
# ZS110A-OTA

发布 *1.0.0*

**2018** 年 **11** 月 **02** 日

# 目录

CHAPTER 1

# 文档介绍

## 1.1 文档目的

介绍本项目中 OTA 实现方式、机制和策略。

## 1.2 术语说明

表 1.1: 术语说明

| 术语 | 说明 |
| --- | --- |
| ZEPHYR | 为所有资源受限设备，构建了针对低功耗、小型内存微处理器设备而进行优化的物联网嵌入式小型、可扩展的实时操作系统（RTOS） |
| OTA | Over-the-Air Technology，本项目中指通过 ble 空中升级设备的固件 |

## 1.3 参考文档

- http://docs.zephyrproject.org/

## 1.4 版本历史

表 1.2: 版本历史

| 日期 | 版本 | 注释 | 作者 |
| --- | --- | --- | --- |
| 2018-08-22 | 1.0 | 初始版本 | ZS110A 项目组 |

CHAPTER 2

# OTA 包

## 2.1 OTA 包生成环境

- 安装 python 2.7.15 版本
- 拷贝\scripts\utils\dd-0.6beta3 到与源码目录无关的路径。例如：C:\
- pc 的环境变量-系统变量-Path 中添加 python 和 dd 的路径。例如：;C:\Python27;C:\dd-0.6beta3

## 2.2 OTA 包生成

运行\samples\voice_rcu\build.bat 后，outdir 目录下会生成 zs110a_*_ota.zip 文件。

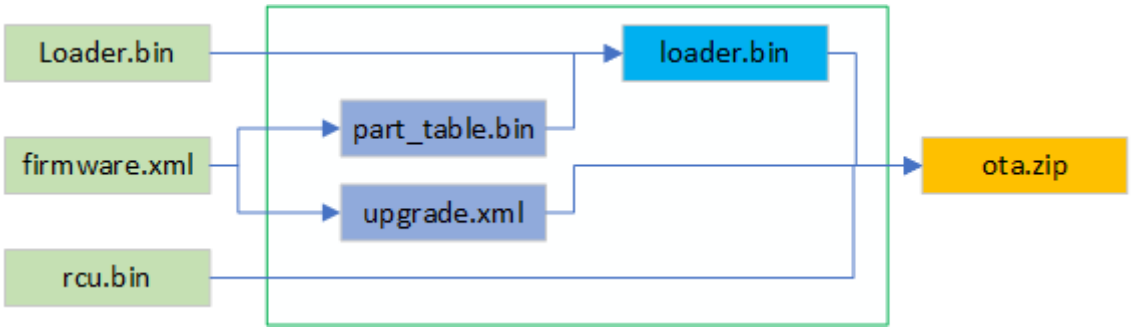


图 2.1: OTA 包生成

## 2.3 OTA 包格式

OTA 包是一个 zip 格式的压缩文件。解压后的目录由一个 OTA 镜像描述文件和多个镜像文件组成。

**以遥控器方案为例，OTA 包解压后包含以下文件：**

- 镜像描述文件：Update.xml
- loader 分区镜像文件：loader.bin
- 应用分区镜像文件：rcu.bin

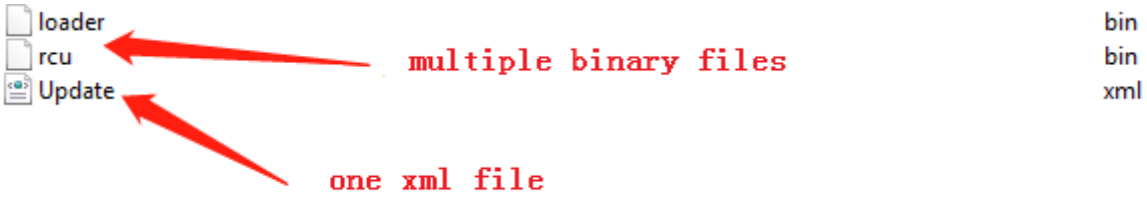


图 2.2: OTA 包含文件

### 2.3.1 镜像描述文件

Update.xml 文件描述了包括待升级的镜像的信息，下面是一个具体示例。

```
<?xml version='1.0' encoding='UTF-8'?>
<partitions>
        <partition>
                <type>BOOT</type>
                <name>fw0_boot</name>
                <file>loader.bin</file>
                <address>0x0</address>
                <fw_id>0</fw_id>
                <crc32>3156977668</crc32>
        </partition>
        <partition>
                <type>BOOT</type>
                <name>fw1_boot</name>
                <file>loader.bin</file>
                <address>0x1000</address>
                <fw_id>1</fw_id>
                <crc32>3156977668</crc32>
        </partition>
        <partition>
                <type>SYSTEM</type>
                <name>fw0_app</name>
```

(continues on next page)

(续上页)

```
                <file>rcu.bin</file>
                <address>0x10000</address>
                <fw_id>0</fw_id>
                <crc32>2230699271</crc32>
        </partition>
        <partition>
                <type>SYSTEM</type>
                <name>fw1_app</name>
                <file>rcu.bin</file>
                <address>0x30000</address>
                <fw_id>1</fw_id>
                <crc32>2230699271</crc32>
        </partition>
        <partitionsNum>4</partitionsNum>
        <version>2.0.00.18073120</version>
</partitions>
```

说明：

1. 格式必须符合标准 XML。
2. version: 描述版本号。
3. partitionsNum：描述分区的数目。
4. partition：描述一个分区的信息，其各个字段由\samples\voice_rcu\firmware.xml 的模板定义，建议不要自行修改改字段的模板定义。
   - type：描述该镜像的类型，合法值包括“BOOT”,“SYSTEM”等。
   - name：由打包脚本自动生成。
   - file：烧写到该分区的镜像文件名。
   - address：分区在 Flash 上的地址。
   - fw_id：镜像 ID。
   - crc32：描述镜像文件的 crc32 校验和。

### 2.3.2 loader.bin 和分区表

- 默认 loader 分区为 4KBytes。
- ota 包中的 loader.bin 是由 loader 工程生成的 loader.bin（<3KBytes）和分区表组合而成。分区表位于 3KBytes 偏移处。
- 分区表的数据结构如下：

```
struct partition_table {
        u32_t magic;
```

(continues on next page)

(续上页)

```
        u16_t version;
        u16_t table_size;
        u16_t part_cnt;
        u16_t part_entry_size;
        u8_t reserved1[4];
        struct partition_entry parts[MAX_PARTITION_COUNT];
        u8_t Reserved2[4];
        u32_t table_crc;
```

分区表数据结构说明：

1. magic: 0x54504341 // ‘ACPT’
2. version: 默认为 0x0000
3. table_size: 分区表的总大小 =16+360+4+4=384 Bytes
4. part_cnt: 实际的分区数
5. part_entry_size: 一个分区的数据结构大小
6. parts[MAX_PARTITION_COUNT]: 预留了 15 个分区的数据结构
7. table_crc：整个分区表的 crc 校验和。

单个分区的结构

```
struct partition_entry {
                u8_t name[8];
                u16_t type;
                u16_t reserve0;
                u32_t offset;
                s16_t seq;
                u16_t reserve1;
                u32_t reserve2;
};

.parts[0] = {
        .name = "fw0_boot",
        .type = BOOT_TYPE,
        .offset = LOADER_A_NOR_ADDR,
},
.parts[1] = {
        .name = "fw1_boot",
        .type = BOOT_TYPE,
        .offset = LOADER_B_NOR_ADDR,
        .seq = -1,
},
.parts[2] = {
        .name = "fw0_app",
        .type = SYSTEM_TYPE,
        .offset = APP_A_NOR_ADDR,
```

(continues on next page)

(续上页)

```
},
.parts[3] = {
        .name = "fw1_app",
        .type = SYSTEM_TYPE,
        .offset = APP_B_NOR_ADDR,
        .seq = -1,
},
.parts[4] = {
        .name = "nv_fact",
        .type = DATA_TYPE,
        .offset = NVRAM_FACTORY_NOR_ADDR,
},
```

分区数据结构说明：

1. name: 分区名。
2. type: 分区类型。

```
enum {
        RESERVE_TYPE = 0,
        BOOT_TYPE = 1,
        SYSTEM_TYPE = 2,
        RECOVREY_TYPE = 3,
        DATA_TYPE = 4,
        DTM_TYPE = 5,
};
```

3. offset: 分区在 Flash 上的偏移地址。
4. seq: 用于区分同类型分区。OTA 升级时会选择 seq 值较小的分区进行升级。

CHAPTER 3

# OTA 流程

## 3.1 OTA 升级流程

为了简化小机端代码逻辑，将更多的如何选择分区，如何调整分区大小等复杂的逻辑留在在移动端。

### 3.1.1 双方具体分工和协作方式如下

**小机端：**

- 提供小机的分区表信息及版本号。
- 实现分区读写和校验的功能。
- 实现重启的功能。

**移动端：**

- 解析 ota 包获取版本号，获取小机端版本号，对比后判断是否需要升级。
- 解析 ota 包内的分区镜像文件，获取小机端的分区信息，选择正确的分区来进行升级。
- 选择备用的 app 分区升级 app 镜像。
- 选择备用 loader 分区升级 loader 镜像（包含更新分区信息），并擦除当前的 loader 分区。

具体流程如下图所示；

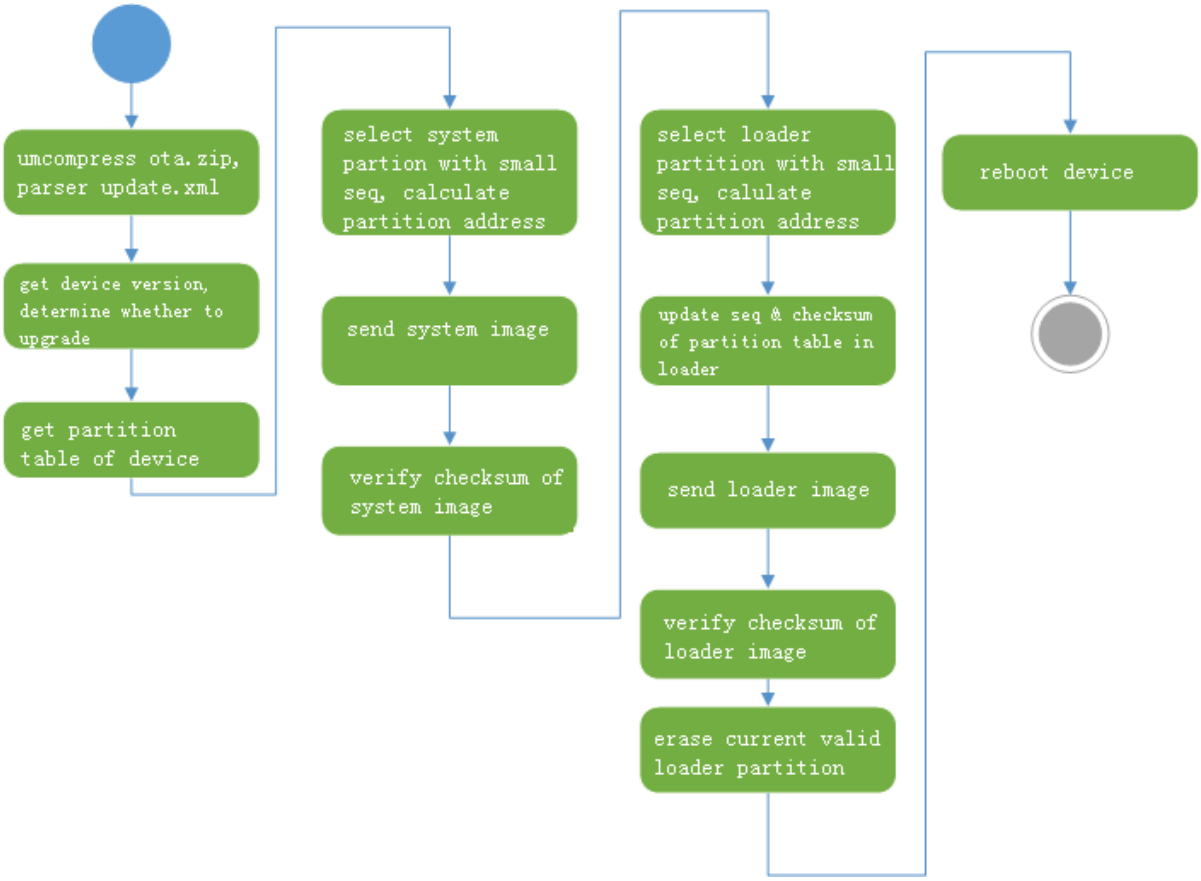


图 3.1: 移动端 OTA 流程

### 3.1.2 A/B 备份机制

A/B 备份机制指存在 2 个相同类型的分区。

以遥控器方案为例：定义了 2 个 BOOT_TYPE 分区和 2 个 SYSTEM_TYPE 分区。

初始版本的分区表中：A 份分区的 seq=0，B 份分区的 seq=-1。

第一次进行 OTA 时，移动端读取到分区表后，会选择 seq 值较小的 B 分区进行升级，升级完成会修改 seq_B。seq_B=seq_A+1。

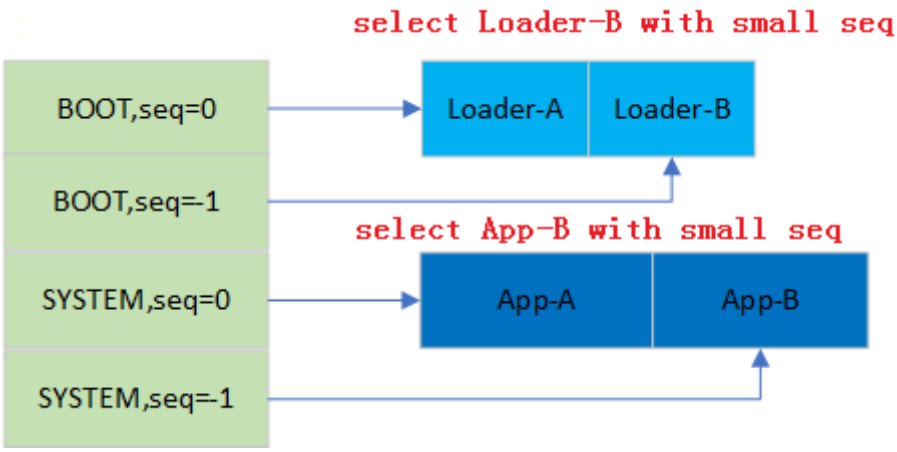


图 3.2: 第一次 OTA

第二次进行 OTA 时，移动端读取分区表后会发现此时 seq 值较小的是 A 分区，所以会选择 A 分区进行升级。

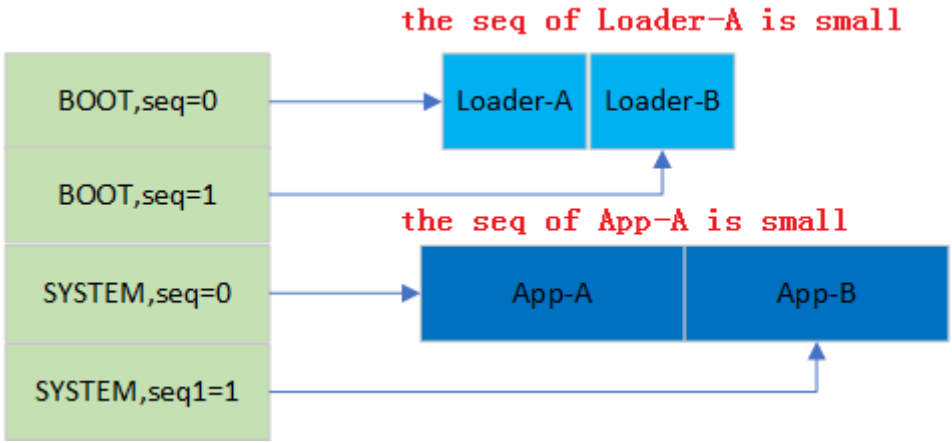


图 3.3: 第二次 OTA

## 3.2 OTA 完成后启动流程

- brom 引导 loader：

  OTA 过程中会檫除当前运行的 BOOT_TYPE 分区，所以正常情况下 brom 也只能引导升级后新的 loader。如果 A/B 同时存在，brom 会优先引导 A 份。

- loader 引导 app：

OTA 过程不会擦除当前运行的 SYSTEM_TYPE 分区，所以可能会有两份正确的 app 镜像同时存在的情况。loader 会尝试读取 2 份 app 镜像。如果 2 份镜像都通过检验，选择其中 seq 值较大的镜像加载运行。

# CHAPTER 4

# OTA Profile

Ota profile 定义了 4 个属性：

- File Transfer Control Characteristic（FTC）：用于 OTA 固件的读和写控制。
- File Transfer Data Characteristic（FTD）：用于 OTA 固件读和写，以保证传送速率。
- Authentication Characteristic（AU）：用于保证 OTA 升级必须是在移动端和小机端协商同意才允许升级。
- Device Configuration（DC）：用于配置和获取一些 BLE 参数，如连接间隔、MTU 等。

## 4.1 传送控制属性-File Transfer Control Characteristic

FTC 用于控制命令的发送和应答。

FTC 命令控制要求采取同步应答方法，除 WDXS_FTC_OP_PACKET_RECEIVED 响应方式为异常应答方式。

WDXS_FTC_OP_PACKET_RECEIVED 设主要用移动端 APP 的传送和设备端计数同步，为了提高 BLE 的传送效率，当前设计为设备端 CPU 稍有空闲时，就将接收到固件大小数据反馈给移动端，同时保证最后一个包的数据统计信息必须反馈给移动端。

FTC 有如下传输控制命令：

| request/response | Opcode | Type | Parameters |
|---|---|---|---|
| OTA_FTC_OP_GET_REQ | 01 | Request | |
| OTA_FTC_OP_GET_RSP | 02 | Response | |
| OTA_FTC_OP_PUT_REQ | 03 | Request | |
| OTA_FTC_OP_PUT_RSP | 04 | Response | |
| OTA_FTC_OP_ERASE_REQ | 05 | Request | |
| OTA_FTC_OP_ERASE_RSP | 06 | Response | |
| OTA_FTC_OP_VERIFY_REQ | 07 | Request | |
| OTA_FTC_OP_VERIFY_RSP | 08 | Response | |
| OTA_FTC_OP_ABORT | 09 | Request | |
| OTA_FTC_OP_EOF | 10 | Response | |
| OTA_FTC_OP_PACKET_RECEIVED | 11 | Response | |
| OTA_FTC_OP_SYSTEM_RESET | 12 | Response | |
| OTA_FTC_OP_GET_VERSION_REQ | 13 | Request | |
| OTA_FTC_OP_GET_VERSION_RSP | 14 | Response | |

- OTA_FTC_OP_GET_REQ

| 0 byte | op | 操作码，用于读取设备端 flash 内容 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3~6 | addr | 地址 |
| 7~10 | len | 长度 |

- OTA_FTC_OP_GET_RSP

| 0 byte | op | 操作码，用于应答移动端操作 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3 | status | 0：正常其它：异常，见状态描述 |

- OTA_FTC_OP_PUT_REQ

| 0 byte | op | 操作码，用于写设备端 flash 内容 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3~6 | addr | 地址 |
| 7~10 | len | 长度 |

- OTA_FTC_OP_PUT_RSP

| 0 byte | op | 操作码，用于应答移动端操作 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3 | status | 0：正常其它：异常，见状态描述 |

- OTA_FTC_OP_ERASE_REQ

| 0 byte | op | 操作码，用于檫除设备端 flash 内容 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3~6 | addr | 地址 |
| 7~10 | len | 长度 |

- OTA_FTC_OP_ERASE_RSP

| 0 byte | op | 操作码，用于应答移动端操作 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3 | status | 0：正常其它：异常，见状态描述 |

- OTA_FTC_OP_VERIFY_REQ

| 0 byte | op | 操作码，用于验证写设备端 flash 数据是否一致 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3~6 | addr | 地址 |
| 7~10 | len | 长度 |
| 11~14 | Crc32 | Crc 信息 |

- OTA_FTC_OP_VERIFY_RSP

| 0 byte | op | 操作码，用于应答移动端操作 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3 | status | 0：正常其它：异常，见状态描述 |

- OTA_FTC_OP_SYSTEM_RESET

| 0 byte | op | 操作码，让设备端重启 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |

- OTA_FTC_OP_GET_VERSION_REQ

| 0 byte | op | 操作码，获取设备端的固件版本信息 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |

- OTA_FTC_OP_GET_VERSION_RSP

| 0 byte | op | 操作码，用于应答移动端操作 |
|---|---|---|
| 1~2 | part_id | 分区 ID，其中 0 为分区表的 ID，其它 ID 暂时未使用 |
| 3 | status | 0：正常其它：异常，见状态描述 |
| 4~19 | version | 字符串如：2.0.00.18070920 |

## 4.2 数据传送属性-File Transfer Data Characteristic

ftd 用于数据文件的传送，通常都是裸的读写数据

## 4.3 授权认证属性-Authentication Characteristic

暂无

## 4.4 设备配置属性-Device Configuration

一个可读可写可通知的 ATT 属性用于支持 OTA 相关配置，如连接参数配置、安全连接配置、连接断开配置、MTU 配置、电池 Level 配置等。

当前 OTA 和应用设计在同一固件中，相关配置已经在应用中配置，该属性节点没有添加相关接口。

CHAPTER 5

# OTA 交互过程

移动端与设备进行 OTA 大致过程为：

1. 移动端获取设备端的信息确认是否需要升级，如获取电量信息、获取设备固件版本信息。
2. 移动端获取设备端的分区表，确认需要升级固件的升级地址。
3. **按照下列操作步骤，循环升级完 OTA 包中的每个 bin。**
    - 移动端发送 put 请求，告诉设备端 xxx.bin 的升级地址及长度
    - 设备端回复 put 应答，告诉移动端已经准备好。
    - 移动端将 xxx.bin 切分多个 data 包（大小跟 mtu 一致），然后发送给设备端
    - 设备端收到数据包后，将数据写到 flash，同时进行计数，并发送一个消息给 Main 线程，Main 线程在空闲时将包计数回复给移动端，用于更新进度条。
    - 设备端将 xxx.bin 接收完后，发送 EOF 给移动端
    - 移动端收到 EOF 后，发送 verify 给设备端。
    - 设备端将 xxx.bin 进行校验，校验成功，回复 verify 应答。
4. 升级包中所有 bin 升级完后，移动端发送檫除命令给设备端，设备檫除完后，回复檫除操作应答。
5. 移动端发送重启命令给设备端，让设备端进行重启。

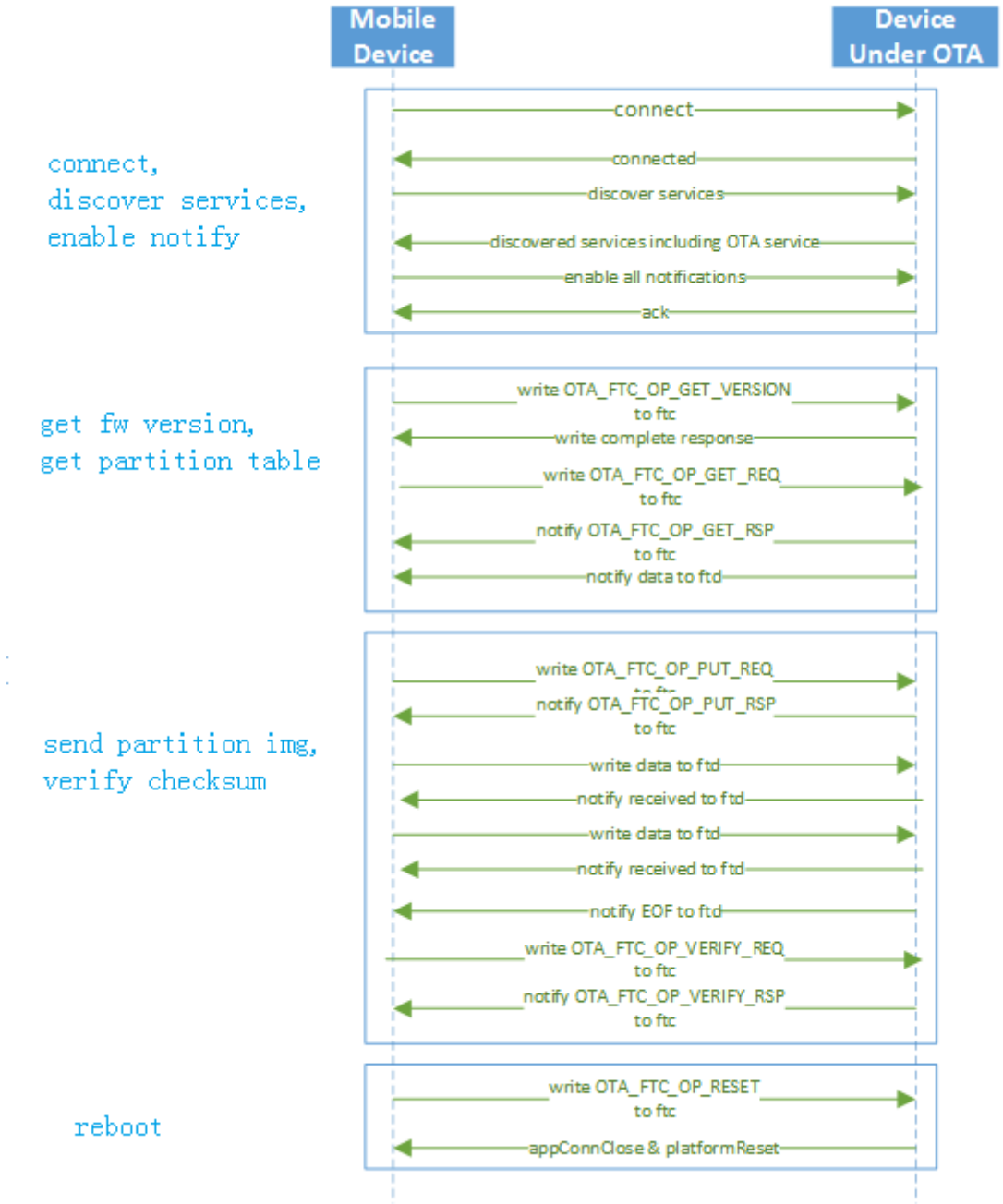
Mobile Device
Device Under OTA
connect,
discover services,
enable notify
connect
connected
discover services
discovered services including OTA service
enable all notifications
ack
get fw version,
get partition table
write OTA_FTC_OP_GET_VERSION to ftc
write complete response
write OTA_FTC_OP_GET_REQ to ftc
notify OTA_FTC_OP_GET_RSP to ftc
notify data to ftd
send partition img,
verify checksum
write OTA_FTC_OP_PUT_REQ
notify OTA_FTC_OP_PUT_RSP to ftc
write data to ftd
notify received to ftd
write data to ftd
notify received to ftd
notify EOF to ftd
write OTA_FTC_OP_VERIFY_REQ to ftc
notify OTA_FTC_OP_VERIFY_RSP to ftc
reboot
write OTA_FTC_OP_RESET to ftc
appConnClose & platformReset

图 5.1: OTA 流程

# List of Figures

# List of Tables