NB-IOT 协议接入说明文档

Version: 1.0

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **时间** | **修订人** | **描述** |
| **1.0.0** | 2017/10/30 | 物联网公司、研究院 | A-创建文档,整理接入方案1.0.0 |

### 资源模型介绍

#### 1.1协议介绍

OneNET提供了采用LWM2M+CoAP协议接入设备的说明文档，用户可以下载学习相关的具体内容，其中包括：

**LWM2M协议的介绍**

LWM2M是OMA组织制定的轻量化的M2M协议。LwM2M定义了三个逻辑实体:

* LWM2M Server 服务器；
* LWM2M Client 客户端，负责执行服务器的命令和上报执行结果；
* LWM2M 引导服务器 Bootstrap Server，负责配置LWM2M客户端。

在这三个逻辑实体之间有4个逻辑接口：

* Device Discovery and Registration：客户端注册到服务器并通知服务器客户端所支持的能力；
* Bootstrap：Bootstrap Server配置Client；
* Device Management and Service Enablement：指令发送和接收；
* Information Reporting：上报其资源信息。

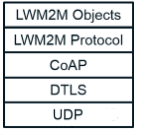


图1-1 LWM2M协议栈

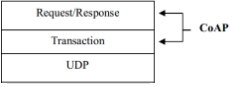
* LWM2M Objects：每个对象对应客户端的某个特定功能实体。LWM2M 规范定义了以下标准Objects，比如
  + urn:oma:lwm2m:oma:2; (LWM2M Server Object)；
  + urn:oma:lwm2m:oma:3; (LWM2M Access Control Object)；
  + 每个object下可以有很多resource，比如Firmware object可以有Firmware版本号，size等resource；
  + Vendor可以自己定义object。
* LWM2M Protocol：定义了一些逻辑操作，比如Read, Write, Execute, Discover or Observe等。

LWM2M协议的具体内容和消息格式可以参考OMA的网站

<https://en.wikipedia.org/wiki/OMA_LWM2M>

**CoAP协议的说明**

CoAP(Constrained Application Protocol)协议是IETF提出的一种面向网络的协议，采用了与HTTP类似的特征，核心内容为资源抽象、REST式交互以及可扩展的头选项等。CoAP协议基于REST构架，REST是指表述性状态转换架构，是互联网资源访问协议的一般性设计风格。为了克服HTTP对于受限环境的劣势，CoAP既考虑到数据报长度的最优化，又考虑到提供可靠通信。一方面，CoAP提供URI，REST式的方法如GET、POST、PUT和DELETE，以及可以独立定义的头选项提供的可扩展性。另一方面，CoAP基于轻量级的UDP协议，并且允许IP多播。为了弥补UDP传输的不可靠性，CoAP定义了带有重传机制的事务处理机制。并且提供资源发现机制，并带有资源描述。



CoAP协议栈示意图

CoAP由UDP作为承载，遵循UDP基本的协议报文格式，UDP数据内容部分按照CoAP协议报文格式进行写入传输。

CoAP协议格式说明如下：

* 【Ver】版本编号，指示CoAP协议的版本号。类似于HTTP 1.0 HTTP 1.1。版本编号占2位，取值为01B。
* 【T】报文类型，CoAP协议定义了4种不同形式的报文：CON报文，NON报文，ACK报文和RST报文。
* 【TKL】CoAP标识符长度。CoAP协议中具有两种功能相似的标识符，一种为Message ID（报文编号），一种为Token（标识符）。其中每个报文均包含消息编号，但是标识符对于报文来说是非必须的。
* 【Code】功能码/响应码。Code在CoAP请求报文和响应报文中具有不同的表现形式，Code占一个字节，它被分成了两部分，前3位一部分，后5位一部分，为了方便描述它被写成了c.dd结构。其中0.XX表示CoAP请求的某种方法，而2.XX、4.XX或5.XX则表示CoAP响应的某种具体表现。
* 【Message ID】报文编号。
* 【Token】标识符具体内容，通过TKL指定Token长度。
* 【Option】报文选项，通过报文选项可设定CoAP主机、CoAP URI、CoAP请求参数和负载媒体类型等等。
* 【1111 1111B】CoAP报文和具体负载之间的分隔符。

CoAP支持多个Option，CoAP的Option的表示方法比较特殊，采用增量的方式描述。一般情况下Option部分包含Option Delta、Option Length和Option Val三部分：

* 【Option Delta】表示Option的增量，当前的Option的具体编号等于之前所有Option Delta的总和。
* 【Option Length】表示Option Val终端设备的具体长度。
* 【Option Val终端设备】表示Option具体内容。



协议报文示意图

CoAP协议报文中具体数值的意义参考CoAP协议：IETF RFC7252。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Object | Object Instance | Resource | 权限 | 值 | 数据类型 |
| 摄像头 (100) | 0 | 最大像素(0) | R | 12,000,000 | Integer |
| 像素(1) | RW | 12,000,000 | Integer |
| 最大光圈(2) | R | 2.2 | Float |
| 最小光圈(3) | R | 32 | Float |
| 光圈(4) | RW | 5.6 | Float |
| 快门(5) | E |  |  |
| 当前图像(6) | R | 二进制图像 | Opaque (二进制) |
| … |  |  |  |
| 1 | 最大像素(0) | R | 8,000,000 | Integer |
| 像素(1) | RW | 8,000,000 | Integer |
| … |  |  |  |
| 光线传感器(101) | 0 | … |  |  |  |
| GPS(102) | 0 | … |  |  |  |
| 陀螺仪(103) | 0 | … |  |  |  |

#### 1.2资源模型

LWM2M 协议定义了三层资源模型：Object, Object Instance, Resource, 每一层都用数字ID来标识。其中，Object 是传感器类别，Object Instance 是传感器具体实例，Resource是传感器的属性和读数等，每个 Resource 具有不同的权限和数据类型，包括可读(R) /可写(W) /可执行(E)。

一个终端设备上可能存在多个 Object, 每个 Object 可能存在多个 Object Instance。

例如，把手机看作一个终端设备，则手机上有摄像头，光线传感器，GPS，陀螺仪等多种传感器，每种传感器便是一个 Object。假定摄像头的 Object ID 为 100，手机上有前后两个摄像头，即两个 Object Instance, ID 分别为 0, 1。每一个摄像头包括了像素，镜头焦距，光圈，快门值，快门等多种 Resource。

手机上的部分资源如下：(Object ID, Resource ID为示例)

可以使用类似于路径的方式来表示某个资源，例如 /100 表示 Object 100 (即摄像头)，/100/0 表示 Object 100, Object Instance 0 (即摄像头0)，/100/0/0 表示 Object 100, Object Instance 0, Resource 0 (即摄像头0的的最大像素)。

IPSO 的文档 <IPSO-Smart-Objects-Starter-Pack>, <IPSO-Smart-Objects-Expansion-Pac>中定义了一些常用传感器的 Object ID 和 Resource ID，终端设备应按照IPSO文档声明终端上的相关资源。

LWM2M 的资源模型与OneNET 的数据模型对应如下：

|  |  |  |
| --- | --- | --- |
| LWM2M | OneNET | 备注 |
| 终端设备 | 设备 Device | 平台为每个设备分配唯一ID |
| Object, Object Instance, Resource | 数据流 | 数据流的名称按照 规则ObjectID\_ObjectInstanceID\_ResourceID 生成，例如手机摄像头的例子中，资源 /100/0/1 对应的数据流为 100\_0\_1 |
| Resource Value | 数据点 | 资源在某个时间点上传的值，与时间戳一起，形成数据点，存储在OneNET |

#### 1.3订阅(Observe机制)

NB-IOT 终端设备通过 LWM2M 协议的订阅(Observe)/上报(Notify) 机制将数据上传到OneNET 平台。订阅可以在 Object 层，Object Instance 层，Resource 层，订阅上层Object时，其所属的所有Object Instance 都可以上报数据；订阅 Object Instance 时，其所属的所有Resource 都可以上报数据。每层订阅都是独立的，由资源路径唯一标识。例如，同时订阅了 /100/0 和 /100/0/0，则 /100/0/0 的数据会上报两次；只取消订阅 /100/0 或者 /100/0/0中的某一个, /100/0/0 的数据还是会上传。

设备上线时，平台会主动下发Observe 消息订阅设备上的所有Object Instance (可以在创建设备时关闭自动订阅)，也可以使用 API 订阅感兴趣的资源。

UE与平台间的通信接口基于LWM2M协议，在LWM2M协议以下基于CoAP协议，通信消息包括四部分：第一部分是发起Bootstrap，Bootstrap Write，Bootstrap完成，第二部分是注册、注销、更新注册消息；第三部分是观测消息、取消观测、消息上报；第四部分是设备管理操作，包括read/write/execute操作。

### SDK使用说明

根据应用的使用方式，基于已经集成SDK的模组, 应用可以采取两种方式：第一种方式是APP集成在模组上，这种场景下应用使用SDK提供的API接口实现；第二种方式是APP工作在自己的MCU上，同时使用集成了SDK的模组来提供OneNet及NBIOT接入的相关功能，这种场景下应用使用SDK的AT接口。

应用厂家使用集成完SDK的芯片和模组分为两种，第一种是使用SDK提供的API接口，这种方式所有的操作都是同步的，要求APP和模组在同一个芯片核内。第二种方式是使用AT接口，这种方式所有操作都是异步的，应用可以在自己的MCU中完成自己的应用的开发。后面分开介绍不同的接口应用的编写。

#### 2.1 API接口模式的应用

SDK的API接口包括对基础通信套件进行初始化、反初始化以及其他操作、相应的配置文件、回调函数以及结构体，SDK提供的API接口如下：

##### 2.1.1 API接口模式的应用

cis\_ret\_t cis\_init (void \*\* context, void \* config, uint32\_t size);

功能：根据输入的config文件进行SDK初始化

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \*\* | 指向void \*指针地址的指针变量，基础通信套件在初始化成功后会将申请到的上下文结构体地址写入该变量中 |
| config | 配置文件信息 | void \* |  |
| size | config的大小 | uint32\_t | typedef unsigned \_int16 uint16\_t; |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_deinit (void \* context);

功能：反初始化操作，释放SDK占用的资源

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_register (void \* context, const cis\_callback\_t\* cb);

功能：SDK初始化完成后，该接口向平台进行注册

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |
| cb | 设备回调 | const cis\_callback\_t\* |  |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_unregister (void \* context);

功能：该接口用于向服务器发起注销操作

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_addobject (void \* context, cis\_oid\_t objectid, const cis\_inst\_bitmap\_t\* bitmap, const cis\_res\_config\_t\* resource);

功能：SDK初始化完成后，使用该接口添加object

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |
| object id | object id | cis\_oid\_t \* |  |
| bitmap | 实例位图 | const cis\_inst\_bitmap\_t\* |  |
| resource | 属性数量及操作数量 | const cis\_res\_config\_t\* |  |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_delobject (void \* context, cis\_oid\_t objectid);

功能：SDK使用该接口删除object

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |
| object id | object id | oid\_t \* |  |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

uint32\_t cis\_pump (void \* context);

功能：用户在初始化完成后，调用该接口驱动SDK运行，SDK在该接口中调度多个任务，包括根据网络回复和状态发起注册消息，根据设定的lifetime定期发起更新注册消息，根据observe参数的设置上报notify消息。

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |

| 返回值 | 返回值含义 |
| --- | --- |
| 0 | 有任务待完成，不能进入休眠模式。 |
| 0xFFFFFFFF | 可随时进入休眠，自定义休眠时间 |
| 1 ~ 0xFFFFFFFF - 1 | 可随时进入休眠，并限定最长休眠时间，单位ms |

cis\_ret\_t cis\_change (void \* context, oid\_t objectid, iid\_t instid, rid\_t rid);

功能：SDK提供该接口，用户使用该接口通知SDK有数值变化。SDK检查当前值是否被平台observe，如果是，检查是否满足observe条件，在条件满足时向平台上报该数据。

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |
| objectid | 对象ID | cis\_oid\_t |  |
| instid | 实例ID | cis\_iid\_t |  |
| rid | 资源ID | cis\_rid\_t |  |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

uint32\_t cis\_update\_reg (void \* context, uint32\_t lifetime, bool withObjects);

功能：SDK提供该接口，用户使用该接口通知SDK更新注册的lifetime。如果withObjects 参数为true，则同时更新object信息，如果withObjects参数为false，则只更新lifetime。

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |
| lifetime | 生命周期 | uint32\_t |  |
| withObjects | 是否同时更新对象信息 | bool | true为更新对象信息，false为不更新对象信息。 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_notify (void\* context, cis\_ret\_t result, cis\_data\_t\* data, uint16\_t count, cis\_mid\_t msgid);

功能：当网络侧发来读、写、执行、观测、设置参数请求时，SDK会调用相关回调函数，并传输相应msgid，待真正操作完成时，应用程序使用cis\_notify接口返回结果(对于读，需要填写result及data字段，其中result表示读操作成功与否，如果成功，则result设为CIS\_CALLBACK\_READ\_SUCCESS，则data字段填入实际读结果，如果失败，则data为空；对于写、执行、观测、设置参数，只需分别将操作结果result设置为CIS\_CALLBACK\_WRITE\_SUCCESS、CIS\_CALLBACK\_EXECUTE\_SUCCESS、CIS\_CALLBACK\_OBSERVE\_SUCCESS、CIS\_CALLBACK\_OBSERVE\_PARAMS\_SUCCESS或相应的错误值，并将data置空)；或者，应用程序应根据网络侧所设置的数据上报策略自行调用相应AT指令接口填入上报数据，而该AT指令最终会调用cis\_notify接口进行数据上报，需要填写data字段，即需要上报的数据，result字段设为CIS\_CALLBACK\_NOTIFY\_SUCCESS。

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |
| result | 操作结果 | cis\_ret\_t |  |
| data | 上报的数据 | cis\_data\_t \* |  |
| count |  | uint16\_t |  |
| msgid | 消息ID | cis\_mid\_t | 该msgid从cis\_on\_observe函数获取 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_version (void \* context, cis\_version\_t\* ver);

功能：获取当前SDK版本时调用本接口获取当前SDK版本

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 上下文信息 | void \* | 上下文结构体地址 |
| ver | 基础通信套件版本 | cis\_version\_t\* | 主版本号和副版本号为cis\_version\_t 结构体里的两个uint8\_t型字段 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_uri\_make(cis\_oid\_t oid, cis\_iid\_t iid, cis\_rid\_t rid, cis\_uri\_t\* uri);

功能：根据输入的Object ID、Instance ID及Resource ID生成URI

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| oid | Object ID | cis\_oid\_t |  |
| iid | Intance ID | cis\_iid\_t |  |
| rid | Resource ID | cis\_rid\_t |  |
| uri | 待生成的URI | cis\_uri\_t\* | SDK将向该指针指向的结构体中填入相应内容 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_uri\_update (cis\_uri\_t\* uri);

功能：根据应用程序填入cis\_uri\_t结构体的Object ID、Instance ID及Resource ID更新结构体flag

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| uri | 待更新的URI | cis\_uri\_t\* | 应用程序在自行更改cis\_uri\_t结构体相关ID后可调用该API，基础通信套件将更新flag字段 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_data\_copy(cis\_data\_t\*\* dst, const cis\_data\_t \* src);

功能：拷贝cis\_data\_t型源数据至目标地址

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| dst | 目标数据 | cis\_data\_t\*\* |  |
| src | 源数据 | const cis\_data\_t \* |  |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cis\_data\_free(cis\_data\_t \* data);

功能：释放cis\_data\_t型指针指向的内存

参数及返回值：

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| data | 待释放数据 | cis\_data\_t\* |  |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

##### 2.1.2 数据结构和回调函数

###### 2.1.2.1 回调函数接口

| 回调函数含义 | 类型 | 描述 |
| --- | --- | --- |
| 读资源值回调 | cis\_read\_callback\_t | typedef cis\_ret\_t (\*cis\_read\_callback\_t)  (const void \* context,  cis\_uri\_t\* uri,  cis\_data\_tdata\_t\* value,  cis\_mid\_t msgid); |
| 写资源值回调 | cis\_write\_callback\_t | typedef cis\_ret\_t (\*cis\_write\_callback\_t)  (const void \* context,  cis\_uri\_t\* uri,  const cis\_data\_t\* value,  cis\_mid\_t msgid); |
| 执行回调 | cis\_exec\_callback\_t | typedef cis\_ret\_t (\*cis\_exec\_callback\_t)  (const void \* context,  cis\_uri\_t\* uri,  const uint8\_t\* value,  uint32\_t length,  cis\_mid\_t msgid); |
| 观测回调 | cis\_observe\_callback\_t | typedef cis\_ret\_t (\*cis\_observe\_callback\_t)  (const void \* context,  cis\_uri\_t\* uri,  bool flag,  cis\_mid\_t msgid); |
| 设置上报策略回调 | cis\_set\_param\_callback\_t | typedef cis\_ret\_t (\*cis\_set\_param\_callback\_t)  (const void \* context,  cis\_uri\_t\* uri,  const cis\_observe\_attr\_t \* attr,  cis\_mid\_t msgid); |
| 发现操作回调 | cis\_discover\_ callback\_t | typedef cis\_ret\_t (\*cis\_discover\_callback\_t)  (const void \* context,  cis\_uri\_t\* uri,  const cis\_data\_t \* value,  cis\_mid\_t msgid); |
| 事件回调 | cis\_event\_callback\_t | typedef void (\*cis\_event\_callback\_t)  (const void \* context,  cis\_event\_t id); |

基础通信套件在收到平台侧读数据请求时，分为三种情况：

1、请求的是read一个object

基础通信套件需要遍历该object下的所有instance，然后对不同的instance调用cis\_read\_callback\_t回调。

2、请求的是read一个object的一个instance

基础通信套件针对该instance下的所有resource调用cis\_read\_callback\_t回调。

3、请求的是read一个object的一个instance的一个resource

基础通信套件针对该resource调用cis\_read\_callback\_t回调。

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 设备上下文 | const void \* |  |
| uri | 设备URI | cis\_uri\_t\* | 表示Object、Instance、Resource ID的URI。若URI的flag其中resource ID位无效，则读该instance下的所有resource；若URI的flag其中instance ID位无效，则读该object下的所有instance |
| value | 设备属性 | cis\_data\_t data\_t\* |  |
| msgid | 消息ID | cis\_mid\_t | 基础通信套件提供一个唯一的消息标识，用来标识从网络侧获取的一条请求。msgid = 0时，该函数同步返回读数据结果；msgid ≠ 0的情形适用于AT指令实现方式，该函数立即返回，读操作真正完成后cis\_notify接口携带该标识异步返回结果。 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_CALLBACK\_READ\_SUCCESS | 成功 |
| CIS\_CALLBACK\_BAD\_REQUEST | 请求无效 |
| CIS\_CALLBACK\_UNAUTHORIZED | 未授权 |
| CIS\_CALLBACK\_NOT\_FOUND | 目标未找到 |
| CIS\_CALLBACK\_METHOD\_NOT\_ALLOWED | 方法不允许 |
| CIS\_CALLBACK\_SERVICE\_UNAVAILABLE | 服务不存在 |

基础通信套件在收到平台侧写数据请求时，基础通信套件调用cis\_write\_callback\_t回调。

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 设备上下文 | const void \* |  |
| uri | 设备URI | cis\_uri\_t\* | 表示Object、Instance的URI，其中flag中关于Resource ID的位会被置空，如果涉及到Resource ID，会在value字段中传入 |
| value | 设备属性 | const cis\_data\_t\* | 如果只写一个属性值，则value的id设置为目标的Resource ID，value的type为该Resource的具体类型(字符型、不透明型、整型、浮点型、布尔型)；如要写某instance的所有属性值，则需将value的type设为cis\_data\_type\_instance类型，并将asChildren的count字段设为该instance所有的属性数量，再将每个属性值填入asChildren的array数组。 |
| msgid | 消息ID | cis\_mid\_t | 基础通信套件提供一个唯一的消息标识，用来标识从网络侧获取的一条请求。msgid = 0时，该函数同步返回写数据结果；msgid ≠ 0的情形适用于AT指令实现方式，该函数立即返回，写操作真正完成后通过cis\_notify接口携带该标识异步返回结果。 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_CALLBACK\_WRITE\_SUCCESS | 成功 |
| CIS\_CALLBACK\_BAD\_REQUEST | 请求无效 |
| CIS\_CALLBACK\_UNAUTHORIZED | 未授权 |
| CIS\_CALLBACK\_NOT\_FOUND | 目标未找到 |
| CIS\_CALLBACK\_METHOD\_NOT\_ALLOWED | 方法不允许 |
| CIS\_CALLBACK\_SERVICE\_UNAVAILABLE | 服务不存在 |

基础通信套件在收到平台侧执行请求时，基础通信套件调用cis\_exec\_callback\_t回调。

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 设备上下文 | const void \* |  |
| uri | 设备URI | cis\_uri\_t\* | 表示Object、Instance、Resource ID的URI |
| value | 表征执行参数的buffer | const uint8\_t\* | 参数解析参见OMA LwM2M协议 |
| length | value的长度 | uint32\_t |  |
| msgid | 消息ID | cis\_mid\_t | 基础通信套件提供一个唯一的消息标识，用来标识从网络侧获取的一条请求。msgid = 0时，该函数同步返回执行结果；msgid ≠ 0的情形适用于AT指令实现方式，该函数立即返回，执行操作真正完成后通过cis\_notify接口携带该标识异步返回结果。 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_CALLBACK\_EXECUTE\_SUCCESS | 成功 |
| CIS\_CALLBACK\_BAD\_REQUEST | 请求无效 |
| CIS\_CALLBACK\_UNAUTHORIZED | 未授权 |
| CIS\_CALLBACK\_NOT\_FOUND | 目标未找到 |
| CIS\_CALLBACK\_METHOD\_NOT\_ALLOWED | 方法不允许 |
| CIS\_CALLBACK\_SERVICE\_UNAVAILABLE | 服务不存在 |

基础通信套件在收到平台侧观测请求时，基础通信套件调用cis\_observe\_callback\_t回调。

基础通信套件在收到平台侧观测请求时，分为三种情况：

1、请求的是观测一个object

基础通信套件需要观测该object下的所有instance，然后根据上报策略来上报变化给平台侧。

2、请求的是观测一个object的一个instance

基础通信套件需要观测该instance下的所有属性，然后根据上报策略来上报变化给平台侧。

3、请求的是观测一个object的一个instance的一个resource

基础通信套件需要观测该指定的resource，然后根据上报策略来上报变化给平台侧。

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 设备上下文 | const void \* |  |
| uri | 设备URI | cis\_uri\_t\* | 表示Object、Instance、Resource ID的URI。若URI的flag其中resource ID位无效，则观测该instance下的所有resource；若URI的flag其中instance ID位无效，则观测该obect下的所有instance |
| flag | 设置取消标志位 | bool | 如果flag 为true，则开启针对指定URI的观测  如果flag为flase，则停止针对指定URI的观测 |
| msgid | 消息ID | cis\_mid\_t | 基础通信套件提供一个唯一的消息标识，用来标识从网络侧获取的一条请求。msgid = 0时，该函数同步返回观测结果；msgid ≠ 0的情形适用于AT指令实现方式，该函数立即返回，观测操作真正完成后通过cis\_notify接口携带该标识异步返回结果。 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_CALLBACK\_OBSERVE\_SUCCESS | 成功 |
| CIS\_CALLBACK\_BAD\_REQUEST | 请求无效 |
| CIS\_CALLBACK\_UNAUTHORIZED | 未授权 |
| CIS\_CALLBACK\_NOT\_FOUND | 目标未找到 |
| CIS\_CALLBACK\_METHOD\_NOT\_ALLOWED | 方法不允许 |
| CIS\_CALLBACK\_SERVICE\_UNAVAILABLE | 服务不存在 |

基础通信套件在收到平台侧上报策略请求时，调用cis\_set\_param\_callback\_t回调。

基础通信套件在收到平台侧设置上报策略请求时，分为三种情况：

1、请求的是对一个object设置上报策略

基础通信套件需要对该object下的所有instance设置上报策略，在上报时检查数据是否满足上报策略。

2、请求的是对一个object的一个instance设置上报策略

基础通信套件需要对该instance下的所有属性设置上报策略，在上报时检查数据是否满足上报策略。

3、请求的是对一个object的一个instance的一个resource设置上报策略

基础通信套件需要对该指定的resource设置上报策略，在上报时检查数据是否满足策略。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 参数含义 | 类型 | 描述 |
| context | 设备上下文 | const void \* | 上下文结构体地址 |
| uri | 设备URI | cis\_uri\_t\* | 表示Object、Instance、Resource ID的URI。若URI的flag其中resource ID位无效，则为该instance下的所有resource设置参数；若URI的flag其中instance ID位无效，则为该obect下的所有instance设置参数 |
| attr | 上报策略参数 | const cis\_observe\_attr\_t \* | 上报策略参数结构体 |
| msgid | 消息ID | cis\_mid\_t | 基础通信套件提供一个唯一的消息标识，用来标识从网络侧获取的一条请求。msgid = 0时，该函数同步返回设置上报策略结果；msgid ≠ 0的情形适用于AT指令实现方式，该函数立即返回，设置上报策略操作真正完成后通过cis\_notify接口携带该标识异步返回结果。 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_CALLBACK\_OBSERVE\_PARAMS\_SUCCESS | 成功 |
| CIS\_CALLBACK\_BAD\_REQUEST | 请求无效 |
| CIS\_CALLBACK\_UNAUTHORIZED | 未授权 |
| CIS\_CALLBACK\_NOT\_FOUND | 目标未找到 |
| CIS\_CALLBACK\_METHOD\_NOT\_ALLOWED | 方法不允许 |
| CIS\_CALLBACK\_SERVICE\_UNAVAILABLE | 服务不存在 |

基础通信套件在收到平台侧Discover请求时，基础通信套件调用cis\_discover\_callback\_t回调。

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 设备上下文 | const void \* |  |
| uri | 设备URI | cis\_uri\_t\* | 表示Object、Instance的URI，其中flag中关于Resource ID的位会被置空 |
| value | 设备属性 | const cis\_data\_t\* | value的id设置为目标的Resource ID，value的type不需要填写。 |
| msgid | 消息ID | cis\_mid\_t | 基础通信套件提供一个唯一的消息标识，用来标识从网络侧获取的一条请求。msgid = 0时，该函数同步返回写数据结果；msgid ≠ 0的情形适用于AT指令实现方式，该函数立即返回，写操作真正完成后通过cis\_notify接口携带该标识异步返回结果。 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_CALLBACK\_WRITE\_SUCCESS | 成功 |
| CIS\_CALLBACK\_BAD\_REQUEST | 请求无效 |
| CIS\_CALLBACK\_UNAUTHORIZED | 未授权 |
| CIS\_CALLBACK\_NOT\_FOUND | 目标未找到 |
| CIS\_CALLBACK\_METHOD\_NOT\_ALLOWED | 方法不允许 |
| CIS\_CALLBACK\_SERVICE\_UNAVAILABLE | 服务不存在 |

基础通信套件会将注册、bootstrap等过程的结果通过cis\_event\_callback\_t回调通知应用。

| 参数名称 | 参数含义 | 类型 | 描述 |
| --- | --- | --- | --- |
| context | 设备上下文 | const void \* | 上下文结构体地址 |
| id | 状态ID | cis\_evt\_t | 事件类型id：  1：注册成功  2：注册失败  3：lifetime超时  4：update成功  -1: 状态错误 |

###### 2.1.2.2 结构体

枚举类型cis\_datatype\_t用来设备属性类型，具体含义如下：

typedef enum

{

cis\_data\_type\_undefine = 0, //未定义类型

cis\_data\_type\_string, //字符串类型

cis\_data\_type\_opaque, //不透明类型

cis\_data\_type\_integer, //整型

cis\_data\_type\_float, //浮点型

cis\_data\_type\_bool, //布尔型

cis\_data\_type\_object, //Object类型

cis\_data\_type\_instance, //Instance类型

}cis\_datatype\_t;

|  |  |  |  |
| --- | --- | --- | --- |
| typedef struct st\_cis\_data  {  uint16\_t id;  cis\_datatype\_t type;  struct  {  uint32\_t length;  uint8\_t\* buffer;  }asBuffer;  union  {  bool asBoolean;  int64\_t asInteger;  double asFloat;  struct  {  uint32\_t count;  cis\_data\_t \* array;  }asChildren;  struct  {  cis\_oid\_t objectId;  cis\_iid\_t instId;  }asObjLink;  } value;  } cis\_data\_t; | 名字 | 类型 | 描述 |
| id | uint16\_t | 相关Object、Instance、Resource的ID |
| type | cis\_datatype\_t | 属性类型 |
| value | union | 对象数据共用体 |
| asBoolean | bool | 布尔型属性值 |
| asInteger | int64\_t | 整型属性值 |
| asFloat | double | 浮点属性值 |
| asBuffer | asBuffer结构体 | buffer类型属性值 |
| asChildren | asChildren结构体 | 涉及到多个Instance或Resource时使用该结构体 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名称 | 参数含义 | 类型 | | 描述 |
| id | 资源ID | cis\_rid\_t | |  |
| type | 数据类型 | cis\_data\_type\_t | | typedef enum em\_cisattr\_data\_type{  cis\_data\_type\_undefine = 0,  类型未定义  cis\_data\_type\_string,  字符串型  cis\_data\_type\_opaque,  不透明类型  cis\_data\_type\_integer,  整型  cis\_data\_type\_float,  浮点型  cis\_data\_type\_bool,  布尔型  cis\_data\_type\_object,  对象类型  cis\_data\_type\_instance,  Instance类型  }cis\_data\_type\_t; |
| 结构体asBuffer | length | uint\_32 | | 无符号32位整形 |
| buffer | uint8\_t\* | | 无符号8位整形指针 |
| union  对象数据共用体value | asBoolean | bool | | 布尔型 |
| asInteger | int64\_t | | typedef signed \_\_int64 int64\_t;整型 |
| asFloat | double | | 双精度浮点型 |
| 结构体asChildren | count | uint32\_t | 无符号32位整形 |
| array | struct cis\_data\_t\* |  |
| 结构体asObjLink | objectId | oid\_t | object标识 |
| instId | iid\_t | instance标识 |

使用该object示例如下：

cis\_data\_t:string:  
                data->id =  id;  
                data->type = cis\_data\_type\_string;  
                data->value.asBuffer.length = strlen(valuestring);  
                data->value.asBuffer.buffer = (uint8\_t\*)strdup(valuestring);  
cis\_data\_t:float  
                data->id = id;  
                data->type = cis\_data\_type\_float;  
                data->value.asFloat= valuedouble;  
  cis\_data\_t:integer  
                data->id = id;  
                data->type = cis\_data\_type\_integer;  
                data->value.asInteger=  valueint;  
cis\_data\_t:bool  
                data->id =id;  
                data->type = cis\_data\_type\_bool;  
                data->value.asInteger=  valueint;

|  |  |  |  |
| --- | --- | --- | --- |
| typedef struct st\_cis\_observe\_attr  {  uint8\_t toSet;  uint8\_t toClear;  uint32\_t minPeriod;  uint32\_t maxPeriod;  float greaterThan;  float lessThan;  float step;  } cis\_observe\_data\_t; | 名字 | 类型 | 描述 |
| toSet | uint8\_t | 设置标志位，为1 |
| toClear | uint8\_t | 清除标志位 |
| minPeriod | uint\_32 | 最小时间间隔，单位为秒 |
| maxPeriod | uint\_32 | 最大时间间隔 |
| greaterThan | float | 观测数据大于 |
| lessThan | float | 观测数据小于 |
| step | float | 观测的数据变化最小区间 |

|  |  |  |  |
| --- | --- | --- | --- |
| typedef struct  st\_cis\_version  {  uint8\_t major;  uint8\_t minor;  } cis\_version\_t; | 名字 | 类型 | 描述 |
| major | uint8\_t | 主版本号 |
| minor | uint8\_t | 副版本号 |

|  |  |  |  |
| --- | --- | --- | --- |
| typedef struct  st\_cis\_inst\_bitmap  {  uint8\_t instanceCount;  uint8\_t instanceBytes;  const uint8\_t\* instanceBitmap;  } cis\_inst\_bitmap\_t; | 名字 | 类型 | 描述 |
| instanceCount | uint8\_t | 实例总数 |
| instanceBytes | uint8\_t | 实例位图实际占用字节数 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 名字 | 类型 | 描述 |
| instanceBitmap | const uint8\_t \* | 表征各实例的位图，若某个实例有效，则对应bit为1，否则为0。实例位图可视为一个数组，数组第0个字节的最高位对应于第0个实例，以此类推 |

|  |  |  |  |
| --- | --- | --- | --- |
| typedef struct  st\_cis\_res\_config  {  cis\_rid\_t attributeCount;  cis\_rid\_t actionCount;  } cis\_res\_config\_t; | 名字 | 类型 | 描述 |
| attributeCount | cis\_rid\_t | 属性总数 |
| actionCount | cis\_rid\_t | 操作总数 |

|  |  |  |  |
| --- | --- | --- | --- |
| typedef structst\_uri  {  uint8\_t flag;  cis\_oid\_t objectId;  cis\_iid\_t instanceId;  cis\_rid\_t resourceId;  }cis\_uri\_t; | 名字 | 类型 | 描述 |
| flag | uint8\_t | Flag用于表征Instance、Resource是否被设置，若设置则表示有效。第0位代表resource，第1位表示instance。 |
| objectId | cis\_oid\_t |  |
| instanceId | cis\_iid\_t |  |
| resourceId | cis\_rid\_t |  |

|  |  |  |  |
| --- | --- | --- | --- |
| typedef struct  st\_cis\_cfg\_sys  {  uint8\_t debug\_mode;  uint8\_t log\_extend\_mode;  uint8\_t log\_output\_mode;  uint8\_t log\_output\_level;  uint16\_t log\_buffer\_size;  cis\_cfgdata\_t ex\_data;  }cis\_cfg\_sys\_t; | 名字 | 类型 | 描述 |
| debug\_mode | uint8\_t | 调试信息开关 |
| log\_extend\_mode | uint8\_t | 扩展信息开关 |
| output\_mode | uint8\_t | 日志输出模式 |
| output\_level | uint8\_t | 日志调试级别 |
| log\_buffer\_size | uint16\_t | 日志缓冲区长度 |
| ex\_data | cis\_cfgdata\_t | 配置信息 |

|  |  |  |  |
| --- | --- | --- | --- |
| typedef struc  st\_cis\_cfg\_vdata  {  uint16\_t len;  uint8\_t\* data;  }; | 名字 | 类型 | 描述 |
| len | uint16\_t | 配置信息长度 |
| data | uint8\_t \* | 配置信息内容 |

##### 2.1.3 应用程序编写

使用SDK编写设备侧的应用包括如下步骤:

1. 准备SDK需要的配置文件，生成相应的二进制数据。

SDK需要的配置文件的具体参数包括如下面所示的要求。应用厂家编写XML格式的配置文件后使用cmiot\_configtool工具来生成对应的二进制数据文件。

cmiot\_configtool 是用于生成基础通信套件配置信息的工具，它将读取XML文件的内容，按固定格式保存为二进制文件，最终将此二进制文件读取到内存中提供给基础通信套件使用。

cmiot\_configtool 的使用方法示例如下：

cmiot\_configtool.exe sample.xml outputfile.bin

cmiot\_configtool 是用于生成基础通信套件配置信息的工具，它将读取XML文件的内容，按固定格式保存为二进制文件。

cmiot\_configtool 的使用方法示例如下：

cmiot\_configtool.exe sample.xml outputfile.bin

cmiot\_configtool 解析的XML文件描述内容举例如下：

<?xml version="1.0" encoding="utf-8"?>

<config>

<item>

<version>1.0</version>

<cfgcnt>3</cfgcnt>

</item>

<item id="0x01">

<data name="Lifetime">600000</data>

</item>

<item id="0x02">

<data name="Mtu">2455</data>

<data name="Linktype">1</data>

<data name="Host">192.168.1.1</data>

<data name="APN">CMIOT</data>

<data name="Username">\*\*\*\*</data>

<data name="Password">\*\*\*\*</data>

<data name="ExData">p1:abcdefg,p2:123456789</data>

</item>

<item id="0x03">

<data name="DebugMode">1</data>

<data name="ExtendMode">1</data>

<data name="OutputMode">2</data>

<data name="OutputLevel">2</data>

</item>

</config>

1. 提供操作需要的回调函数指针，包括如下的回调，这些回调函数里面要完成设备的读、写、执行、观测、发现、设置观测参数等功能。

static cis\_ret\_t cis\_on\_read (void\* context,cis\_uri\_t\* uri,cis\_data\_t\* value,cis\_mid\_t mid);

static cis\_ret\_t cis\_on\_write (void\* context,cis\_uri\_t\* uri,const cis\_data\_t\* value,cis\_mid\_t mid);

static cis\_ret\_t cis\_on\_exec (void\* context,cis\_uri\_t\* uri,const uint8\_t\* value,uint32\_t length,cis\_mid\_t mid);

static cis\_ret\_t cis\_on\_observe (void\* context,cis\_uri\_t\* uri,bool flag,cis\_mid\_t mid);

static cis\_ret\_t cis\_on\_discover (void\* context,cis\_uri\_t\* uri,cis\_data\_t\* value,cis\_mid\_t mid);

static cis\_ret\_t cis\_on\_params (void\* context,cis\_uri\_t\* uri,cis\_observe\_attr\_t parameters,cis\_mid\_t mid);

static void cis\_on\_event (void\* context,cis\_evt\_t eid);

以read消息回调为例,需要在回调函数里面读取真实硬件的信息,然后填写在value数据结构里:

1. 初始化SDK。

调用SDK的API接口进行初始化，在初始化时应用将之前生成的二进制数据的地址提供给SDK。

该功能调用cis\_init接口完成。

1. 添加设备的传感器信息

调用cis\_addobject接口添加传感器对象。

1. 发起设备注册请求消息

调用cis\_register接口来发起注册请求

1. 更新注册消息

调用cis\_update来更新注册信息

1. 驱动SDK运行

调用cis\_pump接口接口驱动基础通信套件运行，基础通信套件在该接口中调度多个任务，包括根据网络回复和状态发起注册消息，根据设定的lifetime定期发起更新注册消息，处理网络消息，调用对应的回调函数。

1. 完成工作后注销注册信息

调用cis\_deregister注销注册消息。

在这种模式中，应用需要完成对应的回调函数，包括网络的读、写、执行、观测、设置参数请求都会通过回调函数将信息回调给应用。

具体应用的编写可以参考应用示例代码：

#### 2.2 AT接口模式的应用

AT指令集是一组命令集合，终端使用这组命令集合来和模组或者芯片进行通信。AT指令由ASCII字符组成，使用前缀“AT”。每个AT请求都必须要求从模组或者芯片获得回复代码，用于回复终端当前请求的执行状态。SDK提供如下统一的接口给应用：

##### 2.2.1 AT接口说明

AT指令集是一组命令集合，终端使用这组命令集合来和模组或者芯片进行通信。AT指令由ASCII字符组成，使用前缀“AT”。每个AT请求都必须要求从模组或者芯片获得回复代码，用于回复终端当前请求的执行状态。SDK提供如下统一的接口给应用：

AT+MIPLCREATE=<totalsize>, <config>, <index>, <currentsize>, <flag> <CR>

功能：该指令创建一个SDK的实例

参数：

<totalsize>：config文件的总长度

<config>：配置文件，参见配置文件格式

<index>：配置文件的序号，考虑到AT指令长度有限，一个完整的配置文件未必能在一条AT指令中发送完成，可以将内容切分成多段，比如分为N段，则从前到后按照降序依次分配序号为N-1~0，按照从大到小序号的顺序每段调用一次AT指令，如此当index为0时意味着该条指令为最后一条配置消息

<currentsize>：当前指令所包含的配置文件长度

<flag>：消息标识

1：第一条消息

2：中间消息

0：最后一条消息

返回值：

<CR><LF>OK<CR><LF>：如果发送的是index不为0的消息，该回复代表消息正确接收

<CR><LF><ref><CR><LF>：如果发送的是index为0的消息，该回复代表消息正确接收，且返回一个创建完成的SDK的一个实例标识，类型为一个无符号整数

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLDELETE=<ref><CR>

功能：该指令删除一个SDK的实例

参数：

<ref>：SDK的一个实例标识，类型为一个无符号整数

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLOPEN=<ref><CR>

功能：该指令向平台发起注册请求

参数：

<ref>：SDK的一个实例标识，类型为一个无符号整数

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLCLOSE=<ref><CR>

功能：该指令向平台发起注销请求

参数：

<ref>：SDK的一个实例标识，类型为一个无符号整数

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLADDOBJ=<ref><objectid>, <instancecount>, <instancebitmap>, <attributecount><CR>

功能：该指令添加一个object

参数：

<ref>：SDK的一个实例标识，类型为一个无符号整数

<objectid>：对象id

<instancecount>：实例个数

<instancebitmap>：实例位图

<attributecount>：属性个数

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLDELOBJ=<ref>，<objectid><CR>

功能：该指令删除一个object

参数：

<ref>：SDK的一个实例标识，类型为一个无符号整数

<objectid>：对象id

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLNOTIFY=<ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, <valuetype>, <len>, <value>, <index>，<flag>,<CR>

功能：该指令通知平台一个数值变化

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id

<objectid>：对象id

<instanceid>：实例id

<resourceid>：资源id

< valuetype >：发送的数据类型

< len >：发送的数据长度

<flag>：消息标识

1：第一条消息

2：中间消息

0：最后一条消息

发送的数据类型要求如下：

string =1,//字符串型

opaque=2,//不透明类型

integer=3,//整型

float=4,//浮点型

bool=5,//布尔型

<len>：读取的数据长度

<value>：发送的数据值

<index>：指令序号。若某个Notify操作需要N条消息组合为一完整指令，则index从N-1至0降序编号，当index编号为0时表示本次Notify指令结束

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLREADRSP=<ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, <valuetype>, <len>, <value>, <index>, <flag><CR>

功能：用户接到+MIPLREAD消息后，需要去读取需要的资源的值，读取到需要的值，用户使用该指令将读取到的值一条一条发给SDK，将msgid置为收到的+MIPLREAD的消息携带的msgid。

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id

<objectid>：对象id

<instanceid>：实例id

<resourceid>：资源id

< valuetype >：发送的数据类型

发送的数据类型要求如下：

string =1,//字符串型

opaque=2,//不透明类型

integer=3,//整型

float=4,//浮点型

bool=5,//布尔型

<len>：读取的数据长度

<value>：发送的数据值

<index>：指令序号。若某个Read操作需要N条消息组合为一完整指令，则index从N-1至0降序编号，当index编号为0时表示本次Read指令结束

<flag>：消息标识

1：第一条消息

2：中间消息

0：最后一条消息

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLWRITERSP=<ref>, <msgid>, <result>, <index><CR>

功能：该指令通知SDK写入的消息结果。用户接到+MIPLWRITE消息后，需要去写入需要的资源的值，同时使用该消息通知SDK写入的结果。调用该命令时将msgid置为对应+MIPLWRITE消息携带的msgid。

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id

<result>：写入操作的结果；“0”代表写入错误，“1”代表写入正确

<index>：对应+MIPLWRITE消息的指令序号

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLEXECUTERSP=<ref>, <msgid>, <result><CR>

功能：该指令通知SDK执行操作的结果。用户接到+MIPLEXECUTE消息后，需要去执行请求的动作，同时使用该消息通知SDK执行的结果。调用该命令时将msgid置为对应+MIPLEXECUTE消息携带的msgid。

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id

<result>：写入操作的结果；“0”代表执行错误，“1”代表执行正确

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLOBSERVERSP=<ref>, <msgid>, <result><CR>

功能：该指令通知SDK观测指令是否有效。当应用程序收到+MIPLOBSERVE消息后，需要去验证该请求是否有效。调用该命令时将msgid置为对应+MIPLOBSERVE消息携带的msgid。

参数：

<ref>：基础通信套件的一个标识，类型为一个无符号整数

<msgid>：消息id

<result>：观测消息是否有效：“0”代表请求无效，“1”代表请求有效

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

AT+MIPLDISCOVERRESP=<ref>, <msgid>, <length>,<valuestring><CR>

功能：该指令回复SDK获取到的指定object的所有属性。

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id

<length>：返回valuestring的长度

<valuestring>:object的属性要求，多个属性之间使用逗号“，”隔开"1101,1102,1103"

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

+MIPLREAD: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, [<count>] <CR>

功能：该指令是一个读取请求消息，如果resourceid =‘-1’，则该指令会携带count

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id，标识该读取请求消息

<objectid>：对象id

<instanceid>：实例id

<resourceid>：资源id，如果为‘-1’，则代表需要读取该instance下的所有资源

<count>：需要读取的资源数

返回值：

无

+MIPLWRITE: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, <valuetype>, <len>, <value>, <index><CR>

功能：该指令是一个写操作请求消息

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id，标识该写入请求消息

<objectid>：对象id

<instanceid>：实例id

<resourceid>：资源id

< valuetype >：待写入的数据类型

待写入的数据类型要求如下：

string =1,//字符串型

opaque=2,//不透明类型

integer=3,//整型

float=4,//浮点型

bool=5,//布尔型

<len>：待写入的数据长度

<value>：待写入的数据值

<index>：指令序号。若某个Write请求需要N条消息组合为一完整指令，则index从N-1至0降序编号，当index编号为0时表示本次Write请求结束。

返回值：

无

+MIPLEXEUTE: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid><CR>

功能：该指令是一个执行操作请求消息

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id，标识该执行请求消息

<objectid>：对象id

<instanceid>：实例id

<resourceid>：资源id

返回值：

无

+MIPLOBSERVE: <ref>, <msgid>, <flag>,<objectid>, [<instanceid>], [<resourceid>] <CR>

功能：该指令是一个观测请求消息

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id，标识该观测请求消息

<flag>: 1 为添加观测，0 为取消观测

<objectid>：对象id

<instanceid>：实例id，如果为‘-1’，则代表观测该object下所有instance下的所有资源

<resourceid>：资源id，如果为‘-1’，则代表观测该instance下的所有资源

返回值：

无

+MIPLPARAMETER: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, <parameter>, <len><CR>

功能：该指令是一个设置策略参数请求消息

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id，标识该观测请求消息

<objectid>：对象id

<instanceid>：实例id, 如果为‘-1’，则代表观测该object下所有instance下的所有资源

<resourceid>：资源id，如果为‘-1’，则代表观测该instance下的所有资源

<parameter>：策略参数，格式为字符串形式

包括如下策略：

pmin=xxx; pmax=xxx; gt=xxx; lt=xxx; stp=xxx

<len>：参数长度

返回值：

无

指令名称：AT+MIPLUPDATE=<ref>,<lifetime>,<withObjectFlag><CR>

功能: 该指令通知SDK发送主动更新注册信息

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<lifetime>：更新的lifetime值，单位为ms，如果为0则表示使用默认的lifetime值

<withObjectFlag>：是否需要同时更新注册的Object对象

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

+MIPLDISCOVER:<ref>, <msgid>, <objectid><CR>

功能：该指令通知mcu需要获取指定object的属性。

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：消息id

<object>：指定的object对象id

AT+MIPLVER? <CR>

功能：该指令获取SDK版本

参数：无

返回值：

<CR><LF><version><CR><LF>：SDK的版本号

+MIPLERROR: <ref>, <msgid>, <errid><CR>

功能：该指令上报一个错误消息

参数：

<ref>：SDK的一个标识，类型为一个无符号整数

<msgid>：引发错误的消息id

<errid>：错误id（可扩充）

0：未知错误

1：系统错误

2：网络异常

3：注册失败

返回值：

<CR><LF>OK<CR><LF>

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。

##### 2.2.2 应用程序编写

该方式与API方式的主要不同在于所有操作都是异步操作，且所有提供的接口都为AT命令。应用需要完成的工作包括如下步骤。

1. 如3.1.1步骤完成配置文件生成
2. 使用AT指令来初始化

AT+MIPLCREATE=<totalsize>, <config>, <index>, <currentsize>, <flag> <CR>

1. 使用AT添加传感器对象

AT+MIPLADDOBJ=<ref><objectid>, <instancecount>, <instancebitmap>, <attributecount><CR>

1. 使用AT指令来发起注册请求

AT+MIPLOPEN=<ref><CR>

1. 注册完成后使用AT指令更新注册消息

AT+MIPLUPDATE=<ref>,<lifetime>,<withObjectFlag><CR>

1. 处理AT的URC指令，包括如下指令

+MIPLREAD: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, [<count>] <CR>

+MIPLWRITE: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, <valuetype>, <len>, <value>, <index><CR>

+MIPLEXEUTE: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid><CR>

+MIPLOBSERVE: <ref>, <msgid>, <flag>,<objectid>, [<instanceid>], [<resourceid>] <CR>

+MIPLPARAMETER: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, <parameter>, <len><CR>

+MIPLDISCOVER:<ref>, <msgid>, <objectid><CR>

+MIPLERROR: <ref>, <msgid>, <errid><CR>

1. 使用AT指令将处理的完成的AT回复消息提供给SDK

AT+MIPLNOTIFY=<ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, <valuetype>, <len>, <value>, <index>，<flag>,<CR>

AT+MIPLREADRSP=<ref>, <msgid>, <objectid>, <instanceid>, <resourceid>, <valuetype>, <len>, <value>, <index>, <flag><CR>

AT+MIPLWRITERSP=<ref>, <msgid>, <result>, <index><CR>

AT+MIPLEXECUTERSP=<ref>, <msgid>, <result><CR>

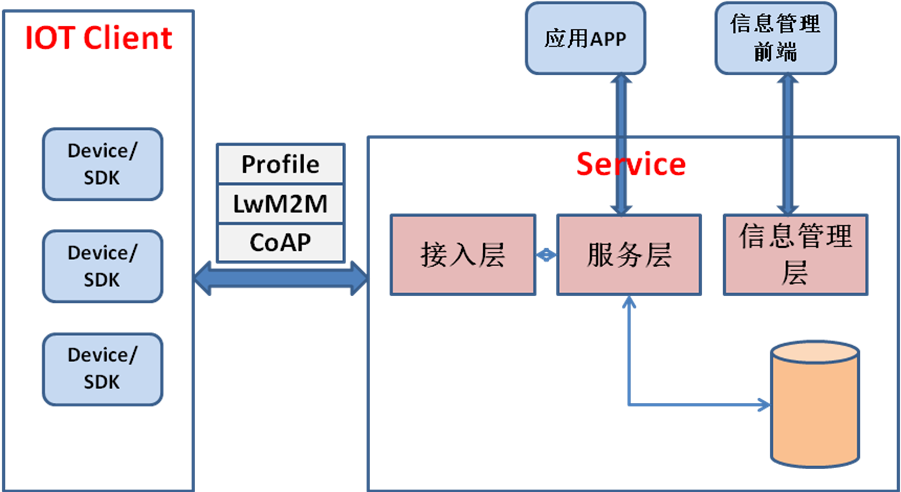
AT+MIPLOBSERVERSP=<ref>, <msgid>, <result><CR>

AT+MIPLDISCOVERRESP=<ref>, <msgid>, <length>,<valuestring><CR>

1. 完成工作后，使用AT注销设备的注册信息

### SDK移植说明

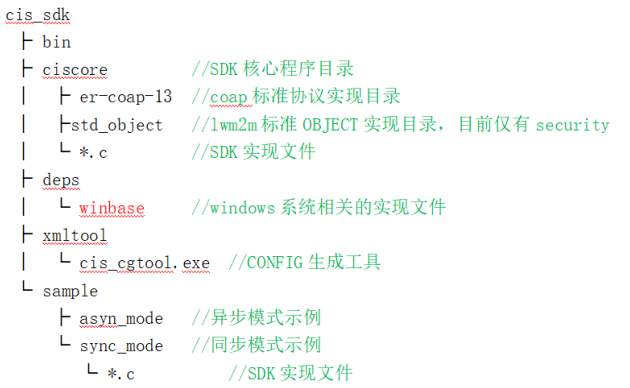
SDK实现数据传输协议中传输层协议CoAP，应用层协议LWM2M协议实现，以及基于IPSO组织制定的Profile规范基础上扩展资源模型，并对应用层提供统一的接口：终端能力开放API接口及AT指令集。



基础通信套件的架构

SDK的架构如图所示，包括了三部分的内容，最下层是基于UDP协议的CoAP实现，中间是基于CoAP的LWM2M协议实现，最上层是在LWM2M协议里面使用的IPSO定义的资源模型，用于对传感器以及传感器属性进行标识，该部分内容遵循IPSO组织制定的Profile规范。

#### 3.1 SDK目录结构



SDK核心目录ciscore，其中包括CoAP及LWM2M协议实现，以及SDK所有接口定义。

#### 3.2 SDK接口文件

SDK核心目录ciscore，其中cis\_api.h为用户侧API接口，移植过程中需要将ciscore目录中文件全部移植至目标平台。

其中cis\_if\_sys.h和cis\_if\_sys.h中定义接口函数需要在不同平台实现供SDK内部调用。

cis\_if\_net.h 为网络相关适配接口函数定义。

cis\_if\_sys.h 为系统相关适配接口函数定义。

#### 3.3网络接口适配

底层网络接口是指基础通信套件需要使用的网络功能，用于建立物理的网络连接，发送和接收网络数据。本章节定义了抽象的通信接口层，提供统一的接口给基础通信套件使用。

##### 3.3.1回调函数

底层网络接口初始化时需要一个参数cisnet\_callback\_t，该参数是一个结构体，其中onEvent用于事件回调，userData指针为当前的用户数据，具体的结构体定义如下：

typedef struct st\_cisnet\_callback\_t{

void \* userData;

cisnet\_event\_callback\_t onEvent;

}cisnet\_callback\_t;

事件回调函数定义如下：

typedef void (\*cisnet\_event\_callback\_t) (cisnet\_t netctx, cisnet\_event\_t id, void \* param, void \* userData);

底层网络需要通知基础通信套件事件时，调用onEvent来通知基础通信套件。

表10‑1 回调onEvent详解

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| netctx | cisnet\_t | 网络上下文 |
| id | cisnet\_event\_t | 事件id |
| param | void \* | 回调参数 |
| userData | void \* | 用户数据 |

其中事件具体定义如下：

typedef enum{

cisnet\_event\_unknow = 0, //发生未知事件

cisnet\_event\_timeout, //网络超时事件

cisnet\_event\_created, //网络链路创建完成事件

cisnet\_event\_connected, //网络连接完成事件(兼容TCP)

cisnet\_event\_disconnect, //网络连接断开事件

}cisnet\_event\_t;

##### 3.3.2结构体

网络配置数据的结构体定义如下：

struct st\_cis\_cfg\_net

{

uint16\_t mtu;

uint8\_t linktype;

cis\_cfgdata\_t host;

cis\_cfgdata\_t apn;

cis\_cfgdata\_t username;

cis\_cfgdata\_t password;

cis\_cfgdata\_t ex\_data;

} cis\_cfg\_net\_t;

其中mtu为最大传输单元的大小，linktype为传输类型，host是连接的目标主机的地址，apn为欲使用的apn名称，username为用户名，password为密码，ex\_data为其他配置信息。

其中cis\_cfgdata\_t的结构定义如下：

struct st\_cis\_cfg\_vdata

{

uint16\_t len; //数据的长度

uint8\_t\* data; //数据的内容

} cis\_cfgdata\_t;

##### 3.3.3底层网络接口

cis\_ret\_t cisnet\_init (const cisnet\_config\_t\* config, cisnet\_callback\_t cb);

功能：用于网络初始化

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| config | const cisnet\_config\_t\* | 网络配置 |
| cb | cisnet\_callback\_t | 网络回调函数结构体 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cisnet\_create (cisnet\_t\* ctx, const char\* host);

功能：用于请求创建链路，该函数为异步方法,当创建链路请求发出后,应立即返回;当链路建立完成后需调用事件回调函数向SDK发出cisnet\_event\_created事件通知

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| ctx | cisnet\_t\* | 网络上下文 |
| host | const char\* | 目标主机地址 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cisnet\_destroy (cisnet\_t ctx);

功能：用于销毁网络

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| ctx | cisnet\_t | 网络上下文 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cisnet\_connect (cisnet\_t ctx);

功能：用于请求网络连接,用于兼容TCP协议；该函数为异步方法,使用TCP协议时,当TCP连接请求发出后,应立即返回;当TCP连接建立完成后需调用事件回调函数向SDK发出cisnet\_event\_connected事件通知,使用UDP协议或其它面向无连接协议时同样需要在此函数中调用事件回调函数向SDK发出cisnet\_event\_connected事件通知后函数返回CIS\_RET\_OK成功.

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| ctx | cisnet\_t | 网络上下文 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cisnet\_disconnect (cisnet\_t ctx);

功能：用于断开网络

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| ctx | cisnet\_t | 网络上下文 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cisnet\_write (cisnet\_t ctx, const uint8\_t\* buffer, uint32\_t length);

功能：用于网络写入缓冲数据

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| ctx | cisnet\_t | 网络上下文 |
| buffer | const uint8\_t\* | 要写入的常量缓冲数据 |
| length | uint32\_t | 缓冲数据长度 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cisnet\_read (cisnet\_t ctx, uint8\_t\*\* buffer, uint32\_t \*length);

功能：用于网络读取缓冲数据

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| ctx | cisnet\_t | 网络上下文 |
| buffer | uint8\_t\*\* | 容纳读取数据的缓冲区 |
| length | uint32\_t\* | 缓冲区长度 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

cis\_ret\_t cisnet\_free (cisnet\_t ctx, uint8\_t\* buffer, uint32\_t length);

功能：用于释放缓冲区,由cisnet\_read中参数返回buffer指针，当SDK使用完成后调用该方法销毁；

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| ctx | cisnet\_t | 网络上下文 |
| buffer | uint8\_t\* | 容纳读取数据的缓冲区 |
| length | uint32\_t | 缓冲区长度 |

| 返回值 | 返回值含义 |
| --- | --- |
| CIS\_RET\_ERROR | 失败 |
| CIS\_RET\_OK | 成功 |

#### 3.4系统接口适配

底层系统接口是基础通信套件需要使用的底层功能，包括操作系统内存、获取系统时间、获取随机数等接口。

cis\_ret\_t cissys\_init (const cis\_cfg\_sys\_t\* config);

功能：用于系统初始化

参数及返回值：

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| config | const cis\_cfg\_sys\_t\* | 系统配置 |

| 返回值类型 | 返回值含义 |
| --- | --- |
| cis\_ret\_t | 初始化成功，返回CIS\_RET\_OK；否则，返回CIS\_RET\_ERROR |

uint32\_t cissys\_gettime ();

功能：用于获取系统时间戳

参数：无

| 返回值类型 | 含义 |
| --- | --- |
| uint32\_t | 获取系统时间毫秒数 |

void cissys\_sleepms (uint32\_t ms);

功能：系统延时函数

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| ms | uint32\_t | 系统延时毫秒数 |

返回值：无

void cissys\_logwrite(uint8\_t\* buffer,uint32\_t length);

功能：用于系统日志写入

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| buffer | uint8\_t\* | 要写入的日志缓冲数据 |
| length | uint32\_t | 缓冲数据长度 |

返回值：无

void \* cissys\_malloc (size\_t length);

功能：用于系统分配内存

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| length | size\_t | 申请内存长度 |

|  |  |
| --- | --- |
| 返回值类型 | 返回值含义 |
| void \* | 分配成功，则返回指向被分配内存的指针；否则，返回NULL |

void cissys\_free (void \* buffer);

功能：用于内存释放

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| buffer | void\_t\* | 要释放的内存 |

返回值：无

void \* cissys\_memcpy (void \* dst, const void \* src, size\_t n);

功能：用于内存复制

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| dst | void \* | 待复制的内存目标地址 |
| src | const void \* | 待复制的内存源地址 |
| n | size\_t | 待复制的内存长度 |

| 返回值类型 | 返回值含义 |
| --- | --- |
| void \* | 指向dst的指针 |

void \* cissys\_memmove (void \* dst, const void \* src, size\_t n);

功能：由src所指内存区域复制n个字节到dst所指内存区域。当内存发生局部重叠的时候，cissys\_memmove保证拷贝的结果是正确的，cissys\_memcpy不保证拷贝的结果的正确。

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| dst | void \* | 待复制的内存目标地址 |
| src | const void \* | 待复制的内存源地址 |
| n | size\_t | 待复制的内存长度 |

| 返回值类型 | 返回值含义 |
| --- | --- |
| void \* | 指向dst的指针 |

int cissys\_memcmp (const void \* s1, const void \* s2, size\_t n);

功能：用于内存比较

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| s1 | void \* | 比较的内存地址一 |
| s2 | void \* | 比较的内存地址二 |
| n | size\_t | 比较的内存长度 |

| 返回值类型 | 返回值含义 |
| --- | --- |
| int | 当buf1<buf2时，返回值小于0；  当buf1==buf2时，返回值等于0；  当buf1>buf2时，返回值大于0。 |

void \* cissys\_memset (void \* s, int c, size\_t n);

功能：用于将内存设置为某个指定的值

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| s | void \* | 要初始化的内存 |
| c | int | 初始化的值 |
| n | size\_t | 要初始化的内存长度 |

| 返回值类型 | 返回值含义 |
| --- | --- |
| void \* | 指向s的指针 |

void cissys\_fault (uint16\_t id);

功能：通知系统发生故障，需要进行处理

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| id | uint16\_t | 系统错误类型 |

id定义：

#define UNKNOW\_ERROR -1 // unknown error

#define MALLOC\_EEROR -2 //malloc error

#define NET\_ERROR -3 //net error

返回值：无

void cissys\_assert (bool flag);

功能：用于系统断言，如果断言条件返回错误，则终止程序执行

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| flag | bool | 断言条件 |

返回值：无

uint32\_t cissys\_rand ();

功能：用于系统生成随机数

参数：无

| 返回值类型 | 含义 |
| --- | --- |
| uint32\_t | 生成的随机整数 |

uint8\_t cissys\_getIMEI (uint8\_t\* buffer, uint32\_t len);

功能：用于获取IMEI号码

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| buffer | uint8\_t\* | IMEI号码 |
| len | uint32\_t | IMEI长度 |

| 返回值类型 | 含义 |
| --- | --- |
| uint8\_t | IMEI的有效长度 |

uint8\_t cissys\_getIMSI (uint8\_t\* buffer, uint32\_t len);

功能：用于获取IMSI号码

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| buffer | uint8\_t\* | IMSI号码 |
| len | uint32\_t | IMSI长度 |

| 返回值类型 | 含义 |
| --- | --- |
| uint8\_t | IMSI的有效长度 |

void cissys\_lock (void \* mutex);

功能：用于锁定当前操作

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| mutex | void\* | 信号量 |

void cissys\_unlock (void \* mutex);

功能：用于释放当前锁定的资源

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| mutex | void\* | 信号量 |

void        cissys\_lockcreate(void\*\* mutex);

功能：用于创建一个信号量

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| mutex | void\*\* | 信号量 |

void        cissys\_lockdestory(void\* mutex);

功能：用于销毁信号量

| 参数 | 类型 | 参数含义 |
| --- | --- | --- |
| mutex | void\* | 信号量 |