U+

U + 联 盟 行 业 标 准

U+智慧生活开放平台
uSDK5.0_Phone_ Android 开发手册

目录

修订记录

修改时间	修改内容	版本	修改人	备注
2018-07-16	创建新版	1.0	张振华	

1. uSDK 简介及 5.X 新增功能

uSDK 是 U+物联平台的移动应用开发套件，它能够支持目前主流的 Android 和 iOS 系统。它允许您运用 Java 或 Object-C 等创建 App 和智能硬件互动。

通过 uSDK 可以实现的基本功能包括智能设备配置入网、搜索、状态查询、控制、接收报警。

1.1. uSDK5.X 变化与新增功能

- uSDK5.X 与 4.X API 基本保持一致，精简了 2.X 历史 API
- 修改了设备属性和报警的方法名
- 提升了高性能路由 SmartLink 配置成功率

1.2. uSDK4.X 工程迁移到 uSDK5.X

修改获得设备属性和报警 API。

2. 实验开发环境搭建

2.1. 官网、QQ 群

官网： 海极网 http://www.haigeek.com，**注册账号成为硬、软件开发者，使用 U+物联开放平台的各项服务及下载资料。**

官方 QQ 群：U+开发者技术讨论群 **457841032。**

2.2. 搭建实验测试环境

准备设备和软件

1.一台 2.4G 无线路由器

2.一台 Android4.0 以上智能手机

3.一台支持 U+协议的智能设备（安全版本智能设备）

4.硬件测试工具 App 或 uSDK4XDemoApp

路由器需要连接互联网，它负责接入 U+设备，提供数据通路。Android 手机主要承载基于 uSDK 的 App 程序。支持 U+协议的智能设备可以是智能空调、洗衣机整机、空气盒子或者 U+Dev 开发板。

实验设备全家福

U+Dev 开发板可以申请进入官方开发群说明您的开发背景，向群主申请。

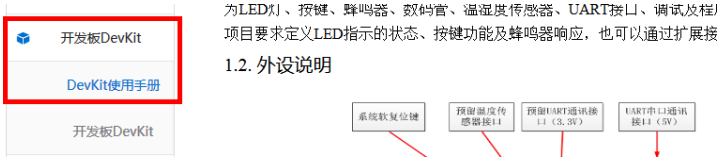
动手搭建环境

1. 将路由器加电并配置连接互联网，打开 2.4G 频段
2. 手机连接路由器 2.4G 频段
3. 智能设备接通电源
4. 操作设备进入待入网模式
5. 使用硬件测试工具 App 或 uSDK3XDemo App 配置智能设备入网
6. 使用硬件测试工具 App 或 uSDK3XDemo App 查看智能设备是否入网

资料下载

硬件测试工具 App

U+Dev 开发板使用说明



配置设备入网演示视频：QQ 群文件

2.3. 搭建程序开发环境

下载 uSDK、Demo、API 文档

导入库文件

配置 uSDK4.X aar 到 gradle 配置文件里。

配置工程包名

建立应用工程时工程包名需要和 U+开发者网站上申请应用填写的应用包名一致。例如：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.partner.uplus.app"
    android:versionCode="2"
    android:versionName="2.1">
```

配置 App 校验信息

将 U+开发者网站申请的 APP_ID,APP_KEY,APP_SECRET 配置到文件中，举例如下：

```
<application>
<meta-data
  android:name="APP_ID"
  android:value="XX-XXXX-XXXX" />
<meta-data
  android:name="SECRET_KEY"
  android:value="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" />
<meta-data
  android:name="APP_KEY"
  android:value="xxxxxxxxxxxxxxxxxxxxxxxxxxxx" />
</application>
```

注意

- APP_KEY 分为“测试”和“生产”。在开发调试阶段用测试的 APP_KEY，在“提交应用”和发布到商店时用生产的 APP_KEY。
- 无论是开发者还是用户，在第一次启动 APP 时手机需要联网。

避免 uSDK 包被混淆

请添加如下配置避免 uSDK 二次混淆：-keep class com.haier.uhome.** {*;}。

uSDK 权限配置

在使用 Android 版 uSDK 进行开发时，需要为 uSDK 配置相关权限：

```
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission
android:name="android.permission.CHANGE_NETWORK_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.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

以上权限 uSDK 仅用于和设备及接入网关通信。

开发测试环境配置及确认

我们在海极网申请开发 App 后，使用**开发者环境**开发测试，产品发布时转入生产环境，参考 4.1 章节配置 uSDK 特性，使 uSDK 工作在恰当的环境里。

U+智能设备（uPlug）默认连接生产环境，App 开发测试阶段，修改路由器 DNS，**智能设备配置到此路由，保证设备连接开发者环境。**

配置开发者环境举例

配置完成后 ping developer.haier.net，查看配置是否成功：

2.4. 范例 demo 程序

海极网提供一款示例 demo 程序，演示 uSDK 配置设备入网及局域网与设备互动的情景。

uSDK5.XDemo App 包含的功能

功能	文件
SmartLink 配置设备入网	MakeDevice2NetSmartLink.java

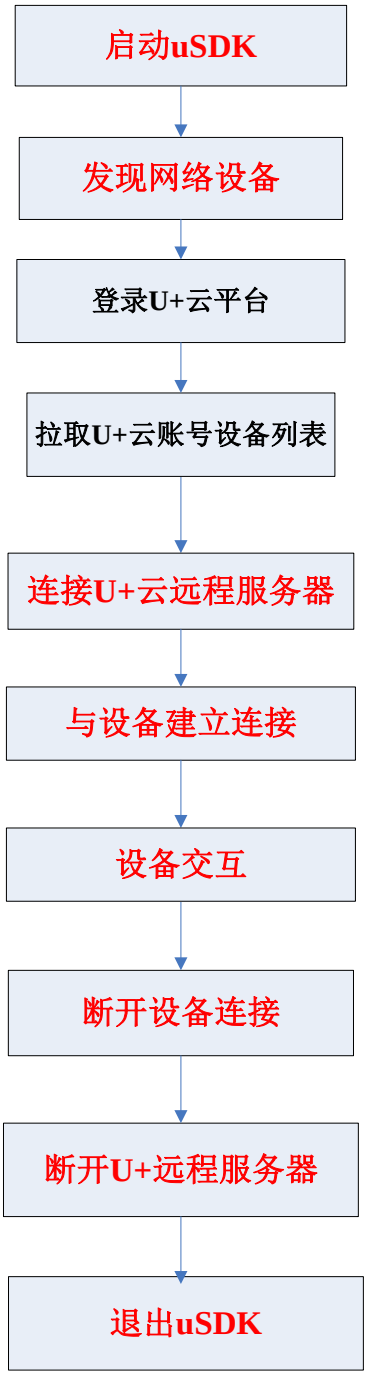
SoftAP 配置设备入网	MakeDevice2NetSoftAP.java
搜索设备功能，并展示设备列表	MainActivity.java
与设备建立连接，显示状态、报警	DeviceStatusActivity.java
设备控制	DeviceControlUDevKitDialog.java DeviceControlElecHeaterDialog.java
U+智能设备自发现(第三代 uPlug:瑞昱)	AppDemo.java
ElecHeater_Dev_DOC_ID_111c120024000810060500418001840000000000000000000000000000000000000000.pdf UDevKit_Dev_DOC_ID_110c1084105085081 4ee0000000000000000000000000000000000000000000000000.pdf	asset 文件夹 电热水器 ID 开发文档 U+开发板 ID 开发文档

3. 业务梳理及开发快速入门

搭建实验环境及 App 开发环境后，本章总结使用 uSDK 的两个常见业务流程：基本业务流程和配置设备入网流程。

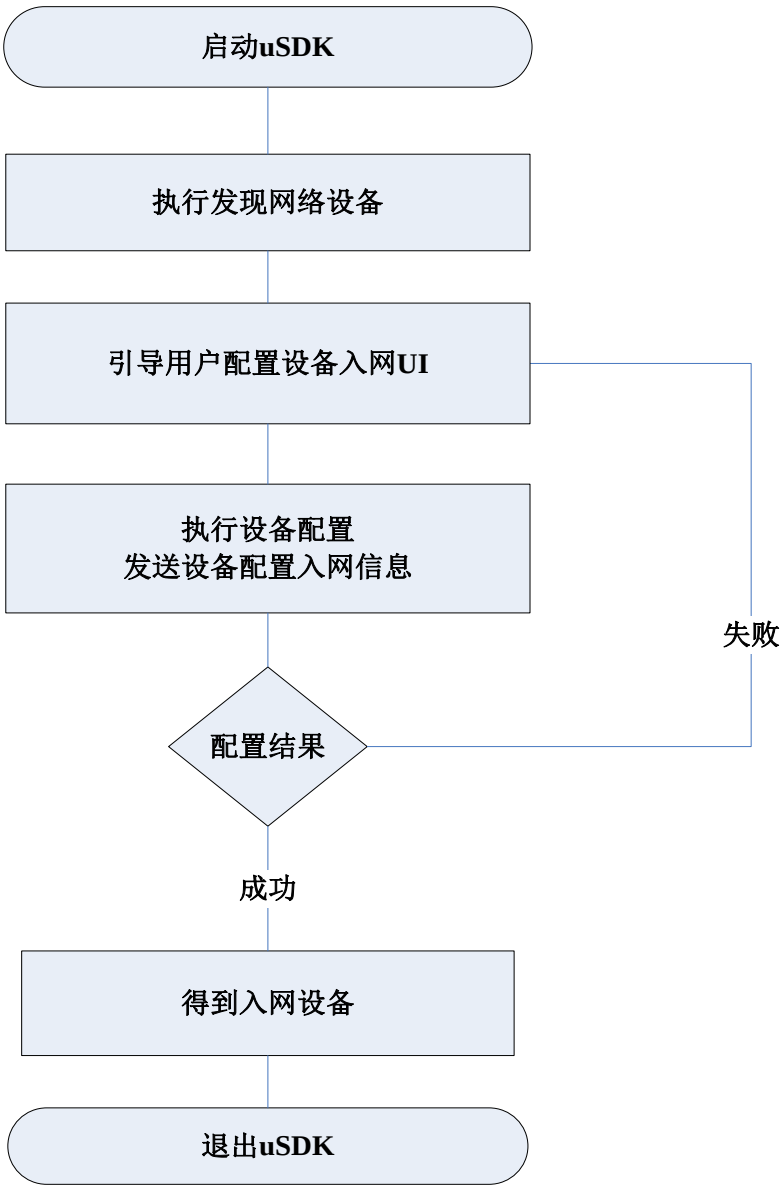
3.1. 基本业务流程

本图简要介绍 uSDK 物联 App 主要业务运行流程，红色文字：App 与 uSDK 交互，黑色文字：App 与 U+云交互：



3.2. 配置入网流程

参考下列流程开发配置设备入网功能，**特别说明：启动 uSDK 是配置入网功能的前提；只使用 uSDK 配置入网功能的 App，完成设备入网后，可执行退出 uSDK。详细介绍请参考章节 4.2 配置设备入网。**



4. uSDK 业务开发详解

4.1. 启动 uSDK

启动 uSDK 是使用 U+物联功能的前提。为 App 配置好海极网验证信息，启动 uSDK。如果果需要，紧接着设定 uSDK 日志级别，下面分别讲解。

预备知识

请先了解章节 2.3 熟悉 App 开发环境的配置。

相关概念和术语

- AppID：在海极网为物联 App 申请的 AppId，用于 App 校验

- AppKey：在海极网申请，用于与 OPEN API 交互时使用
- SecretKey：在海极网申请，uSDK 使用

➢ **启动 uSDK**

首先执行初始化方法，然后执行具体启动方法：

```
//推荐在 Application 对象实现中执行,
//或者在所有 uSDK 操作之前，入参需要是 ApplicationContext
uSDKManager.init(ApplicationContext);

//result 为 uSDKErrorConst.RET_USDK_OK 时启动成功。
uSDKManager.startSDK(new IuSDKCallback() {
        @Override
        public void onCallback(uSDKErrorConst result) {
                if(result == uSDKErrorConst.RET_USDK_OK) {
                    //在此设置 uSDK 特性，例如：关闭 httpDNS，或者设置为 NONE
                }
        }
});
```

➢ **设定 uSDK 日志级别**

uSDK 默认输出所有日志，日志标签是 **uSDK。**uSDK 提供 API 支持 App 设定日志输出级别，uSDK 启动成功后，执行即可：

uSDKManager.initLog(uSDKLogLevelConst level, boolean isWriteToFile, IuSDKCallback)。

注意

1．uSDK 成功启动一次即可，不要多次、多个线程启动

2．uSDK 成功启动之后再调用 uSDK 提供的各类方法

3．启动入参 Context 必须是 **ApplicationContext**

➢ **启用 uSDK 特定功能**

uSDK 启动前，设置特定功能：

1. uSDKManager. SDK_FEATURE_TRACE 开启链式跟踪，uSDKManager. SDK_FEATURE_SAFTE_DNS：开启 httpDNS 功能，uSDKManager. SDK_FEATURE_DEFAULT：开启所有额外功能，uSDKManager.SDK_FEATURE_NONE：关闭所有额外功能。
2. **集成 uSDK 的 App 在生产以外环境以外测试或使用时，在 uSDK 启动成功回调里添加以下语句：**

```
int fea = uSDKManager.SDK_FEATURE_NONE;
//根据需要开启需要特性，注意不开启 uSDKManager. SDK_FEATURE_SAFTE_DNS
fea |= uSDKManager.SDK_FEATURE_TRACE;
```

uSDKManger.enableFeatures(fea);

App 上线时，修改代码允许 httpDNS 特性，例如设置成：uSDKManager.SDK_FEATURE_DEFAULT 允许全特性，或者根据实际需要设置。

4.2. 配置设备入网

配置设备入网就是使用 uSDK 将智能设备加入指定 WIFI 的一项操作。uSDK 支持 SmartLink、SoftAP 两种方式入网。对于每种配置方式，本章将讲解：设备入网的操作过程、程序收集目标无线网络配置信息、普通或安全方式发送配置入网信息、获得配置入网设备实例、中断设备配置入网动作等，最后讲解自带屏幕或操作系统设备入网。

uSDK4.X 支持您使用加密方式发送配置信息，如果您的智能产品包含不支持加密入网的型号，则使用普通配置方式，以保证所有销售产品都能正常入网。如果产品全部支持加密入网，则使用加密入网方式，提升整套产品安全性能。

预备知识

请先了解章节 4.1 启动 uSDK。配置入网演示视频请至 U+开发者 QQ 群下载。

相关术语和概念

- uPlug：为智能设备连接 WIFI 的集成电路板。

SmartLink 入网方式

U+平台 SmartLink 方式是成熟、稳定、快速的设备入网方式。运行时将无线配置信息发送给待入网设备，完成配置入网动作。

➢ **了解设备入网操作过程**

1) 确认当前路由器频段是 2.4G。按说明书指导，操作设备进入配置模式（一般是长按某按键或组合按键），智能设备面板 WIFI 信号灯闪烁（此时 uPlug 指示灯规律快闪），说明设备进入待入网模式。
2) 确认手机稳定连接路由器，设备安装位置 WIFI 信号良好，打开基于 uSDK 的 App（我们要开发的产品），将配置入网信息发送出去。设备收到配置信息后连接路由器，手机在目标网络里等待设备出现。

3) App 显示有新设备出现。
4) App 端设备长时间（2 分钟）不上线，说明操作失败，重新按照 1，2，3 步骤进行。

- **App 收集配置入网信息**

配置入网信息指的就是 WIFI 的 SSID 和密码，App 编程获取 WIFI 名称并显示，密码由使用者输入。

- **App 发送配置入网信息**

App 把无线的名字和密码发给待入网的设备：

```
uSDKDeviceManager.configDeviceBySmartLink(apName, apPass,
null, 100, false, new IuSDKSmartLinkCallback() {

        public void onCallback(uSDKDevice deviceJustJoined , uSDKErrorConst result) {
            if(uSDKErrorConst.RET_USDK_OK == result) {
                System.out.println("Success: " + deviceJustJoined.getDevivceId());

            } else {
                System.out.println("配置设备入网失败了：  "
                    + result .getErrorId() + " :" + result .getValue());
        }
});
```

uSDKDeviceManager.configDeviceBySmartLink：配置设备入网方法。

apName：无线网络名称，不能为空，apPass：无线网络密码，不能为空。

1) apName（即 SSID）最大长度为 31 个字符，最小长度为 1 个字符，内容为可见英文字符
2) apPass（即路由密码）要求 8 至 63 个字符，内容为可见英文字符
3)两项均不支持中文

null：设备 Id（设备 MAC），null：不指定目标设备。

100：配置入网超时时间，单位是秒，推荐 100s。

IuSDKSmartLinkCallback：回调接口。

true：仅能支持配置安全设备入网，**false**：同时支持安全设备和普通设备时使用，请根据产品实际情况配置本参数。

注意

1. uSDKDeviceManager.configDeviceBySmartLink 方法有多个可以指定 MAC、超时时间、是否安全，开发功能时根据业务场景需要选择相应的实现。
2. 重载实现可以指定 TypeId，被配置成功且 TypeId 是指定值的设备将被返回。

3. 重载实现可以指定 uPlusId 数组（TypeId 数组）进行过滤，满足过滤条件且最先返回的设备将被认为是成功入网的设备返回给 App。
4. 重载实现有配合 uTrace 的方法
5. 当配置动作动作尚未完成，再次执行配置 API 时，将返回配置进行中的信息。
6. -21103｜-901 错误码：1.您可以提示使用者重新进行设备配置入网操作；2.如果此时您能得知设备的 MAC 值（例如：二维码或一维码），可以调用 API 查看设备是否存在（章节 4.12），如果能找到，也可以确认为配置入网成功。

- **判断设备入网是否成功**

result==uSDKErrorConst.RET_USDK_OK 时，设备配置入网成功。配置失败时，result 携带配置错误信息。

- **获得配置入网设备实例**

deviceJustJoined 不为 null 时，它就是入网设备对象实例，此时它携带设备网络及产品类型等基本静态信息。

- **中断设备配置入网**

在执行配置设备入网过程中可以调用 API 中断，中断动作结果将通过回调参数通知 App，中断结果返回后，才能启动下一次配置动作。

```
uSDKDeviceManager.stopSmartLinkConfig(new IuSDKCallback() {
    @Override
    public void onCallback(uSDKErrorConst result) {
        System.out.println("取消配置结果：" + result);
    }
);
```

- **发现待入网智能设备&极路由发现待入网设备**

安装有第三代 uPlug 的智能设备第一次接通电源或进入配置模式等待入网时，uSDK3.X-uSDK4.4.01 能够发现它们，我们可以提示使用者，或者引导用户把设备配置入网。

当您使用 uSDK4.2.02 及以上，并连接极路由时，可以接收极路由发现的待入网设备。从 uSDK4.4.02 开始在普通路由（这里指非极路由），不能发现待入网的三代 uPlug，此功能被移除。

普通路由发现待入网智能设备和极路由发现待入网设备共用一个接口。

第三代 uPlug

1.注册接口

```
uSDKDeviceManager.setDeviceScanListener(new IDeviceScanListener() {
    @Override
    public void onDeviceScanned(ConfigurableDevice candidater) {
        System.out.println("uSDK3.x Demo find:" + candidater.getDevIdSubfix());
    }

    @Override
    public void onDeviceRemoved(ConfigurableDevice candidater) {
        System.out.println(candidater.getDevIdSubfix() +
                " has been disappeared!");
    }

});
```

onDeviceScanned：设备发现时

onDeviceRemoved：设备被配置入网或其它原因不能被发现时

2.启动扫描

uSDKDeviceMgr.startScanConfigurableDevice，如果 wifi 没开则会直接报错。

3.停止扫描

uSDKDeviceMgr.stopScanConfigurableDevice，不需要此功能时务必停止扫描。

4.配置设备入网

一般普通路由使用 SmartLink 组合 mac 地址方式配置入网：MAC 结尾 4 个字符：candidater.getDevIdSuffix()，极路由使用免密配置入网，请阅读常见问题回答 5。

注意

1. 其它应用场景不使用带 mac 的配置方式入网
2. 安装三代 uplug 的智能设备同时支持不带 mac 参数的 uSDKDeviceManager.configDeviceBySmartLink 方法。

SoftAP 入网模式

SoftAP 是将智能设备变身为 WIFI 热点，手机连接热点，然后把无线配置信息发送给设备的一种入网方式。

SoftAP 操作或编程时全程务必关闭数据流量，如果用户未关闭 App 需要关闭和提示。

- **了解 SoftAP 设备入网操作过程**

1) 参考使用说明书让设备进入配置模式，此时智能设备工作于 WIFI 热点模式，打开手机 WIFI 列表，就能看见智能设备。
2) 打开通用测试 App（硬件测试工具 App） SoftAP 配置选项卡，显示 uplus-、haier、U-MAC 地址后 4 位组成的 WIFI 热点列表。
3) 点击一项，让手机连接到热点。
4) 连接成功后，显示手机附近的 WIFI 列表，使用者选一个，例如：A。
5) 输入 A 的密码，发送给设备。
6) 操作手机连接到 A，打开 App 等待该设备上线。
7) App 端设备长时间不上线（1 分钟），说明操作失败，重新按照步骤 1 至 6 进行。

- **App 实现 SoftAP 配置入网**

App 实现 SoftAP 配置入网需要完成的工作：1.把路由的 SSID 和密码发送给智能设备；2.在目标路由中发现在设备；

实现步骤

1．App 提示并关闭手机流量，SoftAP 流程必须关闭手机流量。

2．App 显示当前连接无线并提供无线选择功能，询问用户智能设备要加入哪个路由。默认为选择手机当前连接 WIFI。

3．App 收集无线密码，校验路由名称和密码是否符合要求。

4．App 编程或手动连接到设备 WIFI 热点 U-XX。

5．App 准备 SoftAP 配置信息

```
uSDKSoftApConfigInfo softApConfigInfo = new uSDKSoftApConfigInfo.Builder()
                    .routeInfo("ssid", "password")
                    .security(false)
                    .configTimeout(90)
                    .timerValidInBackground(false)
                    [.uplusIDList(typeIdList)]
                    .builder();
```

参数说明：

“ssid”和”password”是无线名称和密码

secuirty：false 非安全配置

configTimeout：SoftAP 配置全过程超时间 90s

uplusIDList：typeId 过滤，过滤 TypeId 时使用，选择性使用

timerValidInBackground：当 App 切换出当前界面时 uSDK 是否继续消耗超时时间

无线名称和密码校验要求：SSID 最大长度为 31 个字符，最小长度为 1 个字符，内容为可见英文字符，passwrod 设置时要求 8 至 63 个字符，内容为可见英文字符，两项均不支持中文。

6．App 进行 SoftAP 配置&检查配置结果

```
final String targetSSID = softApConfigInfo.getSsid();
uSDKDeviceMgr.configDeviceBySoftAp(softApConfigInfo, new IuSDKSoftApCallback() {
 @Override
 public void onSoftApConfigCallback(uSDKDevice targetDevice, uSDKErrorConst result) {
        if(result == uSDKErrorConst.RET_USDK_OK) {
            //App 在此处等待配置最终结果
            //targetDevice：配置入网设备
            Log.d("uSDK4.XDemo", targetDevice.getDeviceId() + “已成功入网”);
        }
 }
 @Override
 public void sendConfigInfoSuccess() {
```

```
        //App 编程或手动将 Android 连接至 targetSSID 等待设备
        Log.d("uSDK4.XDemo", "请立即将手机连接至<" + targetSSID + ">等待设备上线");
    }
});
```

softApConfigInfo：uSDKSoftApConfigInfo 对象

重载方法：uSDKDeviceManger.configDeviceBySoftAp(uSDKSoftApConfigInfo configInfo,TraceNode csNode,IuSDKSoftApCallback callback)

注意：上一次 SoftAP API 执行完成后，才能二次调用。

- **自带屏幕或操作系统设备入网**

对于一些高级、大型家电或智能设备，其自身可能携带操作系统并且带有显示装置，此时我们应将其视为普通计算机，由其自行引导使用者连接网络。

4.3. 发现及管理网络设备

经过上面的步骤，智能设备已经连上 WIFI，App 实现设备管理接口就能实现设备管理业务。本章分别讲解：接收 uSDK 管理设备集变化、设定过滤设备类型、获得可用设备全集。

预备知识

请先了解章节 2“实验开发环境搭建”，章节 3“快速入门”。

相关概念和术语

- 小循环：uSDK 和 U+设备处于同一无线局域网完成设备交互的情形。
- TYPEID :TYPEID 就是设备类型的标识字符串 ,用它可以识别 U+平台上的各种硬件设备。
- 设备连接状态
  1) 未连接：App 还没连上智能设备，表示设备加入 WIFI 已发现。调用 uSDKDevice.getStatus 是 uSDKDeviceStatusConst.STATUS_UNCONNECT。
  2) 离线：设备发现过并且 App 进行了连接，但此时无法收到设备的响应数据。调用 uSDKDevice.getStatus 是 uSDKDeviceStatusConst.STATUS_OFFLINE。
- 设备的属性状态（属性）：
  开机、关机、运行等值。

- **接收 uSDK 管理设备集变化回调**

App 依靠 uSDK 管理智能设备。uSDK 将网络里所有的智能设备都管理起来，形成设备池，设备池里设备的数量会发生变化，例如：配置一台新设备入网，配置过的设备上电加入网络等。uSDK 将以上情况都汇报给 App，App 编程实现设备管理。

App 实现并注册如下接口方法，发现及管理网络设备：

```
uSDKDeviceManager.setDeviceManagerListener(new IuSDKDeviceManagerListener() {

    @Override
    public void onDevicesRemove(List<uSDKDevice> devicesChanged) {
        refreshSightedDeviceList(devicesChanged);
    }

    @Override
    public void onDevicesAdd(List<uSDKDevice> devicesChanged) {
        refreshSightedDeviceList(devicesChanged);
    }

    ……
});
```

devicesChanged：变化的设备集合

onDevicesAdd：在网络里发现新设备

onDevicesRemove：曾经发现的设备现在扫描不到了，它们不可用了

注意

本接口是**即时接口**，在 uSDK 启动前设置。

如果延时设置或再次设定本接口，需要结合查询设备所有列表使用，确保智能设备列表数据齐全。

- **设定设备过滤类型**

App 只关心设备池里的某类设备（例如：酒柜设备）：

1. App 执行如下方法：

```
uSDKDeviceManager. setInterestType(uSDKDeviceTypeConst.WINE_CABINET)
```

2. App 实现管理设备集变化回调：

此时 uSDK 只向 App 上报 uSDKDeviceTypeConst.WINE_CABINET 类型的设备，实现设备过滤。

注意

再次执行 setInterestType 方法，将覆盖已设定的设备过滤类型。

➢ **获得设备池全集**

uSDKDeviceManager.getDeviceList()，本方法也支持设备类型过滤。

4.4. 与设备建立或断开连接

现在我们已经发现设备，调用 API 就可以连接了。本章分别讲解：连接设备、断开设备连接。

预备知识

“启动 uSDK”请参考章节 4.1，“发现及管理网络设备”请参考章节 4.3。

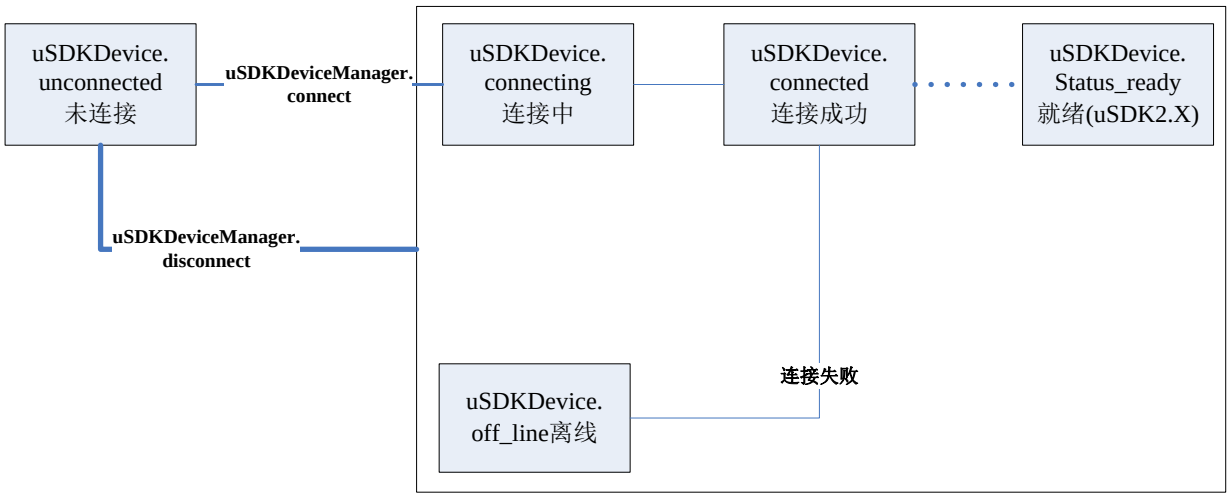
相关概念或术语

- 设备的连接状态：

  连接中(connecting)：App 正在与设备建立连接。

  已连接(connected)：App 已连接智能设备，可以发送控制指令，查询属性状态等，是我们可以交互的设备。

  离线(off_line)：App 不断开与智能设备连接的情况下，智能设备发生异常。



未连接、连接中、已连接、离线都是设备（uSDKDevice）的连接状态。App 连接或断开智能设备将触发连接状态的变化：

1. 智能设备正常入网后是**未连接**（被我们发现）
2. 连接智能设备后，智能设备变为**连接中**
3. 与智能设备连接成功后，变为**已连接**
4. 如果连接失败或通信中断会变为**离线**
5. 断开与智能设备的连接，则变为**未连接**

- **设备连接状态变化接口**

App 先注册监听，然后执行连接方法。智能设备的连接状态会通过回调上报：

```
uSDKDevice.setDeviceListener(new IuSDKDeviceListener() {
        @Override
        public void onDeviceOnlineStatusChange(uSDKDevice device,
                        uSDKDeviceStatusConst status, int statusCode) {
            }
        }
        ……
});
```

status：设备连接结果。

statusCode：设备连接状态变化码，**设备连接状态异常时提示用户或帮助分析问题。**

statusCode 信息表

statusCode 值	含义
50002	App 连接 U+云失败
50003	智能设备连接U+云失败(云端通知设备离线）
50007	心跳超时
50008	App 连接设备失败（本地）
50009	App 读取智能设备数据失败（本地）
50010	App 向智能设备写数据失败（本地）

- **连接设备**

uSDKDevice.connect。

- **断开连接**

不需要某台设备属性数据时，断开设备连接，释放设备资源：

uSDKDevice.disconnect。

- **主动查询连接状态**

uSDKDevice.getStatus()。

注意

1. 断开设备连接，uSDK 将不再上报此设备的任何属性消息给 App。
2. 当用户解绑定设备时，一定执行断开设备连接。
3. Android 设备熄屏或进入后台强制 uSDK 与设备保持连接请执行：

uSDKDeviceManager.setKeepLocalOnline(true)，uSDK 启动成功后执行本方法。

4.5. 获得设备属性状态

现在已经连接成功了，我们就能实时获得设备的最新属性值集合与设备交互了。本章主要讲解：获得设备的连接状态和实时属性值。

预备知识

与设备建立连接请参考章节 4.4

相关术语

- 六位码：六位码是一个键值对，在程序中是 uSDKDeviceAttribute 对象，是 App 和设备沟通交流的语言。举例：20w001 代表开关机功能，App 通过 uSDK 发送 uSDKDeviceAttribute｛20w001:20w001｝就是开机，发送 uSDKDeviceAttribute｛20w001:20w002｝关机，查询设备属性时 uSDKDeviceAttribute{ 20w001 :20w001 } 说明电视开机。
- ID 文档：六位码的集合文档，主要用途是明确设备操作指令及属性集等。

➢ **获得设备的属性状态**

1.App 实现并注册如下接口，准备接收设备属性的实时值：

```
uSDKDevice.setDeviceListener(new uSDKDeviceListener() {
        ……
        @Override
        public void onDeviceAttributeChange(uSDKDevice device,
                        List<uSDKDeviceAttribute> propertiesChanged) {
        }
        ……
});
```

propertiesChanged：变化的属性集合。

2.App 连接智能设备

3.App 根据 ID 文档发送查询指令（发送单命令请参考 4.7 章节）。指令执行成功后，就可以在接口里接收属性数据了。

查询指令举例如下：

60w0ZZ	—	—	查询设备的所有属性和属性值	—

```
uSDKDevice.writeAttribute("60w0ZZ", null, new IuSDKCallback() {
```

```
    @Override
    public void onCallback(uSDKErrorConst result) {
        if(result == uSDKErrorConst.RET_USDK_OK) {
            System.out.println("command succeed！ ");
        }
    }
});
```

- **App 主动获得当前设备所有属性值**

当我们第一次通过接口获得设备的属性状态后，只要设备**已连接或就绪**，调用 uSDKDevcie.getDevcieAttrMap 返回设备最新属性值合集。

4.6. 处理设备故障和报警

我们连接设备后，智能设备或家电如果存在故障或警告，uSDK 会即时将消息推送给 App。设备报警信息用 uSDKDeviceAlarm 承载。报警信息六位码具体含义参考相关 ID 文档。本章讲解：获得报警消息、发送停止报警指令、获得报警解除消息、主动查询设备报警等内容。

预备知识

与设备建立连接请参考章节 4.4。
单命令及如何向设备发送指令请参考章节 4.7。

- **获得报警消息**

实现 uSDKDeviceListener 获得报警消息，报警消息和设备属性状态变化是同一接口。

```
uSDKDevice .setDeviceListener(new uSDKDeviceListener() {

    ……

    @Override
    public void onDeviceAlarm(uSDKDevice myDevice, List<uSDKDeviceAlarm> alarmList) {

        StringBuffer alarmListStr = new StringBuffer();
        for(uSDKDeviceAlarm    deviceAlarm : alarmList) {
            alarmListStr.append(myDevice.getIp()
                    + "报警信息：" + deviceAlarm.getAlarmMessage() );
        }
    }

    ……
```

```
});
```

- **发送停止报警指令**

智能设备或家电可能向 uSDK 规律性快速报警，App 处理完成后，可以向设备发送停止报警指令，用于表示使用者已经了解设备发生故障。停止报警是一条普通单命令，参考 ID 文档编写。

- **获得报警解除消息**

发生故障的设备修理正常后会发送报警解除消息，报警解除就是没有报警，就是正常。报警解除消息与普通报警消息都在 onDeviceAlarm()接口方法产生，App 注意分辨。报警解除的具体值，请参考设备 ID 文档或设备模型文档。

- **主动查询设备报警**

App 可以调用 API 主动查询设备报警信息

```
ArrayList<uSDKDeviceAlarm> mAlarms = mDevice.getAlarmList();
if (mAlarms != null) {
    for (uSDKDeviceAlarm alarm : mAlarms) {
        alarm.getAlarmMessage();
    }
}
```

mDevice：uSDKDevice 对象，App 查询 mDevice 是否有报警信息。

mAlarms：设备当前的报警信息。

4.7. 执行设备控制

我们和设备已经连接了，可以向设备发送控制指令。uSDKDevice 提供 API 用于执行设备控制。执行控制时，可以给设备发送单命令、组命令，设定超时时间，命令执行后会反馈结果，下面将分别讲解。

预备知识

与设备建立连接请参考章节 4.4
获得设备属性状态上报请参考章节 4.5

相关术语和概念

- TYPEID：TYPEID 就是设备类型的标识字符串，可以使用 TYPEID 区分设备类型
- ID 文档：智能设备命令属性集合描述文档
- 标准模型设备文档：智能设备命令属性交互描述文档，海极网创建智能设备时生成

- 单命令：App 只发送一个指令给设备，**例如 ID 文档片段如下**：

204003	0	null 204003	启动	没有处于启动的状态 处于启动的状态	单命令
204004	0	null 204004	暂停	没有处于暂停的状态 处于暂停的状态	单命令

发送｛204003:204003｝代表启动指令，发送｛204004:204004｝代表暂停指令。

设备状态返回时｛204003:｝+｛204004:204004｝，代表当前智能设备是暂停。

- 组命令：App 发送指令集合给设备，**例如 ID 文档版本如下：**

204001	0	null 204001	开机	没有处于开机的状态 处于开机的状态	单命令／组命令
204002	0	null 204002	关机	没有处于关机的状态 处于关机的状态	单命令／组命令

单命令/组命令说明当前指令 204001 可以作为单命令发送 ,同时也可以运用到组命令中。

➢ **与六位码智能设备交互**

发送单命令

```
selecteduSDKDevice.writeAttribute("204003", "214001", new IuSDKCallback() {

    @Override
    public void onCallback(uSDKErrorConst result) {
        if(result == uSDKErrorConst.RET_USDK_OK) {
            System.out.println("Command succeed!");
        }

    }
});
```

selecteduSDKDevice：uSDKDevice 对象实例，代表智能设备。

selecteduSDKDevice.writeAttribute：命令发送方法。

IuSDKCallback：控制结果反馈回调。

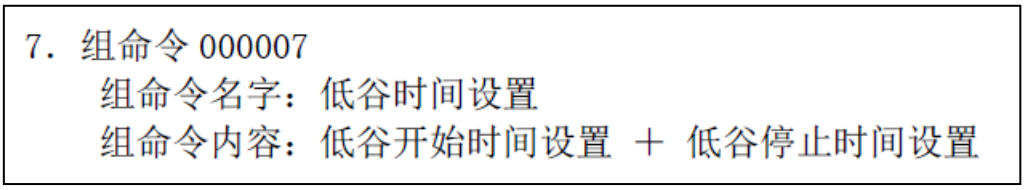
214001：214001：ID 文档描述的控制指令。

注意

App 需要严格遵守 ID 文档规定，命令格式中要求的指令 key、value 不能随意填空，值不能超过文档规定的范围。

发送组命令

16 进制组命令字处理



想执行低谷时间设置功能，就需要执行图片显示的组命令，其中 000007 是组命令字。发送时组命令字使用”**000007”，长度为 6 个字符（不足 6 位的从左补 0），另外组命令字要求全部大写。**

```
List<uSDKArgument> mAttrs = new ArrayList<uSDKArgument>();
mAttrs.add(new uSDKArgument("20600E", "01:02"));
mAttrs.add(new uSDKArgument("20600F", "02:03"));

selecteduSDKDevice.execOperation("000007", mAttrs, 10, new IuSDKCallback() {

    @Override
    public void onCallback(uSDKErrorConst result) {
        if (result != uSDKErrorConst.RET_USDK_OK) {
            System.err.println(result);
        }

    }
});
```

mAttrs：组命令集合

selectedDevice：uSDKDevice 对象实例，发指令给 selectedDevice

selectedDevice.execOperation：命令发送方法

“000007”：组命令标识字

10：超时时间，单位是秒

IuSDKCallback：命令执行结果回调

10 进制组命令字处理

组命令标识字为整形时，先转为 16 进制，再按字符串下发，19807=0x4d5f，不足 6 个字符补 0，得到”004d5f”，转为大写”004D5F”，组命令下发时填写”004D5F”作为组命令字。

```
List<uSDKArgument> mAttrs = new ArrayList<uSDKArgument>();
```

```
mAttrs.add(new uSDKArgument("202003", "202003"));
……

selecteduSDKDevice.execOperation("004D5F", mAttrs, 10, new IuSDKCallback() {

    @Override
    public void onCallback(uSDKErrorConst result) {
        if (result != uSDKErrorConst.RET_USDK_OK) {
            System.err.println(result);
        }

    }
});
```

注意

1．组命令说明中需要的指令放入 mAttrs，mAttrs 没有顺序要求。

2．组命令标识字具体值参考设备的 ID 文档。

3．App 需要严格遵守 ID 文档组命令的具体规定，不能随意添加或者减少指令，命令格式中要求的指令 key 和 value 不能随意填空。

- **与海极网创建智能设备交互（标准模型设备文档）**

海极网支持创建智能设备，创建完成后会产生《XX 设备应用开发文档》，以下将说明如何使用此文档与设备进行交互。

属性

*属性*的含义是此项可以作为智能设备的属性，*可写*列为 T 时，此项可以作为命令发往设备：

名称	数据类型	取值范围		单位	可读	可写	显示名称
		可取值	取值描述				
A	bool	False	关	/	T	T	A功能说明
		True	开				
B	bool	False	禁止发卡	/	T	T	B功能说明
		True	允许发卡				

```
selecteduSDKDevice.writeAttribute("A", "True", new IuSDKCallback() {

    @Override
    public void onCallback(uSDKErrorConst result) {
        if(result == uSDKErrorConst.RET_USDK_OK) {
```

```
                System.out.println("Command succeed!");
            }

        }
});
```

selecteduSDKDevice：uSDKDevice 对象实例，代表智能设备。

selecteduSDKDevice.writeAttribute：命令发送方法。

IuSDKCallback：控制结果反馈回调。

A：True：发送 A 功能开。

注意

发送指令时 Key 和 Value 大小写敏感（例如：" True"）。

可写列为 F 时，此行不能作为指令发送。

操作

操作的含义是 App 端可以发送表格描述的*操作*实现对应的功能，这里包含您在海极网创建的高级命令。

示例一：三种常用指令

getAllProperty	/	/	/	键值对数组	查询所有属性
getAllAlarm	/	/	/	字符串数组	查询所有告警
stopCurrentAlarm	/	/	/	无	停止当前告警

```
List mAttrs = new ArrayList();
selecteduSDKDevice.execOperation("getAllProperty", mAttrs, 10, new IuSDKCallback() {
    @Override
    public void onCallback(uSDKErrorConst result) {
        if (result != uSDKErrorConst.RET_USDK_OK) {
                System.err.println(result);
        }
    }
}
```

示例二：自定义的高级指令，tempPwdLengthX 及其它几项是您设备的属性之一。

Tmpopen	int	tempPwdLengthX	/	无	
	int	tempPwdPart1X	/		
	int	tempPwdPart2X	/		
	int	tempPwdPart3X	/		
	int	tempPwdPart4X	/		

```
List mAttrs = new ArrayList();
```

```
mAttrs.add(new uSDKArgument(“tempPwdLengthX”, “10”));
mAttrs.add(new uSDKArgument(“tempPwdPart1X”, “11”));
mAttrs.add(new uSDKArgument(“tempPwdPart2X”, “12”));
……
selecteduSDKDevice.execOperation("Tmpopen", mAttrs, 10, new IuSDKCallback() {

    @Override
    public void onCallback(uSDKErrorConst result) {
        if (result != uSDKErrorConst.RET_USDK_OK) {
                System.err.println(result);
        }
    }
});
```

selecteduSDKDevice：uSDKDevice 对象实例，代表智能设备。

selecteduSDKDevice.execOperation：命令发送方法。

IuSDKCallback：控制结果反馈回调。

10：控制命令超时时间。

注意

App 需要严格遵守应用开发文档的规定，命令格式中要求的指令 key、value 不能随意填空，值不能超过文档规定的范围。

- **命令超时设定**

uSDK 提供的默认控制方法超时时间为 15 秒，网络及设备良好的情况下瞬间返回。建议超时时间范围：T（单位：秒）5< T <120，App 根据自身业务设定超时间。

- **命令执行结果与设备反馈**

命令执行结果在 IuSDKCallback 回调中给出。uSDKErrorConst.RET_USDK_OK 代表发送成功，uSDKErrorConst.RET_USDK_TIMEOUT_ERR 代表超时。

设备执行成功有可能改变设备的属性值，App 接收设备的最新属性上报即可。

4.8. 远程控制及 U+云平台推送消息

经过前面的开发，我们已经可以在本地和智能设备完美交互了，但我们的手机不能切换路由或者使用 4G，这么做和智能设备的数据通路会立刻切断。现在我们让 App 同时连接远程服务器，手机更换 WIFI 或使用 4G 则直接通过服务器连接设备。

实现远程控制，本章将讲解以下内容：

1．一般的 U+物联 App 如何运行

2．连接用户接入网关

3．连接用户接入关后新绑定设备添加远程控制能力

4．设备解绑时解除设备远程能力

5．账号注销时如何处理

6．如何测试远程功能是否正常

实现设备远程控制预备知识

请参考 4.7 以前所有章节实现小循环与设备交互。

请参考章节 3.1“基本业务流程”或章节 5，确定编程连接用户接入网关时机。

相关概念及术语

- 用户接入网关：U+云支持 App 实现远程功能的服务器软件系统。App 编程使用如下服务器地址和端口：

服务器描述	地址	端口
开发者环境（中国）	usermg.uopendev.haier.net	56821
生产环境（中国）	gw.haier.net	56811
中国以外服务器环境	请自行确定	

- U+云 OPEN API：U+云为开发人员提供的网络 API，主要和用户账号等业务系统交互（OPEN API 开发文档请从海极网下载）。
- accessToken(session)：App 用户账号登录后 OPEN API 分配。
- 绑定、设备绑定：由 App 发起将用户及所属设备数据上送到云并创建设备数据档案的过程，**设备能够被远程控制的前提是已经被用户绑定**。
- 切网：用户由设备所在 A 路由切换到 B 路由，或改用数据网络的行为。
- 小循环 VS 大循环：它们是两种情景的描述。小循环指的是 App 与智能设备在同一无线局域网。大循环是说 App 需要借助 U+云用户接入网关才能和设备进行交互的情景，此时 App 与设备通常不在同一网络。
- 由于加密需要，设备运行设备时间必须准确，当前时间需要超过 2015 年 3 月 1 日

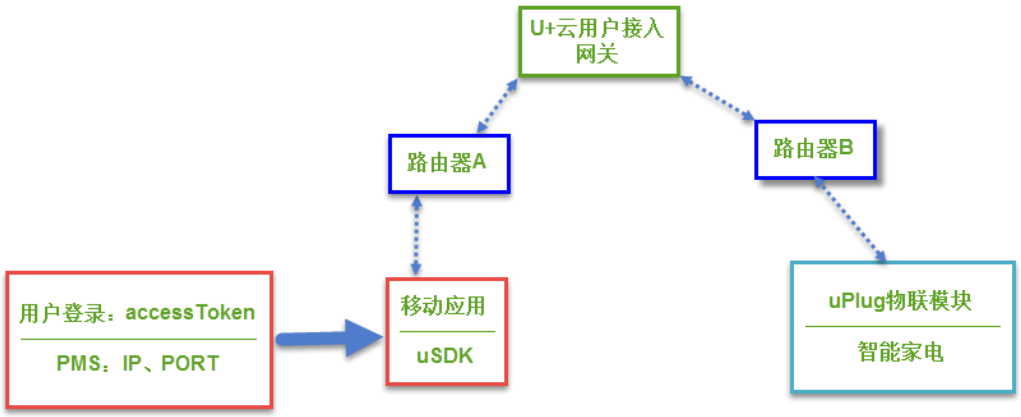


图 大循环

➢ 一般的 U+物联 App 如何运行

章节 5 描述了一般 U+物联 App 的运行过程，本章讲解连接用户接入网关这个步骤。App 执行 uSDKDeviceManager.connectToGateway 连接用户接入网关，此方法需要几个参数：session 是 U+云账号登录后的 accessToken，域名和 port 就是用户接入网关相关的域名和 port；设备信息集合就是用户拥有哪些设备，我们需要做的就是把这些参数凑齐然，参考章节 5 的图示，适时运行方法连接远程服务器。

➢ **连接用户接入网关**

App 在日常使用情景中，连接用户接入网关时，用户可能已经拥有一台智能设备，举例：

1.使用 OPEN API 拉取用户设备列表 json（一台燃气热水器），我们将要使用其中三个红色加粗字段：

```
/*
{"id":"0007A88A527B","status":{"online":true},"location":{"cityCode":null,"l
ongitude":"0.0","latitude":"0.0"},"attrs":{"brand":null,"model":null},"name"
:"燃气热水器
_527B","mac":"0007A88A527B","type":{"type":"18","typeIdentifier":"111c120024
000810180400418002480000000000000000000000000000000000","specialCode":"00418
00248","subType":"04"},"version":{"eProtocolVer":"2.15","smartlink":{"smartL
inkSoftwareVersion":"e_0.1.36","smartLinkHardwareVersion":"G_1.0.00","smartL
inkPlatform":"UDISCOVERY_UWT","smartLinkDevfileVersion":"0.0.0"}}}
*/
```

2.执行连接用户接入网关

使用步骤 1 的五个标红字段，为用户账号下的每台设备都生成一个 uSDKDeviceInfo，每个字段都不能为空和随意填写，**如果用户账号无设备使用 size 为 0 的设备集合**。

```
uSDKDeviceInfo remoteCtrledDeviceInfo = new uSDKDeviceInfo(
        "0007A88A527B",
        "111c120024000810180400418002480000000000000000000000000000000000",
```

```
        true);
```

"0007A88A527B"：deviceId（设备 MAC）
"111c12002400081018040041800248000000000000000000000000000000000000"：设备 uplusId
true：”online”时选 true，其它选 false

```
List<uSDKDeviceInfo> remotedCtrledDeviceCollection =
                    new ArrayList<uSDKDeviceInfo>();
remotedCtrledDeviceCollection.add(remoteCtrledDeviceInfo);
/*
 * 连接用户接入网关
 *  App 开发测试时设置域名和端口连接开发者环境，发布时设置生产环境
 *  uSDKDeviceManager.connectToGateway("token", “usermg.uopendev.haier.net”, 56821
 *                  , remotedCtrledDeviceCollection, new IuSDKCallback() {
 *   开发者环境：“usermg.uopendev.haier.net”, 56821
 *   生产环境（App 发布）：“gw.haier.net”, 56811
 */
uSDKDeviceManager.connectToGateway("abcdefghijklmnopqrstuvwxyz123456",
                    “gw.haier.net”, 56811,remotedCtrledDeviceCollection, new IuSDKCallback() {
        @Override
        public void onCallback(uSDKErrorConst result) {
            if(uSDKErrorConst.RET_USDK_OK == result) {

            } else {
                //方法执行错误原因
                //uSDKErrorConst.ERR_USDK_INVALID_PARAM 错误值需要检查某台设备的 deviceId、
                //uPlusId 是否异常
                System.err.println(result.getValue());
            }
        }
});
```

abcdefghijklmnopqrstuvwxyz123456：accessToken/session。
remotedCtrledDeviceCollection：允许远程控制设备信息集合，账号无设备时使用空集合。

- **远程服务器的连接状态**

uSDKDeviceManager 提供一个回调，可以了解我们当前与远程服务器的连接状态：

```
uSDKDeviceManager.setDeviceManagerListener(new IuSDKDeviceManagerListener() {

        ……
        @Override
        public void onCloudConnectionStateChange(uSDKCloudConnectionState status) {
            //已连接：uSDKCloudConnectionState.CLOUD_CONNECTION_STATE_CONNECTED
        }
```

```
});
```

- **处理绑定一台新设备**

绑定一台新设备后，按“连接用户接入网关”小章节，**将新绑定的设备加入集合，再次连接用户接入网关**，确保新设备远程可用，并按需连接设备以实现控制或查询状态。

- **用户解绑定时解除设备远程能力**

用户解绑定设备成功时，已经失去对设备的控制权，**断开设备连接**，释放设备对象实例，按“连接用户接入网关”小章节**再次连接用户接入网关**。

注意

如果不是用户主动解绑设备，但收到接入网关推送的设备解绑定消息（此时设备可能被他人解绑定），App 需要编写逻辑处理此消息：

```
uSDKDeviceManager.setDeviceManagerListener(
                        new IuSDKDeviceManagerListener() {
    ……

    @Override
    public void onDeviceUnBind(String mac) {
    //建议提示用户设备被解绑
    }
});
```

- **账号注销时的处理**

当用户账号注销时，我们需要调用 uSDKDeviceManager.disconnectToGateway 断开远程链接，清空账号数据。

- **如何测试远程功能是否正常**

手机在配置设备到 A 路由，绑定设备成功。手机或平板电脑切换连接 B 路由或直接使用 2G、3G、4G 数据网络，查询设备状态、进行控制。

注意

1．当用户切网时，uSDK 自动尝试连接用户接入网关，查询设备的最新状态。

2．App 使用 OPEN API 在开发者环境绑定设备，智能设备也要连接开发者境。

3．App 使用 OPEN API 过程中，U+云平台要求 App 有自己的 ClientId。ClientId 和终端的身份验证是相关的，错误或固定的 ClientId 会产生异常行为，**App 可以使用 uSDKManager.getClientId()方法产生 ClientId**。

- **接收 U+云推送消息**

App 获得消息推送需要**先连接用户接入网关**。在 U+云消息推送功能中，uSDK 仅扮演 Client 角色，透传用户接入网关发送来的消息。

实现如下方法接收云的推送消息：

```
uSDKManager.setManagerListener(new IuSDKManagerListener() {
    ……
    @Override
    public void onBusinessMessage(String content) {

    }}
);
```

setManagerListener：uSDK 事件回调

onBusinessMessage：用户接入网关消息推送回调方法

content：消息内容

- **接收 U+云 Session 异常消息**

当 App 得到 Session 异常消息时，可以提示用户登录信息已失效，提示用户重新登录，检查程序逻辑，查看用户登录地址与连接用户接入网关地址是否同一服务器环境。

```
uSDKManager.setManagerListener(new IuSDKManagerListener() {
    //session：失效 session
    @Override
    public void onSessionException(String session) {
    }
});
```

session:异常 session 串

4.9.和获取设备 WIFI 信号强度

uSDK4.2.02 及以上开发者按照章节 4.8 开发完成后，就可以使用 uSDKDevice 实例获取设备的 Wifi 信号强度。

```
uSDKDevice.getDeviceNetQuality(new IuSDKGetDeviceNetQualityCallback() {
    @Override
    public void onDeviceNetQualityGet(
            uSDKErrorConst result, uSDKDeviceNetQuality level, int detailValue) {
            if(result == uSDKErrorConst.RET_USDK_OK) {
                //uSDKDeviceNetQuality.GOOD  信号强度良好

            }
    }
```

```
});
```

IuSDKGetDeviceNetQualityCallback：信号强度方法回调

level：信号强度等级

detailValue：具体数值

4.10. 和带子机设备交互

带子机设备通常有商用空调、星盒等设备，在程序中我们可以和主机和子机进行交互，该类型设备有主机和子机，子机从属于主机，一台主机可以有多台子机。主机拥有 mac，子机没有。

➢ 获得子机实例

当主机已连接后，通过向主机发送查询指令（根据设备开发文档），由设备上报子机的动态，在回调中就可以得到子机实例，子机的数量会动态变化。

```
parentuSDKDevice.setDeviceListener(new IuSDKDeviceListener() {
    ……
    @Override
    public void onSubDeviceListChange(
            uSDKDevice currentDevice, ArrayList<uSDKDevice> childArrayList) {
        ……
    }
});
```

parentuSDKDevice：主机对象

childArrayList：附属子机实例集合

➢ 子机的连接状态及控制

子机只有两种连接状态：离线、已连接。子机只要不是离线状态，我们就可以和子机进行交互。对子机的指令仅能影响子机本身。子机的实例对象也是 uSDKDevice，参考章节 4.8 执行设备控制。

➢ 子机属性及报警

子机的属性和报警将通过回调上报给 App，实现以下接口并设定到子机对象实例上：

```
private class ChildDeviceCtrlListner implements IuSDKDeviceListener {

    @Override
    public void onDeviceAlarm(uSDKDevice uSDKDevice, List<uSDKDeviceAlarm> list) {
        System.out.println("我有子机报警信息……");
    }

    @Override
    public void onDeviceAttributeChange(uSDKDevice childDevice, List<uSDKDeviceAttribute> list) {
        System.out.println("我有子机属性状态信息……");
    }
```

```
        @Override
        public void onDeviceOnlineStatusChange(
                        uSDKDevice uSDKDevice,
                        uSDKDeviceStatusConst uSDKDeviceStatusConst, int i) {
                System.out.println("子机上下线消息：" + uSDKDevice.getDeviceId()
                        + "-" + uSDKDevice.getSubId()
                        + "-" + uSDKDevice.getSubType());
        }

    ……
}
```

- **主机控制**

我们可以像普通设备一样和主机交互，其中一些指令将影响到附属的所有子机。

- **辅助方法**

selectedChildDevice.getMainDevice()：获得主机

selectedChildDevice.isSubDevice()：子机？

selectedChildDevice.isMainDevice()：主机？

selectedChildDevice.getSubType()：子机类型

selectedChildDevice.getSubId()：子机 Id

parentDevice.getSubDeviceList()：获得当前有哪些子机

parentDevice.getSubDeviceById()：根据子机 Id 获得子机实例

4.11. 退出 uSDK

App 不需要使用 U+物联功能时可以停止 uSDK。

- **退出 uSDK**

```
uSDKManager.stopSDK();
```

4.12. 有用的类和方法

uSDKDevice 有用的方法

uSDKDevice.getIP 可以得到设备 IP 地址。

uSDKDevice.getStatus 查询设备的连接状态。

uSDKDevice.getNetType 可以得知设备是本地设备还是远程设备。

uSDKDevice.getDeviceType 查询设备类型，例如冰箱、洗衣机。

uSDKDevice.getUplusId 查询设备 uplusId(typeId)。

uSDKManager.getClientId()得到终端 ClientId。

调用 API 获得 uSDKDevice

uSDKDeviceManager.getDevcieList 获得当前可交互的所有设备。

uSDKDeviceManager.getDevice 使用 DeviceId 获得设备实例。

uSDKDeviceManager.getDeviceList(uSDKDeviceTypeConst.XXXXXXX)按类型取。

获得 uSDK 版本号

uSDKManger.getuSDKVersion()

5.使用 uSDK 绑定

从 4.0 开始 uSDK 开始提供绑定功能，同时 uSDKDevice 提供获取绑定信息 API（对应 uws 接口加密信息字段），方便 App 自行绑定用。

uSDKDevice 的 security 属性：0 为非安全设备，非 0 为安全类设备。安全类设备：设备配置入网成功后，10 分钟内获得绑定 key，安全设备必须绑定才能在 uSDK 中就绪。

5.1. 使用 uSDK 绑定步骤

1. 帐号登录 U+云成功
2. 执行连接用户接入网关，参考 4.8 章节
3. 设备配置入网成功
4. 使用 uSDK 绑定

调用本方法将入网设备绑定设备至 UWS。

```
uSDKDeviceManager.bindDevice(uSDKDevice, "device name", 90,
    new IuSDKCallback() {
        @Override
        public void onCallback(uSDKErrorConst result) {
            //绑定成功
            if(result == uSDKErrorConst.RET_USDK_OK) {
            }
        }
});
```

uSDKDevice：配置成功入网的设备

“device name”：UWS 接口定义的设备名称

90：90 秒，推荐时间 90s，范围 20-120（秒）

result：信息反馈，result 值为 uSDKErrorConst.RET_USDK_OK 时绑定成功。

本方法包含拿绑定信息和绑定两个步骤，本方法自带重试。

5.2. App 获取绑定信息

如果 App 自行编程绑定设备，请参考下列 API 获得绑定信息，示例方法：

```
uSDKDevice.getDeviceBindInfo(userToken, 60, new IuSDKGetDeviceBindInfoCallback() {
    @Override
    public void onDeviceBindInfoGet(uSDKErrorConst result, String info) {
        if(result == uSDKErrorConst.RET_OK) {
            Log.d("uSDK4.XDemo", "绑定 Info " + info);
        }
    }
});
```

userToken：用户登录 U+云成功的 token 值。

60：超时时间秒。

key：**“设备绑定信息”**

重载方法：getDeviceBindInfo(
String token,int timeout,TraceNode csNode,IuSDKGetDeviceBindInfoCallback callback)

注意

1. 本方法自带重试机制，在超时时间内自动重试。
2. 安全类设备：设备绑定是 App 使用 uSDK 与设备本地、远程交互的前提条件。
3. 拿绑定 key 有效时间为 10 分钟，失效后，需要重新进行设备配置，再次获取绑定 key。

5.3. 异常处理

绑定 Info 是设备从 U+云请求的，所以我们依赖于设备正常连接外网，连接 U+云。获取绑定 Info 或绑定失败时参考下表:

错误码	信息	解决建议
27002	账号未登录	账号登录成功
27004	coap 错误	程序重试
-27116	设备本身还没有拿到 Key	等待一会儿
-27117	设备未连接至 U+云	检查外网，等待设备一会儿
-10003	接口超时	程序重试
G20910	云平台返错	重新配置绑定
G20908	云平台返错	重新配置绑定

6. 海外 uSDK 业务指引

- **设定配置文件下载地址**

如果您的 App 主要在中国以外地区使用，请先调用下面准备方法再启动 uSDK，本方法的作用是使您的 App 可以在实际使用区域获得相关数据文件，使 uSDK 正常工作。

预备知识

参考章节 4.1 了解 uSDK 启动内容。

相关 API

uSDKErrorConst uSDKManager. setProfileServiceUrl (String profileServerUrl)

注意

1. profileServerUrl 是以 http 或 https 开头，端口号结尾的 url，例如：http://xxxxxx:9090
2. 方法返回值为 uSDKErrorConst.RET_USDK_OK 说明准备工作成功，不成功终止
3. profileServerUrl 值请根据区域确定

- 设定 uPlug 模块主网关地址及端口

如果您的目标智能设备不能连接本区域的设备接入网关（例如美国销售的智能设备安装中国区域 uPlug 模块），执行下述 API 方法使 uPlug 模块区域适配。

预备知识

配置入网请参考章节 4.2，与设备建立连接请参考章节 4.5。

本方法执行时机与步骤

1.智能设备**每次必须进入配置模式，10 分钟内完成操作。**

2.请连接设备，使设备已连接或就绪。

3.执行本方法修改主网关地址及端口。

4.超过 10 分钟没有成功，需重新进行步骤 1 至 3。

设定 uPlug 主网关地址及端口

```
//以下方法入参均为示例值
public void uSDKDevice.setDeviceGatewayAndPort("gw.haieroet.com", 8569, new IuSDKCallback() {
    @Override
    public void onCallback(uSDKErrorConst result) {
            if(uSDKErrorConst.RET_USDK_OK != result) {
```

```
                System.err.println("method exec failed!");
            }
        }
});
```

“gw.haieroet.com”：以字母开始的主网关域名字符串

8569：端口值

IuSDKCallback：结果回调

第二种设定 uPlug 主网关地址及端口方法

在执行设备 SoftAP 配置时，同时设定 uPlug 主网关地址及端口，示例如下：

uSDKDeviceConfigInfo**.setMainGatewayDomain();**

uSDKDeviceConfigInfo**.setMainGatewayPort();**

注意

1.uSDKDevice.setDeviceGatewayAndPort 方法执行失败请尝试重新执行，建议重试三次。

2.设定 uPlug 主网关地址及端口方法两种只选其一。

- **设定产品无线区域**

根据产品要求的需要为无线模块设定国家或地区，此方法在 SoftAP 写配置信息时执行：

预备知识

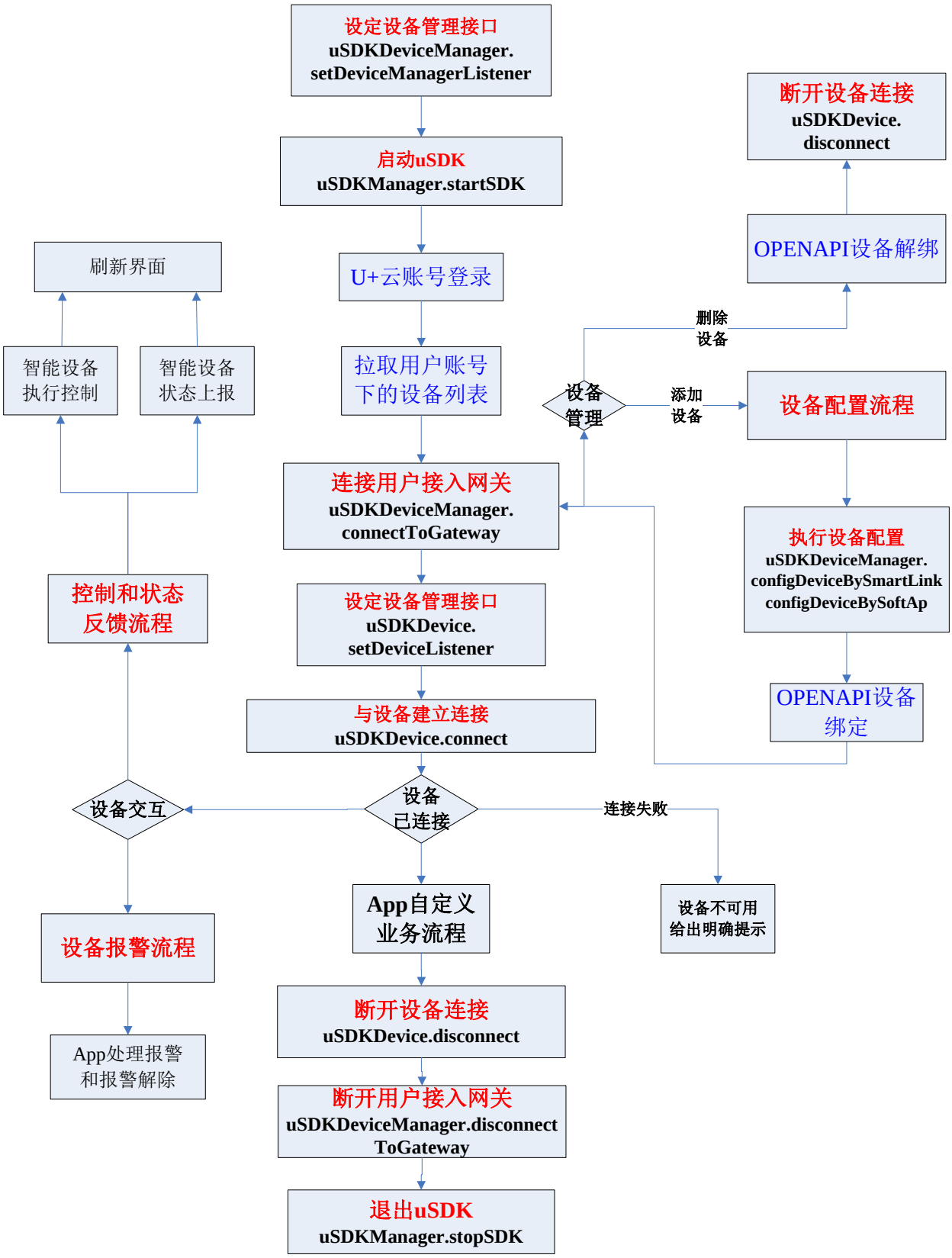
配置入网请参考章节 4.2

设定无线模块国家地区

```
//”JP”:日本，”CN”:中国，”US”:美国，”EU”:欧洲，”WW”:世界
//举例：设定为中国区
uSDKDeviceConfigInfo.setApPassword(“password”);
uSDKDeviceConfigInfo.setCountry("CN");
```

7. 使用 uSDK 控制设备流程说明图

以下流程图适用于使用 uSDK 进行设备配置入网，控制的 App：



8. 链式跟踪系统

8.1. 简介和作用

链式跟踪就是通过一条完整的调用链将散落在不同系统上的“孤立”日志联系在一起，然后通过日志分析，重组还原出更多有价值的信息，用于进行 APP 改进或决策。

链式跟踪的主线就是“调用链”，一条调用链包含了从源头请求(比如 APP 请求、终端设备请求等)到最后底层系统(比如数据库、分布式缓存等)的所有中间环节信息。每次调用，都会在源头请求中产生一个全局唯一的 ID(称为“uTraceId”)，通过网络依次将 uTraceId 传到下一个环节，每一个环节都会生成包含 uTraceId 在内的日志信息并上传到链式跟踪系统中。

8.2. 业务详解

链式跟踪系统中发送节点跟踪请求和发送节点跟踪回应结果是一一对应、成对出现的，形成链式跟踪的完整闭环，与实际业务中的方法执行和方法返回结果是相互对应。

链式跟踪系统目前只支持设备配置、绑定相关业务，uSDK 部分涉及到 SmartLink 配置和 SoftAp 配置的流程。绑定到云平台的相关业务可以参考此部分代码。

- 环境集成

链式跟踪系统依赖统计分析 SDK（uAnalytics），成功启动统计分析 SDK 后，才能正常运行链式跟踪信息的上传。统计分析 SDK 的集成与使用方法，请到海极网中开发者中心中下载相关资料。

地址：http://www.haigeek.com/static/index/index.html

- 创建链式跟踪

每一轮完整（uSDK 的配置开始到云平台的绑定结束）的链式跟踪，都应该以 uTrace 的创建为起点，且整个完整流程开始时只创建一次 uTrace。

初始化创建 uTrace 对象

```
uSDKManager.enableFeatures(uSDKManager.SDK_FEATURE_DEFAULT);
//UTRACE_BIND_BUSSINESS_ID:业务标识，需要 App 自行商定
Trace trace = Trace.createTrace(UTRACE_BIND_BUSSINESS_ID);
```

每次创建 uTrace 对象，都会在源头请求中产生一个全局唯一的 uTraceId，一个完整业务一个 Trace。

发送节点跟踪请求

```
TraceNode getBindInfoKeyTraceBeginNode =
            TraceNode.createCSNode(“getDeviceBindInfo”, “local”);
int result = AppDemo.trace.addTraceNode(getBindInfoKeyTraceBeginNode);
if(result != 0) {
    //说明您调用 uTrace 有错误，主要是调用顺序方面的错误
}
```

createCSNode：App 调用 uSDK 请求
getBindInfoKeyTraceBeginNode：调用 uSDK 获得绑定 Key 方法前创建
addTraceNode：发送节点跟踪请求方法，方法执行前调用。
result ：方法执行返回结果，失败时开发者检查代码

发送节点跟踪回应结果

```
TraceNode getBindInfoKeyResultNode
    = TraceNode.createCRNode(“getDeviceBindInfo”, “local”, getBindInfoKeyTraceBeginNode);
int result = AppDemo.trace.addTraceNode(getBindInfoKeyResultNode);
if(result != 0) {

}
```

createCRNode：App 收到 uSDK API 回应。
getBindInfoKeyResultNode ：业务任务打点，例如获得绑定信息 key 结束。
addTraceNode：发送节点跟踪回应结果的方法，例如：获得绑定信息 Key 结果。
result：方法执行返回结果，失败时开发者需要关注错误码。
getBindInfoKeyTraceBeginNode:本步骤对应的起始点，在此处将本步骤关联起来。

发送节点或服务器要求各项数据

bName，prot 等协议要求必须填写项，有 API 对应，直接使用 API，例如 setBusinessName 等。**其他上传数据，需要时直接调用 add 方法，无须二次数据加工**，例如：

```
TraceNode node ……
node.setBusinessName("bind");
node.setProtocol("local");
//...
node.add("pver","");
node.add("isnp","1");
node.add("model","SKFR-72LW/08FAA21AU1");
node.add("proc","3");
//...
```

链式跟踪系统信息查询

按照如上步骤进行操作，网络良好的情况下，链式跟踪相关节点的信息成功上传到链式跟踪服务器了，链式跟踪服务器的地址、帐号、汇总查询等问题，请联系云平台吴小波，邮箱：wuxiaobo.uh@haier.com。

9. 常见情况问答

1. 为什么我的设备配置不到路由器上？

答 :请首先确认产品说明书要求的路由器频段。市售路由器除了 2.4g 频段还有 5g 频段。目前智能设备只能连接 2.4g 频段 ;其次登录路由器查看负载数是否太多 ,网络条件是否较差；最后尝试修改路由工作模式为 bg 模式，20M 带宽。

2. 远程功能开发时不能获得设备属性是怎么回事？

1.检查 App 账号功能当前在哪个服务器环境，例如：开发者环境、联调环境。
2.检查智能设备所在路由的 DNS，是否和 App 账号功能在同一服务器环境。
3.检查连接用户接入网关时的各项参数是否正确输入，方法是否正常执行。

3. 为什么我连接设备成功之后总会弹报警解除？

报警解除消息代表设备当前没有报警。

4. 为什么我的控制指令失败了？

1.检查发送的指令是否按设备开发文档发送指令，命令 key 和 value 是否填写正确。
2.某些智能设备本身有自己的逻辑，例如，空调送风模式时不能设定温度。
3.更换无线网络或更换一台智能设备。

5. 与极路由结合开发设备配置入网功能

极路由可以安装 U+的插件程序，发现待入网设备并实现无密码配置入网。

前提条件

极路由升级固件至 1.4 版本以上，极路由安装自身市场插件：海尔智能家居，U+设备 WIFI 模块是瑞昱模组（空调：2.5.10 及以上；通用：2.3.30 及以上），uSDK4.2.02 及以上，极路由连接外网。

功能实现

- 发现待入网设备

为了简化开发，发现待入网设备使用已有接口，参考章节 4.2 配置设备入网内容”发现待入网智能设备&极路由发现待入网设备 “，得到待入网设备对象后，您需要先调用如下方法，决定采用何种方式将设备配置入网。

boolean flag = ConfigurableDevice.isSupportNoPwdConfig();当 flag 为 true 时，说明当前连接极路由，可以调用免密配置方法。flag 为 false，您需要参考 4.2 章节进行 SmartLink 或 SoftAP 方式配置入网。

- 无密码配置设备入网

```
String deviceId = ConfigurableDevice.getDevId();
//deviceId：MAC 值要求大写
uSDKDeviceManager.configDeviceByNoPassword(deviceId, 60, new IuSDKConfigCallback() {
    @Override
    public void onConfigCallback(
        uSDKDevice deviceJustCatched, uSDKErrorConst result {
        //result 值为 RET_USDK_OK 时，deviceJustCatched 就是入网成功的设备
        if(uSDKErrorConst.RET_USDK_OK == result) {

        }
    }
});
```

deviceId：设备 Id（Mac 值）。

timeout：建议 60，范围是 30s-120s。

IuSDKConfigCallback：结果回调。

重载方法：

```
uSDKDeviceManager.configDeviceByNoPassword(deviceId, typeId, timeout, IuSDKConfigCallback)
uSDKDeviceManager.configDeviceByNoPassword(
deviceId, typeId, timeout, TraceNode csNode, IuSDKConfigCallback)
```

ConfigurableDevice 对象有用的方法：

```
ConfigurableDevice.isSafe()  判断是否安全模块
ConfigurableDevice.getTag()   得到类似：uplus-haier-sh-2041-v3-sapz 这样的 SSID
ConfigurableDevice.getDevId()  得到全 MAC
ConfigurableDevice.getVersion()  得到 uplus-haier-sh-2041-v3-sapz 中的“v3”
```

10. 附录

10.1. 物联开发术语集

10.1.1. 物联模块 uPlug

按照 U+通信协议制造的物联模块，实现智能设备或智能家电联网。使用 uSDK 实现配置智能家电或设备入网时，智能家电或设备必须连接 uPlug。

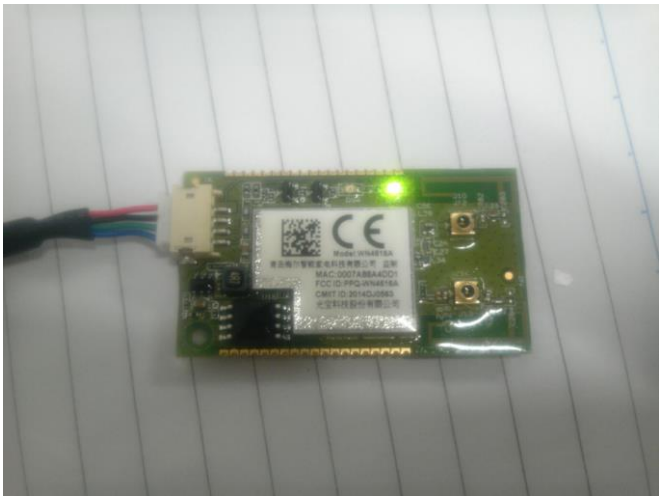


图 uPlug 物联模块

10.1.2. 智能家电、智能硬件或设备底板

智能家电：智能空调、智能酒柜、智能热水器、空气净化器等。

智能硬件：U+平台出品的空气盒子、醛知道等。

设备底板：以上产品的核心电子控制电路板（携带 uPlug 无线物联模块）。

图 智能硬件

10.1.3. 配置、配置设备入网

当用户购买新的智能设备或家电后，智能设备或家电当前并不能自动识别用户家里的无线网络，需要用户进行一些操作，将设备加入网络当中。这一过程就像手机加入某一无线网络，先选择无线，再输入密码。在实际操作过程当中，用户对于智能设备或家电的操作往往是按一个组合按键等，让智能设备进入待命状态，能够监听我们将要发送的无线配置信息。

如果智能设备或家电安装的是 uPlug 无线物联模块，那么配置设备入网时手机或平板电脑需要连接无线路由器。目前 uPlug 无线物联模块并不能很好的支持 5G 频段，所以手机或平板电脑务必连接 2.4G 无线网络。

10.1.4. TYPEID、uPlusId

TYPEID/uPlusId 是设备类型标识字符串，海极网创建智能设备后生成此值，一般存储在智能设备硬件上。

10.1.5. U+云平台 UWS

U+云平台为开发人员提供了一套网络 API，此套 API 主要用于和用户账号系统或 U+云家庭设备模型功能交互。UWS 开发文档请从海极网下载。

10.1.6. U+云平台用户接入网关

U+云平台有一套设备接入网关系统，此系统专职服务于设备连接功能，设备配置入网之后，只要智能设备所在网络可以连接互联网，uPlug 就会自动连接到 U+云平台设备接入网关。当用户应用进行远程连接或控制时，U+云平台设备接入网关也将成为 uSDK 的代理，中转 uSDK 各种设备相关指令。

10.1.7. 绑定、设备绑定

我们需要为用户和他（她）们的设备，在 U+云平台创建一个档案，收集用户的一些基本信息和智能设备的信息，保证他们的对应关系，以便在日后为用户进行增值服务。而由 App 发起将用户数据和设备数据上送到云的过程就叫做设备绑定，在技术层面上讲就是 App 发 http 请求到云，云接受数据并返回一个结果，说明设备绑定到用户账号是否成功。

10.1.8. 六位码

举例：20w001 代表开关机功能，执行开机 20w001，执行关机 20w002。此时应用程序通过 uSDK 发送｛20w001:20w001｝执行开机，发送｛20w001:20w002｝执行关机，而

当查询属性时，发现 20w001 对应的值是 20w001，说明电视开机。六位码就是一个键值对，表示设备的属性状态或者承载 App 的控制指令。

10.1.9. ID 文档、设备模型文档

ID 文档就是六位码的集合文档，主要用途是明确设备操作指令集等。设备模型文档是海极网创建硬件同时生成的文档，主要用途是设备属性和操作指令。

10.1.10. 设备就绪

和智能设备或家电的所有交互都是网络操作，当前智能硬件或智能家电的计算通信能力都是有限的，所以 uSDK 需要移动应用明确下达指令才会和设备建立数据通信链接，链接成功后，uSDK 将会查询指定设备的最新属性状态，并将此状态上报给移动应用，这样的一种状态就是设备就绪，就是设备具备可交互性，设备就绪以后才能通过 uSDK 发送指令。

10.1.11. 控制、状态反馈、设备报警

控制就是 App 通过 uSDK 发送控制指令操作设备。智能设备或家电通过 uPlug 物联模块传送当前自己的最新状态到 uSDK，uSDK 推送消息到移动应用叫做状态反馈。智能设备或家电通过 uPlug 物联模块传送当前自己的报警状态到 uSDK，uSDK 推送设备报警给移动应用是设备报警。

10.1.12. 单命令、组命令

ID 文档定义的一个指令属性。单命令发送时只发送命令本身，组命令发送时按 ID 文档要求发送指令组合。

10.1.13. 切网

切网就是用户操作移动设备切换当前设备使用其它网络，例如：用户使用无线网络，关闭无线网络，打开 3G 数据网络。用户使用无线网络 A，又切换到无线网络 B。

10.1.14. 小循环 VS 大循环

在使用 uSDK 及相关系统进行开发过程当中，可能会听到或看到两个词小循环、大循环，它们是两种情景的描述。小循环指的是移动应用程序与智能设备在同一无线局域网。大循环是说移动应用程序与智能设备不在同一无线局网，数据通路需要完全借助于 U+云平台设备接入网关的情景。