中 国 移 动 通 信 企 业 标 准

QB-XX-XXX-XXXX

中 国 移 动 蜂 窝 物 联 网 终 端 基 础 通 信 套 件 技 术 规 范

China Mobile Cellar IoT Terminal Device Suite

版 本 号 ： 2.0.0

XXXX-XX-XX 发 布　　XXXX-XX-XX 实 施

中国移动通信集团公司　　发布

目　　录

前　言

本标准对 Cellar IoT基础通信套件在技术方面的技术要素、技术特性进行了要求，确保 Cellar IoT基础通信套件能够满足网络运营和业务开展的需求。

本标准主要包括以下几方面内容：基础通信套件的API能力开放接口、基础通信套件的AT指令接口、基础通信套件的终端与平台通信接口、基础通信套件的对底层系统接口、基础通信套件的对底层网络接口等技术要求。本标准是在参考了3GPP国际标准和其他组织有关规范，并结合中国移动的实际业务需求的情况下编写而成。

本标准是中国移动 Cellar IoT技术规范系列标准之一，该系列标准的结构、名称或预计的名称如下：

序号	标准编号	标准名称
[1]	QB-E-064	中国移动NB-IoT终端总体技术规范

本标准的附录A、B、C、D、E、F为资料性附录。

本标准由中移　　号文件印发。

本标准由中国移动通信集团公司技术部提出，集团公司技术部归口。

本标准起草单位：中国移动通信集团公司研究院

本标准主要起草人：龙容、骆正虎、王波、田康、刘琨、李笑如

1.范围

本标准规定了蜂窝物联网终端的基础通信套件的接口要求，供中国移动内部和厂商共同使用，适用于所有蜂窝物联网终端设备。本规范主要包括以下基础通信套件几方面内容：应用层的统一的API接口，应用层的统一的AT指令集，底层系统接口的能力开放接口，底层网络接口的能力开放接口，网络和终端间的通信接口，集成要求，硬件资源需求，在线升级要求，资源模型要求，安全要求。

本标准发布之后入网的所有相关蜂窝物联网终端设备必须遵照执行，规范发布前入网的终端设备视具体需求而定。

2.规范性引用文件

下列文件中的条款通过本标准的引用而成为本标准的条款。凡是注日期的引用文件，其随后所有的修改单（不包括勘误的内容）或修订版均不适用于本标准，然而，鼓励根据本标准达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件，其最新版本适用于本标准。

序号	标准编号	标准名称	发布单位
[1]	GSM 07.05 version 5.5.0	Digital cellular telecommunications system (Phase 2+)	3GPP
[2]	RFC7252	The Constrained Application Protocol (CoAP)	IETF
[3]	OMA-TS-LightweightM2M-V1_0-20151214-C	Lightweight Machine to Machine Technical Specification	Open Mobile Alliance
[4]		中国移动NB-IoT终端总体技术规范	中国移动
[5]		中国移动蜂窝物联网总体安全技术要求	中国移动
[6]		中国移动蜂窝物联网数据与信息交换中心技术规范	中国移动
[6]	Smart Objects Starter Pack1.0	IPSO SmartObject GuideLine	Internet Protocol for Smart Objects (IPSO) Alliance
[7]	Smart Objects Expansion Pack	IPSO SmartObject Alliance Technical Guideline	Internet Protocol for

			Smart Objects (IPSO) Alliance

3.术语、定义和缩略语

下列术语、定义和缩略语适用于本标准：

词语	解释
NB-IoT	Narrow Band Internet of Thing，窄带物联网
CoAP	The Constrained Application Protocol，受限应用协议
UDP	User Datagram Protocol，用户数据报协议
UE	User Equipment，用户设备
IPSO	Internet Protocol for Smart Objects，智能对象因特网协议
LwM2M	Lightweight Machine to Machine，轻量级M2M协议
DTLS	Datagram Transport Layer Security，数据包传输层安全性协议
API	Application Programming Interface，应用编程接口
URI	Uniform Resource Identifier，统一资源标识符
FOTA	Firmware Over The Air,固件空中下载(模组)
SOTA	Software Over The Air 软件空中下载（MCU）
OTA	Over The Air,空中下载技术 （用于空中写卡）
C-IoT	Cellar Internet of Thing，蜂窝物联网

4.总则

在本标准中使用了“必选”、“推荐”、和“可选”等词汇来描述对移动终端产品要求的强调程度。“必选”是指设备产品所必须提供的功能或性能；“推荐”和“后续支持”是指对终端和未来运营很重要，暂时不用，但后续会使用的功能，产品提供的功能或性能；“可选”是指在标准中未作硬性要求，产品可提供也可不提供的功能或性能。

规范中需求除了明确指明为“必选”、“推荐”、“可选”、“后续支持”外，均为必须要求。

5.基本描述

5.1 概述

对于NB-IoT的终端设备，基础通信套件实现UE与平台(如OneNET平台)的通信，包括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。基础通信套件实现CoAP+LwM2M+Profile的协议封装，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。对于非NB-IoT的终端设备，除了使用本规范的接入方式，也可以采用其他的标准连接协议如MQTT，HTTP等接入OneNET。

本规范定义基础通信套件的如下部分，各部分接口根据集成方式不同有不同的集成要求，请参见第11章集成要求：

对应用层的统一API接口如下图中的①；

对应用层的统一的AT指令集如下图中的②；

对底层系统接口的能力开放接口如下图的③；

对底层网络接口的能力开放接口如下图的④；

对网络和终端间的通信接口如下图的⑤；

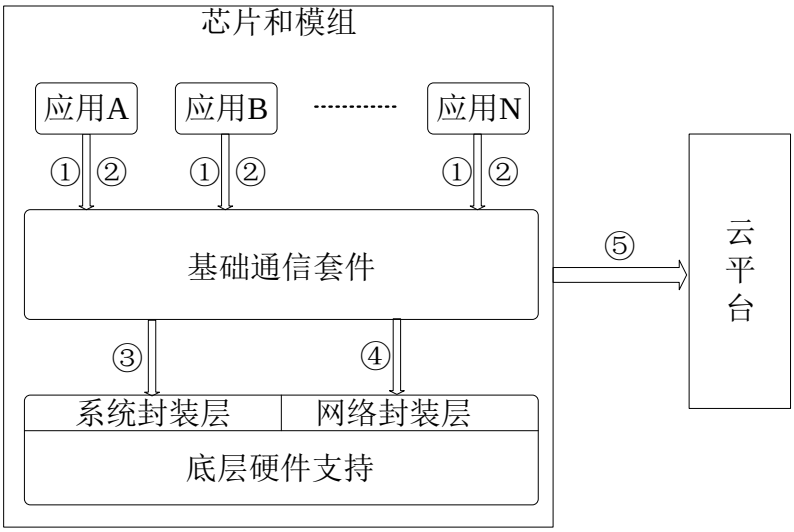


图5-1　基础通信套件接口关系图

本规范定义了运行在终端上的基础通信套件的各层接口及其他的技术要求，其中

第五章介绍基础通信套件的基本概念和架构；

第六章定义基础通信套件的能力开放API接口；

第七章定义基础通信套件的AT接口指令；

第八章定义UE与平台间的通信协议及消息；

第九章定义抽象的底层系统接口；

第十章定义抽象的底层网络接口；

第十一章定义基础通信套件的集成要求；

第十二章定义基础通信套件对硬件资源的要求；

第十四章定义资源模型的要求；

第十五章定义基础通信套件的安全要求。

5.2 架构

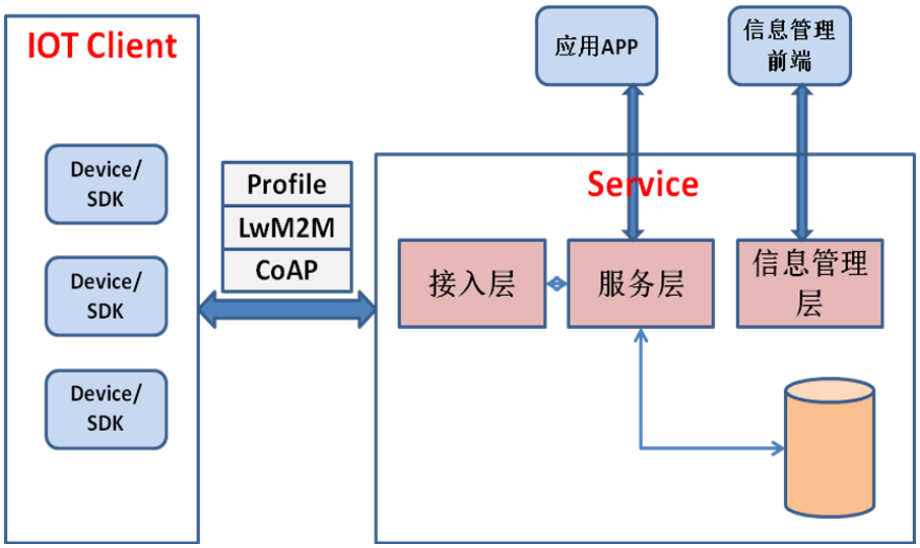


图 5-2 通信套件架构图

基础通信套件的架构如图5-2所示。基础通信套件负责UE和平台间的交互，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。

5.2.1 CoAP 协议

CoAP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。

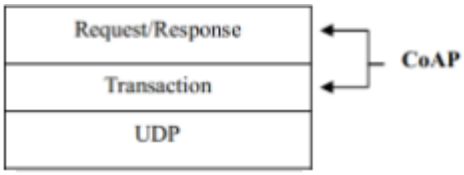


图 5-3 CoAP 协议栈示意图

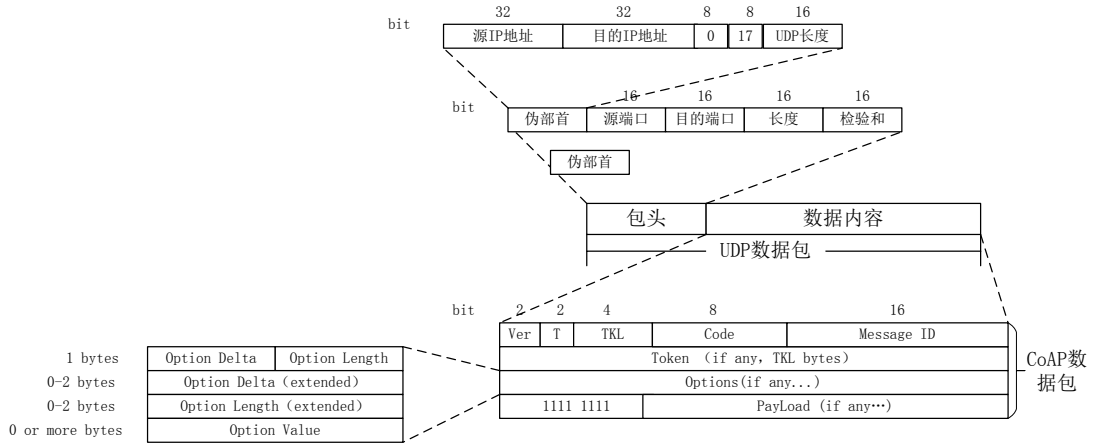


图 5-4 协议报文示意图

传输层为CoAP协议报文格式，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。

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

5.2.2 LwM2M 协议

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。

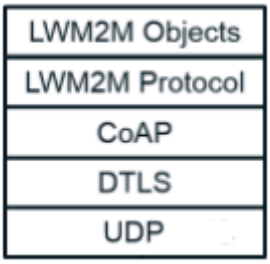


图 5-5 LwM2M 协议栈

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, Create or Delete。

CoAP：是IETF定义的Constrained Application Protocol，用来做LwM2M的传输层，下层是UDP。CoAP有自己的消息头，重传机制等。

DTLS 是指 Datagram Transport Level Security，即数据报安全传输协议，其提供了 UDP 传输场景下的安全解决方案，能防止消息被窃听、篡改、身份冒充等问题。

LwM2M协议具体参见OMA Lightweight M2M v1.0。

6.基础通信套件能力开放 API 接口

基础通信套件需要对外提供应用使用的统一的能力开放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，具体接口使用流程可以参考附录B。

6.1 配置参数要求

基础通信套件初始化时需要对应的配置参数，且按照规范要求的格式组织，在初始化时需要将一个指向该配置数据的指针提供给基础通信套件，基础通信套件获取对应的配置参数来完成初始化工作，当该配置数据的指针参数置 NULL 时，基础通信套件将设置默认配置参数完成初始化。

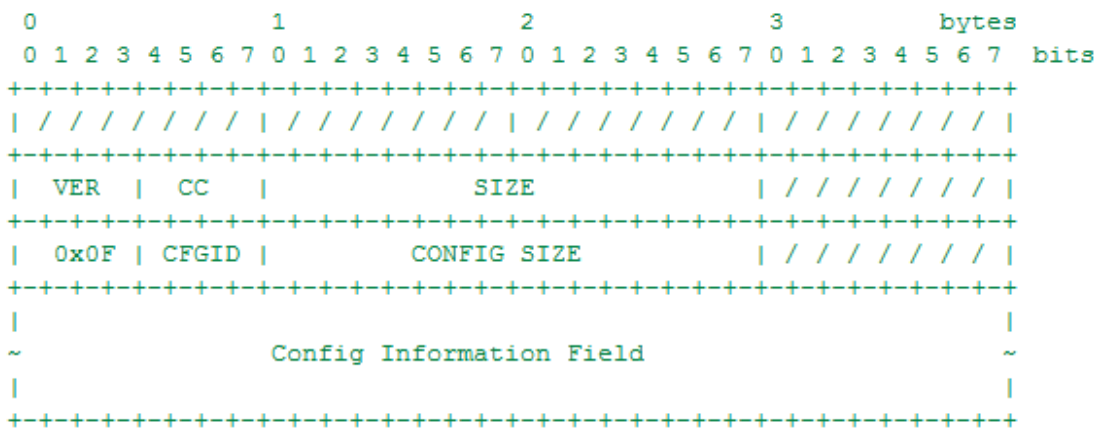


图 6-1 参数 config 结构示意图

[ver]：Version number，配置文件版本号，取值范围Value:0-15

[cc]：Config Count，配置的个数，取值范围Value:0-16

[size]：Config Size of Byte，配置字节大小，取值范围Value:0-65535

[CFGID]：Config ID，配置ID号，取值范围Value:0-15

表 6-1config ID 详解

参数名称	参数含义	描述

cfgid	配置ID号	typedef enum enum_cis_cfgid
		{
		cis_cfgid_init = 0x01,//初始化配置
		cis_cfgid_net = 0x02, //网络配置
		cis_cfgid_sys= 0x03 //系统配置
		} cis_cfgid_t;

[CFGID]：0x01，初始化配置，暂时预留

[CFGID]：0x02，网络配置

[MTU Size]：Maximum Transmission Unit最大传输单元大小，默认为1024

[LinkT]：连接类型，即底层基于UDP还是TCP，默认为UDP

[BandT]：网络类型，默认为1

[APN Len]：APN长度

[APN]：APN

[UserName Len]：用户名长度

[UserName]：用户名

[Password Len]：密码长度

[Password]：密码

[EnableBS]：是否打开BS流程，如果设置为1，则后面的主机地址提供BS的接入机地址，如果为0，则后面的主机地址提供直接接入机地址。

[EnableDTLS]：是否打开DTLS功能，如果设置为1则打开，为0则关闭。注意：打开DTLS功能必须要使能BS功能。

[Host Len]：Bootstrap或者接入主机地址长度

[Host ValueField]：Bootstrap或接入主机内容，格式为“server:port”，现阶段只

支持一个bootstrap服务器

[Userdata Length]：其他配置信息长度

[Userata]:其他配置信息,以“key:value;”的组合形式构成,其中第一个“key:value;”的key必须为AuthCode,value为具体的AuthCode,剩余的“key:value;”根据需求自行填写。基础通信套件不解析除了“AuthCode”外的所有“key:value;”数据，并将这部分剩余的配置数据透传给底层系统进行解析配置。

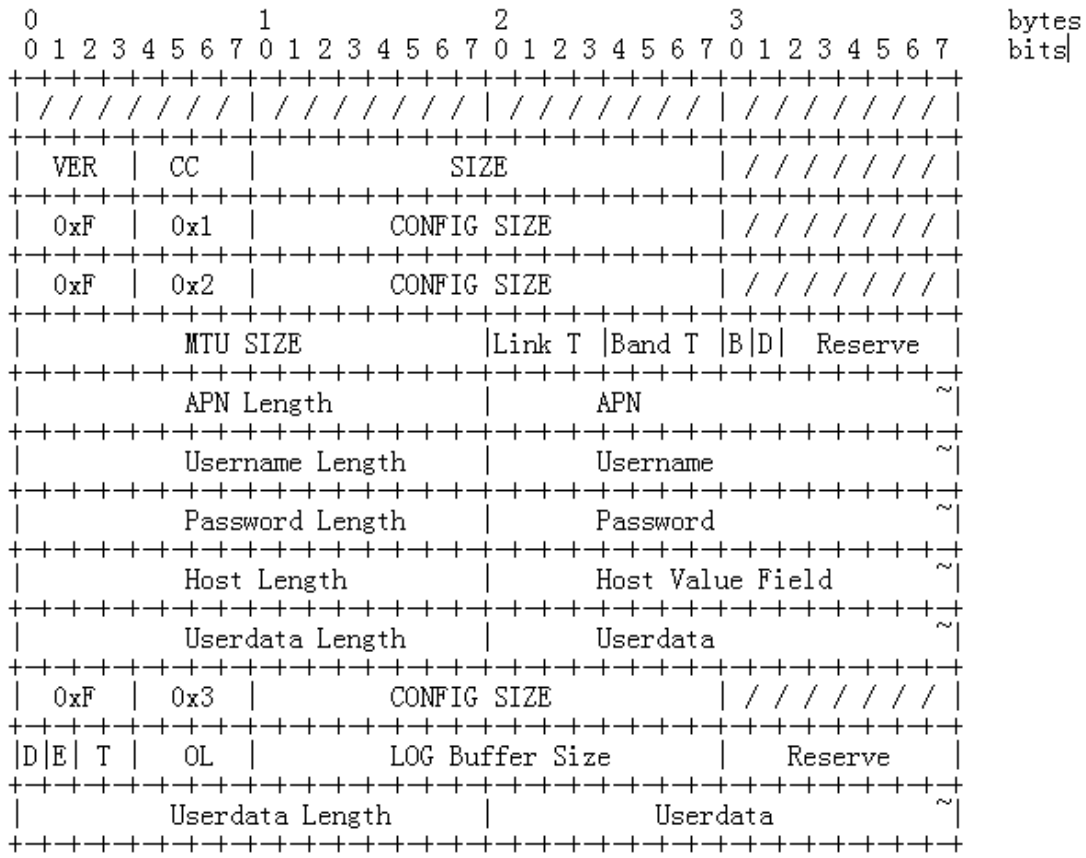


图 6-2config ID 为 2 具体结构示意图

[CFGID]：0x03，系统配置

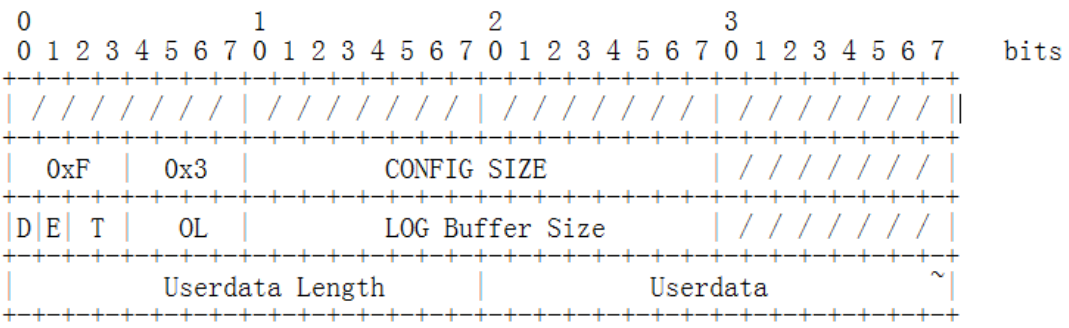


图 6-3config ID 为 3 具体结构示意图

[D]：Debug mode，调试开关，取值为0或1，具体代表如图6-4。

value	mode
0	DISABLE
1	ENABLE

图 6-4 [M]调试模式

[E]：Extend Info Mode，扩展信息开关，取值为0或1，具体代表如图6-5。

value	mode
0	DISABLE
1	ENABLE

图 6-5 [E]扩展信息模式

[T]：Log output mode，日志输出模式，具体模式如图6-6。

value	mode
00	DEFAULE
01	LCD
10	CONSOLE
11	UART

图 6-6 [T]日志输出模式

[OL]：Debug Level of Log Output，日志输出的调试级别，具体模式如图6-7。

value	mode
0001	DEBUG
0010	INFOMATION
0100	WARNING
1000	ERROR

图 6-7 [DL]日志输出的调试级别

[UserDataLen]：其他配置信息长度

[UserData]：其他配置信息。基础通信套件不解析该部分数据，将该配置数据透传给底层系统进行解析配置。

6.2 回调函数接口

基础通信套件定义了如下的回调函数，用于在收到网络侧的操作请求时，将对应的信息通知调用者进行处理。

表 6-2　回调函数列表

回调函数含义	类型	描述
读资源值回调	cis_read_callback_t	typedef cis_coapret_t (*cis_read_callback_t) (const void * context, cis_uri_t* uri, cis_mid_t msgid); 详见表6-3
写资源值回调	cis_write_callback_t	typedef cis_coapret_t (*cis_write_callback_t) (const void * context, cis_uri_t* uri, const cis_data_t* value, cis_attrcount_t attrcount, cis_mid_t msgid); 详见表6-5
执行回调	cis_exec_callback_t	typedef cis_coapret_t (*cis_exec_callback_t) (const void * context, cis_uri_t* uri, const uint8_t* value, uint32_t length, cis_mid_t msgid); 详见表6-7
观测回调	cis_observe_callback_t	typedef cis_coapret_t (*cis_observe_callback_t) (const void * context, cis_uri_t* uri, bool flag, cis_mid_t msgid); 详见表6-9
设置上报策略回调	cis_set_param_callback_t	typedef cis_coapret_t (*cis_set_param_callback_t) (const void * context, cis_uri_t* uri, const cis_observe_attr_t* attr, cis_mid_t msgid); 详见表6-11
发现操作回调	cis_discover_callback_t	typedef cis_coapret_t (*cis_discover_callback_t) (const void * context, cis_uri_t* uri,

回调函数含义	类型	描述
		cis_mid_t msgid); 详见表6-13
事件回调	cis_event_callback_t	typedef void (*cis_event_callback_t) (const void * context, cis_event_t id , void* param); 详见表6-15

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

调用者遍历该object下所有instance的所有资源进行读取操作,对于读取到的每个资源调用cis_response返回读取结果。

2、请求的是read一个object的一个instance

调用者遍历该instance下的所有resource进行读取操作,对于读到的每个资源调用cis_response返回读取结果。

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

调用者进行该resource读取后调用cis_response返回读取结果。

表 6-3cis_read_callback_t 回调函数参数详解

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
msgid	消息ID	cis_mid_t	msgid是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。读操作真正完成后通过6.4节的cis_response接口携带该标识异步返回结果。

表 6-4 函数返回值 cis_coapret_t 详解

返回值	返回值含义
CIS_CALLBACK_READ	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

返回值	返回值含义
CIS_CALLBACK_NOT_ACCEPTABLE	操作不被接受
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，完成写入操作后，调用cis_response返回写入结果。

表 6-5 cis_write_callback_t 回调函数参数详解

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ObjectID、InstanceID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*	详见6.3节的定义。 value的id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)；如果attrcount不是1，则value是一个数组，数组个数为attrcount。
attrcount	一个msgid对应的一组写操作的个数	cis_attrcount_t	用户要根据这个数目完成一次写入操作，最后一次写操作完成后，返回写操作结果给基础通信套件
msgid	消息ID	cis_mid_t	msgid是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。写操作真正完成后通过6.4节的cis_response接口携带该标识异步返回结果。

表 6-6　函数返回值 cis_coapret_t 详解

返回值	返回值含义
CIS_CALLBACK_WRITE	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_NOT_ACCEPTABLE	操作不被接受
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，完成执行后，调用cis_response返回执行结果。

表 6-7 cis_exec_callback_t 回调函数参数详解

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	执行参数的buffer	const uint8_t*	参数解析参见OMA LwM2M协议5.4.5 Execute章节
length	value的长度	uint32_t	
msgid	消息ID	cis_mid_t	msgid是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。执行操作真正完成后通过6.4节的cis_response接口携带该标识返回结果。

表 6-8 函数返回值 cis_coapret_t 详解

返回值	返回值含义
CIS_CALLBACK_EXECUTE	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_NOT_ACCEPTABLE	操作不被接受
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

调用者观测该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

调用者观测该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

调用者观测该指定的resource，然后根据上报策略来上报变化给平台侧。

表 6-9cis_observe_callback_t 回调函数参数详解

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

参数名称	参数含义	类型	描述
			效，则观测该obect下的所有instance
flag	观测标志位	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	msgid是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。观测操作真正完成后通过6.4节的cis_response接口携带该标识返回结果。

表 6-10 函数返回值 cis_coapret_t 详解

返回值	返回值含义
CIS_CALLBACK_ OBSERVE	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_NOT_ACCEPTABLE	操作不被接受
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
　　调用者对该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
　　调用者对该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
　　调用者对该指定的resource设置上报策略，在上报时检查数据是否满足策略。

表 6-11cis_set_param_callback_t 回调函数参数详解

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ObjectID、InstanceID、Resource ID的URI。若URI的flag标识中resource ID位未被设置，则表示为该instance下的所有resource属性设置观测参数；若URI的flag其中instance ID位未被设置，则表示

			为该object下的所有instance设置观测参数
attr	观测策略参数	const cis_observe_attr_t *	观测的策略参数结构体，请参见6.3
msgid	消息ID	cis_mid_t	msgid是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。设置上报策略操作真正完成后通过6.4节的cis_response接口携带该标识返回结果。

表 6-12 函数返回值 cis_coapret_t 详解

返回值	返回值含义
CIS_CALLBACK_OBSERVE_PARAMS	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_NOT_ACCEPTABLE	操作不被接受
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。

表 6-13 cis_discover_callback_t 回调函数参数详解

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
msgid	消息ID	cis_mid_t	msgid是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。发现操作真正完成后通过6.4节的cis_notify接口携带该标识返回结果。

表 6-14 函数返回值 cis_coapret_t 详解

返回值	返回值含义
CIS_CALLBACK_DISCOVER	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_NOT_ACCEPTABLE	操作不被接受

返回值	返回值含义
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。当前支持的事件类型包含：

```
#define     CIS_EVENT_BASE                  (cis_evt_t)0x00
#define     CIS_EVENT_BOOTSTRAP_START       CIS_EVENT_BASE + 1
#define     CIS_EVENT_BOOTSTRAP_SUCCESS     CIS_EVENT_BASE + 2
#define     CIS_EVENT_BOOTSTRAP_FAILED      CIS_EVENT_BASE + 3
#define     CIS_EVENT_CONNECT_SUCCESS       CIS_EVENT_BASE + 4
#define     CIS_EVENT_CONNECT_FAILED        CIS_EVENT_BASE + 5
#define     CIS_EVENT_REG_SUCCESS           CIS_EVENT_BASE + 6
#define     CIS_EVENT_REG_FAILED            CIS_EVENT_BASE + 7
#define     CIS_EVENT_REG_TIMEOUT           CIS_EVENT_BASE + 8
#define     CIS_EVENT_LIFETIME_TIMEOUT      CIS_EVENT_BASE + 9
#define     CIS_EVENT_STATUS_HALT           CIS_EVENT_BASE + 10
#define     CIS_EVENT_UPDATE_SUCCESS        CIS_EVENT_BASE + 11
#define     CIS_EVENT_UPDATE_FAILED         CIS_EVENT_BASE + 12
#define     CIS_EVENT_UPDATE_TIMEOUT        CIS_EVENT_BASE + 13
#define     CIS_EVENT_UPDATE_NEED           CIS_EVENT_BASE + 14
#define     CIS_EVENT_RESPONSE_FAILED       CIS_EVENT_BASE + 20
#define     CIS_EVENT_RESPONSE_SUCCESS      CIS_EVENT_BASE + 21
#define     CIS_EVENT_NOTIFY_FAILED         CIS_EVENT_BASE + 25
#define     CIS_EVENT_NOTIFY_SUCCESS        CIS_EVENT_BASE + 26
```

表 6-15cis_event_callback_t 回调函数参数详解

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	错误ID	cis_evt_t	事件类型，可扩展。具体见前面定义
param	错误参数	void*	错误返回携带的参数

6.3 结构体要求

枚举类型cis_datatype_t用来设置属性类型，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_datatype_t;
```

表 6-16 数据结构 cis_data_t*详解

typedef struct st_cis_data { cis_rid_t id; cis_datatype_t type; struct { uint32_t length; uint8_t* buffer; }asBuffer; union { bool asBoolean; int64_t asInteger; double asFloat; } value; } cis_data_t;	名字	类型	描述
	id	cis_rid_t	相关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

使用该object示例如下：

```
cis_data_t：string
        data->id  =   id;
      data->type  =  cis_data_type_string;
data-> asBuffer.length  =  strlen(valuestring);
        data-> 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;
cis_data_t:透明类型
        data->id  =id;
        data->type  =  cis_data_type_opaque;
        data->value.length  =  valueopaque.length;
        data->value.buffer  =  valueopaque.buffer;
```

表 6-17 数据结构 cis_data_t 参数详解

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_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

表 6-18 结构体 st_cis_observe_attr 详解

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

表 6-19 结构体 st_cis_inst_bitmap 详解

typedef struct	名字	类型	描述

st_cis_inst_bitmap { uint8_t instanceCount; uint8_t instanceBytes; const uint8_t* instanceBitmap; } cis_inst_bitmap_t;	instanceCount	uint8_t	实例总数
	instanceBytes	uint8_t	实例位图实际占用字节数
	instanceBitmap	const uint8_t *	表示各实例的位图，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

表 6-20 结构体 st_cis_res_count 详解

tpyedef struct st_cis_res_count
{
cis_attrcount_t attrCount;
cis_actcount_t actCount;
};
	名字	类型	描述

	attrCount	cis_attrcount_t	可读写资源的个数
	actCount	cis_actcount_t	可执行资源的个数

表 6-21 结构体 st_cis_version 详解

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

结构体uri用来记录object/instance/resource的信息。
其数据结构如下：

表 6-22 结构体 cis_uri_t 详解

typedef structst_uri { uint8_t flag; cis_oid_t objectId; cis_iid_t instanceId;	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

cis_rid_t resourceId; }cis_uri_t;			resource，第1位表示instance。
	objectId	cis_oid_t	
	instanceId	cis_iid_t	
	resourceId	cis_rid_t	

表 6-23 结构体 cis_cfg_sys_t 详解

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

表 6-24 结构体 cis_cfgdata_t 详解

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

6.4 接口要求

[需求编号]：TS-UE-SUITE-SERVRQ-00001
[需求描述]：初始化接口
函数原型：
cis_init (void** context, void* config,uint16_t configLen);
功能：根据输入的config文件进行基础通信套件初始化
参数及返回值：

表 6-24 cis_init 参数详解

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

参数名称	参数含义	类型	描述
config	配置文件信息	void *	可置NULL，当该参数为NULL时,基础通信套件将采用默认配置进行初始化，详见6.1节
config_len	config的数据的长度	uint16_t	当config参数为NULL,该参数应置0

表 6-25 cis_init 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00002
[需求描述]：反初始化接口
函数原型：
cis_ret_t cis_deinit (void ** context);
功能：反初始化操作，释放基础通信套件占用的资源
参数及返回值：

表 6-26 cis_deinit 参数详解

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

表 6-27 cis_deinit 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00003
[需求描述]：注册接口
函数原型：
cis_ret_t cis_register (void * context, cis_time_t lifetime,const cis_callback_t* cb);
功能：基础通信套件初始化完成后，该接口向平台进行注册
参数及返回值：

表 6-28 cis_register 参数详解

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

参数名称	参数含义	类型	描述
lifetime	生命周期	cis_time_t	生命周期，单位为秒
cb	设备回调	const cis_callback_t*	详见第10章

表 6-29 cis_register 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00004
[需求描述]：注销接口
函数原型：
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：

表 6-30 cis_unregister 参数详解

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

表 6-31 cis_unregister 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：

[需求编号]：TS-UE-SUITE-SERVRQ-00005
[需求描述]：添加object接口
函数原型：
cis_ret_t cis_addobject (void * context, cis_oid_t objectid, const cis_inst_bitmap_t* bitmap，const cis_res_count_t* rescount);
功能：基础通信套件初始化完成后，使用该接口添加object，基础通信套件会将用户添加的object注册到平台
参数及返回值：

表 6-32 cis_addobject 参数详解

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*	每一位表示一个实

参数名称	参数含义	类型	描述
			例，有效地实例为1，无效的为0。
rescount	资源的数量	const cis_res_count_t*	资源的可读写，可执行个数，参见上文结构体定义

表 6-33 cis_addobject 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00006
[需求描述]：删除object接口
函数原型：
cis_ret_t cis_delobject (void * context, cis_oid_t objectid);
功能：基础通信套件使用该接口删除object
参数及返回值：

表 6-34 cis_delobject 参数详解

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 *	

表 6-35 cis_delobject 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00007
[需求描述]：运行驱动接口
函数原型：
uint32_t cis_pump (void * context);
功能：用户在初始化完成后，调用该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，根据observe参数的设置上报notify消息。
参数及返回值：

表 6-36 cis_pump 参数详解

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

表 6-37 cis_pump 返回值详解

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s

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00008

[需求描述]：注册更新接口

函数原型：

uint32_t cis_update_reg (void * context, uint32_t lifetime, bool withObjects);

功能：基础通信套件提供该接口，用户使用该接口通知基础通信套件更新注册的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：

表 6-38 cis_update_reg 参数详解

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。

表 6-39 cis_update_reg 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00009

[需求描述]：用于处理读、写、执行、观测、设置订阅参数的返回

函数原型：

cis_ret_t cis_response (void* context,const cis_uri_t* uri,const cis_data_t* value,cis_mid_t mid,cis_coapret_t result);

功能：当网络侧发来读、写、执行、观测、设置参数等请求时，基础通信套件会调用相关回调函数，并在回调中携带相应msgid，待操作完成时，应用程序使用cis_response接口返

回结果。如果操作成功：对于读操作，如果是这组读操作的最后一条返回，则result设为CIS_RESPONSE_READ，如果不是最后一条消息，result设置为CIS_RESPONSE_CONTINUE。uri是读取的当前数据的标识，msgid是该条请求的标识，使用对应callback携带的msgid；对于写、执行、观测、设置参数，分别将操作结果result设置为CIS_RESPONSE_WRITE、CIS_RESPONSE_EXECUTE、CIS_RESPONSE_OBSERVE、CIS_RESPONSE_OBSERVE_PARAMS，uri和value为null。如果请求的操作失败，包括读、写、执行、观测、设置参数，则将uri和value置为空，result设置为对应的错误结果。

参数及返回值：

表 6-40 cis_response 参数详解

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_coapret_t	#define CIS_CALLBACK_CONFORM CIS_COAP_206_CONFORM #define CIS_CALLBACK_BAD_REQUEST CIS_COAP_400_BAD_REQUEST #define CIS_CALLBACK_UNAUTHORIZED CIS_COAP_401_UNAUTHORIZED #define CIS_CALLBACK_NOT_FOUND CIS_COAP_404_NOT_FOUND #define CIS_CALLBACK_METHOD_NOT_ALLOWED CIS_COAP_405_METHOD_NOT_ALLOWED #define CIS_CALLBACK_NOT_ACCEPTABLE CIS_COAP_406_NOT_ACCEPTABLE #define CIS_CALLBACK_SERVICE_UNAVAILABLE CIS_COAP_503_SERVICE_UNAVAILABLE
uri	对应数据的uri	const cis_uri_t*	该操作的值对应的uri
data	上报的数据	cis_data_t *	
msgid	消息ID	cis_mid_t	该msgid从cis_on_observe函数获取

表 6-41 cis_response 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：

[需求编号]：TS-UE-SUITE-SERVRQ-00010

[需求描述]：用于主动数据上报

函数原型：

cis_ret_t cis_notify (void* context,const cis_uri_t* uri,const cis_data_t* value,cis_mid_t mid,cis_coapret_t result,bool needAck);

功能：应用程序根据网络侧所设置的数据上报策略调用cis_notify接口进行数据上报，一组上报操作最多可以上报一个instance的数据，如果需要上报多个instance的数据，需要将数据分为多组进行上报。另外，每一次调用上报一个resource的数据，用户需要提供对应observe消息的msgid，该resouce对应的uri，对于一组上报，如果不是最后一条消息，则result字段应为CIS_NOTIFY_CONTINUE，如果是最后一条消息，result应该为CIS_NOTIFY_CONTENT。requestACK为保留参数，表示该组上报消息是否需要以CON形式上报。

参数及返回值：

表 6-40 cis_notify 参数详解

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_coapret_t	
uri	对应数据的uri	const cis_uri_t*	该条操作的值对应的uri
data	上报的数据	cis_data_t *	
msgid	消息ID	cis_mid_t	该msgid从cis_on_observe函数获取
requestACK	是否要求ACK	bool	是否要求ACK，默认为false

表 6-41 cis_notify 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：

[需求编号]：TS-UE-SUITE-SERVRQ-00011

[需求描述]：获取基础通信套件版本接口

函数原型：

cis_ret_t cis_version(void * context, cis_version_t* ver);

功能：获取当前基础通信套件版本时调用本接口获取当前基础通信套件版本

参数及返回值：

表 6-42 cis_version 参数详解

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

表 6-43 cis_version 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00012
[需求描述]：URI生成接口
函数原型：
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：

表 6-44 cis_uri_make 参数详解

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*	基础通信套件将向该指针指向的结构体中填入相应内容

表 6-45 cis_uri_make 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00013
[需求描述]：URI更新接口
函数原型：
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：

表 6-46 cis_uri_update 参数详解

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

表 6-47 cis_uri_update 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SERVRQ-00014

[需求描述]：初始化接口

函数原型：

cis_init_ota(void** context, void* config,uint16_t configLen, cis_time_t ota_timeout_duration);

功能：根据输入的config文件进行基础通信套件初始化

参数及返回值：

表 6-24 cis_init 参数详解

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 *	可置NULL，当该参数为NULL时,基础通信套件将采用默认配置进行初始化，详见6.1节
config_len	config的数据的长度	uint16_t	当config参数为NULL,该参数应置0
ota_timeout_duration	空写超时时间	Int32_t	物联网卡空中写卡超时时间，即启动一次空写任务,未发生写卡动作的超时退出时间,时间单位为秒。

表 6-25 cis_init 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：该初始化接口区别于TS-UE-SUITE-SERVRQ-00001的基础套件初始化接口，本初始化函数应用于中移动物联网卡的空中写卡升级，仅建立与空中写卡流程相关的SDK上下文。ota_timeout_duration为单次空中写卡的等待超时时间，单位为秒，终端在进入空写状态后若未完成写卡则超时退出。

[需求编号]：TS-UE-SUITE-SERVRQ-00015

[需求描述]：调用该函数向SDK返回sota升级的异步事件
函数原型：
cis_ret_t cis_notify_sota_result (void* context,uint8_t eventid)
功能：参数及返回值：

表 6-50 cis_notify_sota_result 参数详解

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
eventid	回调事件id	uint8_t	写flash成功，eventid为1 写flash失败，eventid为2 擦除flash成功，eventid为3 擦除flash失败，eventid为4 升级失败， eventid为8

表 6-51 cis_notify_sota_result 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求编号]：TS-UE-SUITE-SERVRQ-00016
[需求描述]：把MCU传递过来的版本信息和升级空间大小保存进flash中
函数原型：
cis_ret_t cis_set_sota_info(char* version,uint16_t size)
功能：参数及返回值：

表 6-51 cis_set_sota_info 参数详解

参数名称	参数含义	类型	描述
version	MCU版本信息	char *	MCU版本信息
size	MCU的升级空间大小	uint16_t	MCU的升级空间大小，单位KB

表 6-52 cis_set_sota_info 返回值详解

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

7.AT 指令接口

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。基础通信套件提供如下统一的接口给应用和URC事件给MCU：

[需求编号]：TS-UE-SUITE-ATRQ-00001

[需求描述]：

指令名称：AT+MIPLCREATE=<totalsize>, <config>, <index>, <currentsize>, <flag><CR>

功能：该指令创建一个基础通信套件的实例

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>+MIPLCREATE:<ref><CR><LF>

<CR><LF>OK<CR><LF>：

只有接到<index>=0的消息才会返回<ref>参数，该回复代表消息正确接收，且返回一个创建完成的基础通信套件的一个实例标识，类型为一个无符号整数。

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

errid可包含：

601参数错误

602状态错误

100未知错误

等错误返回，可扩展,厂家的扩展码大于650。

[需求澄清]：如果只有一条配置消息，则index和flag都为0。该接口为可选要求。

[需求编号]：TS-UE-SUITE-ATRQ-00002

[需求描述]：

指令名称：AT+MIPLCREATE<CR>

功能：该指令创建一个基础通信套件的实例

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>+MIPLCREATE:<ref><CR><LF>

<CR><LF>OK<CR><LF>：

消息正确接收返回一个创建完成的基础通信套件的一个实例标识,类型为一个无符号整数。

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

errid可包含：

601参数错误

602状态错误

100未知错误

等错误返回，可扩展,厂家的扩展码大于650。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00003

[需求描述]：

指令名称：AT+MIPLDELETE=<ref><CR>

功能：该指令删除一个基础通信套件的实例

参数：

<ref>：基础通信套件的一个实例标识，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。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00004

[需求描述]：

指令名称：AT+MIPLOPEN=<ref>，<lifetime>[，<timeout>] <CR>

功能：该指令向平台发起注册请求

参数：

<ref>：基础通信套件的一个实例标识，类型为一个无符号整数

<lifetime>:生命周期，单位为秒

<timeout>:注册的超时时长，单位为s

返回值：

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

<CR><LF>+CIS ERROR:<errid><CR><LF>：返回错误。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00005

[需求描述]：

指令名称：AT+MIPLCLOSE=<ref><CR>

功能：该指令向平台发起注销请求

参数：

<ref>：基础通信套件的一个实例标识，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。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00006
[需求描述]：
指令名称：AT+MIPLADDOBJ=<ref>,<objectid>, <instancecount>, <instancebitmap>, <attributecount>, <actioncount><CR>
功能：该指令添加一个object
参数：
<ref>：基础通信套件的一个实例标识，类型为一个无符号整数
<objectid>：对象id
<instancecount>：实例个数
<instancebitmap>：实例位图，字符串格式，每一个字符表示为一个实例，其中1表示可用，0表示不可用。例如当前添加的object有5个实例，其中，1，3可用，则实例位图为00101
<attributecount>：属性个数
<actioncount>：操作个数
返回值：
<CR><LF>OK<CR><LF>
<CR><LF>+MIPLERROR:<errid><CR><LF>：返回错误。errid定义同前面。
[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00007
[需求描述]：
指令名称：AT+MIPLDELOBJ=<ref>，<objectid><CR>
功能：该指令删除一个object
参数：
<ref>：基础通信套件的一个实例标识，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。errid定义同前面。
[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00008
[需求描述]：
指令名称：AT+MIPLNOTIFY=<ref>, <msgid>，<objectid>, <instanceid>, <resourceid>, <valuetype>, <len>, <value>,<index>, <flag> [, <ackid>] <CR>
功能：该指令通知基础通信套件一个数值变化
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
<objectid>：对象id
<instanceid>：实例id
<resourceid>：资源id
<valuetype >：发送的数据类型

<len>：上报的数据长度

<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

<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，<value>两端加双引号，如“abc”;<length>参数为字符串长度，即双引号内字符串长度，不包括双引号。

opaque=2,//不透明类型，<value>两端不加双引号，<length>参数为字节长度。

integer=3,//整型，<length>为该整型占用的字节数，可选2字节、4字节、8字节

float=4,//浮点型，<length>为该float占用的字节数，当前可选4字节和8字节

bool=5,//布尔型，0代表false，1代表true，<length>参数为1。

举例如下：

如果是string类型的value，则该指令如下

AT+MIPLREADRSP=<ref>, <msgid>, <result> [, <objectid>, <instanceid>, <resourceid>, 1, 4, “test”, <index>, <flag>]<CR>

如果是opaque类型的value，则该指令如下

AT+MIPLREADRSP=<ref>, <msgid>, <result> [, <objectid>, <instanceid>, <resourceid>, 2, 4, F1F2F3F4, <index>, <flag>]<CR>

如果是interger类型的value，则该指令如下

AT+MIPLREADRSP=<ref>, <msgid>, <result> [, <objectid>, <instanceid>, <resourceid>, 3, 2，800, <index>, <flag>]<CR>

如果是float类型的value，则该指令如下

AT+MIPLREADRSP=<ref>, <msgid>, <result> [, <objectid>, <instanceid>, <resourceid>, 4, 4，3.14159, <index>, <flag>]<CR>

如果是bool类型的value，则该指令如下

AT+MIPLREADRSP=<ref>, <msgid>, <result> [, <objectid>, <instanceid>, <resourceid>, 5, 1，0, <index>, <flag>]<CR>

<ackid>:MCU指定该消息以CON形式上报，如果该条Notify消息成功，则基础通信套件会将收到的返回用+MIPLEVENT返回。

返回值：

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

<CR><LF>+CIS ERROR:<errid><CR><LF>：返回错误。errid定义同前面。

[需求澄清]：如果对于同一条消息，只有一条notify指令，则index和flag都为0

一组上报消息最多只能上报一个instance的信息，如果需要上报多个instance的信息，需要按照instance对消息进行分组上报。

例如：收到了observe/3301， 该object有3个instance0，1，2，每个instance有4个可读的resource。用户需要将该object下的所有resource都上报时，至少需要分为三组数据上报。三组数据分别为：instance0的四个resource为一组，分别使用index为3，2，1，0上报三组数据，这三组数据msgid相同。

[需求编号]：TS-UE-SUITE-ATRQ-00009

[需求描述]：

指令名称:AT+MIPLREADRSP=<ref>, <msgid>, <result> [,<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基础通信套件,msgid值为收到的+MIPLREAD的消息携带的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>：读取操作的结果；可返回代码如下：

1 2.05 Content　　读取操作正确完成

11　　4.00 Bad Request

12　　4.01 Unauthorized

13　　4.04 Not Found

14　　4.05 Method Not Allowed

15　　4.06 Not Acceptable

<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，<length>为该整型占用的字节数

float=4,//浮点型，<length>为该float占用的字节数，当前可选4个字节和8个字节

bool=5,//布尔型，0代表false，1代表true<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。errid定义同前面。

[需求澄清]：如果对于同一条消息，只有一条notify指令，则index和flag都为0；如果result不是1，则<objectid>, <instanceid>, <resourceid>, <valuetype>, <len>, <value>, <index>, <flag>参数省略。result不为1代表该条读取消息结束，如果读取请求的消息中的某个资源已经通知了基础通信套件,则仅将能成功读取的消息通知给基础通信套件，并由基础通信套件上报到网络并结束本次读取过程；如果在通知错误的result之前没有读取到任何数据,则上报读取错误结果给平台并结束本次读取操作。所有资源都读取错误时，

MCU只需要回复一条错误result给通信套件，基础通信套件将上报读取错误回复给平台。

value和type的填写示例请参见AT+NOTIFY中描述

[需求编号]：TS-UE-SUITE-ATRQ-00010

[需求描述]：

指令名称：AT+MIPLWRITERSP=<ref>, <msgid>, <result><CR>

功能：该指令通知基础通信套件写入的消息结果。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基础通信套件写入的结果。调用该命令时msgid值为对应+MIPLWRITE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>：写入操作的结果；可返回代码如下：

2 2.04Changed写操作正确完成

11 4.00 Bad Request

12 4.01 Unauthorized

13 4.04 Not Found

14 4.05 Method Not Allowed

返回值：

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

<CR><LF>+CIS ERROR:<errid><CR><LF>：返回错误。errid定义同前面。

[需求澄清]：同一个消息id的write返回只需要一次结果返回。

[需求编号]：TS-UE-SUITE-ATRQ-00011

[需求描述]：

指令名称：AT+MIPLEXECUTERSP=<ref>, <msgid>, <result><CR>

功能：该指令通知基础通信套件执行操作的结果。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基础通信套件执行的结果。调用该命令时将msgid置为对应+MIPLEXECUTE消息携带的msgid。

参数：

<ref>：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>：执行操作的结果；可返回代码如下：

2 2.04Changed执行操作正确完成

11 4.00 Bad Request

12 4.01 Unauthorized

13 4.04 Not Found

14 4.05 Method Not Allowed

返回值：

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

<CR><LF>+CIS ERROR:<errid><CR><LF>：返回错误。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00012

[需求描述]：

指令名称：AT+MIPLOBSERVERSP=<ref>, <msgid>, <result><CR>

功能：该指令通知基础通信套件观测指令是否有效。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。该功能也包括cancelobserve的回复

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>：观测消息或者取消观察消息是否成功：可返回代码如下：

1 2.05Content操作正确完成

11 4.00 Bad Request

12 4.01 Unauthorized

13 4.04 Not Found

14 4.05 Method Not Allowed

15 4.06 Not Acceptable

返回值：

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

<CR><LF>+CIS ERROR:<errid><CR><LF>：返回错误。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00013

[需求描述]：

指令名称：AT+MIPLDISCOVERRSP=<ref>,<msgid>,<result> ,<length>,<valuestring><CR>

功能：该指令回复基础通信套件获取到的指定object的所有属性。

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>：发现操作的结果；可返回代码如下：

1 2.05 Content 操作正确完成

11 4.00 Bad Request

12 4.01 Unauthorized

13 4.04 Not Found

14 4.05 Method Not Allowed

15 4.06 Not Acceptable

<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。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00014

[需求描述]：

指令名称：AT+MIPLPARAMETERRSP=<ref>, <msgid>, <result><CR>

功能：该指令通知基础通信套件执行操作的结果。用户接到+MIPLPARAMETER消息后，需要去执行请求的动作，同时使用该消息通知基础通信套件执行的结果。调用该命令时msgid值为对应+ MIPLPARAMETER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>：设置订阅参数操作的结果；可返回代码如下：

1　　2.04Changed操作正确完成

11　　4.00 Bad Request

12　　4.01 Unauthorized

13　　4.04 Not Found

14　　4.05 Method Not Allowed

返回值：

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

<CR><LF>+CIS ERROR: <errid><CR><LF>：返回错误。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00015

[需求描述]：该指令通知基础通信套件发送主动更新注册信息

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

参数：

<ref>：基础通信套件的一个标识，类型为一个无符号整数

<lifetime>：更新的lifetime值，单位为s，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。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00016

[需求描述]：

指令名称：AT+MIPLVER? <CR>

功能：该指令获取基础通信套件版本

参数：无

返回值：

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

<CR><LF>+MIPLVER:version<CR><LF>：基础通信套件的版本号，如“10.01”。

<CR><LF>+CIS ERROR:<errid><CR><LF>：返回错误。errid定义同前面。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00017

[需求描述]：

指令名称：+MIPLREAD: <ref>, <msgid>, <objectid>, <instanceid>, <resourceid>,

<CR>

功能：该指令是一个读取请求消息参数：

<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，标识该读取请求消息

<objectid>：对象id

<instanceid>：实例id，如果为”-1”,则代表需要读取该object下的所有资源

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

[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00018

[需求描述]：

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

功能：该指令是一个写操作请求消息

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，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

<flag>：如果一条写入的消息value需要分多条指令上报，则消息标识需要根据当前的位置填写，如下：

1：第一条消息

2：中间消息

0：最后一条消息

<index>:一组写操作的序号，从大到小递减，比如N，N-1，…,0,最后一条的时候为0。

[需求澄清]：如果该msgid对应的消息只有一条，且不分条，则<flag>和<value>都为0

[需求编号]：TS-UE-SUITE-ATRQ-00019

[需求描述]：

指令名称：+MIPLEXECUTE:<ref>,<msgid>,<objectid>,<instanceid>,<resourceid>[,<len>,<arguments>]<CR>

功能：该指令是一个执行操作请求消息

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，标识该执行请求消息
<objectid>：对象id
<instanceid>：实例id
<resourceid>：资源id
<len>:参数的字符长度
<arguments>:执行的字符串格式参数
[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00020
[需求描述]：
指令名称：+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>：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，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
[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00021
[需求描述]：
指令名称：+MIPLPARAMETER: <ref>,<msgid>,<objectid>,<instanceid>,<resourceid>, <len>, <parameter><CR>
功能：该指令是一个设置策略参数请求消息
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，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
<len>：参数长度
<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
[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00022
[需求描述]：
指令名称：+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>：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
<object>：指定的object对象id
[需求澄清]：无

[需求编号]：TS-UE-SUITE-ATRQ-00023
指令名称：+MIPLEVENT: <ref>, <evtid>[,<extend>, <ackid>]<CR>
功能：该指令上报一个状态事件
参数：
<ref>：基础通信套件的一个标识，类型为一个无符号整数
<evtid>：事件id,十进制显示。

事件类型id，可扩展：

```
#define     CIS_EVENT_BASE                      (cis_evt_t)0x00
#define     CIS_EVENT_BOOTSTRAP_START           CIS_EVENT_BASE + 1
#define     CIS_EVENT_BOOTSTRAP_SUCCESS         CIS_EVENT_BASE + 2
#define     CIS_EVENT_BOOTSTRAP_FAILED          CIS_EVENT_BASE + 3
#define     CIS_EVENT_CONNECT_SUCCESS           CIS_EVENT_BASE + 4
#define     CIS_EVENT_CONNECT_FAILED            CIS_EVENT_BASE + 5
#define     CIS_EVENT_REG_SUCCESS               CIS_EVENT_BASE + 6
#define     CIS_EVENT_REG_FAILED                CIS_EVENT_BASE + 7
#define     CIS_EVENT_REG_TIMEOUT               CIS_EVENT_BASE + 8
#define     CIS_EVENT_LIFETIME_TIMEOUT          CIS_EVENT_BASE + 9
#define     CIS_EVENT_STATUS_HALT               CIS_EVENT_BASE + 10
#define     CIS_EVENT_UPDATE_SUCCESS            CIS_EVENT_BASE + 11
#define     CIS_EVENT_UPDATE_FAILED             CIS_EVENT_BASE + 12
#define     CIS_EVENT_UPDATE_TIMEOUT            CIS_EVENT_BASE + 13
#define     CIS_EVENT_UPDATE_NEED               CIS_EVENT_BASE + 14
#define     CIS_EVENT_RESPONSE_FAILED           CIS_EVENT_BASE + 20
#define     CIS_EVENT_RESPONSE_SUCCESS          CIS_EVENT_BASE + 21
#define     CIS_EVENT_NOTIFY_FAILED             CIS_EVENT_BASE + 25
#define     CIS_EVENT_NOTIFY_SUCCESS            CIS_EVENT_BASE + 26
```

<ackid>:当返回为CIS_EVENT_NOTIFY_SUCCESS时，携带Notify时下发的对应ackid。
<extend>:扩展参数，CIS_EVENT_RESPONSE_FAILED和CIS_EVENT_RESPONSE_FAILED事件时会携带该对应指令的msgid;CIS_EVENT_UPDATE_NEED事件时会携带LIFETIME剩余时间（单位秒）
[需求澄清]：
示例如下：
当事件类型为CIS_EVENT_UPDATE_NEED，LIFETIME剩余时间30秒
+MIPLEVENT: 1, 14，30<CR>
当事件类型为CIS_EVENT_RESPONSE_FAILED，Message Id为1234
+MIPLEVENT: 1, 21，1234<CR>

[需求编号]：TS-UE-SUITE-ATRQ-00024

[需求描述]：

指令名称：AT+OTASTART=<timeout_duration><CR>

功能：该指令触发空中写卡升级流程，并在写卡成功、写卡失败或写卡超时后以+OTAFINISH:N的形式返回命令。

返回值：

<CR><LF>OK<CR><LF>：

<CR><LF>+OTAFINISH:<res><CR><LF>

res结果码包含：

0　写卡成功完成

1　设备已有写卡成功的历史记录，无需写卡

30　平台侧未有该设备的写卡请求登记或登记异常

55　设备未发起写卡超时退出

56　设备已发起写卡后超时退出

57　设备写卡后的IMSI号码未改变或新号码非法错误

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

errid可包含：

601参数错误

602状态错误

100未知错误

等错误返回，可扩展,厂家的扩展码大于650。

[需求澄清]：AT+OTASTART作为申请空中写卡任务的触发，不支持与OneNET的用户应用连接同时存在，即不与AT+MIPLCREATE、AT+MIPLOPEN同时工作。

[需求编号]：TS-UE-SUITE-ATRQ-00025

[需求描述]：

指令名称：AT+OTASTATE=<ota_flag><CR>

AT+OTASTATE?<CR>

AT+OTASTATE=?<CR>

功能：该指令可配置、查询空写历史完成状态标志，该标志记录设备是否已成功完成了空中写卡流程。

返回值：

<CR><LF>OK<CR><LF>：

<CR><LF>+OTASTATE:<flag><CR><LF>：

flag标志码包含：

0 未进行空中写卡

1 已完成空中写卡

<CR><LF>+OTASTATE:<0-1><CR><LF>

[需求澄清]：空写完成状态标志应存储在非易失存储器，当设备未进行写卡时Flag值为0，此时通过AT+OTASTART=t触发写卡流程时，终端通过OneNET与物联网公司芯片管理平台交互,进行空中写卡;当首次写卡完成后,SDK会将Flag值置1并存储,此后再收到AT+OTASTART=t触发写卡流程时，SDK直接对外返回状态码+OTAFINISH:1提示设备已有写卡成功记录，无需写卡，结束写卡流程，且不与OneNET平台交互。

[需求编号]：TS-UE-SUITE-ATRQ-00026
指令名称AT+SOTARESULT=<result><CR>
功能：该指令将MCU升级时的产生的事件通知给模组
参数：result
<result>：事件id
返回：<CR><LF>OK<CR><LF>
[需求澄清]：MCU在一下5种情况下，需要发送该AT报文
1. 把接收到的升级报文写入 flash 中，成功，result 为 1
2. 把接收到的升级报文写入 flash 中，失败，result 为 2
3. 收到擦除 FLASH 的事件后，擦除 FLASH 成功，result 为 3
4. 收到擦除 FLASH 的事件后，擦除 FLASH 失败，result 为 4
5. 升级失败，比如软件包建议失败等，result 为 8
[需求编号]：TS-UE-SUITE-ATRQ-00027
指令名称AT+SOTAINFO=<”version”>,<upgrade space><CR>
功能：该指令将MCU当前的版本号和升级空间大小通知给模组
参数：
<“version”>：版本信息，字符串类型，不超过16字节
<upgrade space>:预留的升级空间大小，整型，单位KB
返回：<CR><LF>OK<CR><LF>
[需求澄清]：MCU需要在调用AT+MIPLCREATE指令之后，AT+MIPLOPEN指令之前调用该接口。

8.UE 与平台通信接口

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，通信消息包括三部分：
第一部分是注册、注销、更新注册消息；
第二部分是观测消息、取消观测、消息上报；
第三部分是设备管理操作，包括read/write/execute/discover操作。

8.1 协议适配要求

表 8-1 协议适配要求

协议功能	平台状态	终端要求
Client Registration Interface（客户端注册）	支持	UE注册默认选择的Binding Mode 为 U（UDP）
Device Management	支持	UE 要求能支持 Read, Write,

Interface（设备管理）		Execute 指令
Resource Observe and Information Reporting Interface（资源订阅与数据上报）	支持	UE 要求支持 Observe/Cancel observe /Notify
Resource Model（资源模型）	支持	资源模型中的object和resource都定义在IPSO Smart Object标准文档，如果无法找到对应的定义，需要向平台申请扩展的资源模型定义
Data Format（数据传输格式）	支持	见Data Format表
Transport Layer Bingding & Encoding（传输层模式绑定与编码）	支持	UE要求支持U（UDP）模式

数据传输的类型如下表：

表 8-2 Data Format 表

Data Format	IANA Media Type	Numeric Content-Formats [CoAP]
Plain Text	text/plain	0
Opaque	application/octet-stream	42
TLV	application/vnd.oma.lwm2m+tlv	11542

TLV示例如下：

08 00 13

C1 01 01

C4 02 3F 80 00 00

C1 03 01

C5 04 54 65 73 74 31

TLV即Type-Length-Value格式，其中Type字段表示报文类型，Length字段表示报文长度，Value字段表示报文内容。

具体定义如下：

表 8-3 TLV 格式定义

字段	格式与长度	说明
Type	8bit，其中第7位为最高位，第0位为最低位	第 7 和第 6 位表征 ID 的类型：00 表示 Object Instance，11表示Resource，01、10暂未使用。 第5位表示ID的长度：0表示ID为8bit，1表示ID为16bit。

		第4和第3为表征Length的类型：00表示没有Length字段，第2-第0位即为内容长度；01表示Length字段为8bit，此时应忽略Type字段第2-第0位；10表示Length字段为16bit，此时应忽略Type字段第2-第0位；11表示Length字段为24bit，此时应忽略Type字段第2-第0位。
ID	8bit或16bit无符号整型值	Object Instance或Resource的ID
Length	与Type字段中长度部分对应的0-24bit无符号整型值	内容字段的长度
Value	报文内容	

字段解析如下表：

表 8-4 TLV 示例解析

Type	ID	Length	Value	说明
08(0b00 00 10 00)	0x00	0x13(19字节)	以下四行内容	Object Instance
C1(0b11 00 00 01)	0x01	-	0x01	Resource
C4(0b11 00 01 00)	0x02	-	0x3F 80 00 00	Resource
C1(0b11 00 00 01)	0x03	-	0x01	Resource
C5(0b11 00 01 01)	0x04	-	0x54 65 73 74 31	Resource

UE读取信息上报的可以是一个Object Instance或者一个Resource，平台侧下发的写消息是一个或者多个Resource。

8.2 Bootstrap 流程及要求

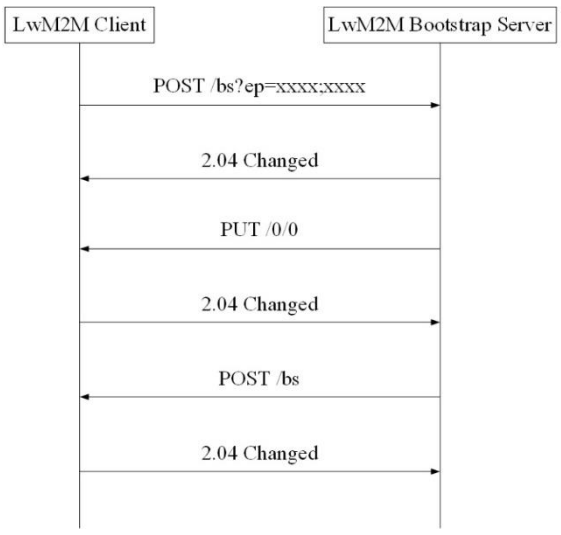


图8-1 Bootstrap流程

如图8-1所示，平台关于NB-IoT的bootstrap流程主要有设备上传bootstrap request请求，bootstrap server下发bootstrap write请求，bootstrap server下发Bootstrap Finish请求。下面是对三个请求接口的说明。

8.2.1 Bootstrap Request

[需求编号]：TS-UE-SUITE-PFRQ-00001

[需求描述]：

请求的主要作用为设备连接Bootstrap Server，请求Bootstrap Server分配接入机地址。

表 8-5 Bootstrap Request 接口定义

操作	Bootstrap request
LwM2M参数	/bs?ep={endpoint name}
参数说明	{endpoint name}：必选，设备的endpoint name（endpointname 字段包含两部分内容，前面字符为UE的IMEI号码，后面为UE的IMSI的号码，两部分内容由特殊的分隔符分开，比如“；”）
CoAP-Method	POST
CoAP-Option	Option 1：Uri-Path（11）：bs，说明：括号里面的为Option编号 Option 2：Uri-Query（15）：{endpoint name}
CoAP-payload	
Success	2.04 Changed
Failure	4.00 Bad Request, 4.15 Unsupported content format

[需求澄清]：无

8.2.2 Bootstrap Write

[需求编号]：TS-UE-SUITE-PFRQ-00002

[需求描述]：

该接口由Bootstrap Server发起，用于Bootstrap服务器分配一个接入机地址，写入设备。写入资源路径为/0/0的Security资源模型中。Security Object的resource id为0的资源用于承载接入机地址。

表 8-6 Bootstrap Write 接口定义

操作	Bootstrap write
LwM2M参数	/{Object ID}/{Object Instance ID}/{Resource ID}
参数说明	{Object ID}：必选，标识Object {Object Instance ID}：可选，标识Instance，若未携带该参数，则payload中必须指定Object Instance ID {Resource ID}：可选，标识Resource，若未携带该参数，则payload中必须指定Resource ID
CoAP-Method	PUT
CoAP-Option	Option 1：Uri-Path（11）：{Object ID} Option 2：Uri-Path（11）：{Object Instance ID}

	Option 3：Uri-Path（11）：{Resource ID} Option 4：Content-Format（12）：格式为Content Format表中的几种格式，如application/vnd.oma.lwm2m+tlv
CoAP-payload	数据格式为Option4中指定的数据格式
Success	2.04 Changed
Failure	4.00 Bad Request

资源模型简要列表如下：

Resource definitions

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
0	LWM2M Server URI		Single	Mandatory	String	0-255 bytes		Uniquely identifies the LWM2M Server or LWM2M Bootstrap Server, and is in the form: "coaps: //host:port", where host is an IP address or FQDN, and port is the UDP port of the Server.
1	Bootstrap Server		Single	Mandatory	Boolean			Determines if the current instance concerns a LWM2M Bootstrap Server (true) or a standard LWM2M Server (false)
2	Security Mode		Single	Mandatory	Integer	0-3		Determines which UDP payload security mode is used 0: Pre-Shared Key mode 1: Raw Public Key mode 2: Certificate mode 3: NoSec mode

图8-2　Security Object资源模型图

[需求澄清]：无

8.2.3 Bootstrap Finish

[需求编号]：TS-UE-SUITE-PFRQ-00003

[需求描述]：

该接口由Bootstrap Server发起，用于通知设备整个bootstrap过程完成。

表 8-7　Bootstrap Finish 接口定义

操作	Bootstrap Finish
LwM2M参数	/bs
CoAP-Method	POST
CoAP-Option	Option 1：Uri-Path（11）：bs，说明：括号里面的为Option编号
CoAP-payload	
Success	2.04 Changed
Failure	4.00 Bad Request

[需求澄清]：无

8.3 设备注册及注销

8.3.1 设备注册

[需求编号]：TS-UE-SUITE-PFRQ-00004

[需求描述]：

UE在每次完成Bootstrap流程后向平台发起注册请求，流程如下：

1、UE向服务器端发送注册请求，服务器端地址为Bootstrap过程中由Bootstrap Server向UE写入的Security object中的resource id为0的资源内容，即接入机地址；

2、服务器端接收到UE的注册请求，获取到endpoint name，通过对endpoint进行鉴权，如果鉴权成功，则返回success，否则鉴权失败；

3、UE接入后服务器端会记录UE的状态信息（如生存时间），并设置UE的状态为ONLINE。

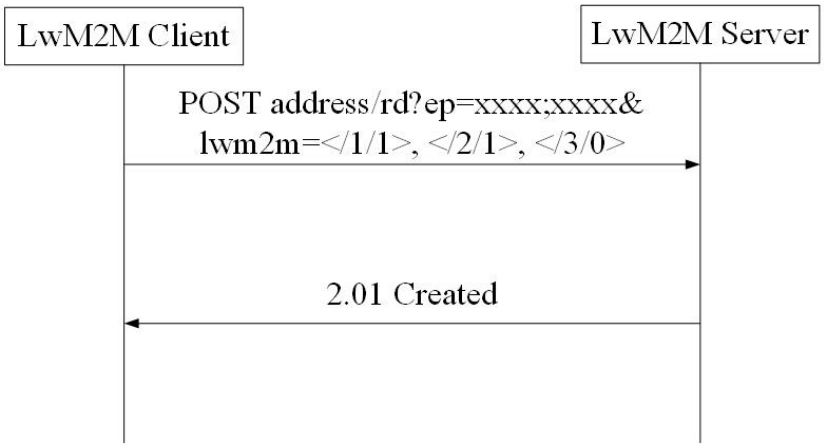


图8-3　设备注册

表 8-8 注册消息接口详细说明

操作	Register（设备注册消息）
LwM2M参数	address/rd?ep={endpoint name}&lt={lifetime} &lwm2m={version}&b={binding}&{ObjectLinks}
参数说明	{endpoint name}：必选，即为注册的endpoint name（endpointname字段包含两部分内容，前面字符为UE的IMEI号码，后面为UE的IMSI的号码，两部分内容由特殊的分隔符分开，比如“；”） {lifetime}：可选，默认值为86400 s（24小时） {version}：可选，默认为1.0 {binding}：可选，默认为U（即UDP） {ObjectLinks} ：必选，如</1/1>, </2/1>, </3/0>
CoAP-Method	POST
CoAP-Option	Option 1：Uri-Path（11）：rd，说明：括号里面的为Option编号 Option 2：Content-Format（12）：application/link-format Option 3：Uri-Query（15）：{binding} Option 4：Uri-Query（15）：{lifetime} Option 5：Uri-Query（15）：{endpoint name}

CoAP-payload	LwM2M协议相关参数加上{ObjectLinks}， 例如</>;rt="oma.lwm2m",</1/0>,</3/0>,</6/0>
Success	2.01 Created
Failure	4.00 Bad Request, 4.03 Forbidden

[需求澄清]：无

8.3.2 设备注册更新消息

[需求编号]：TS-UE-SUITE-PFRQ-00005

[需求描述]：

UE在lifetime到期前需要向平台侧发送注册更新消息，交互流程如下：

1、UE向平台侧发送Update消息，并携带UE注册时平台返回的location、相应的更新信息(如UElifetime)以及ObjectLinks等；

2、平台侧对消息进行鉴权，如果鉴权成功，则将对UE的相关注册信息进行更新并刷新lifetime。

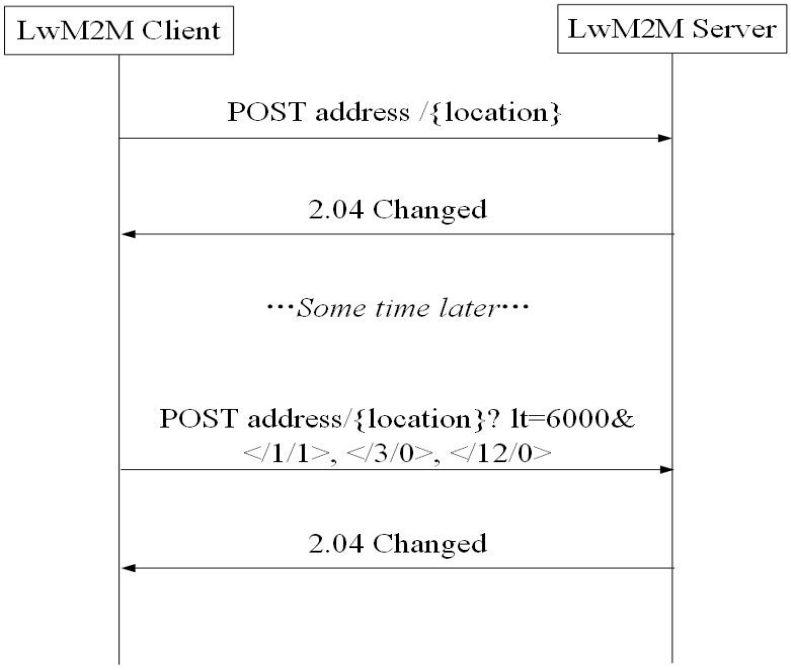


图8-4 设备注册更新消息

表 8-9 注册消息更新接口详细说明

操作	Update(设备注册更新)
LwM2M参数	address /{location} ? lt={lifetime}&lwm2m={version}&b={bingding}&{objectLinks}
参数说明	{lifetime}：可选，默认值为Register消息中的lifetime {version}：可选，默认为1.0 {bingding}：可选，默认为U（UDP） {ObjectLinks} ：可选，如</1/1>, </2/1>, </3/0>

CoAP-Method	POST
CoAP-Option	Option 1：Uri-Path（11）：{location} Option 3：Uri-Query（15）：{binding} Option 4：Uri-Query（15）：{lifetime}
CoAP-payload	lwm2m协议相关参数加上{ObjectLinks}， 例如</>;rt="oma.lwm2m",</1/0>,</3/0>,</6/0>
Success	2.04 Changed
Failure	4.00 Bad Request,4.04 Not Found

[需求澄清]：无

8.3.3 设备注销消息

[需求编号]：TS-UE-SUITE-PFRQ-00006

[需求描述]：

1、UE向平台侧发送De-register消息，并携带UE注册时平台返回的location；

2、平台根据location找到对应的设备，删除UE端的状态信息并设置UE状态为下线状态（OFFLINE）。

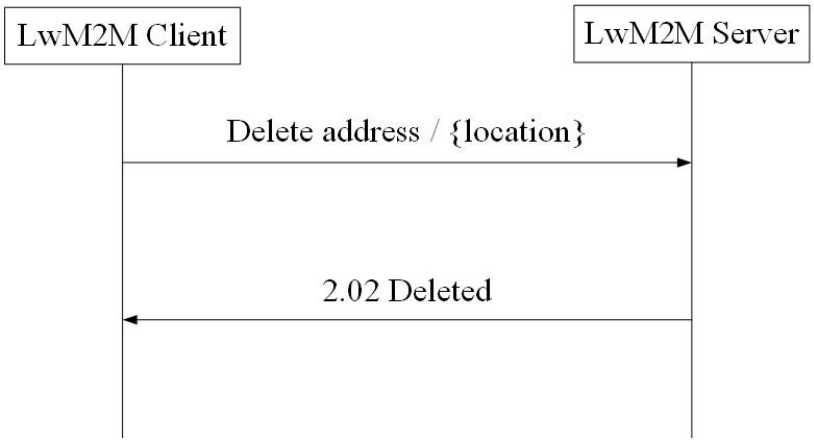


图 8-5 设备注销消息

表 8-10 注销注册消息接口详细说明

操作	De-register（设备注销消息）
LwM2M参数	address / {location}
CoAP-Method	DELETE
CoAP-Option	Option 1：Uri-Path（11）：{location}
CoAP-payload	
参数说明	{location}：必选，需要注销的设备；
Success	2.02 Deleted
Failure	4.00 Bad Request, 4.04 Not Found

[需求澄清]：无

8.4 消息观察、取消观察、消息上报、设定 Notify 策略

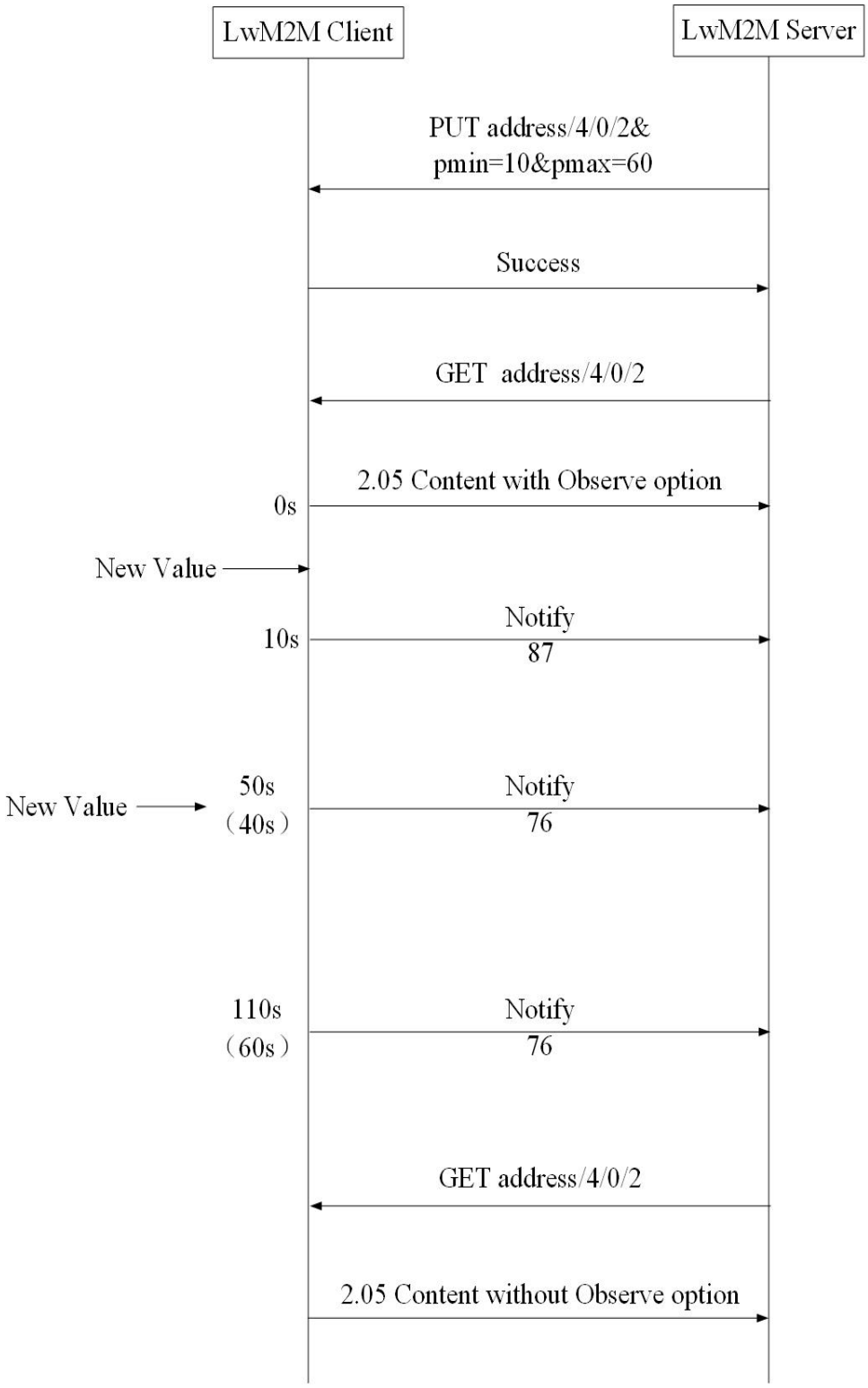


图8-6设定Notify策略、消息观察、取消观察、消息上报

8.4.1 设定 Notify 策略

[需求编号]：TS-UE-SUITE-PFRQ-00007

[需求描述]：

服务器端会主动向UE发送观察参数设置消息（Write Attributes）来配置观测资源的策略，UE收到该消息则返回Success。

表 8-11 设定 Notify 策略详细说明

操作	Write Attributes（）
LwM2M参数	address/{ObjectID}/{Object Instance ID}/{Resource ID}&pmin={Minimum Period}& pmax={Maximum Period}&gt={Greater Than}& lt={Less Than}& stp={Step}
参数说明	{Object ID}：必选，如UE上的传感器类型 {Object Instance ID}：可选，该类型传感器的编号 {Resource ID}：可选，该传感器的某种类型的数据，如温度的当前值，最大值等 {Minimum Period}：可选，两次上报的最小间隔 {Maximum Period}：可选，两次上报的最大间隔 {Greater Than}：可选，上报的数值需要大于的数值 {Less Than}：可选，上报的数值需要小于的数值 {Step}：可选，两次上报数据最小要求的值的变化
CoAP-Method	PUT
CoAP-Option	Option 1：Uri-Path（11）：{Object ID} Option 2：Uri-Path（11）：{Object Instance ID} Option 3：Uri-Path（11）：{Resource ID} Option 4：Uri-Query（15）：{Minimum Period} Option 5：Uri-Query（15）：{Maximum Period} Option 6：Uri-Query（15）：{Greater Than} Option 7：Uri-Query（15）：{Less Than} Option 8：Uri-Query（15）：{Step}
CoAP-payload	
Success	2.04 Changed
Failure	4.00 Bad Request, 4.04 Not Found, 4.01 Unauthorized, 4.05 Method Not Allowed

[需求澄清]：无

8.4.2 观察消息

[需求编号]：TS-UE-SUITE-PFRQ-00008

[需求描述]：

1、服务器端主动向UE发送观测消息（Observe）来观测资源列表中所有可观测的资源，UE收到该消息后，检查需要观测的资源是否可以被订阅且状态正常，如果是则返回Success；

2、UE在返回Success后，会按照数据上报规则，上传携带订阅资源具体数值的Notify消息（即相应观测的传感器数据值）；

表 8-12 观察消息详细说明

操作	Observe（观察消息）
LwM2M参数	address /{Object ID}/{Object Instance ID}/{Resource ID}
参数说明	{Object ID}：必选，如UE上的传感器类型 {Object Instance ID}：可选，该类型传感器的编号 {Resource ID}：可选，该传感器的某种类型的数据，如温度的当前值等
CoAP-Method	GET
CoAP-Option	Option 1：Observe（6）：0（订阅） Option 2：Uri-Path（11）：{Object ID} Option 3：Uri-Path（11）：{Object Instance ID} Option 4：Uri-Path（11）：{Resource ID} Option 5：Accept（17）：Content Format表中的类型，如application/vnd.oma.lwm2m+tlv参数表示订阅后数据上报的格式。
CoAP-payload	
Success	2.05 Content with Observe option
Failure	4.00 Bad Request, 4.04 Not Found, 4.01 Unauthorized, 4.05 Method Not Allowed

[需求澄清]：无

8.4.3 取消观察消息

[需求编号]：TS-UE-SUITE-PFRQ-00009

[需求描述]：

1、服务器端会主动向UE发送取消观察消息（Cancel Observe）来取消对资源列表中资源的观察，例如Cancel Observe /1/0/5，UE收到该消息则返回Success；

2、UE在返回Success后，会停止上报/1/0/5的Notify消息。

表 8-14 取消观察消息详细说明

操作	Cancel Observe（取消观察消息）
LwM2M参数	address /{Object ID}/{Object Instance ID}/{Resource ID}
参数说明	{Object ID}：必选，如UE上的传感器类型 {Object Instance ID}：可选，该类型传感器的编号 {Resource ID}：可选，该传感器的某种类型的数据，如温度的当前值，最大值等
CoAP-Method	GET
CoAP-Option	Option 1：Observe（6）：1（取消订阅） Option 2：Uri-Path（11）：{Object ID} Option 3：Uri-Path（11）：{Object Instance ID} Option 3：Uri-Path（11）：{Resource ID}

表 8-14 取消观察消息详细说明（续）

操作	Cancel Observe（取消观察消息）
CoAP-payload	
Success	2.05 Content without Observe option
Failure	4.00 Bad Request, 4.04 Not Found, 4.01 Unauthorized, 4.05 Method Not Allowed
LwM2M参数	address /{Object ID}/{Object Instance ID}/{Resource ID}

[需求澄清]：无

8.4.4 数据上报

[需求编号]：TS-UE-SUITE-PFRQ-00010

[需求描述]：

在满足观察规则的情况下，上报服务侧需要的信息，直到平台侧取消观察。

表 8-15 数据上报详细说明

操作	Notify（数据上报）
LwM2M参数	
CoAP-Method	Asynchronous Response
CoAP-Option	Option 1：Observe（6）：2（数据上报） Option 2：Content-Format（12），即为Observe Accept Option中设置的数据格式
CoAP-payload	{newValue}：上报的数据值
Success	2.05 Content（with Values）
Failure	

[需求澄清]：无

8.5 设备管理消息

UE资源操作的前提是UE和服务器端保持连接，即lifetime未过期。UE上的资源都具有一定权限，比如可读（R），可写（W），可执行（E）或其组合，如果对一个不支持R或者不支

持W的资源进行读写操作则会返回Failure消息到平台侧。

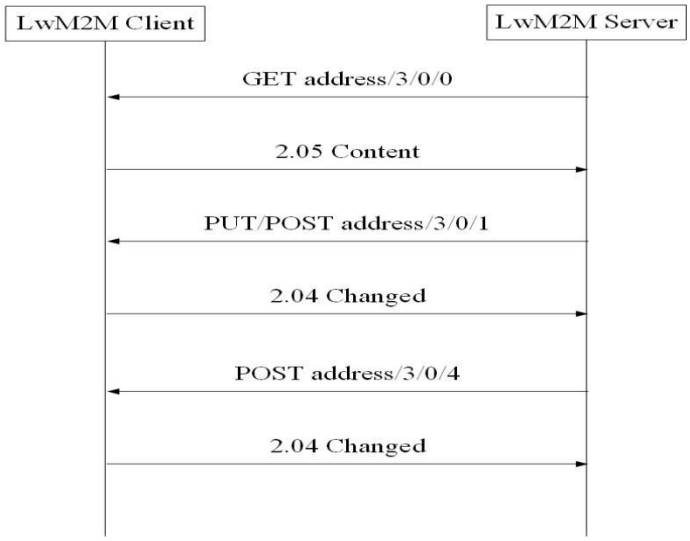


图 8-6 设备管理消息（Read/Write/Execute）

8.5.1 读取资源

[需求编号]：TS-UE-SUITE-PFRQ-00011

[需求描述]：

1、服务器端向UE发送Read消息

2、UE收到Read消息后，基础通信套件获取相应的资源属性值，根据Read消息的格式，将读取到的资源值返回给平台侧，并携带返回码Success。

表 8-16 读取资源详细说明

操作	Read（资源读取）
LwM2M参数	address /{Object ID}/{Object Instance ID}/{Resource ID}
参数说明	{Object ID}：必选，如UE上的传感器类型 {Object Instance ID}：可选，该类型传感器的编号 {Resource ID}：可选，该传感器的某种类型的数据，如温度的当前值，最大值等
CoAP-Method	GET
CoAP-Option	Option 1：Uri-Path（11）：{Object ID} Option 2：Uri-Path（11）：{Object Instance ID} Option 3：Uri-Path（11）：{Resource ID} Option 4：Accept（17）：第1节中的Content Format表中的类型，如application/vnd.oma.lwm2m+tlv参数是表示读取数据的格式为tlv
CoAP-payload	
Success	2.05 Content
Failure	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed

[需求澄清]：无

8.5.2 写入资源

[需求编号]：TS-UE-SUITE-PFRQ-00012

[需求描述]：

1、服务器向UE发送Write消息，如Write /6/0/5 并在消息的负载中携带资源的属性值，资源属性值是单个的，具体根据Write消息的格式；

2、UE接收到Write消息后，查看对应的资源是否可写，如果可写则将数据写入资源中，然后返回Success给平台侧，否则返回Failure消息。

表 8-17 写入资源详细说明

操作	Write（资源写入）
LwM2M参数	address /{Object ID}/{Object Instance ID}/{Resource ID}/{NewValue}
参数说明	{Object ID}：必选，如UE上的传感器类型 {Object Instance ID}：必选，该类型传感器的编号 {Resource ID}：可选，该传感器的某种类型的数据，如温度的当前值等 {NewValue}：必选，写入的资源属性值
CoAP-Method	PUT/POST
CoAP-Option	Option 1：Uri-Path（11）：{Object ID} Option 2：Uri-Path（11）：{Object Instance ID} Option 3：Uri-Path（11）：{Resource ID} Option 4：Content-Format（12）：格式为Content Format表中的几种格式，如application/vnd.oma.lwm2m+tlv
CoAP-payload	Resource ID和Value都封装成TLV格式放在payload里
Success	2.04 Changed
Failure	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed

[需求澄清]：对于同一个资源需要写入的数据超过长度限制，基础通信套件会将该写入的数据拆分为多个消息发送给MCU，如果需要写入的资源中包含一个或者以上的写入数据超过长度限制，那么则无法一次批量写入。即超过长度限制的资源只支持单独写操作。

8.5.3 执行资源

[需求编号]：TS-UE-SUITE-PFRQ-00013

[需求描述]：

1、服务器向UE发送Execute消息，如Execute /6/0/5 并在消息的负载中携带对资源的具体操作，比如，重启（Reboot），重置（Reset）等，命令格式为字符串形式；

2、当UE接收到Execute消息后，查看对应的资源是否可执行，解析相应的命令串，如果执行成则返回Success，否则返回Failure消息到平台侧。

表 8-18 执行资源详细说明

操作	Observe（资源执行）
LwM2M参数	address /{Object ID}/{Object Instance ID}/{Resource ID}/{Arguments}
参数说明	{Object ID}：必选，如UE上的传感器类型 {Object Instance ID}：必选，该类型传感器的编号 {Resource ID}：必选，该传感器的某种类型的数据，如温度的当前值，最大值等 {Arguments}：可选，待执行的命令，如重启，关机等
CoAP-Method	POST
CoAP-Option	Option 1：Uri-Path（11）：{Object ID} Option 2：Uri-Path（11）：{Object Instance ID} Option 3：Uri-Path（11）：{Resource ID}
CoAP-payload	{Arguments}，格式为Content Format表中的text/plain格式
Success	2.04 Changed
Failure	4.00 Bad Request, 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed

[需求澄清]：无

8.5.4 资源发现

[需求编号]：TS-UE-SUITE-PFRQ-00014

[需求描述]：

1、服务器向UE发送Discover消息；

2、当UE接收到Discover消息后，根据请求的URI，返回对应的资源URI，如</6/0/1>,</6/0/2>。

表 8-19 资源发现详细说明

操作	Discover（资源发现）
LwM2M参数	address /{Object ID}/{Object Instance ID}/{Resource ID}
参数说明	{ Object ID }：必选，如设备上的传感器类型； { Object Instance ID }：可选，该类型传感器的编号； {Resource ID}：可选，该传感器的某种类型的数据，如温度的当前值，最大值等
CoAP-Method	POST
CoAP-Option	Option 1：Uri-Path（11）：{Object ID} Option 2：Uri-Path（11）：{Object Instance ID} Option 3：Uri-Path（11）：{Resource ID}

表 8-19 资源发现详细说明（续）

操作	Discover（资源发现）
CoAP-payload	{Arguments}，格式为Content Format表中的text/plain格式
Success	2.05Content
Failure	4.00 Bad Request， 4.01 Unauthorized, 4.04 Not Found, 4.05 Method Not Allowed
LwM2M参数	address /{Object ID}/{Object Instance ID}/{Resource ID}

[需求澄清]：无

9.底层系统接口

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。

[需求编号]：TS-UE-SUITE-SYSRQ-00001

[需求描述]：设置系统初始化参数

函数原型：

cis_ret_t cissys_init(constcis_cfg_sys_t* config);

功能：用于系统初始化

表 9-1 cissys_init 参数说明

参数	类型	参数含义
config	const cis_cfg_sys_t*	系统配置

表 9-2cissys_init 返回值说明

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

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00002

[需求描述]：获取系统时间戳

函数原型：

uint32_t cissys_gettime();

功能：用于获取系统时间戳

参数：无

表 9-3 cissys_ gettime 返回值说明

返回值类型	含义
uint32_t	获取系统时间毫秒数

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00003
[需求描述]：系统延时接口
函数原型：
void cissys_sleepms(uint32_t ms);
功能：系统延时函数

表 9-4 cissys_ sleepms 参数说明

参数	类型	参数含义
ms	uint32_t	系统延时毫秒数

返回值：无
[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00004
[需求描述]：写入系统日志接口
函数原型：
void cissys_logwrite(uint8_t* buffer,uint32_t length);
功能：用于系统日志写入

表 9-5 cissys_ logwrite 参数说明

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
[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00005
[需求描述]：内存分配接口
函数原型：
void * cissys_malloc(size_t length);
功能：用于系统分配内存

表 9-6 cissys_ malloc 参数说明

参数	类型	参数含义
length	size_t	申请内存长度

表 9-7 cissys_ malloc 返回值说明

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

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00006
[需求描述]：内存释放接口
函数原型：
void cissys_free(void * buffer);
功能：用于内存释放

表 9-8 cissys_ free 参数说明

参数	类型	参数含义
buffer	void_t*	要释放的内存

返回值：无
[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00007
[需求描述]：内存复制接口
函数原型：
void * cissys_memcpy (void * dst, const void * src, size_t n);
功能：用于内存复制

表 9-9 cissys_ memcpy 参数说明

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

表 9-10 cissys_ memcpy 返回值说明

返回值类型	返回值含义
void *	指向dst的指针

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00008
[需求描述]：内存移动接口
函数原型：
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

表 9-11 cissys_ memmove 参数说明

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

表 9-12 cissys_ memmove 返回值说明

返回值类型	返回值含义
void *	指向dst的指针

[需求澄清]：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。

[需求编号]：TS-UE-SUITE-SYSRQ-00009
[需求描述]：内存比较接口
函数原型：
int cissys_memcmp (const void * s1, const void * s2, size_t n);
功能：用于内存比较

表 9-13 cissys_ memcmp 参数说明

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

表 9-14 cissys_ memcmp 返回值说明

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。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-000010
[需求描述]：内存初始化接口
函数原型：
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

表 9-15 cissys_ memset 参数说明

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

表 9-16 cissys_ memset 返回值说明

返回值类型	返回值含义
void *	指向s的指针

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00011
[需求描述]：系统故障处理接口
函数原型：
void cissys_fault(uint16_t id);
功能：通知系统发生故障，需要进行处理

表 9-17 cissys_ fault 参数说明

参数	类型	参数含义
id	uint16_t	系统错误类型

id定义：
#define UNKNOW_ERROR 0 //unknown error
#define MALLOC_EEROR 1 //malloc error
#define NET_ERROR 2 //net error
返回值：无
[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00012
[需求描述]：支持系统断言接口
函数原型：
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

表 9-18 cissys_ assert 参数说明

参数	类型	参数含义
flag	bool	断言条件

返回值：无
[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00013
[需求描述]：支持系统随机数生成接口
函数原型：uint32_t cissys_rand();
功能：用于系统生成随机数
参数：无

表 9-19 cissys_ rand 参数说明

返回值类型	含义
uint32_t	生成的随机整数

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00014
[需求描述]：支持获取IMEI号码接口
函数原型：
uint8_t cissys_getIMEI(uint8_t* buffer, uint32_t len);
功能：用于获取IMEI号码

表 9-20 cissys_ getIMEI 参数说明

参数	类型	参数含义
buffer	uint8_t*	IMEI号码
len	uint32_t	IMEI长度

表 9-21 cissys_ getIMEI 返回值说明

返回值类型	含义
uint8_t	IMEI的有效长度

[需求澄清]： 无

[需求编号]：TS-UE-SUITE-SYSRQ-00015
[需求描述]：支持获取IMSI号码接口
函数原型：
uint8_t cissys_getIMSI(uint8_t* buffer, uint32_t len);
功能：用于获取IMSI号码

表 9-22 cissys_ getIMSI 参数说明

参数	类型	参数含义
buffer	uint8_t*	IMSI号码
len	uint32_t	IMSI长度

表 9-23 cissys_ getIMSI 返回值说明

返回值类型	含义
uint8_t	IMSI的有效长度

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00016
[需求描述]：锁定当前操作
函数原型：
cis_ret_t cissys_lock(void* mutex,uint32_t waitms);
功能：用于锁定当前操作

表 9-24 cissys_ getIMSI 参数说明

参数	类型	参数含义
mutex	void*	信号量
waitms	uint32_t	阻塞等待时间

表 9-25 cissys_lock 返回值说明

返回值类型	含义
CIS_RET_TIMEOUT	阻塞超时
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00017
[需求描述]：释放锁定的信号量
函数原型：
void cissys_unlock (void * mutex);
功能：用于释放当前锁定的资源

表 9-26 cissys_unlock 参数说明

参数	类型	参数含义
mutex	void*	信号量

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00018
[需求描述]：创建信号量
函数原型：
void cissys_lockcreate(void** mutex);
功能：用于创建一个信号量

表 9-27 cissys_lockcreate 参数说明

参数	类型	参数含义
mutex	void**	信号量

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00019
[需求描述]：销毁信号量
函数原型：
void cissys_lockdestory(void* mutex);
功能：用于销毁信号量

表 9-28 cissys_lockdestory 参数说明

参数	类型	参数含义
mutex	void*	信号量

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00020
[需求描述]：保存缓冲区数据到FLASH内存区，保证休眠或断电情况下数据不丢失。

函数原型：
bool cissys_save(uint8_t* buffer,uint32_t length);
功能：用于保存缓冲区数据

表 9-29 cissys_save 参数说明

参数	类型	参数含义
buffer	uint8*	需要存储的数据
length	uint32_t	长度

表 9-30 cissys_ save 返回值说明

返回值类型	含义
bool	true：保存操作成功；false：保存操作失败

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00021
[需求描述]：读取保存的缓冲区数据
函数原型：
bool cissys_load(uint8_t* buffer,uint32_t length);
功能：用于读取保存的缓冲区数据

表 9-31 cissys_load 参数说明

参数	类型	参数含义
buffer	uint8*	需要存储的数据
length	uint32_t	长度

表 9-32 cissys_ load 返回值说明

返回值类型	含义
bool	true：读取操作成功；false：读取操作失败

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00022
[需求描述]：支持获取用户配置的DTLS PSK
函数原型：
uint8_t cissys_getPSK(char* buffer,uint32_t maxlen)
功能：用于读取保存在文件中的PSK

表 9-33 cissys_getPSK 参数说明

参数	类型	参数含义
buffer	uint8_t*	返回的PSK的缓存指针
len	uint32_t	PSK的最大长度

表 9-34 cissys_getPSK 返回值说明

返回值类型	含义
uint8_t	PSK的有效长度

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00023
[需求描述]：获取网络制式
函数原型：
int8_t cissys_getBearer();
功能：用于读取当前网络的网络制式

表 9-35 cissys_getBearer 返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无
[需求编号]：TS-UE-SUITE-SYSRQ-00024
[需求描述]：获取当前网络链路使用率
函数原型：
int8_t cissys_getUtilization()
功能：用于当前网络链路使用率

表 9-36 cissys_getUtilization 返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00025
[需求描述]：获取信号强度质量
函数原型：
int8_t cissys_getCSQ()
功能：获取信号强度质量

表 9-37 cissys_getCSQ 返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00026
[需求描述]：获取覆盖等级
函数原型：
int8_t　　cissys_getECL()
功能：获取覆盖等级

表 9-38 cissys_getECL 返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00027
[需求描述]：获取当前PLMN
函数原型：
int8_t　　cissys_getPLMN()
功能：获取当前PLMN

表 9-39 cissys_getPLMN()返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00028
[需求描述]：获取注册小区下行频点
函数原型：
int8_t　　cissys_getEarfcn()
功能：获取小区下行频点

表 9-40 cissys_getEarfcn ()返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00029
[需求描述]：获取服务小区物理小区识别码
函数原型：
int16_t　　cissys_getPCI()
功能：获取服务小区物理小区识别码

表 9-41 cissys_getPCI()返回值说明

返回值类型	含义
int16_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00030
[需求描述]：获取最强同频邻小区物理小区识别码

函数原型：

int16_t　　cissys_getPCIFrist()

功能：获取最强同频邻小区物理小区识别码

表 9-42 cissys_getPCIFrist()返回值说明

返回值类型	含义
int16_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00031

[需求描述]：获取次强同频邻小区物理小区识别码

函数原型：

int16_t　　cissys_getPCISecond()

功能：获取次强同频邻小区物理小区识别码

表 9-43 cissys_getPCISecond()返回值说明

返回值类型	含义
int16_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00032

[需求描述]：获取最强同频邻小区参考信号接收功率

函数原型：

int8_t　　cissys_getRSRPFrist()

功能：获取最强同频邻小区参考信号接收功率

表 9-44 cissys_getRSRPFrist()返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00033

[需求描述]：获取次强同频邻小区参考信号接收功率

函数原型：

int8_t　　cissys_getRSRPSecond()

功能：获取次强同频邻小区参考信号接收功率

表 9-45 cissys_getRSRPSecond()返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00034

[需求描述]：获取当前设备参考信号接收功率

函数原型：

int8_t　　cissys_getRSRP()

功能：获取当前设备参考信号接收功率

表 9-46 cissys_getRSRP()返回值说明

返回值类型	含义
int8_t	

[需求澄清]：无

[需求编号]：TS-UE-SUITE-SYSRQ-00035
[需求描述]：获取设备GPS纬度信息
函数原型：
int8_t cissys_getLatitude()
功能：获取设备GPS纬度信息

表 9-47 cissys_getLatitude()返回值说明

返回值类型	含义
int8_t	

[需求澄清]：可选实现

[需求编号]：TS-UE-SUITE-SYSRQ-00036
[需求描述]：获取设备GPS经度信息
函数原型：
int8_t cissys_getLongitude()
功能：获取设备GPS纬度信息

表 9-48 cissys_getLongitude ()返回值说明

返回值类型	含义
int8_t	

[需求澄清]：可选实现

[需求编号]：TS-UE-SUITE-SYSRQ-00037
[需求描述]：获取设备GPS高度信息
函数原型：
int8_t cissys_getAltitude()
功能：获取设备GPS高度信息

表 9-49 cissys_getAltitude ()返回值说明

返回值类型	含义
int8_t	

[需求澄清]：可选实现

10. 底层网络接口

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。

10.1 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;
```

底层网络需要通知基础通信套件事件时，调用onEvent通知。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);
```

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_connected,    //网络连接成功事件
    cisnet_event_disconnected,  //网络连接断开事件
}cisnet_event_t;
```

10.2 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;
```

10.3 底层网络接口

[需求编号]：TS-UE-SUITE-NWRQ-00001
[需求描述]：网络初始化
函数原型：
cis_ret_t cisnet_init (const cisnet_config_t* config, cisnet_callback_t cb);
功能：用于网络初始化

表 10-2cisnet_init 参数说明

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，具体参见10.1节

表 10-3 cisnet_init 返回值说明

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-NWRQ-00002
[需求描述]：支持创建网络接口
函数原型：
cis_ret_tcisnet_create(cisnet_t* ctx, const char* host);
功能：用于创建网络

表 10-4 cisnet_create 参数说明

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

表 10-5 cisnet_create 返回值说明

返回值	返回值含义
CIS_RET_ERROR	失败

返回值	返回值含义
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-NWRQ-00003
[需求描述]：支持销毁网络接口
函数原型：
cis_ret_tcisnet_destroy(cisnet_t ctx);
功能：用于销毁网络

表 10-6 cisnet_destroy 参数说明

参数	类型	参数含义
ctx	cisnet_t	网络上下文

表 10-7 cisnet_destroy 返回值说明

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-NWRQ-00004
[需求描述]：支持网络连接接口，如果是UDP承载，该接口为空函数
函数原型：
cis_ret_tcisnet_connect(cisnet_t ctx);
功能：用于网络连接

表 10-8 cisnet_connect 参数说明

参数	类型	参数含义
ctx	cisnet_t	网络上下文

表 10-9 cisnet_connect 返回值说明

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-NWRQ-00005
[需求描述]：支持断开网络接口，如果是UDP承载，该接口为空函数

函数原型：
cis_ret_tcisnet_disconnect(cisnet_t ctx);
功能：用于断开网络

表 10-10 cisnet_disconnect 参数说明

参数	类型	参数含义
ctx	cisnet_t	网络上下文

表 10-11 cisnet_disconnect 返回值说明

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-NWRQ-00006
[需求描述]：支持网络写入接口
cis_ret_tcisnet_write(cisnet_t ctx,const uint8_t* buffer,uint32_t length);
功能：用于网络写入缓冲数据

表 10-12 cisnet_write 参数说明

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

表 10-13 cisnet_write 返回值说明

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-NWRQ-00007
[需求描述]：支持网络读取接口
函数原型：
cis_ret_tcisnet_read(cisnet_t ctx, uint8_t** buffer, uint32_t *length);
功能：用于网络读取缓冲数据

表 10-14 cisnet_read 参数说明

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

表 10-15 cisnet_read 返回值说明

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-NWRQ-00008
[需求描述]：支持缓冲区释放接口
函数原型：
cis_ret_tcisnet_free(cisnet_t ctx, uint8_t* buffer, uint32_t length);
功能：用于释放缓冲区

表 10-16 cisnet_free 参数说明

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

表 10-17 cisnet_free 返回值说明

返回值	返回值含义
CIS_RET_ERROR	失败
CIS_RET_OK	成功

[需求澄清]：无

[需求编号]：TS-UE-SUITE-NWRQ-00009
[需求描述]：获取网络附着状态
函数原型：
bool cisnet_attached_state();
功能：用于获取网络附着状态

表 10-18cisnet_free 返回值说明

返回值	返回值含义
true	已附着

返回值	返回值含义
false	未附着

[需求澄清]：无

11. 集成要求

基础通信套件的实现包括两种形式：

1、集成中国移动SDK来完成基础通信套件的功能。

根据用户使用方式,可以分为两种场景：一种是APP和模组或芯片集成在同一个应用核，如图11-1所示；另一种是APP和模组或芯片集成在不同MCU，如图11-2所示。

为了满足这两种使用场景，采用集成中国移动SDK的的芯片和模组厂家，需要实现第7章的AT指令集（即下面场景图中的AT指令接口②），第9章的底层系统抽象接口（下面场景图中的系统接口③），第10章的底层网络层的抽象接口（下面场景图中的网络接口④）；

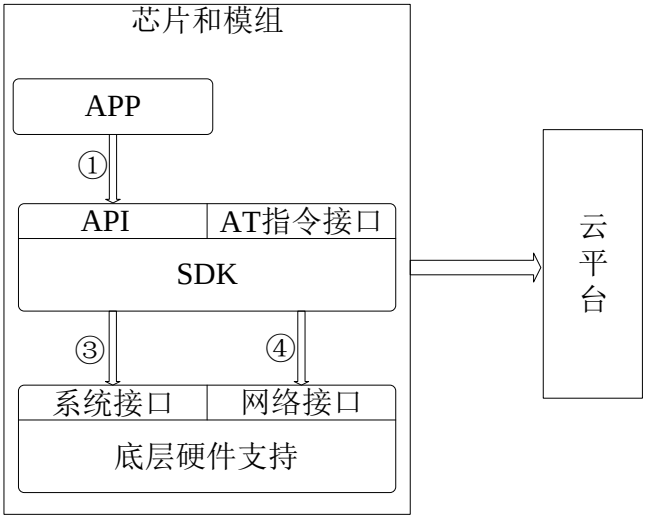


图11-1　集成中国移动提供的SDK使用场景一

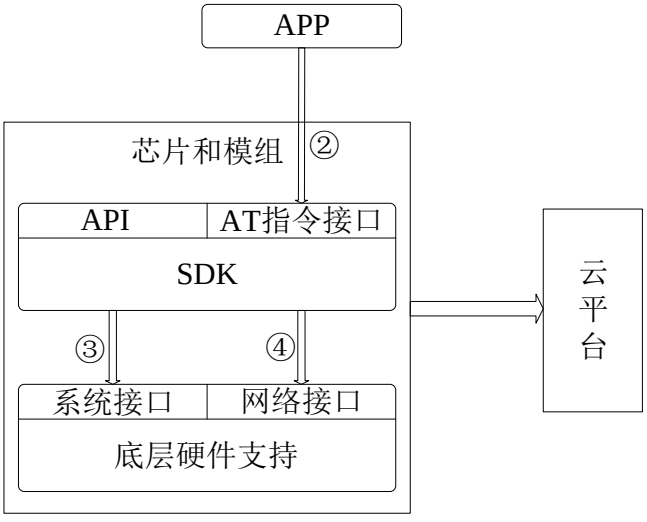


图11-2　集成中国移动提供的SDK使用场景二

2、芯片和模组厂家自行完成基础通信套件的功能。

根据用户使用方式，同样分为两种场景：一种是APP和模组或芯片集成在同一MCU，如图11-3所示；另一种是APP和模组或芯片集成在不同MCU，如图11-4所示。

为了满足这两种使用场景，使用该方式的芯片和模组厂家需要实现第6章的对API接口抽象层（下面场景图中的能力开放API①），第7章的AT指令集（即下面场景图中的AT指令接口②），第8章的UE与平台通信接口（下面场景图中的⑤）。

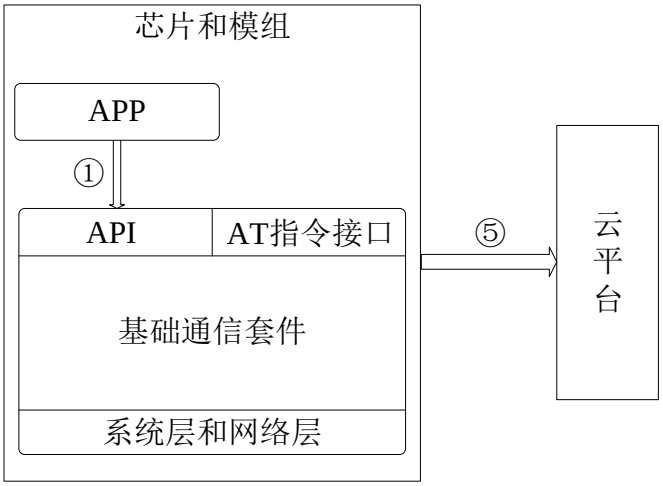


图11-3　厂家自行实现基础通信套件的使用场景一

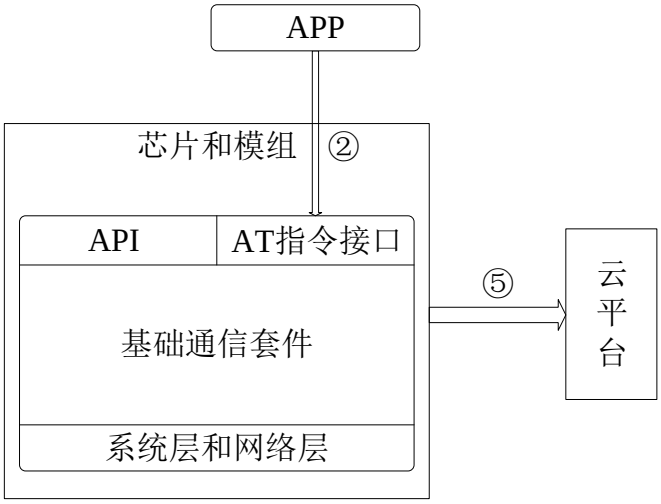


图11-4　厂家自行实现基础通信套件的使用场景二

12. 硬件资源的要求

[需求编号]：TS-UE-SUITE-HWRQ-00000

[需求描述]：基础通信套件要求提供如下的硬件配置：

处理速率要求：处理器需要最低达到32MHz。

Flash容量要求：如果UE不支持第15章的安全要求，Flash需要最低达到60KB；如果UE支持第15章的安全要求，Flash需要最低达到100KB。

RAM容量要求：如果UE不支持第15章的安全要求，RAM需要最低达到5KB；如果UE支持第15章的安全要求，Flash需要最低达到10KB。

[需求澄清]：无

13.在线升级要求

[需求编号]：TS-UE-SUITE-FOTA-00001

[需求描述]：FOTA用于设备模组的远程升级，SDK的FOTA基于LwM2M协议，只用于建立服务器和UE之间的下载通道和进行数据包传输。设计遵循LwM2M对于FOTA的定义及流程，并扩展了device 对象和Firmware 对象，FOTA分为两个阶段：数据包下载和设备升级。其中下载流程使用LwM2M定义的Object来进行管理。当前使用包push方式来下发数据包。即服务器主动推送 /5/0/0 资源的写操作。升级流程由系统提供能力和相应的事件回调。

FOTA的升级流程参见附录E。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00002

[需求描述]：SOTA用于设备MCU的远程升级，SDK的SOTA能力基于自身的FOTA升级协议，如果平台选择对MCU进行升级，则SDK会向MCU触发相应的升级事件，并将升级报文通过AT接口的格式传递至MCU，MCU需要调用AT接口完成和SDK的升级交互。

SOTA的升级流程参见附录F。

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00003

[需求描述]：终端的空中写卡，指通过空中升级的方式完成嵌入式SIM卡（eSIM）从临时码号至正式码号的变更，即通过OneNET以及网络信令的交互完成用户卡的个人信息变更。

OTA空中写卡的流程参见附录F。

[需求澄清]：无

13.1FOTA/SOTA 设备接口需求

升级状态枚举用于定义升级过程的状态：

```
typedef enum{
    FOTA_STATE_IDIL,//空闲状态
    FOTA_STATE_DOWNLOADING,//下载状态
    FOTA_STATE_DOWNLOADED,//下载完成状态
    FOTA_STATE_UPDATING,//升级状态
}cis_fw_state;
```

升级结果定义枚举用于定义升级过程中的各种结果：

```
typedef enum
{
    FOTA_RESULT_INIT,                   //0 初始结果
    FOTA_RESULT_SUCCESS,                //1 升级成功
    FOTA_RESULT_NOFREE,                 //2 空间不足
```

```
    FOTA_RESULT_OVERFLOW,              //3 下载过程中内存溢出
    FOTA_RESULT_DISCONNECT,            //4 下载过程中链接断开
    FOTA_RESULT_CHECKFAIL,             //5 校验失败
    FOTA_RESULT_NOSUPPORT,             //6 不支持的数据包
    FOTA_RESULT_URIINVALID,            //7 无效的uri
    FOTA_RESULT_UPDATEFAIL,            //8 升级失败
    FOTA_RESULT_PROTOCOLFAIL          //9 不支持的协议
}cissys_fw_result_type_t;
```

回调事件枚举用于定义底层接口操作的返回值：

```
typedef enum
{
    cissys_event_unknow = 0,//未知
    cissys_event_fw_erase_success,          //擦除成功
    cissys_event_fw_erase_fail,  //擦除失败
    cissys_event_write_success,             //写入成功
    cissys_event_write_fail,//写入失败
    cissys_event_fw_validate_success,        //校验成功
    cissys_event_fw_validate_fail, //校验失败
    cissys_event_fw_update_success,    //升级成功
    cissys_event_fw_update_fail,//升级失败
}cissys_event_t;
```

回调函数定义：

```
typedef    cis_ret_t    (*cissys_event_callback_t)(cissys_event_t    id,void*
param,void* userData);
typedef struct st_cissys_callback
{
    void* userData;
    cissys_event_callback_t onEvent;
}cissys_callback_t;
```

该函数用于在基础通信套件对底层设备的异步调用结果返回。

升级信息结构体：

```
typedef struct cis_fw_context{
    uint8_t state;  // 升级状态
    uint8_t result;  // 升级结果
    int savebytes;//已保存的升级包的字节数
}cis_fw_context_t;
```

13.2 FOTA/SOTA 系统适配接口定义:

[需求编号]：TS-UE-SUITE-FOTA-00001

[需求描述]：获取当前固件版本号
函数原型：
uint32_t　　cissys_getFwVersion(uint8_t** version);
功能：用于获取当前固件版本号

表 13-1 cissys_getFwVersion 返回值说明

返回值	返回值含义
0	获取失败
xx.xx	版本号

[需求澄清]：返回值为版本号长度，分为两部分，中间用.隔开，如1.9

[需求编号]：TS-UE-SUITE-FOTA-00002
[需求描述]：获取设备电量
函数原型：
uint32_t　　cissys_getBatteryLevel();
功能：用于获取设备电量

表 13-2 cissys_getBatteryLevel 返回值说明

返回值	返回值含义
0	获取失败
uint32_t	版本号

[需求澄清]：获取电量的百分比*100以后的数值，比如电量还剩30%，则返回30

[需求编号]：TS-UE-SUITE-FOTA-00003
[需求描述]：获取设备电压
函数原型：
uint32_t　　cissys_getFwBatteryVoltage();
功能：用于获取设备电压

表 13-3 cissys_getFwBatteryVoltage 返回值说明

返回值	返回值含义
0	获取失败
uint32_t	电压值

[需求澄清]：电压单位为mV

[需求编号]：TS-UE-SUITE-FOTA-00004
[需求描述]：获取小区设备号
函数原型：
uint32_t　　cissys_getCellId();
功能：用于获取小区设备号

表 13-4 cissys_getCellId 返回值说明

返回值	返回值含义
uint32_t	小区设备号

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00005
[需求描述]：获取信号强度
函数原型：
uint32_t　　cissys_getRadioSignalStrength();
功能：用于获取当前信号强度

表 13-5 cissys_getRadioSignalStrength 返回值说明

返回值	返回值含义
uint32_t	信号强度

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00006
[需求描述]：获取当前升级包可用存储空间，单位为KB
函数原型：
uint32_t　　cissys_getFwAvailableMemory();
功能：用于获取当前升级包可用存储空间

表 13-6 cissys_getFwAvailableMemory 返回值说明

返回值	返回值含义
0	获取失败
uint32_t	升级包可用存储空间

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00007
[需求描述]：擦除设备存储区
函数原型：
bool　　cissys_eraseFwFlash ();
功能：用于擦除设备存储区

表 13-7 cissys_eraseFwFlash 返回值说明

返回值	返回值含义
0	擦除操作正确
1	擦除操作出错

[需求澄清]：具体的擦出操作结果会从回调返回，返回结果定义在cissys_event_t，擦除成功返回cissys_event_fw_erase_success，擦出失败返回cissys_event_fw_erase_fail。

[需求编号]：TS-UE-SUITE-FOTA-00008
[需求描述]：检查下载完成的升级包的完整性
函数原型：
bool　　cissys_checkFwValidation();
功能：用于检查下载完成的升级包的完整性

表 13-8 cissys_checkFwValidation 返回值说明

返回值	返回值含义
0	函数执行成功
1	函数执行出错

[需求澄清]：具体的操作结果会从回调返回，返回结果定义在cissys_event_t，完整性检查成功返回cissys_event_fw_validate_success，完整性检查失败返回cissys_event_fw_validate_fail。

[需求编号]：TS-UE-SUITE-FOTA-00009
[需求描述]：下载包写入函数
函数原型：
int32_t　　cissys_writeFwBytes(uint32_t size, uint8_t* buffer);
功能：用于将收到的升级包写入指定地址
参数：
size：缓冲区长度
buffer：需要写入的缓冲区

表 13-9 cissys_writeFwBytes 返回值说明

返回值	返回值含义
0	函数执行成功
1	函数执行出错

[需求澄清]：具体的操作结果会从回调返回，返回结果定义在cissys_event_t，完整性检查成功返回cissys_event_fw_write_success，完整性检查失败返回cissys_event_fw_write_fail。

[需求编号]：TS-UE-SUITE-FOTA-00010
[需求描述]：执行升级操作函数
函数原型：
bool　　cissys_updateFireware ();
功能：调用设备接口开始升级流程

表 13-10 cissys_updateFireware 返回值说明

返回值	返回值含义
0	函数执行成功
1	函数执行出错

[需求澄清]：具体的操作结果会从回调返回，返回结果定义在cissys_event_t，完整性检查成功返回cissys_event_fw_update_success，完整性检查失败返回cissys_event_fw_update_fail。

[需求编号]：TS-UE-SUITE-FOTA-00011
[需求描述]：设置升级状态
函数原型：
bool cissys_setFwState(uint8_t state);
功能：用于保存当前的固件升级状态到设备
参数：state定义请参考cis_fw_state，包含四个升级过程状态

表 13-11 cissys_getFwVersion 返回值说明

返回值	返回值含义
0	函数执行成功
1	函数执行出错

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00012
[需求描述]：设置升级结果
函数原型：
bool cissys_setFwUpdateResult(uint8_t result);
功能：用于保存升级结果到设备
参数：state定义请参考cissys_fw_result_type_t，包含10个升级过程结果

表 13-12 cissys_setFwUpdateResult 返回值说明

返回值	返回值含义
0	读取成功
1	读取出错

[需求澄清]：设备也可以根据状态更新该升级结果，比如链接断开等结果

[需求编号]：TS-UE-SUITE-FOTA-00013
[需求描述]：读取设备上升级相关信息
函数原型：
bool cissys_readContext(cis_fw_context_t * context);
功能：用于读取设备上的升级相关信息
参数：需要读取的内容，请参考cis_fw_context_t结构体，包括了升级状态，升级结果，已存储的升级包字节数

表 13-13 cissys_readContext 返回值说明

返回值	返回值含义
0	读取成功
1	读取出错

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00014
[需求描述]：读取已存储的升级包字节数
函数原型：

```
int         cissys_getFwSavedBytes();
```

功能：用于读取已存储的升级包字节数

表 13-14 cissys_getFwSavedBytes 返回值说明

返回值	返回值含义
int	存储的数据字节数长度
-1	未存储数据

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00015
[需求描述]：保存升级包字节数
函数原型：

```
void cissys_savewritebypes ();
```

功能：由SDK调用，用于把已经接收到的升级包字节数保存进flash中，

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00016
[需求描述]：读取已存储的升级包字节数
函数原型：

```
void cissys_ClearFwBytes ();
```

功能：由SDK调用，把保存进flash中的升级包字节数清零

[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00017
[需求描述]：把当前升级MCU的标志保存进flash中
函数原型：

```
void        cissys_setSwState(bool ismcu)
```

功能：将当前升级MCU的标志保存进flash中
参数：当前是否升级MCU的标志
[需求澄清]：当平台下发升级MCU的指令下来后，SDK会将该标志位保存进flash中，当出现掉电续传的情况，依旧能保证升级数据流的流向。

[需求编号]：TS-UE-SUITE-FOTA-00018
[需求描述]：读取已存储的升级包字节数
函数原型：

```
bool cissys_getSwState();
```

功能：把升级MCU的标志从flash中读出来

[需求澄清]：设备由于各种原因重新连接平台后，将会调用该函数把上一次的升级标志读出来。

[需求编号]：TS-UE-SUITE-FOTA-00019
[需求描述]：将MCU的当前版本号写进flash中
函数原型：
void cissys_setSotaVersion(char* version);
功能：将MCU的当前版本号写进flash中保存
参数：版本信息指针
[需求澄清]：防止掉电导致SDK丢失MCU的版本信息

[需求编号]：TS-UE-SUITE-FOTA-00020
[需求描述]：将MCU的当前版本号从flash中读出来
函数原型：
uint32_t cissys_getSotaVersion(char** version);
功能：将MCU的当前版本号从flash中读出来
参数：存储版本信息的指针
返回值：版本信息的长度
[需求澄清]：无

[需求编号]：TS-UE-SUITE-FOTA-00021
[需求描述]：将MCU的升级空间大小保存进flash中
函数原型：
void cissys_setSotaMemory(uint32_t size);
功能：将MCU的升级空间大小保存进flash中
参数:MCU升级空间的大小
[需求澄清]：防止掉电导致SDK丢失MCU的升级空间大小

[需求编号]：TS-UE-SUITE-FOTA-00022
[需求描述]：将MCU的升级空间大小从flash中读出来
函数原型：
uint32_t cissys_getSotaMemory(void);
功能：将MCU的升级空间大小保存进flash中
返回值：MCU升级空间的大小
[需求澄清]：无

13.3 OTA 系统适配接口

[需求编号]：TS-UE-SUITE-OTA-00001
[需求描述]：读取保存的缓冲区数据
函数原型：

uint8_t cissys_getEID(uint8_t* buffer,uint32_t maxlen);
功能：用于读取物联网用户识别卡EID

表 9-35 cissys_getEID 参数说明

参数	类型	参数含义
buffer	uint8*	EID号码
length	uint32_t	EID长度

表 9-36 cissys_ load 返回值说明

返回值类型	含义
uint8_t	EID的有效长度

[需求澄清]：EID标识文件位于eSIM主文件路径（3F00）下，文件标识：2F02。

[需求编号]：TS-UE-SUITE-OTA-00002
[需求描述]：查询和关闭PSM功能
函数原型：
int8_t cissys_remove_psm (void);
功能：用于空中写卡过程中关闭NBiot终端设备的PSM功能

表 9-37 cissys_remove_psm 返回值说明

返回值类型	含义
int8_t	0 - 修改PSM策略成功 1 - PSM当前未使能，无需修改 -1 - 修改PSM策略失败

[需求澄清]:为保障写卡环境,在空写流程中加入了NBiot终端的PSM检查和关闭接口，旨在暂时关闭终端PSM策略，待写卡完成后恢复原策略。

[需求编号]：TS-UE-SUITE-OTA-00003
[需求描述]：恢复PSM策略
函数原型：
int8_t cissys_recover_psm (void);
功能：用于空中写卡结束后恢复NBiot终端设备的PSM策略

表 9-38 cissys_recover_psm 返回值说明

返回值类型	含义
int8_t	0 - 恢复PSM策略成功 1 - PSM未改变，无需恢复 -1 - 恢复PSM策略失败

[需求澄清]：无

[需求编号]：TS-UE-SUITE-OTA-00004
[需求描述]：查询空中写卡成功历史标志位
函数原型：
uint8_t cis_ota_history_state cissys_quary_ota_finish_state (void);
功能：用于查询终端空中写卡完成标志位

表 9-39 cis_ota_history_state cissys_quary_ota_finish_state 返回值说明

返回值类型	含义
uint8_t	0 - 终端未空中写卡成功 1 - 终端已成功完成空中写卡

[需求澄清]：无

[需求编号]：TS-UE-SUITE-OTA-00005
[需求描述]：修改空中写卡成功历史标志位
函数原型：
uint8_t cissys_change_ota_finish_state(uint8_t cis_ota_history_state flag);
功能：用于修改空中写卡成功历史标志位

表 9-40 cissys_change_ota_finish_state 参数说明

参数	类型	参数含义
flag	uint8_t	0 - 终端未空中写卡成功 1 - 终端已成功完成空中写卡

[需求澄清]：空中写卡只应用与从临时码号至正式码号的单次变更，当终端已完成空写后，再次调用空写接口时，SDK应返回已完成状态，不再与平台交互。特殊情形下，如重新焊接eSIM，则可以先将空写成功标志置0，再次发起写卡。

13.4 SDK 与 MCU 的接口消息定义:

升级过程中MCU与SDK的UARTAT指令：
#define CIS_EVENT_FIRMWARE_DOWNLOADING CIS_EVENT_BASE + 40
设备进入下载升级包状态
#define CIS_EVENT_FIRMWARE_DOWNLOAD_FAILED CIS_EVENT_BASE + 41
设备下载升级包失败
#define CIS_EVENT_FIRMWARE_DOWNLOADED CIS_EVENT_BASE + 42
设备下载升级包完成
#define CIS_EVENT_FIRMWARE_UPDATING CIS_EVENT_BASE + 43
设备进入升级状态
#define CIS_EVENT_FIRMWARE_UPDATE_SUCCESS CIS_EVENT_BASE + 44
设备升级成功
#define CIS_EVENT_FIRMWARE_UPDATE_FAILED CIS_EVENT_BASE + 45
设备升级失败

```
#define          CIS_EVENT_FIRMWARE_UPDATE_OVER    CIS_EVENT_BASE + 46
设备升级完成
#define          CIS_EVENT_FIRMWARE_DOWNLOAD_DISCONNECT  CIS_EVENT_BASE + 47
设备下载中断
#define     CIS_EVENT_FIRMWARE_ERASE_SUCCESS        CIS_EVENT_BASE + 48
设备擦除完成
#define     CIS_EVENT_FIRMWARE_ERASE_FAIL            CIS_EVENT_BASE + 49
设备擦除失败
#define     CIS_EVENT_FIRMWARE_TRIGGER               CIS_EVENT_BASE + 50
针对非自有FOTA通道的固件升级触发
#define     CIS_EVENT_SOTA_DOWNLOADING          CIS_EVENT_BASE + 51
通知MCU软件下载开始
#define     CIS_EVENT_SOTA_DOWNLOAED            CIS_EVENT_BASE + 52
通知MCU软件下载完成
#define     CIS_EVENT_SOTA_FLASHERASE           CIS_EVENT_BASE + 53
通知MCU擦除升级用的FLASH
#define     CIS_EVENT_SOTA_UPDATING             CIS_EVENT_BASE + 54
通知MCU开始执行升级动作

#define     CIS_EVENT_CMIOT_OTA_START               CIS_EVENT_BASE + 60
开始进行空中写卡
#define     CIS_EVENT_CMIOT_OTA_SUCCESS             CIS_EVENT_BASE + 61
空中写卡成功
#define     CIS_EVENT_CMIOT_OTA_FAIL                CIS_EVENT_BASE + 62
空中写卡失败
#define     CIS_EVENT_CMIOT_OTA_FINISH              CIS_EVENT_BASE + 63
空中写卡完成
```

14. 资源模型 Profile 要求

IPSO组织针对CoAP+LwM2M的物联网传输模型，制定了一套适用于LwM2M的Profile规范《Smart Objects Starter Pack1.0》。该规范定义了一个公共的智能事物模型，其中定义了18种智能事物类型，包括温度传感器、灯光控制、加速度、电子传感器等通用的UE类型。同时，IPSO为了完善对于Profile的规范，还提供了《Smart Objects Expansion Pack》规范进行了扩展。

IPSO包含两层定义，包括对Object id和Resource id定义，以及一个具体Object所能对应的resource类型。要求object id和resource id定义不重复。一个object可以有多个resource id。一个object对应一组resource的集合。

可以从平台查询最新的支持的profile的列表，如果现有的profile不满足需求，需要向平台申请新的object、resource资源定义。

基础通信套件支持如下的对象和资源定义。

[需求编号]：TS-UE-SUITE-RMODEL-00001
[需求描述]：基础通信套件支持资源模型Device对象的的读取操作
对象ID：3
资源要求如下：

Resource Id	名称	操作权限	数据类型	范围	单位	备注
3	Firmware Version	R	String			模组固件版本号
9	Battery Level	R	Integer	0-100	%	电量百分比,用于设置升级策略
10	Memory Free	R	Integer		KB	模组侧FOTA可用的储空间,用于设置升级策略
26261	MCU Memory Free	R	Integer		KB	MCU侧可用的SOTA存储空间
19	Sotfware Version	R	String			MCU固件版本号

[需求编号]：TS-UE-SUITE-RMODEL-00001
[需求描述]：基础通信套件支持资源模型FirmwareUpdate对象的的读写执行操作
对象ID：5
资源要求如下：

Resource Id	名称	操作权限	数据类型	范围	单位	备注
0	Package	W	Opaque			升级包,采用推（PUSH）方式升级时,平台直接将升级包写入该

						资源
1	Package URI	RW	String	0-255 字节		升级包 URI，采用拉（PULL）方式升级是，平台将升级包的地址写入该资源
2	Update	E				执行升级操作，固件包下载完成后平台自动下发 execute 消息触发升级
3	State	R	Integer	0-3		0: 空闲（未升级或者升级完成） 1：下载中 2：下载完成 3：升级中
5	Update Result	R	Integer	0-9		0: 初始状态 1: 升级成功 2: 空间不足 3：下载过程中内存溢出 4: 下载过程中链接断开 5: 包完整性检查失败 6: 固件包不支持 7：无效 URI

						8：升级失败 9：协议不支持
8	Firmware Update Protocol Support	R	Integer	0,1		采用拉的方式获取固件包时，终端支持的协议，目前只支持推，平台忽略该资源 0: CoAP 1: CoAPS
9	Firmware Update Delivery Method	R	Integer	0-2		固件包传输方式，目前只支持推，平台忽略该资源 0：拉 1：推 2：推拉均可
26500	Transferred Bytes	R	Integer			已传输字节数，用于断点续传
26501	Switch to Download State	E				切换系统状态，准备下载固件包
26550	Switch to Mcu Update	E				切换到SOTA升级功能
26600	Firmware Trigger	E				针对非自有FOTA通道的触发

[需求编号]：TS-UE-SUITE-RMODEL-00003

[需求描述]：基础通信套件支持资源模型ConnectivityMonitoring对象的的读取操作

对象ID：4

资源要求如下：

资源 ID	名称	操作	数据类型	范	单位	备注

		权限		围		
2	RadioSignal Strength	R	Integer		dBm	信号强度
8	Cell ID	R	Integer			小区ID
6035	RSRP	R	Integer		dBm	服务小区参考信号接收功率
6038	SINR	R	Integer		dB	信号与干扰加噪声比

[需求编号]：TS-UE-SUITE-RMODEL-00004
[需求描述]：基础通信套件支持资源模型CMCCMonitoring对象的的读取操作
对象ID：10290
资源要求如下：

资源 ID	名称	操作权限	数据类型	范围	单位	备注
0	Network Bearer	R	Integer			网络制式 0:GSM cellular network 1:TD-SCDMA cellular network 2:WCDMA cellular network 3:CDMA2000 cellular network 4:WiMAX cellular network 5:LTE-TDD cellular network 6:LTE-FDD cellular network 7：NB-IoTcellularnetwork 8：eMTC cellularnetwork 9~20: Reserved for other type cellular network 21~40 Wireless Bearers 21: WLAN network 22: Bluetooth network 23: IEEE 802.15.4 network 24~40: Reserved for other type local wireless network 41~50 are Wireline Bearers 41: Ethernet

						42: DSL 43: PLC 44~50: reserved for others type wireline networks.
6	LinkUtilization	R	Integer	0-100	%	当前链路的平均使用率
30	CSQ	R	Integer			信号强度质量
31	ECL	R	Integer			覆盖等级
6030	plmnID	R	Integer			PLMN(MCC/MNC)
6032	dlEarfcn	R	Integer			下行频点
6034	PCI	R	Integer			服务小区物理小区识别码
20628	PCI1	R	Integer			最强同频邻小区物理小区识别码
20629	PCI2	R	Integer			次强同频邻小区物理小区识别码
20630	RSRP1	R	Integer			最强同频邻小区参考信号接收功率
20631	RSRP2	R	Integer			次强同频邻小区参考信号接收功率
5513	Latitude	R	String			GPS纬度
5514	Longitude	R	String			GPS经度
5515	Altitude	R	String			GPS高度

[需求编号]：TS-UE-SUITE-RMODEL-00005

[需求描述]：基础通信套件支持资源模型POWERUPLOG对象的的读取操作

对象ID：3351

资源要求如下：

资源 ID	名称	操作权限	数据类型	范围	单位	备注
2	IMEI	R	String			IMEI号码
3	IMSI	R	String			IMSI号码
4	MSISDN	R	String			EID号码

[需求编号]：TS-UE-SUITE-RMODEL-00006

[需求描述]：基础通信套件支持资源模型CmdhDefEcValues对象的的读写操作

对象ID：2051

资源要求如下：

资源 ID	名称	操作权限	数据类型	范围	单位	备注
2	RequestOrigin	RW	String			空中写卡下行命令通道
3	RequestContext	RW	String			空中写卡上行

						响应通道

15. 安全要求

基础通信套件应支持DTLS协议的实现，在SDK和平台之间建立安全通道，用于认证和数据安全传输。

CoAP协议的Client应同样是DTLS协议的client，当DTLS握手协议完成之后，client发起第一条CoAP请求消息。CoAP协议的所有信息应作为DTLS协议的应用层数据进行传输。

当终端需要释放资源时，可以选择关闭DTLS连接，但通常情况下DTLS连接应一直保持。如果CoAP协议检测到多次重传失败，则启动下层链路重建。

具体方案可参见蜂窝物联网系列规范《中国移动蜂窝物联网总体安全技术要求》第七章“基础通信套件安全通信能力要求”。

16. 编制历史

版本号	更新时间	主要内容或重大修改	编制人	技术审核人	部门审核人
0.0.1	2017.8.16	0.0.1版本	龙容、骆正虎	刘聪	郭晓岩
1.0.0	2017.12.2	1.0.0版本	龙容、骆正虎、王波、田康、刘琨、李笑如	刘聪	程卫东
2.0.0	2018.11.15	2.0.0版本	龙容、骆正虎、白杰	袁园	

附录A（资料性附录） 接口使用数据流图示例

A.1. 设备注册接口数据流向

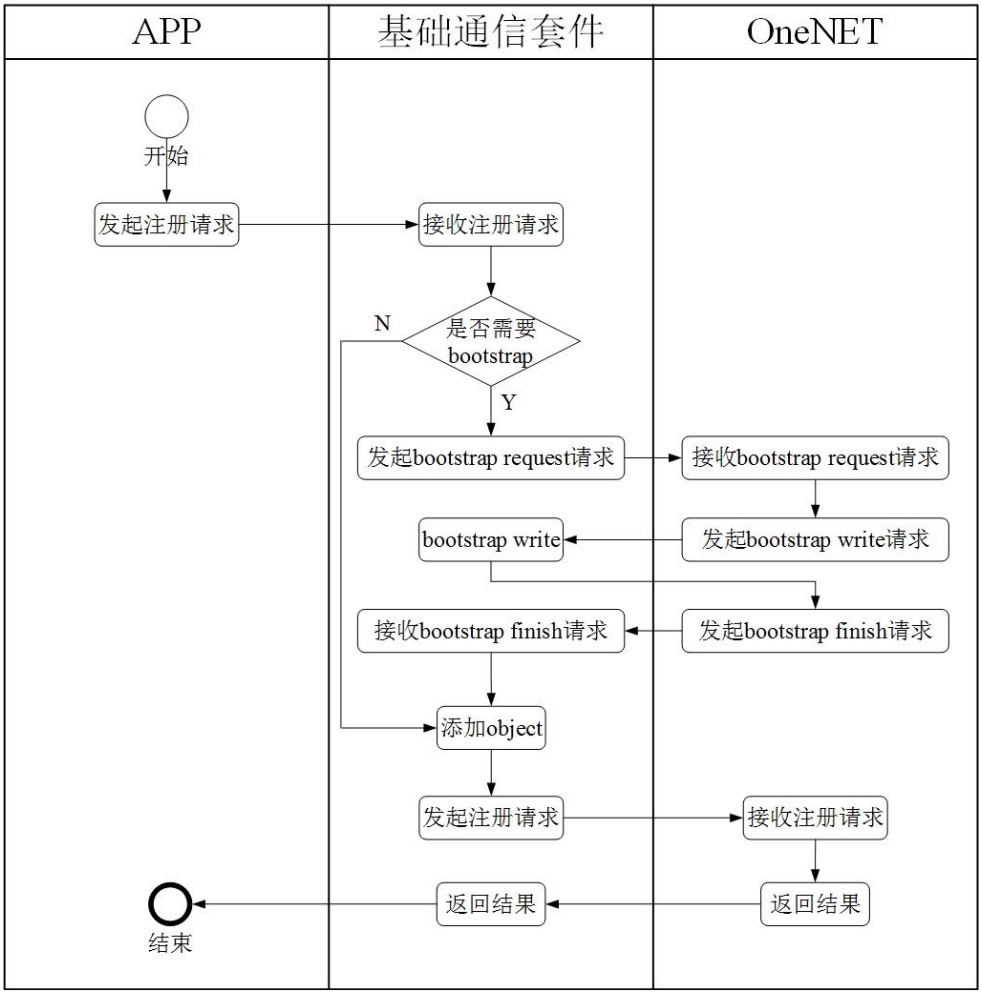


图 B-1 设备注册接口数据流向图

设备注册接口数据流向说明：

1) 设备用户发起注册请求
2) API 接口接收注册请求，得到请求参数信息
3) 判断是否需要bootstrap。若需要，则先进行bootstrap request、bootstrap write、bootstrap finish 流程，再添加 object；若不需要，则直接添加 object
4) 向平台发起注册请求
5) 平台返回结果

A.2. 设备注销接口数据流向

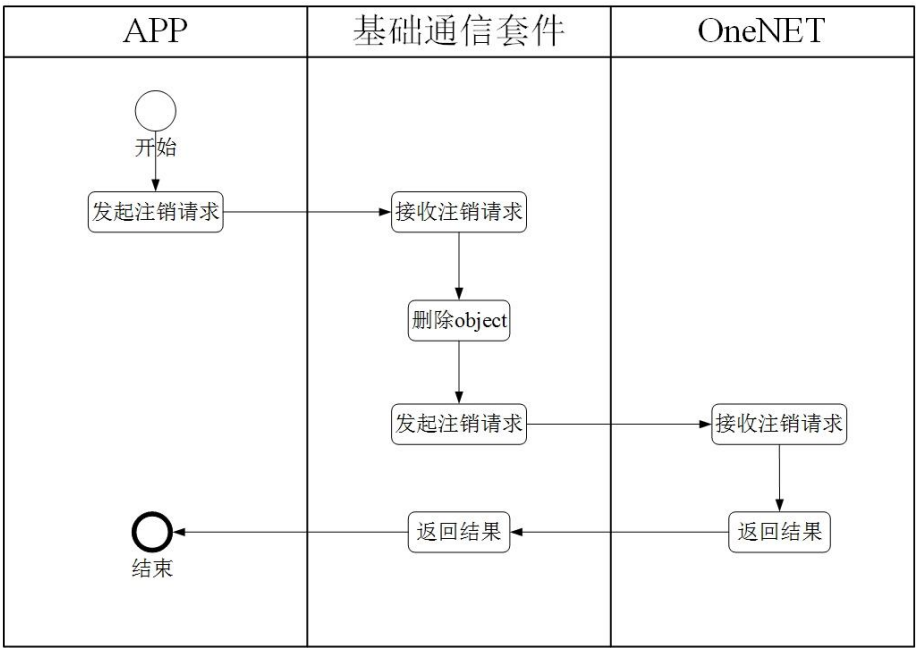


图 B-2 设备注销接口数据流向图

设备注销接口数据流向说明：

1) 设备用户发起注销请求
2) API 接口接收注销请求，得到请求参数信息
3) 删除 object，向平台发起注销请求
4) 平台返回结果

A.3. 设备资源读取接口数据流向

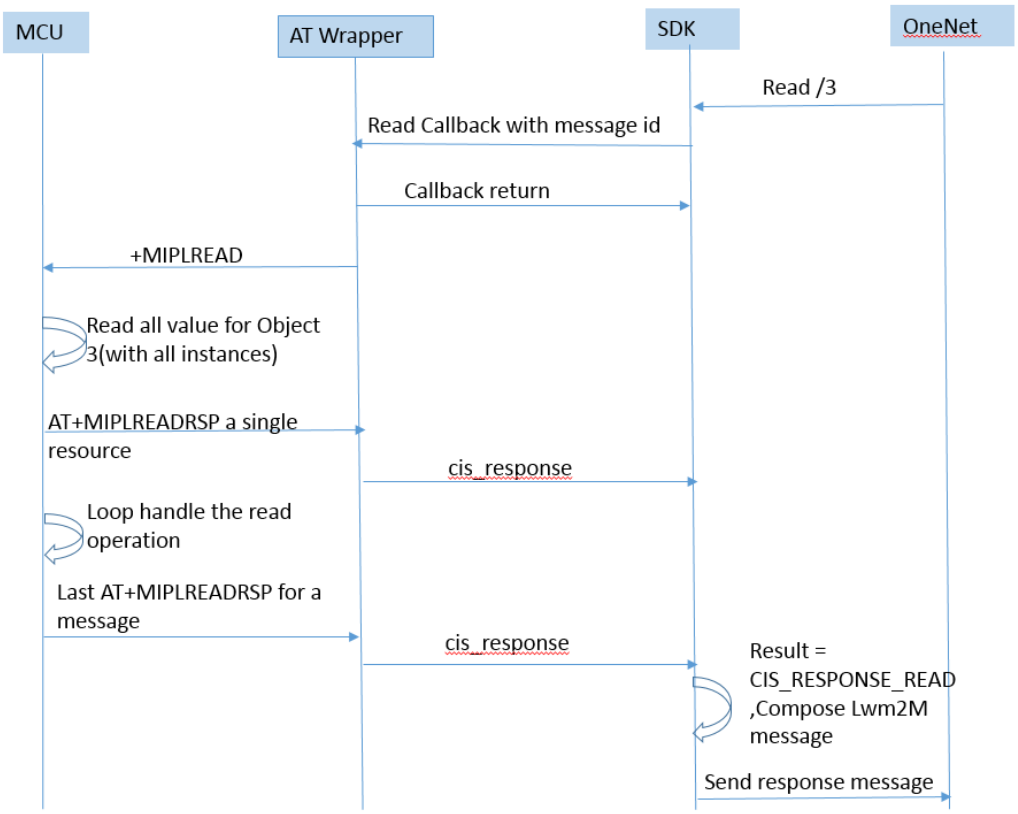


图 B-3 设备资源读取接口数据流向图示例

设备资源读取接口数据流向说明：

1) 平台发起 Read 请求。
2) 基础通信套件得到请求参数信息，调用 cis_on_read 回调函数。
3) 设备用户执行 cis_on_read 回调函数。
4) ATWrapper 返回回调执行结果。
5) ATWrapper 申请需要的空间。
6) ATWrappper 上报+MIPLREAD 事件给 MCU。
7) MCU 将读取的消息发送给 ATWrapper。
8) ATWrapper 通过 API 接口 cis_notify 返回函数值给基础通信套件
9) 基础通信套件将获取的数据组装
10) 基础通信套件将数据包发送给平台

附录B（资料性附录） Config文件生成工具

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 解析的XML文件描述内容如下：

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

<config>
<item>
<version>1.0</version>
<cfgcnt>3</cfgcnt>
</item>

<item id="1"></item>

<item id="2">
<data name="Mtu">2455</data>
<data name="Linktype">1</data>
    <data name="Bandtype">1</data>
<data name="APN">CMIOT</data>
    <data name="Username">****</data>
<data name="Password">****</data>
    <data name="Host">183.200.40.39</data>
    <data name="Userdata">***</data>
</item>

<item id="3">
<data name="LogEnabled">1</data>
<data name="LogExtOutput">1</data>
<data name="LogOutputType">2</data>
<data name="LogOutputLevel">2</data>
    <data name="LogBufferSize">200</data>
    <data name="Userdata"></data>
</item>

</config>
```

附录C（资料性附录）基础通信套件接口文件范例

C.1. cis_api.h

```
#ifndef _CIS_API_H_
#define _CIS_API_H_

#include "cis_config.h"
#include "cis_def.h"

#ifdef __cplusplus
extern "C" {
#endif

struct st_cis_context;



typedef struct st_cis_version       cis_version_t;
typedef struct st_cis_data          cis_data_t;
typedef struct st_cis_observe_attr  cis_observe_attr_t;
typedef struct st_cis_inst_bitmap   cis_inst_bitmap_t;
typedef struct st_cis_res_count     cis_res_count_t;
typedef struct st_cis_callback      cis_callback_t;
typedef struct st_uri               cis_uri_t;

typedef  cis_coapret_t  (*cis_read_callback_t)                          (void*
context,cis_uri_t* uri,cis_mid_t mid);
typedef  cis_coapret_t  (*cis_write_callback_t)                         (void*
context,cis_uri_t*      uri,const      cis_data_t*      value,cis_attrcount_t
attrcount,cis_mid_t mid);
typedef  cis_coapret_t  (*cis_exec_callback_t)                          (void*
context,cis_uri_t* uri,const uint8_t* buffer,uint32_t length,cis_mid_t mid);
typedef   cis_coapret_t   (*cis_observe_callback_t)                     (void*
context,cis_uri_t* uri,bool flag,cis_mid_t mid);
typedef   cis_coapret_t   (*cis_discover_callback_t)                    (void*
context,cis_uri_t* uri,cis_mid_t mid);
typedef   cis_coapret_t   (*cis_set_params_callback_t)                  (void*
context,cis_uri_t* uri,cis_observe_attr_t parameters,cis_mid_t mid);
typedef void        (*cis_event_callback_t)          (void* context,cis_evt_t
id,void* param);
```

```
#define LIFETIME_INVALID      ((uint32_t)0xFFFFFFFF)
#define LIFETIME_LIMIT_MIN    ((uint32_t)0x0000000F)
#define LIFETIME_LIMIT_MAX    ((uint32_t)0x0FFFFFFF)

#define PUMP_RET_CUSTOM       ((uint32_t)0xFFFFFFFF)
#define PUMP_RET_NOSLEEP      ((uint32_t)0x00)

#define URI_INVALID           ((uint16_t)0xFFFF)
#define URI_MAX_ID            ((uint16_t)0xFFFF)
#define URI_FLAG_OBJECT_ID    (uint8_t)0x04
#define URI_FLAG_INSTANCE_ID  (uint8_t)0x02
#define URI_FLAG_RESOURCE_ID  (uint8_t)0x01

#define CIS_URI_IS_SET_INSTANCE(uri) (((uri)->flag & URI_FLAG_INSTANCE_ID) != 0)
#define CIS_URI_IS_SET_RESOURCE(uri) (((uri)->flag & URI_FLAG_RESOURCE_ID) != 0)

struct st_uri
{
    uint8_t      flag;           // indicates which segments are set
    cis_oid_t    objectId;
    cis_iid_t    instanceId;
    cis_rid_t    resourceId;
};

CIS_API  cis_ret_t       cis_uri_make                (cis_oid_t  oid,cis_iid_t
iid,cis_rid_t rid, cis_uri_t* uri);
CIS_API cis_ret_t    cis_uri_update       (cis_uri_t* uri);


typedef enum
{
    cis_data_type_undefine = 0,
    cis_data_type_string,
    cis_data_type_opaque,
    cis_data_type_integer,
    cis_data_type_float,
    cis_data_type_bool,
}cis_datatype_t;

struct st_cis_callback
{
    cis_read_callback_t      onRead;
    cis_write_callback_t     onWrite;
```

```
    cis_exec_callback_t      onExec;
    cis_observe_callback_t   onObserve;
    cis_discover_callback_t   onDiscover;
    cis_set_params_callback_t onSetParams;
    cis_event_callback_t     onEvent;
};

struct st_cis_data
{
    cis_rid_t             id;
    cis_datatype_t      type;

    struct
    {
        uint32_t    length;
        uint8_t*    buffer;
}asBuffer;

    union
    {
        bool            asBoolean;
        int64_t         asInteger;
        double          asFloat;

    } value;
} ;


struct st_cis_version
{
    uint8_t     major;
    uint8_t     minor;
};

struct st_cis_observe_attr
{
    uint8_t     toSet;
    uint8_t     toClear;
    cis_time_t  minPeriod;
    cis_time_t  maxPeriod;
    float       greaterThan;
    float       lessThan;
    float       step;
};
```

```
struct st_cis_inst_bitmap
{
    cis_instcount_t instanceCount;
    cis_instcount_t instanceBytes;
const uint8_t*  instanceBitmap;
};



struct st_cis_res_count
{
    cis_attrcount_t attrCount;
    cis_actcount_t  actCount;
};


CIS_API cis_ret_t    cis_init            (void** context,void* config,uint16_t
config_len);
CIS_API cis_ret_t    cis_deinit          (void** context);

CIS_API  cis_ret_t       cis_register               (void*  context,cis_time_t
lifetime,const cis_callback_t* cb);
CIS_API cis_ret_t    cis_unregister      (void* context);

CIS_API  cis_ret_t       cis_addobject               (void*  context,cis_oid_t
objectid,const cis_inst_bitmap_t* bitmap,const cis_res_count_t* rescount);
CIS_API cis_ret_t    cis_delobject       (void* context,cis_oid_t objectid);

CIS_API uint32_t     cis_pump            (void* context);
CIS_API  cis_ret_t      cis_version              (void*  context,cis_version_t*
version);

CIS_API  cis_ret_t       cis_update_reg             (void*  context,cis_time_t
lifetime,bool withObjects);

CIS_API cis_ret_t    cis_response        (void*    context,const    cis_uri_t*
uri,const cis_data_t* value,cis_mid_t mid,cis_coapret_t result);
CIS_API cis_ret_t    cis_notify          (void*    context,const    cis_uri_t*
uri,const cis_data_t* value,cis_mid_t mid,cis_coapret_t result,bool needAck);



/*
```

```
 * Event code
 * @cis_evt_t
 */
#define     CIS_EVENT_BASE                  (cis_evt_t)0x00
#define     CIS_EVENT_BOOTSTRAP_START       CIS_EVENT_BASE + 1
#define     CIS_EVENT_BOOTSTRAP_SUCCESS     CIS_EVENT_BASE + 2
#define     CIS_EVENT_BOOTSTRAP_FAILED      CIS_EVENT_BASE + 3
#define     CIS_EVENT_CONNECT_SUCCESS       CIS_EVENT_BASE + 4
#define     CIS_EVENT_CONNECT_FAILED        CIS_EVENT_BASE + 5

#define     CIS_EVENT_REG_SUCCESS           CIS_EVENT_BASE + 6
#define     CIS_EVENT_REG_FAILED            CIS_EVENT_BASE + 7
#define     CIS_EVENT_REG_TIMEOUT           CIS_EVENT_BASE + 8

#define     CIS_EVENT_LIFETIME_TIMEOUT      CIS_EVENT_BASE + 9
#define     CIS_EVENT_STATUS_HALT           CIS_EVENT_BASE + 10
#define     CIS_EVENT_UPDATE_SUCCESS        CIS_EVENT_BASE + 11
#define     CIS_EVENT_UPDATE_FAILED         CIS_EVENT_BASE + 12
#define     CIS_EVENT_UPDATE_TIMEOUT        CIS_EVENT_BASE + 13


#define     CIS_EVENT_RESPONSE_FAILED       CIS_EVENT_BASE + 20
#define     CIS_EVENT_NOTIFY_FAILED         CIS_EVENT_BASE + 21



/*
 * Error code
 * @cis_ret_t
 */
#define     CIS_RET_BASE                    (cis_ret_t)0x00
#define     CIS_RET_OK                      CIS_RET_BASE
#define     CIS_RET_NO_ERROR                CIS_RET_BASE
#define     CIS_RET_ERROR                   CIS_RET_BASE - 1
#define     CIS_RET_FAILED                  CIS_RET_BASE - 2
#define     CIS_RET_INVILID                 CIS_RET_BASE - 3
#define     CIS_RET_EXIST                   CIS_RET_BASE - 4
#define     CIS_RET_PARAMETER_ERR           CIS_RET_BASE - 5
#define     CIS_RET_MEMORY_ERR              CIS_RET_BASE - 6
#define     CIS_RET_NOT_FOUND               CIS_RET_BASE - 7
#define     CIS_RET_TIMEOUT                 CIS_RET_BASE - 8
#define     CIS_RET_COUNTOUT                CIS_RET_BASE - 9
#define     CIS_RET_PENDING                 CIS_RET_BASE - 10
#define     CIS_RET_UNSUPPORTIVE            CIS_RET_BASE - 11
```

```
/*
 *COAP result code
 *@cis_cbret_t
 */
#define     CIS_COAP_204_CHANGED                (uint8_t)0x44
#define     CIS_COAP_205_CONTENT                (uint8_t)0x45
#define     CIS_COAP_206_CONFORM                (uint8_t)0x46
#define     CIS_COAP_231_CONTINUE               (uint8_t)0x5F
#define     CIS_COAP_400_BAD_REQUEST            (uint8_t)0x80
#define     CIS_COAP_401_UNAUTHORIZED           (uint8_t)0x81
#define     CIS_COAP_404_NOT_FOUND              (uint8_t)0x84
#define     CIS_COAP_405_METHOD_NOT_ALLOWED     (uint8_t)0x85
#define     CIS_COAP_406_NOT_ACCEPTABLE         (uint8_t)0x86
#define     CIS_COAP_503_SERVICE_UNAVAILABLE    (uint8_t)0xA3


#define     CIS_CALLBACK_CONFORM                CIS_COAP_206_CONFORM
#define     CIS_CALLBACK_BAD_REQUEST            CIS_COAP_400_BAD_REQUEST
#define     CIS_CALLBACK_UNAUTHORIZED           CIS_COAP_401_UNAUTHORIZED
#define     CIS_CALLBACK_NOT_FOUND              CIS_COAP_404_NOT_FOUND
#define     CIS_CALLBACK_METHOD_NOT_ALLOWED     CIS_COAP_405_METHOD_NOT_ALLOWED
#define     CIS_CALLBACK_NOT_ACCEPTABLE         CIS_COAP_406_NOT_ACCEPTABLE
#define     CIS_CALLBACK_SERVICE_UNAVAILABLE    CIS_COAP_503_SERVICE_UNAVAILABLE


#define     CIS_RESPONSE_READ                   CIS_COAP_205_CONTENT
#define     CIS_RESPONSE_WRITE                  CIS_COAP_204_CHANGED
#define     CIS_RESPONSE_EXECUTE                CIS_COAP_204_CHANGED
#define     CIS_RESPONSE_OBSERVE                CIS_COAP_205_CONTENT
#define     CIS_RESPONSE_OBSERVE_PARAMS         CIS_COAP_204_CHANGED
#define     CIS_RESPONSE_DISCOVER               CIS_COAP_205_CONTENT
#define     CIS_RESPONSE_CONTINUE               CIS_COAP_231_CONTINUE
#define     CIS_RESPONSE_BAD_REQUEST            CIS_COAP_400_BAD_REQUEST
#define     CIS_RESPONSE_UNAUTHORIZED           CIS_COAP_401_UNAUTHORIZED
#define     CIS_RESPONSE_NOT_FOUND              CIS_COAP_404_NOT_FOUND
#define                                         CIS_RESPONSE_METHOD_NOT_ALLOWED
CIS_COAP_405_METHOD_NOT_ALLOWED
#define     CIS_RESPONSE_NOT_ACCEPTABLE         CIS_COAP_406_NOT_ACCEPTABLE
#define                                         CIS_RESPONSE_SERVICE_UNAVAILABLE
CIS_COAP_503_SERVICE_UNAVAILABLE
```

```
#define     CIS_NOTIFY_CONTENT                  CIS_COAP_205_CONTENT
#define     CIS_NOTIFY_CONTINUE                 CIS_COAP_231_CONTINUE



#ifdef __cplusplus
};
#endif


#endif//_CIS_API_H_
```

C.2. cis_def.h

```
#ifndef _CIS_DEFINITION_HEADER_
#define _CIS_DEFINITION_HEADER_

//some pre-defined macro
//#define  LWM2M_LITTLE_ENDIAN
#define  LWM2M_BIG_ENDIAN


////////////////////////////////////////
#if defined( CIS_WIN32 )
#    if defined( CIS_EXPORT )
#        define CIS_API __declspec(dllexport)
#    endif
#    if defined( CIS_IMPORT )
#        define CIS_API __declspec(dllimport)
#    endif  // defined(NB_EXPORTS)
#elif defined( CIS_LINUX )
// Do not upstream, this will be added later upstream,
// after this file is renamed to base_export.h. Not
// cherry-picking that since the #define's also change
#define CIS_API __attribute__((visibility("default")))

#elif defined( CIS_EMBED )
#include <stdbool.h>
#include <time.h>
#else
#error "Please select the target platform used in your application."
```

```
#endif


#if !defined( CIS_API )
#define     CIS_API
#endif

#include <stdint.h>
#include <stdlib.h>

#ifndef TRUE
#define  TRUE (1)
#endif

#ifndef FALSE
#define  FALSE (0)
#endif

typedef int32_t      cis_evt_t;
typedef int32_t      cis_ret_t;
typedef uint8_t      cis_coapret_t;
typedef uint16_t     cis_oid_t;
typedef uint16_t     cis_iid_t;
typedef uint16_t     cis_rid_t;
typedef uint32_t     cis_mid_t;   //callback message id

typedef cis_iid_t    cis_instcount_t;
typedef cis_rid_t    cis_attrcount_t;
typedef cis_rid_t    cis_actcount_t;
typedef int32_t      cis_time_t;  //normally in second unit

/*
 *
 */
#ifndef     CIS_VERSION_MAJOR
#define     CIS_VERSION_MAJOR                2
#endif
#ifndef     CIS_VERSION_MINOR
#define     CIS_VERSION_MINOR                3
#endif
#ifndef     CIS_VERSION_MICRO
#define     CIS_VERSION_MICRO                0
#endif
```

```
#ifndef     DM_VERSION_MAJOR
#define     DM_VERSION_MAJOR                1
#endif
#ifndef     DM_VERSION_MINOR
#define     DM_VERSION_MINOR                1
#endif
#ifndef     DM_VERSION_MICRO
#define     DM_VERSION_MICRO                1
#endif

#ifndef     CIS_ENABLE_UPDATE_MCU
#define     CIS_ENABLE_UPDATE_MCU             0
#else
#undef      CIS_ENABLE_UPDATE_MCU
#define     CIS_ENABLE_UPDATE_MCU             1
#endif

#ifndef     CIS_ENABLE_UPDATE
#define     CIS_ENABLE_UPDATE                 0
#else
#undef      CIS_ENABLE_UPDATE
#define     CIS_ENABLE_UPDATE                 1
#endif

#ifndef     CIS_ENABLE_MONITER
#define     CIS_ENABLE_MONITER                0
#else
#undef      CIS_ENABLE_MONITER
#define     CIS_ENABLE_MONITER                1
#endif

#ifndef      CIS_ENABLE_CMIOT_OTA
#define      CIS_ENABLE_CMIOT_OTA             0
#else
#undef       CIS_ENABLE_CMIOT_OTA
#define      CIS_ENABLE_CMIOT_OTA             1
#endif

#ifndef     CIS_ENABLE_DM
#define     CIS_ENABLE_DM                   0
#else
#undef      CIS_ENABLE_DM
#define     CIS_ENABLE_DM                   1
#endif
```

```
#ifndef     CIS_ENABLE_AUTH
#define     CIS_ENABLE_AUTH                 0
#else
#undef      CIS_ENABLE_AUTH
#define     CIS_ENABLE_AUTH                 1
#endif


//default
#define CIS_ENABLE_BOOTSTRAP
//#define CIS_ENABLE_MEMORYTRACE

/*
 *
 */
#ifndef     CIS_ENABLE_BOOTSTRAP
#define     CIS_ENABLE_BOOTSTRAP                0
#else
#undef      CIS_ENABLE_BOOTSTRAP
#define     CIS_ENABLE_BOOTSTRAP                1
#endif

#ifndef     CIS_ENABLE_BLOCK
#define     CIS_ENABLE_BLOCK                    0
#else
#undef      CIS_ENABLE_BLOCK
#define     CIS_ENABLE_BLOCK                    1
#endif

#ifndef     CIS_ENABLE_MEMORYTRACE
#define     CIS_ENABLE_MEMORYTRACE              0
#else
#undef      CIS_ENABLE_MEMORYTRACE
#define     CIS_ENABLE_MEMORYTRACE              1
#endif

#ifndef     CIS_ENABLE_LOG
#define     CIS_ENABLE_LOG                      0
#else
#undef      CIS_ENABLE_LOG
#define     CIS_ENABLE_LOG                      1
#endif
```

```
#define     CIS_CONFIG_LOCK_INFINITY              0xFFFFFFFF
#define     CIS_CONFIG_MEMORYTRACE_TIME           (10)
#define     CIS_CONFIG_CALLBACK_TIMEOUT           (10)

#define     CIS_CONFIG_CONNECT_RETRY_TIME         (2)
#define     CIS_CONFIG_REG_INTERVAL_TIME          (5)

#define     CIS_COFNIG_REG_QUERY_SIZE             (256)
#define     CIS_COFNIG_REG_PAYLOAD_SIZE           (512)

// the maximum payload transfered by block1 we accumulate per server
#define     CIS_CONFIG_BLOCK1_SIZE_MAX            (2048)

#ifndef     CIS_ENABLE_DTLS
#define     CIS_ENABLE_DTLS                     0
#else
#undef      CIS_ENABLE_DTLS
#define     CIS_ENABLE_DTLS                     1
#endif

#ifndef     CIS_DTLS_LOG
#define     CIS_DTLS_LOG                      0
#else
#undef      CIS_DTLS_LOG
#define     CIS_DTLS_LOG                      1
#endif

#ifndef     CIS_LITTLE_ENDIAN
#define     CIS_LITTLE_ENDIAN                   0
#else
#undef      CIS_LITTLE_ENDIAN
#define     CIS_LITTLE_ENDIAN                   1
#endif


#ifndef     CIS_ENABLE_PSK
#define     CIS_ENABLE_PSK                      0
#else
#undef      CIS_ENABLE_PSK
#define     CIS_ENABLE_PSK                      1
#endif
#endif //_CIS_DEFINITION_HEADER_
```

C.3. cis_config.h

```
#ifndef _NB_DEBUG_H_
#define _NB_DEBUG_H_


#   include <stdio.h>
#   include <stdarg.h>

#include "cis_if_sys.h"
#include "cis_api.h"
#include "cis_def.h"


#define LOG_EXOUTPUT_TIME_MSK     (0x01)
#define LOG_EXOUTPUT_FILE_MSK     (0x02)
#define LOG_EXOUTPUT_LINE_MSK     (0x04)




#define LOG_LEVEL_ERROR      (0x00)
#define LOG_LEVEL_WARNING    (0x01)
#define LOG_LEVEL_INFO       (0x02)
#define LOG_LEVEL_DEBUG      (0x03)


#define LOG_TEXT_SIZE_MIN                   (64)
#define LOG_TEXT_SIZE_MAX                   (2048)
#define LOG_PREFIX_SIZE_MAX                 (32)
//////////////////////////////////////////////////////////////////////////
#define LOG_DUMP_ENABLED                    1
#define LOG_OUTPUT_EXTINFO                  0

//////////////////////////////////////////////////////////////////////////
#ifndef LOG_TAG
#define LOG_TAG             "cis"
#endif

//////////////////////////////////////////////////////////////////////////
#define STRINGIFY(x)  #x
#define STRCONCAT(x, y)  #x#y
```

```
#define AJOIN(X, Y) A_DO_JOIN(X, Y)
#define A_DO_JOIN(X, Y) A_DO_JOIN2(X, Y)
#define A_DO_JOIN2(X, Y) X##Y

extern int utils_strlen(const char *str);
extern int utils_vsnprintf(char *buf, size_t buf_size, const char *fmt, va_list
args);
extern int utils_snprintf(char* buffer, size_t size, const char* format, ...);

extern bool     gLogEnabled;
extern uint8_t  gLogLevel;
extern uint8_t  gLogExoutput;
extern uint16_t gLogBufferLength;

static inline void logOutput(uint32_t level, const char *tag, const char*
format, ...)
{
    va_list va;
    int len, len2,space;
    char* pBuffer;

if(!gLogEnabled ||
        level > gLogLevel)
        return;

if(tag == NULL)
        tag = "TAG";

pBuffer = (char*)cissys_malloc(gLogBufferLength);
    /*space "\n\0"*/
    space = gLogBufferLength - 2;
    len = utils_snprintf(pBuffer, LOG_PREFIX_SIZE_MAX, "#%s,%d:", tag,level);
    utils_snprintf( pBuffer, LOG_PREFIX_SIZE_MAX, "#%s,%d:", tag,level);
    space -= len;
    va_start(va, format);
    len2 = utils_vsnprintf( pBuffer+len, space ,format, va);
    va_end(va);
    len += (len2>0?len2:space);

    if(len < gLogBufferLength - 2){
        pBuffer[len] = '\n';
        pBuffer[len+1] = '\0';
    }else{
        pBuffer[gLogBufferLength - 5] = '.';
```

```
        pBuffer[gLogBufferLength - 4] = '.';
        pBuffer[gLogBufferLength - 3] = '?';
        pBuffer[gLogBufferLength - 2] = '\n';
        pBuffer[gLogBufferLength - 1] = '\0';
    }

    cissys_logwrite((uint8_t*)pBuffer,utils_strlen(pBuffer));
    cissys_free(pBuffer);
};



static inline void logData(const char* format, ...)
{
    va_list va;
    int len;
    char* pBuffer;

if(!gLogEnabled)
        return;

pBuffer = (char*)cissys_malloc(gLogBufferLength);

    va_start(va, format);
    len = utils_vsnprintf( pBuffer,gLogBufferLength - 1,format, va);
    va_end(va);

if(len < gLogBufferLength - 1){
        pBuffer[len+1] = '\0';
    }

    cissys_logwrite((uint8_t*)pBuffer,utils_strlen(pBuffer));
    cissys_free(pBuffer);
};


#if CIS_ENABLE_LOG
#   if LOG_OUTPUT_EXTINFO

static inline const char* cutFileName(const char* s){
    uint32_t len = utils_strlen(s);
while((*(s + len) != 0x5C && *(s + len) != 0x2F) && len > 0) len--;
if(len <= 0)return "";
    return (s + len + 1);
```

```
};

#       define LOGE(fmt, ...) logOutput( LOG_LEVEL_ERROR,   LOG_TAG,
STRCONCAT( "%s, line:%d, ", fmt), cutFileName(__FILE__), __LINE__, ##__VA_ARGS__)
#       define   LOGW(fmt,   ...)   logOutput(   LOG_LEVEL_WARNING,   LOG_TAG,
STRCONCAT( "%s, line:%d, ", fmt), cutFileName(__FILE__), __LINE__, ##__VA_ARGS__)
#       define LOGI(fmt, ...) logOutput( LOG_LEVEL_INFO,    LOG_TAG,
STRCONCAT( "%s, line:%d, ", fmt), cutFileName(__FILE__), __LINE__, ##__VA_ARGS__)
#       define LOGD(fmt, ...) logOutput( LOG_LEVEL_DEBUG,   LOG_TAG,
STRCONCAT( "%s, line:%d, ", fmt), cutFileName(__FILE__), __LINE__, ##__VA_ARGS__)
#       define LOG_PRINT(fmt, ...) logData( fmt, ##__VA_ARGS__)
#   else
#       define LOGE(fmt, ...) logOutput( LOG_LEVEL_ERROR,   LOG_TAG,        fmt,
##__VA_ARGS__)
#       define  LOGW(fmt,  ...)  logOutput(  LOG_LEVEL_WARNING,  LOG_TAG,  fmt,
##__VA_ARGS__)
#       define LOGI(fmt, ...) logOutput( LOG_LEVEL_INFO,    LOG_TAG,        fmt,
##__VA_ARGS__)
#       define LOGD(fmt, ...) logOutput( LOG_LEVEL_DEBUG,   LOG_TAG,        fmt,
##__VA_ARGS__)
#       define LOG_PRINT(fmt, ...) logData( fmt, ##__VA_ARGS__)
#   endif
#else
#   define LOGE(fmt, ...) do {} while(0)
#   define LOGW(fmt, ...) do {} while(0)
#   define LOGI(fmt, ...) do {} while(0)
#   define LOGD(fmt, ...) do {} while(0)
#   define LOG_PRINT(fmt, ...) do {} while(0)
#endif

#if LOG_DUMP_ENABLED
#   define DUMP_BUF(b,l)    log_dump(NULL,b,l,0)
#   define LOG_BUF(t,b,l)   log_dump(t,b,l,0)
#else
#   define DUMP_BUF(b,l)    do {} while(0)
#   define LOG_BUF(t,b,l)   do {} while(0)
#endif//LOG_DUMP_ENABLED


#ifdef __cplusplus
extern "C" {
#endif

void log_config(bool enable,uint8_t exoutput,uint8_t level,uint16_t bufsize);
```

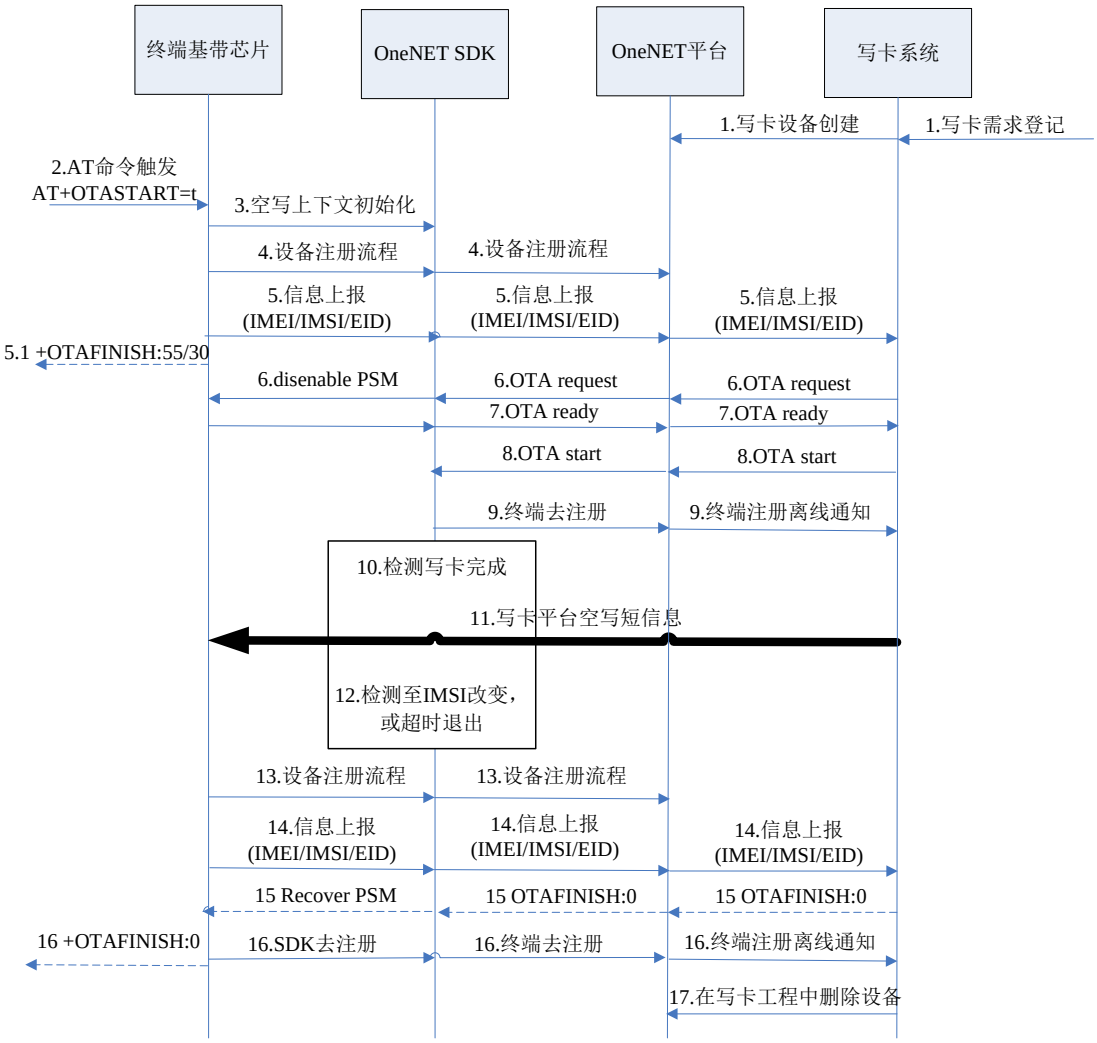
```
#if LOG_DUMP_ENABLED
void log_dump(const char* title,const uint8_t * buffer,int length,int indent);
#endif//LOG_DUMP_ENABLED


#ifdef __cplusplus
};
#endif

#endif
```

附录D （资料性附录）终端空中写卡流程说明

终端的空中写卡，指通过空中升级的方式完成嵌入式 SIM 卡（eSIM）从临时码号至正式码号的变更，即通过 OneNET 以及网络信令的交互完成用户卡的个人信息变更。
集成 OneNET SDK 的设备则具备空写（OTA）能力，空写流程由 AT 命令 AT+OTASTART 触发，由返回值+OTAFINISH:N 结束，AT+OTASTART 作为申请空中写卡任务的触发，不支持与 OneNET 的用户应用连接同时并发使用，即不与 AT+MIPLCREATE、AT+MIPLOPEN 发起的连接同时工作。
首次进行空写的设备（OTASTATEFLAG=0）执行完整的空写流程如图 D-1 所示：



图D-1 终端首次空写流程

流程解析：

1. 中移物联网公司应用空中写卡技术支持对物联网卡从临时码号至正式码号的单向变更。写卡需求登记则是写卡需求方向物联网公司芯片管理平台提交登记待写卡设备信息和目标码号信息。芯片管理平台在验证信息无误和芯片流转过程完整后会在 OneNET 平台的专项写卡工程中创建对应写卡虚拟设备。

2. 终端侧的空写流程由 AT 命令或对应 API 接口触发，其中 AT 命令格式为 AT+OTASTART=t。 t 为本次在 OneNET 上等待写卡请求的时间，单位是秒，如果超时未收到平台任何命令，则从 OneNET 上离线，注销初始化的上下文信息，并对外返回写卡完成超时+OTAFINISH:55 的指示。超时时间 t 根据用户应用场景可配置，为正确接收写卡系统下发的写卡命令，建议设置大于等于 20 秒，典型值为 120 秒。

3. - 4. 设备执行空写初始化时使用 EID+IMSI 构成 DeviceName 在 OneNET 上注册上线。

5. SDK 使用 objectID：3351 的对象管理设备信息，SDK 在初始化时，通过终端接口查询 eSIM 的 IMSI 和 EID 信息，以及终端设备 IMEI 号码，并在平台执行发现时上报设备侧的 IMSI/IMEI/EID。该终端信息在 OneNET 平台下发订阅 objectID：3351 后立即上报，其中，IMEI 上报使用 ResourceID：2、IMSI 上报使用 ResourceID：3、EID 上报使用 ResourceID：4。

5.1 此步骤为空写过程的超时退出，即若未收到任何平台命令，则超时终止本次写卡流程，

对外返回设备未发起写卡超时退出指示，+OTAFINISH:55。此外，当芯片管理平台侧存在该设备的写卡需求登记，在收到终端的上报信息后将直接发起后续流程。而对未登记写卡需求的设备，或登记信息有误，则平台不会触发后续写卡流程，且下发 OTAFINISH:3 系列命令，则终端退出本次写卡流程，对外返回平台侧未有该设备的写卡请求登记或登记异常，+OTAFINISH: 30。

6.7.8.9.该部分为执行写卡动作前平台侧与终端侧通过OneNET的双向验证和交互流程，保证写卡执行的可靠性和安全性。交互过程中，对NBiot终端的PSM策略进行了查验，若为使能状态，则暂时关闭终端PSM功能，在空写退出时恢复。SDK使用objectID：2051的对象管理设备平台命令和回复。平台下行命令使用对ResourceID：2的写（write）命令触发，终端上行响应使用对ResourceID：3的通知（notify）完成。命令格式和响应方式如下

序号	命令名称	下行命令	上行响应
1	写卡准备命令	OTAREQ:XXXXXX	
2	写卡准备就绪响应		OTAREADY:XXXXXX
3	写卡短信触发通知	OTASTART:XXXXXX	
4	写卡完成指示命令	OTAFINISH:N: XXXXXX	

①其中，XXXXXX为写卡系统平台下发的校验码序列，终端上行OTAREADY响应时应携带与OTAREQ相同的命令序列码，写卡系统平台在校验通过会，调度触发写卡短信。

②写卡完成命令指示码N用于指示当前写卡完成结果或平台状态，具体内容如流程描述介绍。

10.12. SDK 在写卡执行阶段检测终端的写卡完成状况，推荐使用轮询 IMSI 号码变化的方式，或其他底层同步/异步通知机制。DEMO 中以 10S 间隔检测 IMSI 变更，若始终码号无变化，则在持续查询 20 次后超时退出。

11.该步骤为 eSIM 写卡的空中写卡底层执行动作，临时码号会接收到物联网写卡平台下发的空中写卡短信息，通过 PP-Download 标准协议执行卡数据更新操作，并在个人数据更新为正式码号后，重新附属、激活无线网络。

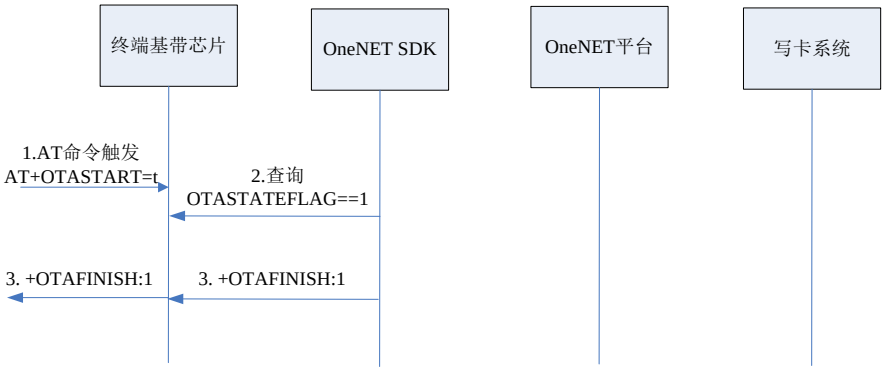
13.14.SDK 在第 12 步流程退出后，重新连接 OneNET 并上报当前卡数据信息，过程与第 4,5 步骤相同。

15.芯片管理平台根据终端上报的信息判断空中写卡的完成状态，若已完成空写，且信息验证无误则对终端下发空中写卡完成成功命令（OTAFINISH:0），提示空写流程完结，空写产品生命周期结束。当终端成功完成写卡后，需由终端标记固化完成状态，作为本设备已完成空中写卡的标志，当下一次收到外部 AT+OTASTART=t 时，则无需再与 OneNET 交互，直接返回终端已写卡成功指示（+OTAFINISH：1）。若平台判断写卡失败，则下发空中写卡完成失败命令（OTAFINISH:2）。收到平台侧的空写成功和空写失败命令后，则分别通过 AT 接口对外返回（+OTAFINISH：0）和（+OTAFINISH：57）结果作为指示。若未收到平台任何结果命令，则等待 SDK 超时退出，并返回写卡后的超时退出指示（+OTAFINISH：56）。

16.空写完成后，从 OneNET 平台下线，根据前文描述对用户返回对应+OTAFINISH：N 写卡完成状态指示。注销空写 SDK 的上下文。

对于已成功完成过空中写卡的终端设备（OTASTATEFLAG=1），当再次重复触发写卡时，则直接返回已成功完成写卡的指示（+OTAFINISH:1），无需再与 OneNET 和芯片管理平台交互。对于该状态标志，需开放另一个 AT 命令 AT 命令提供更改和查询，AT+OTASTATE=n、AT+OTASTATE=?、AT+OTASTATE?

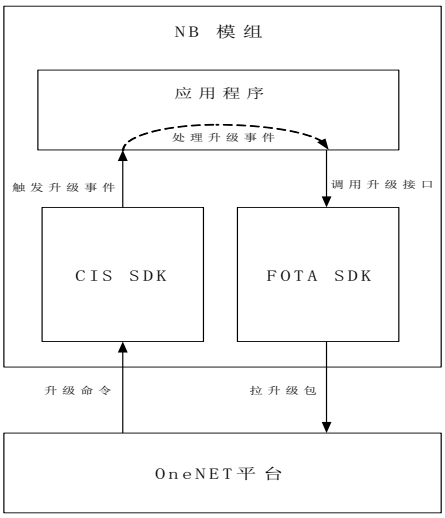
对已空写成功的设备进行再次反复写卡的流程如图 D-2 所示。



图D-2再次空写流程

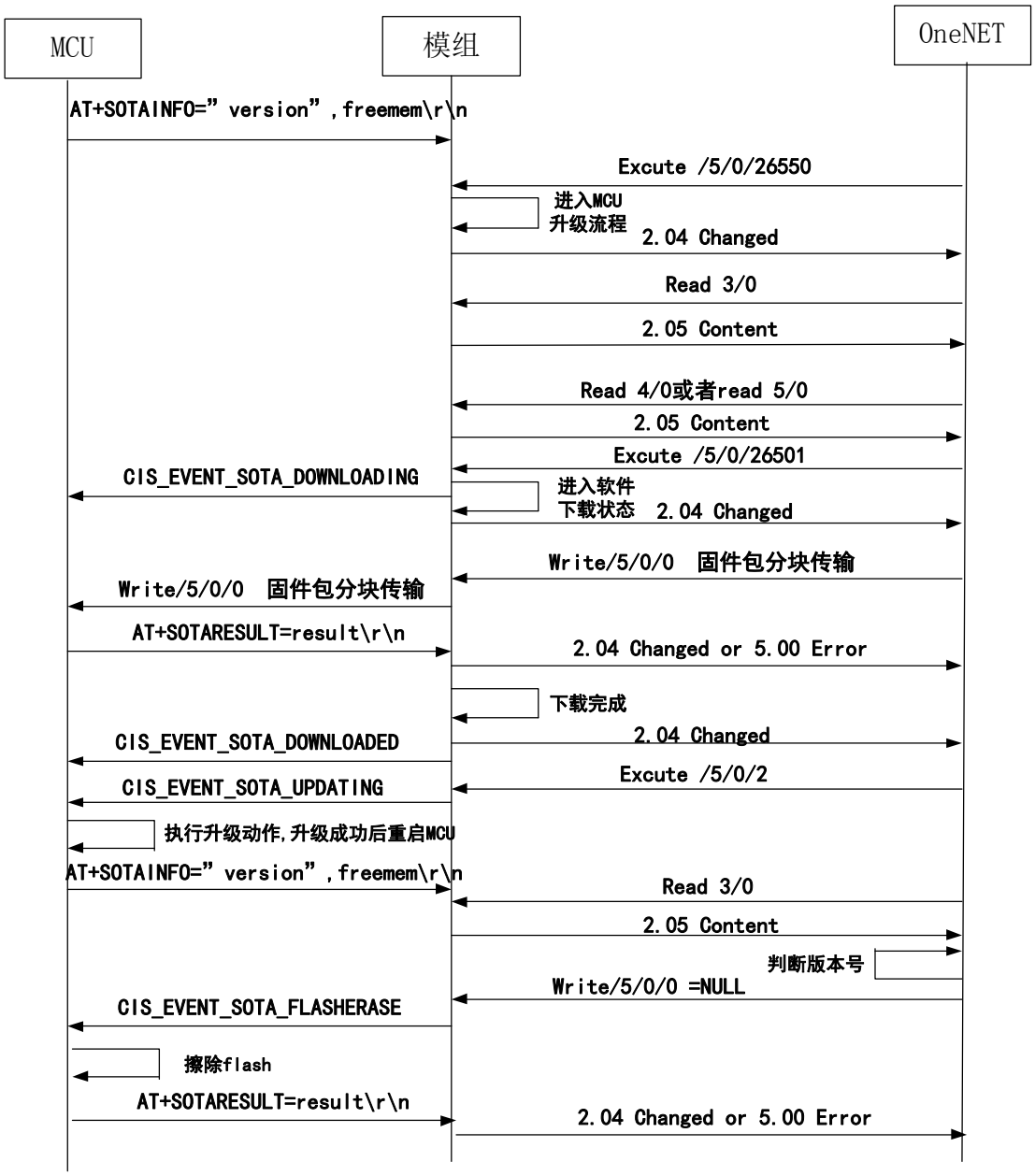
附录E （资料性附录）非自有FOTA通道升级触发说明

非自有FOTA通道升级触发的机制如下图所示，OneNET平台检测到有新的升级包，将会向CIS SDK发送升级命令，SDK收到升级命令后会触发CIS_EVENT_FIRMWARE_TRIGGER事件，模组应用程序收到该触发事件后调用FOTA SDK提供的升级接口完成和OneNET平台的升级交互。



图E-1非自有FOTA通道升级触发机制

附录F （资料性附录）SOTA通道升级流程说明



MCU需要实现以下的动作

1. MCU 上电后，在登录平台之前，应该先使用 AT+SOTAINFO 指令将 MCU 当前的版本号和升级空间大小发送给模组；
2. 当收到 CIS_EVENT_SOTA_DOWNLOADING 事件时，说明平台后面将要下发升级包，需要做好升级准备；
3. 当收到 CIS_EVENT_SOTA_DOWNLOAED 事件时，说明软件包软件包已经下载完成；
4. 当收到 CIS_EVENT_SOTA_FLASHERASE 事件时，需要擦除升级用的 flash，并通过 AT+SOTACB 指令将擦除结果返回给模组；
5. 当收到写 5/0/0 请求时，需要将收到的升级报文按序写入升级 flash 中，并通过

AT+SOTARESULT 指令将写入结果返回给模组;

6. 当收到 CIS_EVENT_SOTA_UPDATING 事件时，需要执行升级动作,如果升级失败则需要调用 AT+SOTARESULT 指令通告模组。