19 14:36:11 HTTP INF: UHttpFsm(5)[1:CONNECTING] run SCK msg(1:TCP\_CONN) to [CONNECT] ok   
05-19 14:36:12 HTTP TRC: recv data length 626:   
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   
HTTP/1.1 200 OK   
MSGHDL: C62429B5DB3ECCC4FF2ED3931BA0569D   
X-Frame-Options: SAMEORIGIN   
Content-Type: application/json;charset=UTF-8   
Transfer-Encoding: chunked   
Date: Thu, 19 May 2016 06:36:50 GMT   
Server: OpenAS   
Set-Cookie: SESSION\_COOKIE\*; Expires=Thu, 01-Jan-1970 00:00:01 GMT; path=/   
   
146   
{"accessToken":"9dd1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*4-4f7\*\*\*\*197a1","tokenType":"bearer","timeout":"43199","refreshToken":"b613\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*a-\*\*\*\*487b4f9c","mqttClientId":"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*d-9fa7-da2ab60ec53d","addrHAServer":"   
UGP INF: ---Remov Device success ---.

# 设备业务开发

基本概念



开发步骤

开发者可直接将Demo在目标设备上运行起来，并基于Demo的已有代码做必要的修改，完成所需功能。

说明

本文基于Demo介绍设备绑定接口的使用，其他接口的介绍，可参考《华为IoT Agent Lite API参考(C).docx》。

1.将Demo在目标设备上运行起来

参考[将Demo在目标设备上运行起来](#_ZH-CN_TOPIC_0037842501)的方法，将设备的Demo在目标设备上运行起来。

说明

直连设备与网关使用的是同一个Demo。

直连设备可以认为是一个简化版的网关，开发流程与网关类似。

2.设备初始化

在发起业务前，需要先初始化Agent Lite相关资源。

设备启动，进行直连设备的初始化。调用API接口IOTA\_Init(HW\_CHAR \*pcWorkPath, HW\_CHAR \*pcLogPath)，初始化Agent Lite资源，具体API的参数使用参考Agent LiteAPI接口文档。

代码示例参考[设备初始化](#_ZH-CN_TOPIC_0037842502)。

3.设备绑定

设备第一次接入IoT联接管理平台时需要进行绑定操作，从而将设备关联到平台上。开发者通过传入设备序列号以及设备信息，将设备绑定到IoT联接管理平台。绑定操作需要通过IoT连接管理平台北向接口配合操作，通常有两种绑定方式：

* 提前在平台上预制设备的设备序列号，设备初始启动后发起绑定流程，将设备绑定到IoT联接管理平台中。
* 通过APP终端扫描设备序列号生成的二维码图片动态绑定设备，扫码后APP通过命令动态往平台注册设备的序列号，之后重启设备进行绑定。

开发描述和代码示例参考[网关绑定](#_ZH-CN_TOPIC_0037842503)。

调测结果

设备业务开发的调测结果与网关业务开发的调测结果类似。

开发样例（Demo库）

参考Agent Lite Demo(C).zip中的Demo工程。

# 参考

常见问题(FAQ)

1. 未进行二维码扫码

当开始进行设备绑定流程时，如果没有进行APP没有进行扫码，只单独的进行程序的运行，会产生设备还没有注册的错误信息，具体如下所示：

49   
{"error\_code":"100410","error\_desc":"The device has not registered yet."}

解决的办法是执行程序时用APP扫二维码进行绑定登录。

1. 验证码过期

验证码是有有效时间的，网关需要在有效时间内接入平台，当有效时间后设备还没有绑定到平台，会产生验证码过期的错误信息，具体如下所示：

42   
{"error\_code":"100411","error\_desc":"The verifyCode has expired."}

解决的办法是修改esn.ini文件中的验证码，然后用这个验证码重新生成二维码进行APP扫码绑定流程。