**OneOS OTA组件移植使用说明**

Version: 1.0

Date: 2023-08-10



[[iot.10086.cn](http://www.spreadtrum.com)](http://iot.10086.cn/)

**修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **编号** | **版本号** | **修订日期** | **修订描述** | **修改人** |
| 1 | 1.0 | 2023-08-10 | 平台，APP和BootLoader合并初始版本 | OneOS Team |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |

目 录

[1 概述 4](#_Toc142557954)

[2 OTA APP移植 5](#_Toc142557955)

[2.1 静态代码组织 5](#_Toc142557956)

[2.2 接口说明 6](#_Toc142557957)

[2.2.1 系统适配接口 6](#_Toc142557958)

[2.2.2 OTA接口 17](#_Toc142557959)

[2.2.3 功能控制定义 22](#_Toc142557960)

[3 BootLoader适配 25](#_Toc142557961)

[3.1 资源 25](#_Toc142557962)

[3.2 数据结构 25](#_Toc142557963)

[3.3 接口描述 26](#_Toc142557964)

[3.4 返回码表 28](#_Toc142557965)

[3.5 适配流程 29](#_Toc142557966)

[3.6 参数配置 31](#_Toc142557967)

[3.7 注意事项 32](#_Toc142557968)

[3.8 升级主流程示例 33](#_Toc142557969)

[4 升级操作流程 34](#_Toc142557970)

[4.1 云平台创建项目 34](#_Toc142557971)

[4.2 本地配置 36](#_Toc142557972)

[4.3 生成差分包 36](#_Toc142557973)

[4.3.1 本地打包 36](#_Toc142557974)

[4.3.2 平台差分 37](#_Toc142557975)

[4.4 升级测试 41](#_Toc142557976)

[4.4.1 一键升级demo 41](#_Toc142557977)

[4.4.2 查询是否有新版本 42](#_Toc142557978)

[4.4.3 有新版本后下载 42](#_Toc142557979)

[5 参考资料 43](#_Toc142557980)

[6 FAQ 44](#_Toc142557981)

# 概述

远程升级（Firmware Over The Air）可通过无线网络实现固件升级。OneOS的OTA采用差分升级算法，采用此方式可极大降低升级包的大小。完整的OTA解决方案包括云平台及端侧软件两部分。

云平台服务提供制作差分包、差分包测试及发布、版本管理、查询升级日志等功能。

OTA端侧方案包括BootLoader和APP，BootLoader负责保存引导、差分还原，APP负责制作差分包压缩文件（前置版本压缩包、目标版本压缩包）、版本监测、固件差分包下载、版本校验、日志上报。

本文档包含OTA APP组件移植说明，及BootLoader的适配说明，以及云平台OTA服务的使用说明。

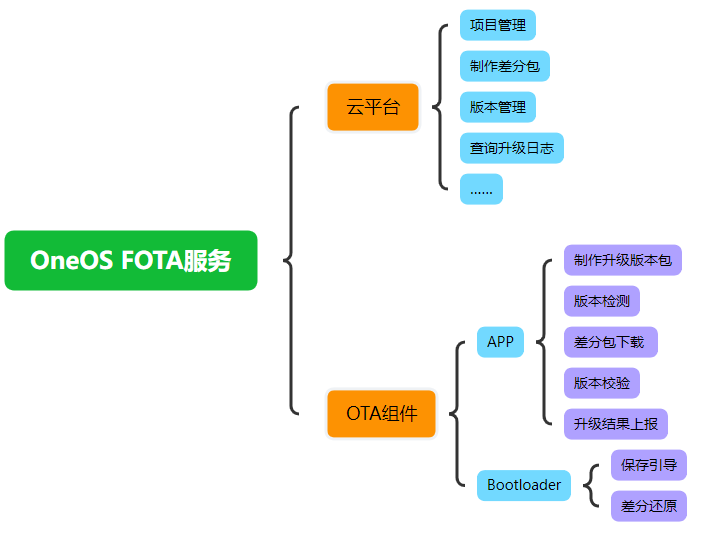


图1.1 OTA组成

# OTA APP移植

## 静态代码组织

CMIOT OTA 代码分为以下几部分：

1. lib目录：OTA所使用的库文件，需要链接到烧写的bin文件中
2. os\_port适配目录：包含移植时需要适配的系统接口及定义
3. ota目录：OTA APP主体代码，包含用户调用接口声明及OTA lib依赖接口实现
4. Port目录：OTA APP依赖OS的部分，使用os\_port中定义的接口
5. Source目录：包含了用户使用的接口，其中部分需要用户自行修改。另外pack子目录存放用于制作差分升级包压缩文件的脚本
6. port\_demo：存放已适配的平台demo代码
7. oneos\_config.h文件用于组件配置，需要用户根据实际情况修改产品ID及密钥

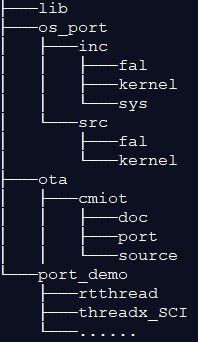


图2.1 OTA组件代码结构

具体使用说明请参照以下章节。

## 接口说明

### 系统适配接口

需要适配的接口包括fal(flash操作接口)，kernel(内存、task等)，sys(socket相关内容)三部分，以下接口如果未适配，则需要用户在新平台上自行实现。

#### flash接口

位于os\_port\inc\fal\os\_fal\_part.h、os\_fal.h

1. **重要结构体**

typedef struct os\_fal\_part\_info

{

char name[OS\_FAL\_DEV\_NAME\_MAX]; /\* flash分区名，必须要定义 \*/

char flash\_name[OS\_FAL\_DEV\_NAME\_MAX]; /\* 该分区所属flash名 \*/

os\_uint32\_t offset; /\* 该分区在flash的偏移地址，必须要定义 \*/

os\_uint32\_t size; /\* 该分区字节大小，必须要定义 \*/

os\_uint32\_t flags; /\* 未使用 \*/

}os\_fal\_part\_info\_t; /\* 分区信息表 \*/

struct os\_fal\_flash /\* flash信息表 \*/

{

char name[OS\_FAL\_DEV\_NAME\_MAX]; /\* flash名 \*/

os\_uint32\_t page\_size; /\* flash最小的读写字节大小，必须要定义 \*/

os\_uint32\_t block\_size; /\* flash最小擦除字节大小，必须要定义 \*/

os\_uint32\_t capacity; /\* flash容量字节大小，必须要定义 \*/

void \*priv;

};

typedef struct os\_fal\_part

{

os\_fal\_part\_info\_t \*info; /\* 指向分区信息表，必须要定义 \*/

os\_fal\_flash\_t \*flash; /\* 指向包含本分区的flash信息表，必须要定义 \*/

os\_uint32\_t flags; /\* 未使用 \*/

}os\_fal\_part\_t;

若平台不支持FAL，可以自定义os\_fal\_part\_t结构体变量，手动填入必须要定义的成员，实现相关接口。

1. **os\_fal\_part\_find**

该函数用于根据分区名获取flash分区信息。原型如下：

fal\_part\_t \* os\_fal\_part\_find(const char \*name);

| 参数 | 说明 |
| --- | --- |
| name | 分区名，目前只会使用“download”分区，定义在cmiot\_cfg.h CMIOT\_ONEOS\_UPDATE. |
| 返回 | 说明 |
| fal\_part\_t \* | 分区详细信息,详见结构体定义 |
| NULL | 获取失败 |

1. **os\_fal\_part\_read**

该函数用于读取flash分区特定地址数据。原型如下：

int os\_fal\_part\_read(os\_fal\_part\_t \*part, os\_uint32\_t addr, os\_uint8\_t \*buf, os\_size\_t size);

| 参数 | 说明 |
| --- | --- |
| part | Flash分区结构体 |
| addr | 读取数据偏移地址（相对于part分区首地址偏移） |
| buf | 数据存储缓冲区 |
| size | 读取数据长度 |
| 返回 | 说明 |
| >0 | 读取数据长度,一般为size长度 |
| 其他 | 读取数据失败 |

1. **os\_fal\_part\_write**

该函数用于向flash分区特定地址写入数据。原型如下：

int os\_fal\_part\_write(os\_fal\_part\_t \*part, os\_uint32\_t addr, const os\_uint8\_t \*buf, os\_size\_t size);

| 参数 | 说明 |
| --- | --- |
| part | Flash分区结构体 |
| addr | 写数据偏移地址（相对于part分区首地址偏移） |
| buf | 数据存储缓冲区 |
| size | 写入数据长度 |
| 返回 | 说明 |
| >0 | 写入数据长度,一般为size长度 |
| 其他 | 写入数据失败 |

1. **os\_fal\_part\_erase**

该函数用于擦除flash分区某块地址。原型如下：

int os\_fal\_part\_erase(os\_fal\_part\_t \*part, os\_uint32\_t addr, os\_size\_t size);

| 参数 | 说明 |
| --- | --- |
| part | Flash分区结构体 |
| addr | 擦除数据偏移地址（相对于part分区首地址偏移） |
| size | 擦除块长度 |
| 返回 | 说明 |
| <0 | 擦除失败 |
| 其他 | 擦除成功 |

#### kernel列表

位于os\_port\inc\kernel\os\_assert.h

1. **OS\_ASSERT**

ASSERT异常处理函数。

void OS\_ASSERT(os\_bool\_t condition);

| 参数 | 说明 |
| --- | --- |
| condition | 异常判断条件 |
| 返回 | 说明 |
| 无 | 无 |

位于os\_port\inc\kernel\os\_memory.h

1. **os\_malloc**

从heap中申请分配指定大小byte的内存空间。

void \*os\_malloc(os\_size\_t size)

| 参数 | 说明 |
| --- | --- |
| Size | 申请内存大小，单位byte |
| 返回 | 说明 |
| Null | 申请失败 |
| 非null | 分配的内存首地址 |

1. **os\_free**

释放指定地址的内存

void os\_free(void \*ptr)

| 参数 | 说明 |
| --- | --- |
| ptr | 待释放内存地址 |
| 返回 | 说明 |
| 无 | 无 |

1. **os\_realloc**

调整指定内存块大小，如果size为0则等同于os\_free, 如果指定内存地址为null则等同于os\_malloc.

void \*os\_realloc(void \*ptr, os\_size\_t size)

| 参数 | 说明 |
| --- | --- |
| ptr | 指定内存块地址 |
| Size | 调整后大小 |
| 返回 | 说明 |
| Null | 调整失败 |
| 其他 | 调整后的内存地址 |

1. **os\_calloc**

从heap申请一块内存组，每个元素内存大小相同，并初始化为0.

void \*os\_calloc(os\_size\_t count, os\_size\_t size)

| 参数 | 说明 |
| --- | --- |
| count | 申请的内存组的元素数量 |
| Size | 每个元素内存大小 |
| 返回 | 说明 |
| Null | 申请失败 |
| 非null | 分配的内存首地址 |

位于os\_port\inc\kernel\os\_task.h

1. **os\_task\_msleep**

使当前任务进入休眠状态

os\_err\_t os\_task\_msleep(os\_uint32\_t ms)

| 参数 | 说明 |
| --- | --- |
| Ms | 休眠时间，单位ms |
| 返回 | 说明 |
| 0 | 休眠成功 |
| 非0 | 休眠失败 |

位于os\_port\inc\kernel\os\_util.h

1. **os\_kprintf**

该函数用于打印日志，OTA日志依赖此接口，其函数原型如下：

void os\_kprintf(const char \*fmt, ...);

| 参数 | 说明 |
| --- | --- |
| fmt | 标准打印参数 |
| 返回 | 说明 |
| 无 | 无 |

位于os\_port\inc\kernel\ring\_buff.h

1. **Ring buffer接口**

目前ring buffer已经实现在\src\kernel\ring\_buff.c。如果需要修改，请实现所有的接口。

#### sys列表

socket接口位于os\_port\inc\sys\socket.h

1. **os\_socket**

该函数用于创建一个socket链接。原型如下：

int os\_socket(int domain, int type, int protocol);

| 参数 | 说明 |
| --- | --- |
| domain | 协议族，一般使用AF\_INET |
| type | Socket类型，包括以下三类：  #define SOCK\_STREAM 1  #define SOCK\_DGRAM 2  #define SOCK\_RAW 3 |
| protocol | 通常为0 |
| 返回 | 说明 |
| >=0 | 创建成功的socket描述符fd |
| -1 | 创建失败 |

1. **os\_closesocket**

该函数用于销毁一个socket链接。原型如下：

int os\_closesocket(int fd);

| 参数 | 说明 |
| --- | --- |
| fd | socket描述符 |
| 返回 | 说明 |
| 0 | 销毁成功 |
| 其他 | 销毁失败，查看返回值确定原因 |

1. **os\_connect**

该函数用于和远端建立tcp连接。原型如下：

int os\_connect(int fd, const struct sockaddr \*name, socklen\_t namelen);

| 参数 | 说明 |
| --- | --- |
| fd | socket描述符 |
| name | 远端服务器地址，详见struct sockaddr |
| namelen | 地址长度 |
| 返回 | 说明 |
| 0 | 连接成功 |
| -1 | 连接失败 |

1. **os\_send**

该函数用于发送tcp数据。原型如下：

int os\_send(int fd, const void \*data, os\_size\_t size, int flags);

| 参数 | 说明 |
| --- | --- |
| fd | socket描述符 |
| data | 待发送数据buffer |
| size | 待发送数据字节长度 |
| flags | 一般为0 |
| 返回 | 说明 |
| >0 | 发送数据长度 |
| -1 | 发送失败 |

1. **os\_sendto**

该函数用于发送udp数据。原型如下：

int os\_sendto(int fd, const void \*data, os\_size\_t size, int flags, const struct sockaddr \*to, socklen\_t tolen);

| 参数 | 说明 |
| --- | --- |
| fd | socket描述符 |
| data | 待发送数据buffer |
| size | 待发送数据字节长度 |
| flags | 一般为0 |
| to | 数据发送目的地 |
| tolen | 地址长度 |
| 返回 | 说明 |
| >0 | 发送数据长度 |
| -1 | 发送失败 |

1. **os\_setsockopt**

该函数用于设置socket选项。原型如下：

int os\_setsockopt(int fd, int level, int optname, const void \*optval, socklen\_t optlen);

|  |  |
| --- | --- |
| 参数 | 说明 |
| fd | socket描述符 |
| level | 选项定义的层次,目前仅使用SOL\_SOCKET。 |
| optname | 待设置选项,目前仅使用SO\_RCVTIMEO |
| optval | 指向存放选项值缓冲区 |
| optlen | 缓冲区长度 |
| 返回 | 说明 |
| 0 | 设置成功 |
| 其他 | 设置失败 |

1. **os\_recv**

该函数用于接收tcp数据。原型如下：

int os\_recv(int fd, void \*mem, os\_size\_t len, int flags);

| 参数 | 说明 |
| --- | --- |
| fd | socket描述符 |
| mem | 接收数据buffer |
| len | 接收数据buffer字节长度 |
| flags | 一般为0 |
| 返回 | 说明 |
| >0 | 接收数据长度 |
| -1 | 接收失败 |

1. **os\_recvfrom**

该函数用于接收udp数据。原型如下：

int os\_recvfrom(int fd, void \*mem, os\_size\_t len, int flags, struct sockaddr \*from, socklen\_t \*fromlen);

| 参数 | 说明 |
| --- | --- |
| fd | socket描述符 |
| mem | 接收数据buffer |
| len | 接收数据buffer字节长度 |
| flags | 一般为0 |
| from | 数据来源地址 |
| fromlen | 作为输入时，fromlen常置为sizeof(struct sockaddr)；当输出时，fromlen包含实际存入buf中的数据字节数 |
| 返回 | 说明 |
| >0 | 接收数据长度 |
| -1 | 接收失败 |

1. **os\_gethostbyname**

该函数用于根据主机名获取主机地址。原型如下：

struct hostent \* os\_gethostbyname(const char \*name);

| 参数 | 说明 |
| --- | --- |
| name | 主机名字符串 |
| 返回 | 说明 |
| struct hostent \* | 主机地址,详见结构体定义 |
| NULL | 获取失败 |

### OTA接口

#### API列表

相关接口位于ota\cmiot\source\inc\cmiot\_cfg.h、cmiot\_main.h、cmiot\_id.h

注：以下说明中“**无需调用**”表示该接口无需OTA使用人员主动调用，OTA组件在需要的时候会自动调用，使用者只用根据自己的需求来重写。“**用户可更改**”表示用户可以根据平台实际情况移植时修改。

| **文件名** | **接口** | **说明** |
| --- | --- | --- |
| cmiot\_cfg.h | cmiot\_get\_try\_time | OTA组件socket的接收超时时间，单位毫秒，无需调用 |
| cmiot\_cfg.h | cmiot\_get\_try\_count | OTA组件在解码服务器的数据失败时，或端侧处理失败时，重新向服务器请求的次数，无需调用 |
| cmiot\_cfg.h | cmiot\_get\_utc\_time | 当前的UTC时间戳，暂时未用到，目前可保持默认值，无需调用 |
| cmiot\_cfg.h | cmiot\_download\_name | DOWNLOAD分区的名字，**用户不可更改**，无需调用 |
| cmiot\_cfg.h | cmiot\_get\_product\_id | 获取云平台对应产品ID，在oneos\_config.h中配置，无需调用 |
| cmiot\_cfg.h | cmiot\_get\_product\_sec | 获取云平台对应产品密钥，在oneos\_config.h中配置，无需调用 |
| cmiot\_main.h | cmiot\_upgrade | 检查新版本并下载固件包,**供用户调用** |
| cmiot\_main.h | cmiot\_report\_upgrade | 上报升级结果,**供用户调用** |
| cmiot\_main.h | cmiot\_check\_version | 检查是否有新版本,**供用户调用** |
| cmiot\_main.h | cmiot\_only\_download | 存在新版本后调用此接口下载新版本,**供用户调用** |
| cmiot\_id.h | cmiot\_get\_uniqueid | 获取端侧的设备ID，用于OTA升级时区分不同的终端设备，**需要用户根据实际情况修改** |

#### API说明

1. **cmiot\_get\_try\_time**

该函数用于获取socket的接收超时时间，单位毫秒，可根据实际网络情况调整，其函数原型如下：

cmiot\_uint32 cmiot\_get\_try\_time(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| cmiot\_uint32 | socket接收超时时间 |

1. **cmiot\_get\_try\_count**

该函数用于获取在解码服务器的数据失败时，或端侧处理失败时，重新向服务器请求的次数 ，函数原型如下：

cmiot\_uint8 cmiot\_get\_try\_count(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| cmiot\_uint8 | 重试次数 |

1. **cmiot\_get\_utc\_time**

该函数用于获取当前的UTC时间戳，函数原型如下：

cmiot\_uint32 cmiot\_get\_utc\_time(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| cmiot\_uint32 | 当前UTC时间戳 |

1. **cmiot\_download\_name**

该函数用于获取DOWNLOAD分区的名字，该分区存储下载的升级包，该函数原型如下：

cmiot\_char \*cmiot\_download\_name(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| cmiot\_char \* | DOWNLOAD分区名字，不能为空 |

1. **cmiot\_get\_product\_id**

该函数用于获取OTA产品ID，定义在oneos\_config.h中，与OTA云平台上对应。该函数原型如下：

cmiot\_char \*cmiot\_get\_product\_id(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| cmiot\_char \* | 产品ID字符串 |

1. **cmiot\_get\_product\_sec**

该函数用于获取OTA产品密钥，定义在oneos\_config.h中，与OTA云平台上对应。该函数原型如下：

cmiot\_char \*cmiot\_get\_product\_sec(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| cmiot\_char \* | 产品密钥字符串 |

1. **cmiot\_upgrade**

该函数检测新版本，如果新版本存在会下载固件升级包，其函数原型如下：

cmiot\_int8 cmiot\_upgrade(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| E\_CMIOT\_SUCCESS | 检测到新版本并下载成功，重启终端后会进入还原流程 |
| E\_CMIOT\_FAILURE | 无网络等异常原因 |
| E\_CMIOT\_NOMEMORY | 内存不足，初始化失败 |
| E\_CMIOT\_LAST\_VERSION | 当前版本已经是最新 |
| E\_CMIOT\_PAKAGE\_TOO\_LARGE | 检测到新版本，但是版本数据过大，无法升级 |

1. **cmiot\_report\_upgrade**

该函数用于上报升级结果，其函数原型如下：

cmiot\_int8 cmiot\_report\_upgrade(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| E\_CMIOT\_SUCCESS | 升级结果上报成功 |
| E\_CMIOT\_FAILURE | 无网络等异常原因上报失败 |
| E\_CMIOT\_NO\_UPGRADE | 未检测到升级，无需上报 |

1. **cmiot\_check\_version**

该函数用于检测是否存在新版本，其函数原型如下：

cmiot\_int8 cmiot\_check\_version(char version[CMIOT\_VERSION\_NAME\_MAX]);

| 参数 | 说明 |
| --- | --- |
| version | 如果新版本检测成功，存放新版本号，字符长度最大CMIOT\_VERSION\_NAME\_MAX |
| 返回 | 说明 |
| E\_CMIOT\_SUCCESS | 检测成功 |
| E\_CMIOT\_FAILURE | 无网络等异常原因 |
| E\_CMIOT\_NOMEMORY | 内存不足，初始化失败 |
| E\_CMIOT\_LAST\_VERSION | 当前版本已经是最新 |
| E\_CMIOT\_PAKAGE\_TOO\_LARGE | 检测到新版本，但是版本数据超限，无法升级 |
| E\_CMIOT\_NEW\_VERSION\_EXIST | 新版本已下载，请重启升级 |

1. **cmiot\_only\_download**

在检测到新版后，该函数用于下载新版本差分包，其函数原型如下：

cmiot\_int8 cmiot\_only\_download(void);

| 参数 | 说明 |
| --- | --- |
| 无 | 无 |
| 返回 | 说明 |
| E\_CMIOT\_SUCCESS | 下载成功，重启终端后会进入还原流程 |
| E\_CMIOT\_FAILURE | 无网络等异常原因 |
| E\_CMIOT\_NOMEMORY | 内存不足，初始化失败 |

1. **cmiot\_get\_uniqueid**

该函数返回端侧的设备ID给调用者，用于OTA升级时区分不同的终端设备，需要用户根据实际情况修改，其函数原型如下：

void cmiot\_get\_uniqueid(cmiot\_char \*uid)

| 参数 | 说明 |
| --- | --- |
| \*uid | 存储本设备ID的buffer |
| 返回 | 说明 |
| 无 | 无 |
| 其他 | uid长度不应超过CMIOT\_MID\_MAXLEN（31） |

### 功能控制定义

相关控制宏定义于oneos\_config.h

#define CMIOT\_DEFAULT\_TRY\_TIME 5000 /\* socket连接超时值，单位ms，可不修改 \*/

#define CMIOT\_DEFAULT\_NETWORK\_PROTOCOL 2 /\* 网络类型选择：1.udp, 2.http，不建议修改\*/

#define CMIOT\_FOTA\_ALGORITHM 4 /\* 差分还原算法，不能修改！ \*/

#define CMIOT\_FOTA\_AVAILABLE\_RAM\_SIZE 0x80000 /\* OTA还原可用RAM空间，平台使用，根据实际情况修改\*/

#define CMIOT\_DEFAULT\_SECTOR\_SIZE 0x10000 /\* flash最小擦除大小值，单位byte，根据实际情况修改\*/

#define CMIOT\_DEFAULT\_SEGMENT\_SIZE\_INDEX 5 /\* 差分包下载块大小，不建议修改 \*/

/\* **以下两项要根据平台配置修改**，详见第4.1章 \*/

#define CMIOT\_FOTA\_SERVICE\_PRODUCT\_ID "8BURTiV0X4"

#define CMIOT\_FOTA\_SERVICE\_PRODUCT\_SEC "2E32F161BC4C3D9C325B1D2CAD726DC1"

#define CMIOT\_UPDATE\_SIZE 0xA0000 /\* 下载差包分区大小，平台使用，根据实际情况修改\*/

#define CMIOT\_FOTA\_CONTRAST\_BLOCK\_SIZE 64 /\*差分包对比大小，单位KB，需要和还原算法一致，不建议修改\*/

#define CMIOT\_FIRMWARE\_VERSION "1.0" /\* 指示版本号，升级只能从低版本往高版本方向进行，如1.1->1.2 \*/

/\*以下配置需要进行升级的bin文件及其对应地址，按以下格式新增或删除bin文件，最多支持20个bin文件进行升级，在打包升级包时会使用，根据实际情况修改，详见4.3.1\*/

#define CMIOT\_FOTA\_FILENAME1 "mmi\_res\_128x128.bin"

#define CMIOT\_FOTA\_ADDRESS1 0x90000004

#define CMIOT\_FOTA\_FILE\_MORE1

#define CMIOT\_FOTA\_FILENAME2 "UIS8910\_ROM\_16MB\_DS\_DEBUG\_delta\_nv.bin"

#define CMIOT\_FOTA\_ADDRESS2 0x90000100

#define CMIOT\_FOTA\_FILE\_MORE2

#define CMIOT\_FOTA\_FILENAME3 "UIX8910\_UIS8910\_ROM\_16MB\_DS\_DEBUG\_cp.bin"

#define CMIOT\_FOTA\_ADDRESS3 0x90000101

#define CMIOT\_FOTA\_FILE\_MORE3

#define CMIOT\_FOTA\_FILENAME4 "UIX8910\_UIS8910\_ROM\_16MB\_DS\_DEBUG\_stone.bin"

#define CMIOT\_FOTA\_ADDRESS4 0x80000003

注：如果有更多的bin需要升级，仿照上面添加即可：

#define CMIOT\_FOTA\_FILE\_MOREx

#define CMIOT\_FOTA\_FILENAMExx “xxxxx”

#define CMIOT\_FOTA\_ADDRESSxx 0xxxxxxxxx

# BootLoader适配

## 资源

在bootloader阶段，需要依赖的资源有：

①内存，用于存放还原过程中的数据，通常内存空间越大越好；

②堆栈，程序必须运行在堆栈已经完成正确初始化的C环境下；

③flash操作接口，包括下载及升级相关的存储介质的读/写/擦除接口。

## 数据结构

1. Flash操作接口结构

typedef struct ota\_fal\_flash\_ops

{

ota\_int32 (\*init)(void);

ota\_int32 (\*read)(ota\_uint32 addr, ota\_uint8 \*buf, ota\_size\_t size);

ota\_int32 (\*write)(ota\_uint32 addr, const ota\_uint8 \*buf, ota\_size\_t size);

ota\_int32 (\*erase)(ota\_uint32 addr, ota\_size\_t size);

} ota\_fal\_flash\_ops\_t;

1. Flash硬件信息结构

typedef struct ota\_fal\_flash

{

ota\_int8 name[FAL\_NAME\_MAX]; /\* flash 名称 \*/

ota\_size\_t size; /\* flash 大小 \*/

ota\_size\_t erase\_size; /\* flash 最小擦除单位 \*/

ota\_fal\_flash\_ops\_t ops; /\* flash 读写擦操作函数 \*/

} ota\_fal\_flash\_t;

1. Flash分区信息结构

typedef struct ota\_fal\_part\_info

{

ota\_int8 name[FAL\_NAME\_MAX]; /\* 分区 名称 \*/

ota\_int8 flash\_name[FAL\_NAME\_MAX]; /\* 分区 所在flash名称 \*/

ota\_uint32 address; /\* 分区 地址 \*/

ota\_uint32 size; /\* 分区 大小 \*/

ota\_uint32 part\_type; /\* 分区 类型 \*/

} ota\_fal\_part\_info\_t;

1. Flash地址映射信息结构

typedef struct ota\_fal\_map

{

ota\_uint32 logic\_addr; /\* 分区 逻辑地址 \*/

ota\_uint32 phy\_addr; /\* 分区 物理地址 \*/

} ota\_fal\_map\_t;

## 接口描述

1. void ota\_printf(const ota\_int8 \*fmt, ...)

功能描述：LOG格式化输出接口，升级过程中的LOG信息，以及返回值通过该接口输出。

该接口需用户实现

1. void ota\_update\_proc\_bar(ota\_int32 percentage)

功能描述：升级进度条显示

输入参数：ota\_int32 percentage： 当前升级进度：0~100

该接口需用户实现，在升级过程中被调用，不需要应用代码主动调用。通常是将进度条通过屏幕显示，因此需要在该函数中实现屏幕显示进度条的功能。

1. ota\_int32 ota\_heap\_init(ota\_uint8 \*buf, ota\_size\_t buf\_size, ota\_size\_t block\_size)

功能描述：堆空间初始化接口

输入参数：ota\_uint8 \*buf：用于OTA还原升级的内存起始地址；

ota\_size\_t buf\_size：用于OTA还原升级的内存大小；

ota\_size\_t block\_size：内存块大小；

返回值： 0：成功，小于0：失败

当bootloader中堆不可用时，可以使用该接口使用内置堆管理。提供给还原升级的内存应尽可能大，除了系统栈所需内存，和预留静态占用的内存外，剩余内存可以都用于还原升级。该内存可以通过定义一个静态数组的方式进行预分配。 为了提高内存的利用率，block\_size不宜过大，buf\_size需与CMIOT\_FOTA\_AVAILABLE\_RAM\_SIZE保持一致，内置堆管理模块会对buf\_size 和block\_size进行合理性检查，当初始化返回失败时，根据提示log进行调整即可。

1. ota\_int32 ota\_fal\_init(ota\_fal\_flash\_t \*flash\_tab, ota\_size\_t flash\_tab\_size, ota\_fal\_part\_info\_t \*part\_tab, ota\_size\_t part\_tab\_size)

功能描述：注册分区信息

输入参数：ota\_fal\_flash\_t \*flash\_tab：flash信息，包含flash名称，大小，擦除大小，读写擦操作函数。

ota\_size\_t flash\_tab\_size：flash列表大小

ota\_fal\_part\_info\_t \*part\_tab：分区信息，包含分区名称，分区所在flash名称，分区起始地址，分区大小和分区类型

ota\_size\_t part\_tab\_size：分区信息表大小

返回值： 0：成功，小于0：失败

当需要进行地址映射时，分区表中的地址应当使用逻辑地址。

1. ota\_int32 ota\_fal\_map\_init(ota\_fal\_map\_t \*map\_table, ota\_size\_t map\_table\_size)

功能描述：分区地址映射初始化

输入参数：ota\_fal\_map\_t \*map\_table：逻辑地址与物理地址映射表

ota\_size\_t map\_table\_size：逻辑地址与物理地址映射表大小

返回值： 0：成功，小于0：失败

若某些分区使用逻辑地址，某些分区使用物理地址，则使用物理地址的分区，逻辑地址与物理地址保持一致即可。

1. ota\_int32 ota\_start\_up(ota\_uint32 try\_times)

功能描述：启动升级

输入参数：ota\_uint32 try\_time：升级重试次数。建议至少重试一次

返回值： 升级结果返回码

## 返回码表

表 1-1 返回码表

|  |  |
| --- | --- |
| 状态码 | 说明 |
| 1000 | 升级成功 |
| 1001 | 不需要升级 |
| 1002 | 内存不足 |
| 1003 | 无效升级包 |
| 1004 | 前置版本校验错误 |
| 1005 | 升级包类型错误 |
| 1006 | 升级包不存在 |
| 1007 | DL分区空间不足 |
| 1008 | 未找到分区 |
| 1009 | 分区读取错误 |
| 1010 | 分区写入错误 |
| 1011 | 分区擦除错误 |
| 1012 | 分区数据备份错误 |
| 1013 | 分区数据还原错误 |
| 1014 | 流信息初始化错误 |
| 1015 | 头信息初始化错误 |
| 1016 | 解压头错误 |
| 1017 | 解压数据体错误 |
| 1018 | 还原块长度错误 |
| 1019 | 差分包数据校验错误 |
| 1501 | 新APP块数据校验错误 |
| 1502 | 覆盖线差值计算错误 |
| 1503 | 构建新数据错误 |
| 1504 | DL备份区域擦除失败 |
| 1505 | 解析块还原编号错误 |
| 1506 | 解析块覆盖线数量错误 |
| 1507 | 新APP分区数据校验错误 |

## 适配流程

1. Flash操作接口

ota\_int32 (\*init)(void);

flash的初始化一般在bootloader硬件初始化阶段已完成，init接口为NULL；若需要在fal初始化中完成，则可以在init接口中完成flash的硬件初始化。成功返回0，失败返回-1。

ota\_int32 (\*read)(ota\_uint32 addr, ota\_uint8 \*buf, ota\_size\_t size);

数据读取接口，成功返回读取长度，失败返回-1

ota\_int32 (\*write)(ota\_uint32 addr, const ota\_uint8 \*buf, ota\_size\_t size);

数据写入接口，成功返回写入长度，失败返回-1

ota\_int32 (\*erase)(ota\_uint32 addr, ota\_size\_t size);

数据擦除接口，其中addr按block对齐，size不是block的整数倍时，按照最小对齐block擦除，成功返回0，失败返回-1

将操作函数，flash名称，容量，块大小（擦除大小）填写至ota\_fal\_flash\_t类型的表中。如果有多个不同的flash设备，根据实际情况，实现相应的操作函数，并将硬件信息添加到flash表中即可。

1. 填写分区表

ota\_fal\_part\_info\_t类型的分区表中包含了需要升级的“APP”分区以及“download”分区，除了download分区名称是固定的，其他“APP”的名称可以任意命名。每个分区，包含了分区名称，所在flash名称，起始地址（如果使用了逻辑地址需填写逻辑地址），分区大小，分区类型（“APP”必须为FAL\_UPGRADE\_PART类型，“download”必须为FAL\_DOWNLOAD\_PART类型），分区名称不可重复。

1. 配置参数

根据实际情况，修改oneos\_config.h中的配置参数（见2.1.6参数配置章节）

1. 初始化地址映射（必要时）

若使用逻辑地址，需要填写ota\_fal\_map\_t类型的地址映射表，表中必须包含分区表中所有的分区地址映射信息，若某些分区不使用逻辑地址，则逻辑地址与物理地址保持一致。通过ota\_fal\_map\_init接口完成地址映射初始化。

1. 初始化分区信息

将flash表和分区表作为参数，调用ota\_fal\_init接口完成分区信息初始化。

1. 初始化堆空间（必要时）

若使用内置堆管理，则需要调用ota\_heap\_init接口进行堆空间初始化。

1. 检查升级状态

在bootloader启动时，读取download分区最后一个块起始存放的前2个成员数值，判断是否需要进入还原升级：

struct cmiot\_update\_state

{

unsigned char inited;

unsigned short update\_result;

};

update\_result等于100，表示需要进入还原升级

1. 启动升级

调用ota\_start\_up接口，启动升级。对比返回码，可以查看升级结果以及错误码

## 参数配置

在oneos\_config.h中，用户可配置的FOTA相关的几个重要参数如下：

1. 可用内存大小CMIOT\_FOTA\_AVAILABLE\_RAM\_SIZE

示例：#define CMIOT\_FOTA\_AVAILABLE\_RAM\_SIZE 0x80000

支持十进制/十六进制，行尾注释前必须加空格。该宏定义了fota还原升级时真实可用的内存大小，在条件允许的情况下，内存越大越好。

1. Download分区大小CMIOT\_UPDATE\_SIZE

示例：#define CMIOT\_UPDATE\_SIZE 0xA0000

支持十进制/十六进制，行尾注释前必须加空格。该宏定义了差分包下载空间（即download分区）大小，在制作差分包时会对其进行校验，如果差分包的实际大小超过该值定义的大小，会提示制作差分包失败，用户需提前规划好download分区大小，不允许升级版本时再作改变。如果空间有限，可以采取多次少量进行升级。

1. Flash擦除大小CMIOT\_DEFAULT\_SECTOR\_SIZE

示例：#define CMIOT\_DEFAULT\_SECTOR\_SIZE 0x10000

支持十进制/十六进制，行尾注释前必须加空格。该宏定义了FLASH的块擦除大小，必须和实际flash的块擦除大小一致。制作差分包和还原升级，会以此作为计算和存储的基本单位。

1. 固件版本CMIOT\_FIRMWARE\_VERSION

示例：#define CMIOT\_FIRMWARE\_VERSION "1.0"

字符串，行尾注释前必须加空格。该宏定义了固件版本，升级版本时，版本号需同时进行迭代，版本最大支持128字节长度。

1. 下载块大小CMIOT\_DEFAULT\_SEGMENT\_SIZE\_INDEX

示例：#define CMIOT\_DEFAULT\_SEGMENT\_SIZE\_INDEX 3

十进制/十六进制，行尾注释前必须加空格。该宏定义了下载差分包时默认的buff大小，该值越大，单次下载的数据包长度更大，总的下载时间会减少，最大可设置为5。

1. 升级文件CMIOT\_FOTA\_FILENAMEx，CMIOT\_FOTA\_ADDRESSx

示例：#define CMIOT\_FOTA\_FILENAME1 "mmi\_res\_240x240\_child\_circular.bin"

#define CMIOT\_FOTA\_ADDRESS1 0x90000004

CMIOT\_FOTA\_FILENAMEx文件名为字符串，行尾注释前必须加空格。与实际文件名称对应，制作版本包时，会根据该名称检索文件。如果是单bin，则x为空，如果是多bin，x=1，2...，文件名必须是以常见的二进制文件结尾，如’\*.bin’,’\*.img’等，若差分文件为其他后缀，可以在文件末尾添加’.bin’。CMIOT\_FOTA\_ADDRESSx文件地址，必须是十六进制格式，如果使用逻辑地址，也应当符合地址的特征，如长度为8位。

1. 全量/差分升级CMIOT\_FOTA\_USE\_FULLx

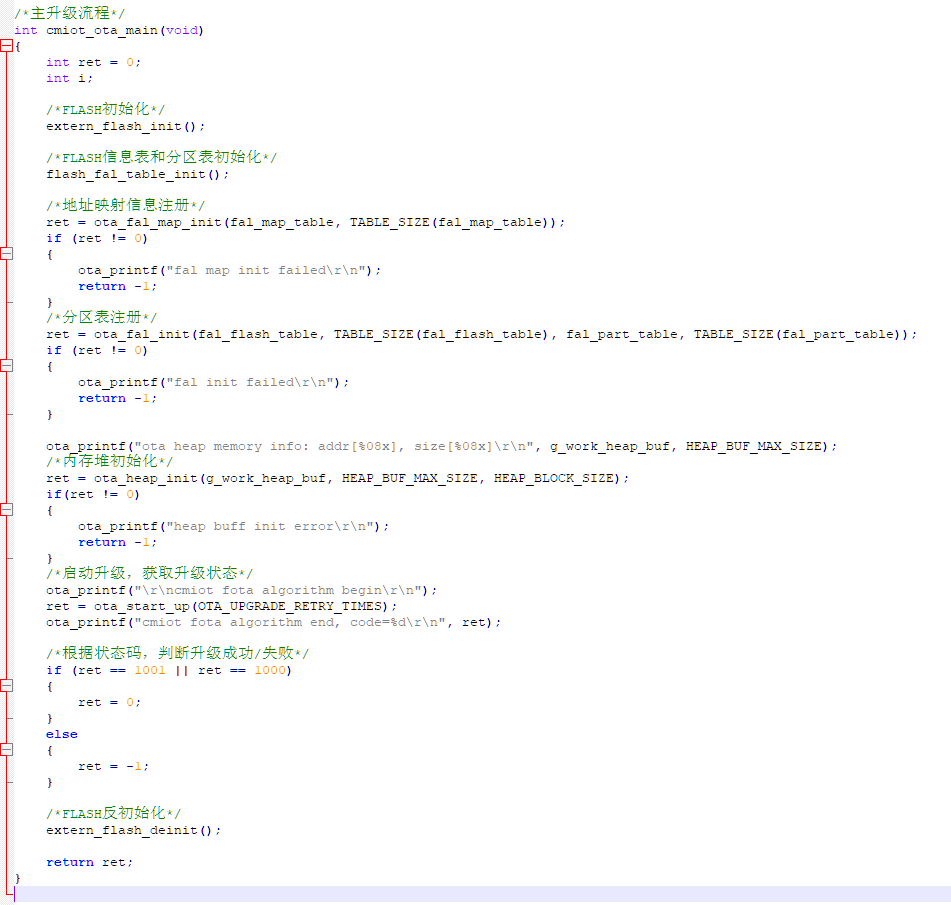
示例：#define CMIOT\_FOTA\_USE\_FULL2

CMIOT\_FOTA\_USE\_FULLx，表示对应的CMIOT\_FOTA\_FILENAMEx升级模式为全量升级。默认为差分升级。当目标版本需要做全量升级时，可定义该宏。

## 注意事项

1. download分区除差分包存放区域外，还另外需要两个区域用于备份和存放信息，所以download分区大小至少需要三个扇区；
2. flash操作接口适配时，当读写成功时返回读写数据的长度，读写失败时，返回-1；擦除失败时返回-1，擦除成功返回0；
3. 在没有完善堆可使用ota\_heap\_init初始化升级使用内存，有完善堆空间时可不使用。

## 升级主流程示例



# 升级操作流程

## 云平台创建项目

确保账号有权限后，登录[CMS云平台](https://os.iot.10086.cn/cms/index)，在主界面点击产品开发-产品管理-新建产品按钮：



图4.1 CMS平台入口

填写产品名称，选择芯片型号，选择所要使用的服务、通信方式以及协议，完成产品创建：



图4.2 CMS项目创建

完成产品创建后可以在cms产品界面查看刚刚创建的产品的详细信息，其中产品ID与产品秘钥要配置到端侧oneos\_config中：

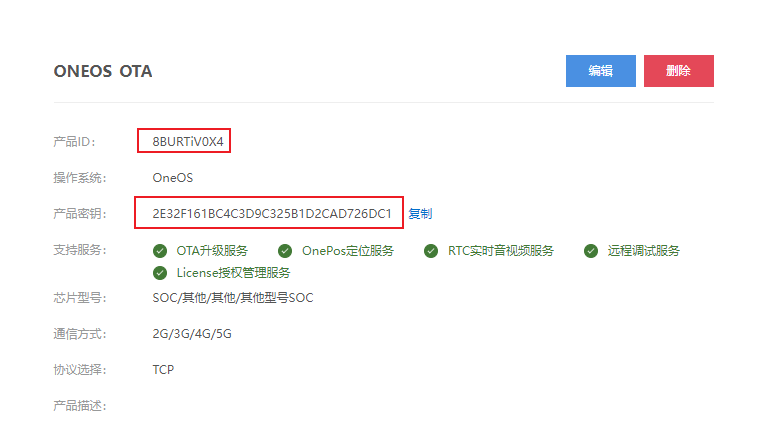


图4.3 CMS产品信息

## 本地配置

根据项目信息修改**oneos\_config.h**中相关宏定义：

#define CMIOT\_FOTA\_SERVICE\_PRODUCT\_ID "8BURTiV0X4" /\* 对应平台中的产品ID \*/

#define CMIOT\_FOTA\_SERVICE\_PRODUCT\_SEC "2E32F161BC4C3D9C325B1D2CAD726DC1" /\* 对应平台中的产品密钥 \*/

## 生成差分包

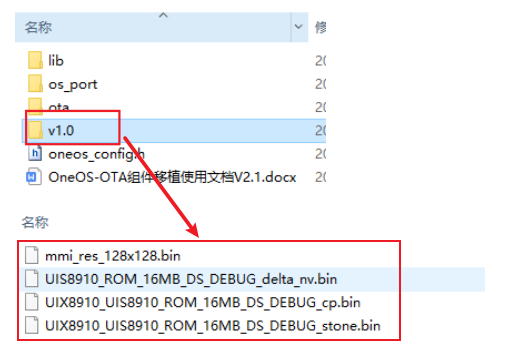
### 本地打包

整个工程编译完成后，通过编写脚本或手动执行cmd命令，命令格式为：

oneos\ota\cmiot\source\pack\cmiot\_pack.bat “压缩软件路径” {oneos\_config.h路径} {bin文件夹路径}

|  |  |
| --- | --- |
| 命令 | 参数说明 |
| oneos\ota\cmiot\pack\cmiot\_pack.bat | 使用打包的批处理文件.bat 文件路径 |
| “压缩软件路径” | 压缩软件绝对路径，支持WinRAR，7zip |
| {oneos\_config.h路径} | 该版本对应的头文件，用于读取版本信息，差分算法等 |
| {bin文件夹路径} | 需要升级的bin文件路径，与3.3 oneos\_config.h中定义的bin要保持一致 |

以目前设置为例，oneos\_config.h中定义了需要升级的bin文件有4个，"mmi\_res\_128x128.bin"、"UIS8910\_ROM\_16MB\_DS\_DEBUG\_delta\_nv.bin"、"UIX8910\_UIS8910\_ROM\_16MB\_DS\_DEBUG\_cp.bin" 和"UIX8910\_UIS8910\_ROM\_16MB\_DS\_DEBUG\_stone.bin"，首先在ota同级目录建立文件夹v1.0（对应当前版本号），并把bin文件复制到此：



然后在cmd窗口进到ota同级目录，输入上述命令：

**>.\ota\cmiot\source\pack\cmiot\_pack.bat "C:\Program Files\7-Zip\7z.exe" oneos\_config.h .\v1.0\**

在ota同级目录就会生成一个.zip压缩包，命名是**“版本号\_年月日\_时分秒.zip”**。

一般升级至少要有两个版本，一个当前设备的版本，另一个升级目标的版本。可以通过版本号来识别，修改oneos\_config.h中CMIOT\_FIRMWARE\_VERSION定义后，执行打包命令就可以看到对应版本号及时间戳的升级压缩包。两个版本对应两个压缩包，然后再到平台上制作差分包。



### 平台差分

回到OTA平台，点击FOTA升级-版本管理-添加版本：

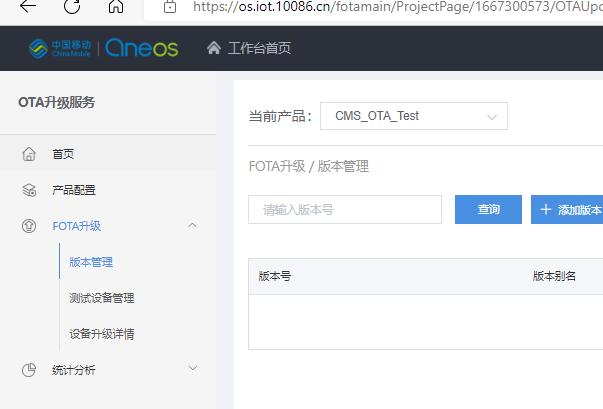


图4.4 FOTA版本管理入口

输入版本号，别名，描述，点击上传上一步打包的压缩文件，点确定：

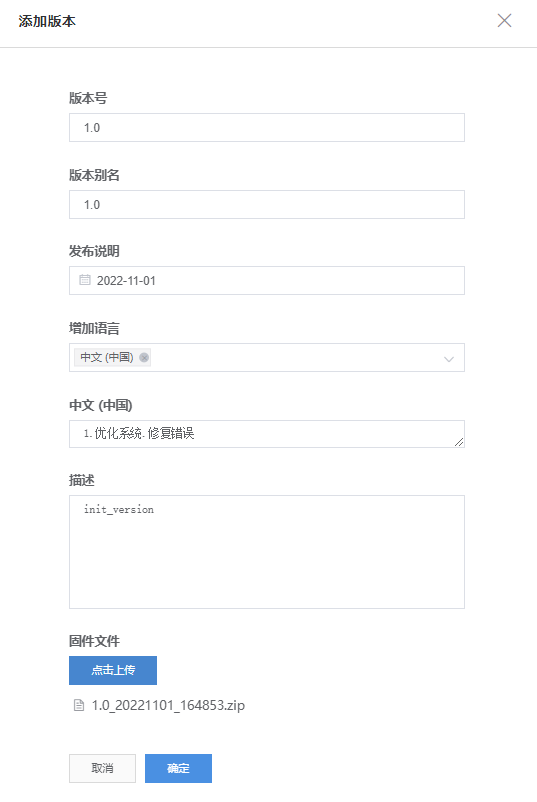


图4.5 FOTA上传版本

同样上传2.0的版本后，可以看到对应的版本信息：



图4.6 FOTA版本信息

接下来配置升级任务，点击配置升级任务-配置，然后选择升级类型为差分，制作差分包为线上自动差分，可用RAM保持默认，RAM值是根据oneos\_config.h中配置获取：

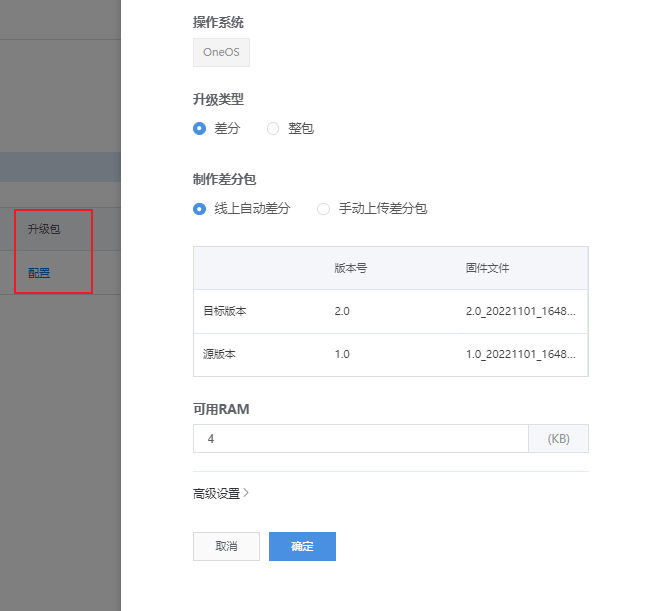


图4.7 FOTA版本差分设置

点击确定生成差分升级包，制作完成后回到配置升级任务页，此时升级任务状态为待测试：

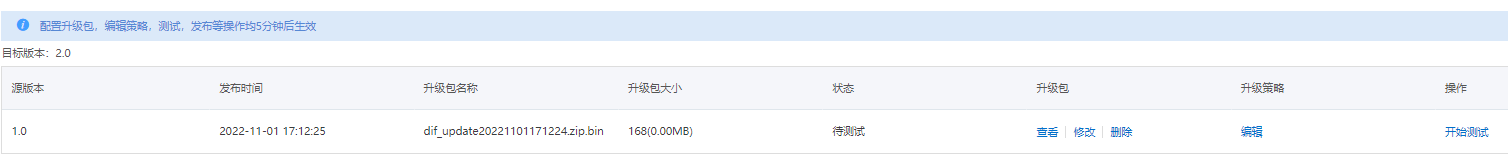


图4.8 FOTA差分包信息

点击开始测试后，进入测试中状态，此时测试设备可以检测到新的版本（5分钟后）。测试完成后，点击通过测试，发布版本。



图4.9 FOTA版本发布

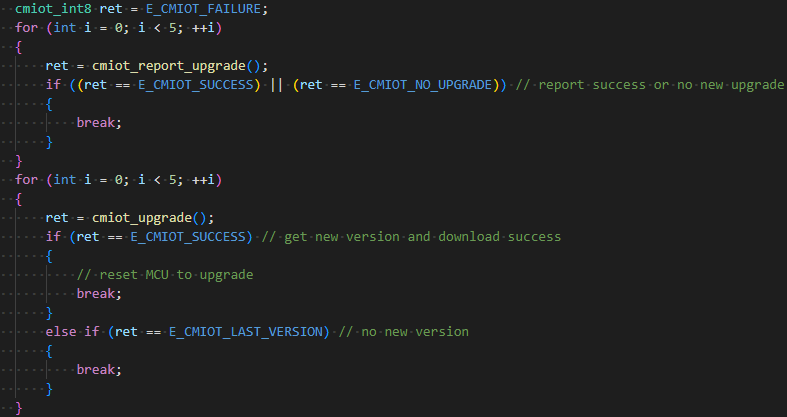
## 升级测试

### 一键升级demo

1. 引用头文件

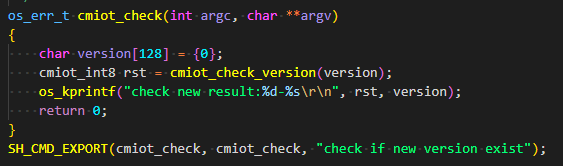
#include “cmiot\_main.h”

1. 添加上报升级结果和升级更新的代码

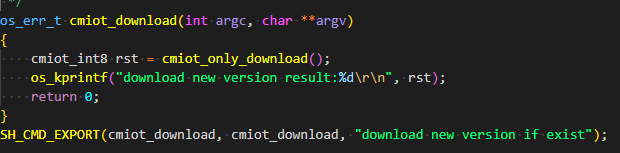


注：for循环重复几次是避免网络问题导致连接失败；升级检测并下载成功后由应用决定是否重启，如果重启，之后会进入bootloader进行固件升级。

### 查询是否有新版本



### 有新版本后下载



# 参考资料

[1]CMS工作台：[文档中心 (10086.cn)](https://os.iot.10086.cn/v2/doc/detailPage/documentHtml?proId=2000000002&proName=CMS%E5%B7%A5%E4%BD%9C%E5%8F%B0&idss=177&versionName=v1.0&versionId=3000000002)

[2]FOTA服务：[文档中心 (10086.cn)](https://os.iot.10086.cn/v2/doc/detailPage/documentHtml?proId=2000000004&proName=FOTA%E6%9C%8D%E5%8A%A1&idss=199&versionName=v1.0&versionId=3000000003)

[3]视频教程：[OneOS学院 (10086.cn)](https://os.iot.10086.cn/college/)

[4]OTA平台文档中心：<https://os.iot.10086.cn/otaplatform/DocumentCenterViewPage>

# FAQ

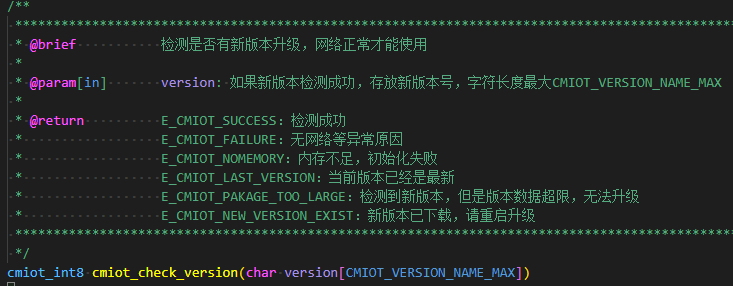
## APP移植

### 如何调用升级接口？

OTA组件只提供机制，策略性的东西需要用户来定，比如可以每天检测一次，也可以每周检测一次，可以下载完就重启升级，也可以等别的业务完成再重启升级。

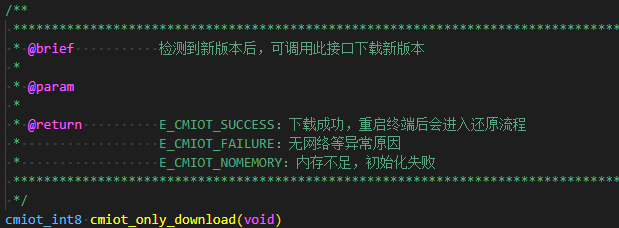
1、检测是否有新版本

MMI调用此接口检测是否有新版本，如果有可以进行下载，否则显示当前为最新版本



2、下载新版本

检测到有新版本后，调用此接口进行下载。

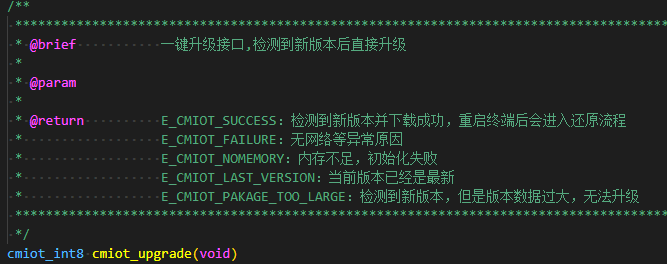


3、下载完成后重启

MMI调用设备重启接口，具体实现由SDK提供

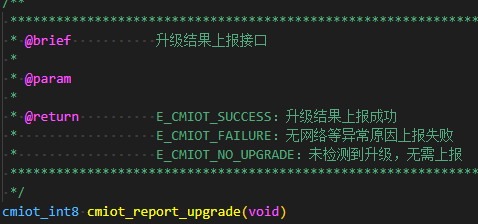
4、一键升级

此接口为一键升级功能，包含检测新版本，下载新版本过程



1. 上报升级结果

升级成功后需要上报升级结果，建议每次开机后调用此接口。



### 调用cmiot\_upgrade()接口总是超时是什么原因？

请根据网络适当加大接收超时时间，对应超时时间的宏为CMIOT\_DEFAULT\_TRY\_TIME，在cmiot\_config.h文件里。

### 如何设置设备device ID？

参考3.2.2.11接口，用户自己设定，用于云平台识别终端设备。

### 假如文件下载快完成了，突然中断了，下次下载会从头开始下载吗？

不会的，OTA支持断点续传，会从中断的地方开始下载。

### cmiot\_upgrade 内部返回码

当接口执行不成功时，会有日志打印具体的错误码，表示具体的异常情况。日志开头为：Maybe some errors, process termination, error code:，冒号后面为具体错误码，其详细情况如下：

|  |  |
| --- | --- |
| 内部错误码 | 错误码说明 |
| 1000 | 执行成功 |
| 1001 | 产品ID无效，请检查项目ID和平台是否一致 |
| 1002 | 项目不存在，需要对比终端配置的Oem、Model、Device type、Platform和平台是否一致。 |
| 1005 | 终端Flash操作失败，终端擦写读flash有问题，需要检查相关的驱动程序或硬件 |
| 1008 | MID长度错误，MID最少4个字符，最多31个字符 |
| 1010 | 达到设备访问上限，该设备的单日访问量达到上限了 |
| 1015 | 达到用户访问上限，该设备所属用户的单日访问量达到上限了 |
| 2001 | 签名不对，需要对比终端配置的Product Id、Product secret和平台是否一致，如果一致，擦除flash下载分区再测试 |
| 2101 | 没有新版本，该设备正在运行的版本已经是最新版本 |
| 2103 | 设备未注册，一般测试时出现，擦除flash下载分区再测试 |

## BootLoader适配

Q：如何合理的配置download分区大小？

download分区配置为需要升级的尺寸总和的10~50%之间，若空间非常紧张，建议设置为最大应用尺寸的20%。

Q：可用RAM大小会对性能有哪些影响？

ram空间不足时，第一，可能导致还原过程中内存分配失败，从而增加升级失败的风险；第二，不会启用二次压缩，相同的版本，制作的差分包会更大，分配给算法ram空间配置尽可能大，最小不低于3个block（最小擦除）大小。

Q：升级过程中断电会有什么后果？

升级过程中断电，重新上电后会继续升级。但是升级过程中断电有可能带来其他风险，因此推荐在升级在电量充足的条件下进行，并且过程中明确的提示用户不要断电。

Q：哪些因素会影响升级的时间？

第一，版本差异大小。升级前后版本差异越小，需要更新的flash越少，擦写flash的总次数越少，耗时也就越少；第二，存储介质擦写耗时，升级相同的版本，存储介质擦写耗时越少，总的耗时越少。

Q：在一个平台上适配的还原库，可以直接应用到其他项目上吗？

主要取决于以下几个方面：1.芯片架构，2.编译工具，3.算法配置；如果这几项都一样即可可以直接应用到其他项目上。