XR871 OTA Developer Guide

Revision 1.0

Nov 3, 2017

Declaration

Revision History

Version	Data	Summary of Changes
1.0	2017-11-3	Initial Version

Table 1-1 Revision History

Contents

Tables

Figures

1 概述

OTA 模块为系统提供在线升级固件的功能。此文档用以解释说明 OTA 模块相关概念和定义，介绍并指导开发者使用 SDK 中的 OTA 方案。OTA 模块在一定程度上依赖 Image 模块。与 Image 相关的内容可以参考文档《XR871 Image Developer Guide》。

1.1 OTA 升级原理

SDK 中的 OTA 方案通过对两个 Image 区域进行乒乓操作实现对固件的升级，两个 Image 区域如下图所示。

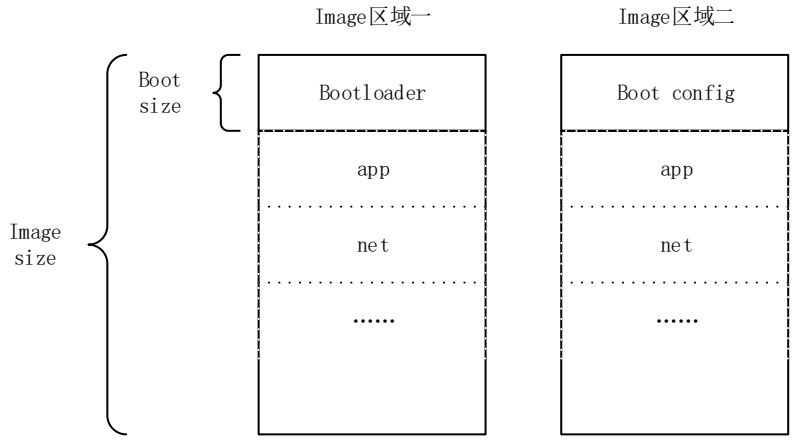

图 1-1 OTA 方案两个 Image 区域示意图

两个 Image 区域大小相同，均为 Image size。Image size 应该大于或等于实际固件的大小。此外，Image 区域应该与 Flash 可擦除块对齐。Image 区域一的数据与实际固件数据相同，前 Boot size 大小的区域为 Bootloader。Image 区域二的前 Boot size 大小的区域为 Boot config。Boot size 之后区域的数据与实际固件数据相同（即 Image 区域二中没有 Bootloader，对应位置为 Boot config）。因此 OTA 升级时，Bootloader 不变，始终加载 Image 区域一的 Bootloader。Bootloader 之后的数据将会在每次 OTA 升级成功后交替加载两个区域的固件数据。

系统启动时，加载 Image 区域一的 Bootloader，Bootloader 根据 Boot config 的记录，判断后续加载哪个 Image 区域的固件数据。

OTA 升级固件时，系统根据 Boot config 的记录，判断本次更新哪个 Image 区域的固件。固件更新成功并校验通过后，更新 Boot config，并重启系统。系统重启后会根据更新后的 Boot config 加载最新的固件数据。如果固件更新或校验失败，则不修改 Boot config，系统下次启动仍然加载 OTA 升级前的固件数据。

1.2 相关概念和定义

1.2.1 Boot config

系统启动时，从 Boot config 获取加载哪个 Image 区域固件的信息。Boot config 中主要存储结构体类型 ota_cfg_t 的数据，结构体类型 ota_cfg_t 的两个成员类型分别为 ota_image_t 和 ota_state_t。上述 3 个类型数据在 image.h 中定义如下：

```
typedef enum ota_image {
    OTA_IMAGE_1ST = 1,
    OTA_IMAGE_2ND = 2,
} ota_image_t;

typedef enum ota_state {
    OTA_STATE_UNVERIFIED    = 0,
    OTA_STATE_VERIFIED      = 1,
} ota_state_t;

typedef struct ota_cfg {
    ota_image_t   image;
    ota_state_t   state;
} ota_cfg_t;
```

ota_image_t 是 Image 区域的序号，OTA_IMAGE_1ST 表示 Image 区域一，OTA_IMAGE_2ND 表示 Image 区域二。ota_state_t 是固件校验状态，OTA_STATE_UNVERIFIED 表示固件校验失败，OTA_STATE_VERIFIED 表示固件校验成功。

从 Boot config 读取到 ota_cfg_t 类型数据后，其两个成员 ota_image_t 和 ota_state_t 的值组合如下：

1. OTA_IMAGE_1ST 和 OTA_STATE_VERIFIED：最新已校验通过的固件位于 Image 区域一，则系统启动从 Image 区域一加载固件，OTA 升级更新 Image 区域二的固件。
2. OTA_IMAGE_1ST 和 OTA_STATE_UNVERIFIED：Image 区域一的固件未校验通过，则系统启动从 Image 区域二加载固件，OTA 升级更新 Image 区域一的固件。
3. OTA_IMAGE_2ND 和 OTA_STATE_VERIFIED：最新已校验通过的固件位于 Image 区域二，则系统启动从 Image 区域二加载固件，OTA 升级更新 Image 区域一的固件。
4. OTA_IMAGE_2ND 和 OTA_STATE_UNVERIFIED：Image 区域二的固件未校验通过，则系统启动从 Image 区域一加载固件，OTA 升级更新 Image 区域二的固件。

1.2.2 OTA protocol

OTA protocol 表示 OTA 升级时下载固件的协议。在 image.h 中定义如下：

```
typedef enum ota_protocol {
    OTA_PROTOCOL_FILE    = 0,
    OTA_PROTOCOL_HTTP    = 1,
} ota_protocol_t;
```

当前 SDK 中支持通过 Http 和 File 两种协议下载固件。

1.2.3 OTA verify

OTA verify 表示对下载完固件的校验算法。为保证固件在 OTA 下载和烧写 Flash 过程中不出错，可采用 CRC32、MD5、SHA1 或 SHA256 算法对固件进行校验。

```
typedef enum ota_verify {
    OTA_VERIFY_NONE     = 0,
    OTA_VERIFY_CRC32    = 1,
    OTA_VERIFY_MD5      = 2,
    OTA_VERIFY_SHA1     = 3,
    OTA_VERIFY_SHA256   = 4,
} ota_verify_t;
```


2 使用说明

2.1 代码位置

相关代码请参考：

sdk-code/include/sys/ota.h

sdk-code/src/ota/

2.2 接口说明

下面对 OTA 模块提供的接口进行简要说明。

1. 初始化 OTA 模块。输入参数 param 为初始化参数结构体指针，初始化参数可通过 Image 模块提供的接口 image_get_ota_param()获得。参数主要包括两个 Image 区域的位置、大小以及 Bootloader 区域大小等信息。执行成功，返回 OTA_STATUS_OK；执行失败，返回 OTA_STATUS_ERROR。

```
ota_status_t ota_init(image_ota_param_t *param);
```

2. 反初始化 OTA 模块。

```
void ota_deinit(void);
```

3. 读取 Boot config 信息，系统启动时和固件升级时根据 Boot config 信息选择 Image 区域。输入参数为 config 的指针。执行成功，返回 OTA_STATUS_OK；执行失败，返回 OTA_STATUS_ERROR。

```
ota_status_t ota_read_cfg(ota_cfg_t *cfg);
```

4. 写入 Boot config 信息，完成对下载固件的校验后更新 Boot config 信息。输入参数为 config 的指针。执行成功，返回 OTA_STATUS_OK；执行失败，返回 OTA_STATUS_ERROR。

```
ota_status_t ota_write_cfg(ota_cfg_t *cfg);
```

5. 通过指定协议下载固件。输入参数 protocol 为所选择的下载固件的协议。输入参数 url 为固件统一资源定位符。下载成功，返回 OTA_STATUS_OK；下载失败，返回 OTA_STATUS_ERROR。

```
ota_status_t ota_get_image(ota_protocol_t protocol, void *url);
```

6. 通过指定算法校验下载的固件，并根据校验结果更新 Boot config 信息。即使不采用其他校验算法也应该调用该接口来更新 Boot config 信息（详情参见示例）。输入参数 verify 为指定的校验算法。输入参数 value 为正确固件通过指定校验算法得到的校验值。执行成功（指函数执行成功，不等价于校验通过），返回 OTA_STATUS_OK；执行失败（指函数执行失败，不等价于校验不通过），返回 OTA_STATUS_ERROR。

```
ota_status_t ota_verify_image(ota_verify_t verify, uint32_t *value);
```

7. 重启系统。

```
void ota_reboot(void);
```


2.3 使用示例

2.3.1 系统启动

下面是系统启动时 Bootloader 中调用 OTA 模块接口的例子：

```
//初始化 OTA 模块，参数 param 通过接口 image_get_ota_param()获得
ota_init(&param);

//读 Boot config 信息
if (ota_read_cfg(&cfg) != OTA_STATUS_OK) {
    ……                                  //出错处理
}

//根据 Boot config 信息选择 Image 区域
if (((cfg.image == OTA_IMAGE_1ST) && (cfg.state == OTA_STATE_VERIFIED))
    || ((cfg.image == OTA_IMAGE_2ND) && (cfg.state == OTA_STATE_UNVERIFIED))) {
    *image_seq = IMAGE_SEQ_1ST;         //选择 Image 区域一
} else if (((cfg.image == OTA_IMAGE_2ND) && (cfg.state == OTA_STATE_VERIFIED))
        ||((cfg.image == OTA_IMAGE_1ST) && (cfg.state == OTA_STATE_UNVERIFIED)))
{
    *image_seq = IMAGE_SEQ_2ND;         //选择 Image 区域二
} else {
    ……                                  //出错处理
}
```

2.3.2 升级固件

以 Http 协议下载固件为例，假设固件的 url 为 http://192.168.1.100/OTA/xr_system.img，升级固件的示例如下：

```
char url[] = "http://192.168.1.100/OTA/xr_system.img";

//通过 Http 协议下载固件
if (ota_get_image(OTA_PROTOCOL_HTTP, url) != OTA_STATUS_OK) {
    ……                          //出错处理
}

//校验固件，此处选择不进行其他校验算法校验，调用该接口将更新 Boot config 信息
if (ota_verify_image(OTA_VERIFY_NONE, NULL)  != OTA_STATUS_OK) {
    ……                          //出错处理
}

//重启系统
ota_reboot();
```

2.3.3 协议扩展

OTA 模块已提供 Http 和 File 两种协议下载固件，此外还支持扩展其他协议。扩展协议的方式为：

1. 在文件 ota.h 中将扩展的协议补充到数据类型 ota_protocol_t。

```
typedef enum ota_protocol {
    OTA_PROTOCOL_FILE    = 0,
    OTA_PROTOCOL_HTTP    = 1,
    OTA_PROTOCOL_XXXX    = 2,
} ota_protocol_t;
```

2. 实现扩展协议下载固件的两个回调函数，回调函数类型在文件 ota_i.h 中定义如下。可参考 Http 协议和 File 协议对两个回调函数的实现（ota_http.h、ota_http.c、ota_file.h、ota_file.c）。

```
typedef ota_status_t (*ota_update_init_t)(void *url);
typedef ota_status_t (*ota_update_get_t)(uint8_t *buf, uint32_t buf_size,
                                         uint32_t *recv_size, uint8_t *eof_flag);
```

3. 将扩展协议的两个回调函数注册到文件 ota.c 的函数 ota_get_image()中。

```
case OTA_PROTOCOL_FILE:
    return ota_update_image(url, ota_update_file_init, ota_update_file_get);
case OTA_PROTOCOL_HTTP:
    return ota_update_image(url, ota_update_http_init, ota_update_http_get);
case OTA_PROTOCOL_XXXX:
    return ota_update_image(url, ota_update_xxxx_init, ota_update_xxxx_get);
```

