公众号【IOT物联网小镇】

点击 IOT物联网小镇

在最近的两篇文章中，我们从概念和流程上梳理了: 一个终端设备如何把一个固件，安全无误的从服务器上，下载到本地。

文章链接在此:

物联网设备OTA软件升级之：升级包下载过程之旅

物联网设备OTA软件升级之：完全升级和增量升级

这篇文章就继续往下深入，以一个实际的 ESP32 项目，来完整的梳理一下 OTA 升级的全过程。

主要包括下面 3 部分内容:

> 1. AWS 平台上，部署一个 OTA 升级任务时，需要完成哪些步骤;

> 2. ESP32 模组中，关于 Flash 分区和 OTA 升级控制过程和代码说明;
> 3. 如何通过 ESP32，给与之相连的 MCU 进行 OTA 升级;

PS: 在下面的内容中，终端设备指的就是 ESP32 模组。

ESP32 Flash 分区

其实 ESP32 的官方文档的过程描述，已经是非常的详细了。

不仅把每一个操作的步骤都写的很清楚，而且把一些可能遇到的错误，都会做一些善意的提醒。

下面这部分内容，基本上是来源于官方的文档。

我们这里只是把一些与本文相关的、比较重要的内容摘录在这里。

首先要了解的，肯定是 Flash 的分区信息了。

所有的固件、数据，都要存储在 Flash 中，它是一个系统的记忆部件，离开了它，再怎么聪明的 CPU 都无用武之地。

关于分区表，ESP32 中预定义了 2 份分区表，分别对应：是否存在 OTA 功能这两种情况，截图如下：

没有 OTA 的分区表：

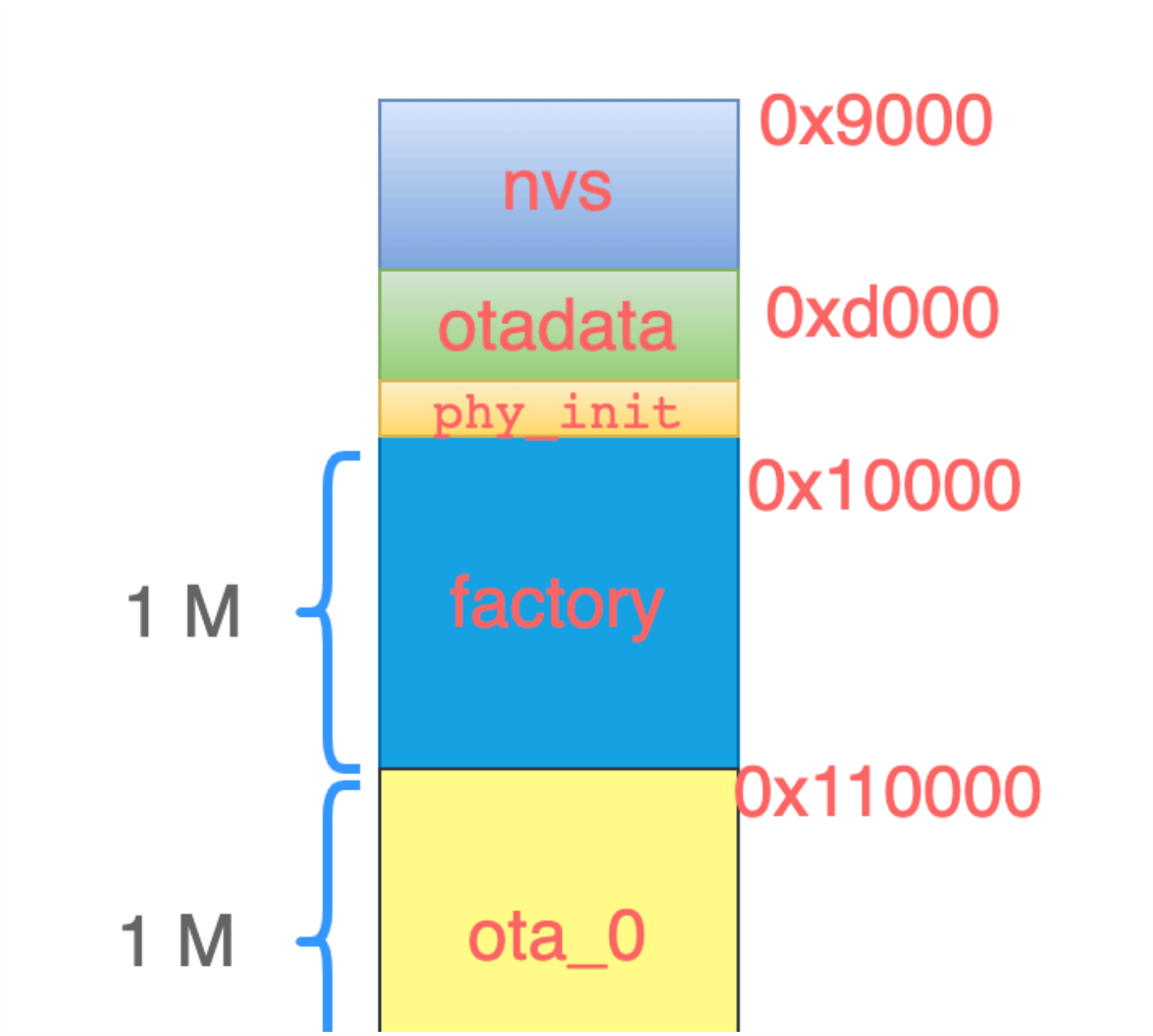
```
# ESP-IDF Partition Table
# Name,   Type, SubType, Offset,  Size,   Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
```

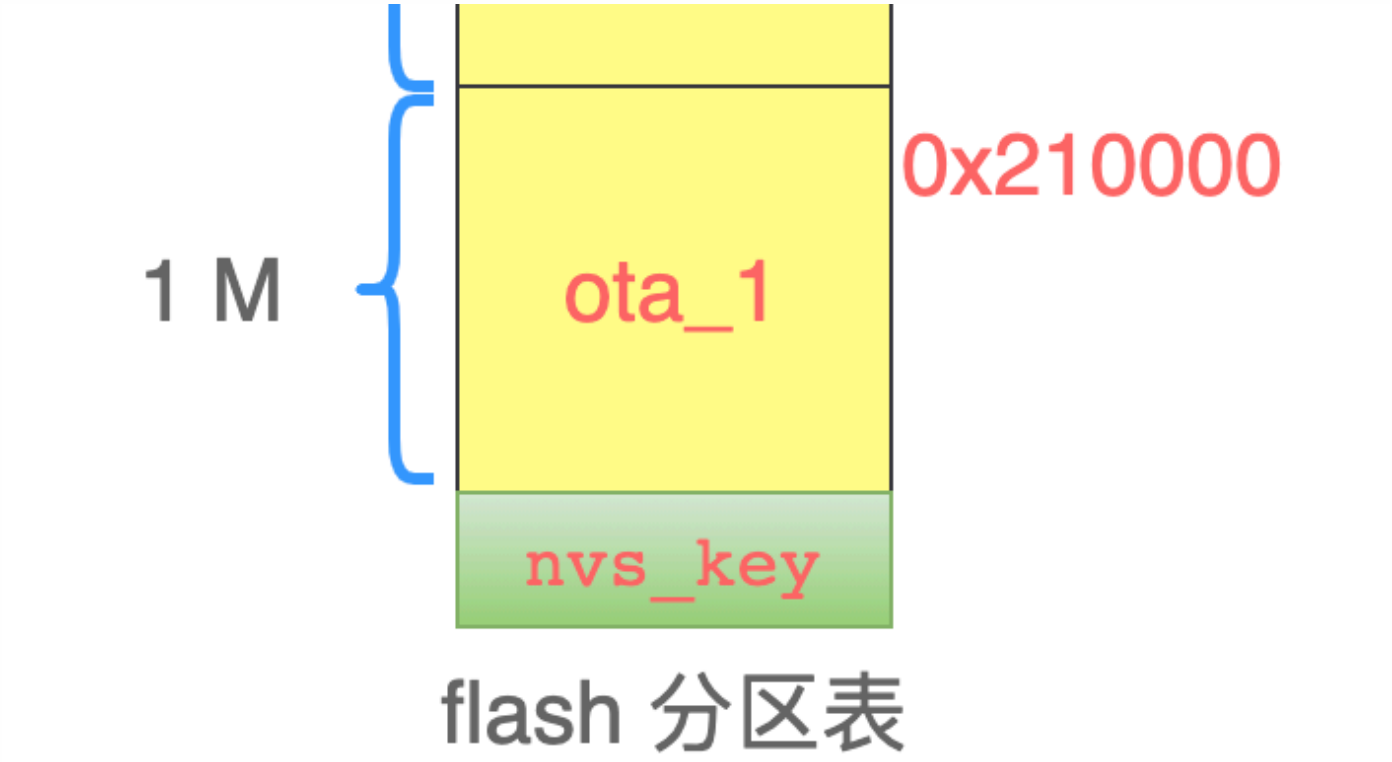
有 OTA 功能的分区表：

```
# ESP-IDF Partition Table
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x4000,
otadata,  data, ota,     0xd000,  0x2000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000,  1M,
ota_0,    app,  ota_0,   0x110000, 1M,
ota_1,    app,  ota_1,   0x210000, 1M,
```

官方的文档链接在这里。

既然我们是在描述 OTA 过程，那肯定就是以带有 OTA 功能的这个分区表为准了。





在这张分区表中，一共定义了 3 个应用程序分区：

> factory 分区;
> ota_0 分区;
> ota_1 分区;

这三个分区的类型都是 app，但具体 app 的类型不相同。

其中，位于 0x10000 偏移地址处的为出厂应用程序（factory），其余两个为 OTA 应用程序（ota_0，ota_1）。

名为 otadata 的数据分区，用于保存 OTA 升级时需要的数据。

启动加载器会查询该分区(otadata)的数据，以判断：应该从哪个 OTA 应用程序分区来加载程序。

如果 otadata 分区为空(说明这台设备还没有进行过 OTA 升级)，则会执行出厂程序，也就是执行 factory 分区中的固件程序。

如果 otadata 分区非空，则启动加载器将加载这个分区中的数据，进而判断: 启动哪个 OTA 镜像文件。

AWS 平台部署 OTA 升级任务

AWS 平台按照不同的业务类型，划分为不同的服务。这样处理起来，流程更规范，操作步骤也更多，当然也更赚钱一些！

从上一篇文章中可以看到，当一个新的固件准备好之后，需要做 2 件事情：

> 1. 把固件(bin 文件)和一个固件描述文件(json格式的文本文件)，上传到 S3 云存储服务器上;
> 2. 在 AWS Core 任务管理中，新建一个升级任务(会得到一个 Job ID)。在这个任务中需要选择：
>
>> (1) 步骤1中上传的 json 文件;

> (2) 哪些终端设备需要升级;

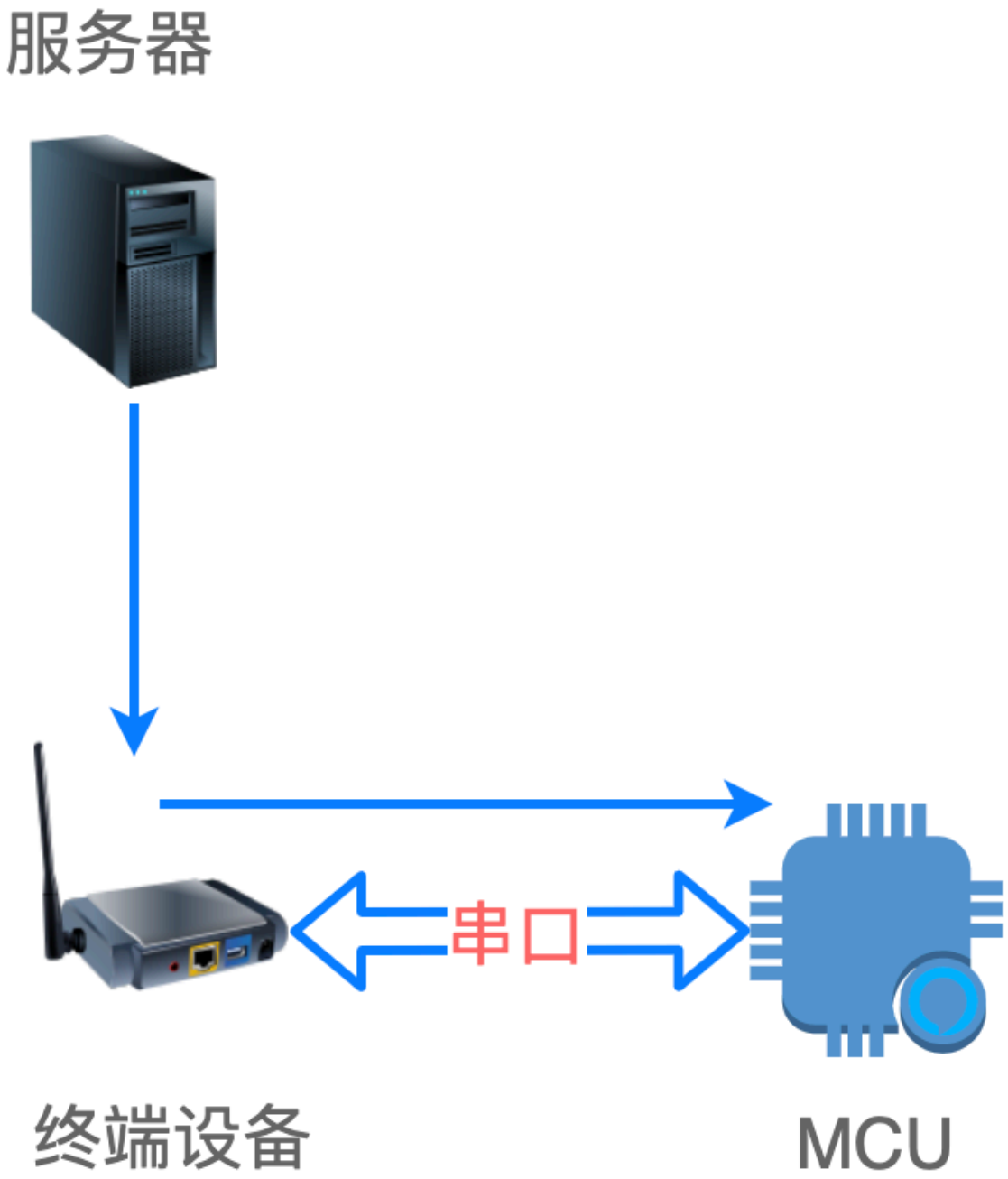
json 格式的固件描述文档，格式大概如下(可以根据实际的业务需求进行修改):

```
{
  "product": "产品名称",
  "group": "设备分组",
  "firmware":
  [
     {
        "ota_type": "esp32",
        "url": "http://xxx/esp32-v1.1.0.bin",
        "md5": "xxx"
     }
  ]
}
```

不知道您是否注意到：在 firmware 字段中，使用的是数组([...])，而不是对象({...})?

这样来组织的原因是，OTA 升级不仅仅可以对 ESP32 模组中的固件进行升级("ota_type": "esp32")，还可以对其他的一些固件或用户数据进行更新。

比如：更新 ESP32 串口连接的 MCU 中的固件程序。



对了，一个终端在通过网络连接到云平台时，都有一个唯一的 ID 编号，一般都是利用 ESP32 模组上的网卡 MAC 地址来作为唯一 ID。

当完成以上步骤时，在服务器端，就存在着一个升级任务关系链：

也就是说：一个 Job ID 就对应着一次 OTA 升级任务。终端设备在进行 OTA升级过程中，就是从这个 Job ID 开始的。

ESP32 OTA 升级的触发

ESP32 与 AWS 平台之间，是通过 MQTT 协议进行通信的。

因此，当运营人员创建了一个 OTA 升级任务后，所有相关的终端设备，必须从某个预先确定好的主题(topic)中，接收到 OTA 升级通知指令。

例如一个可能的 topic： $aws/things/xxx/job/notify

其中的 xxx，代表终端设备的 MAC 地址，只有这样，每一个设备才能够接收到属于自己的命令。

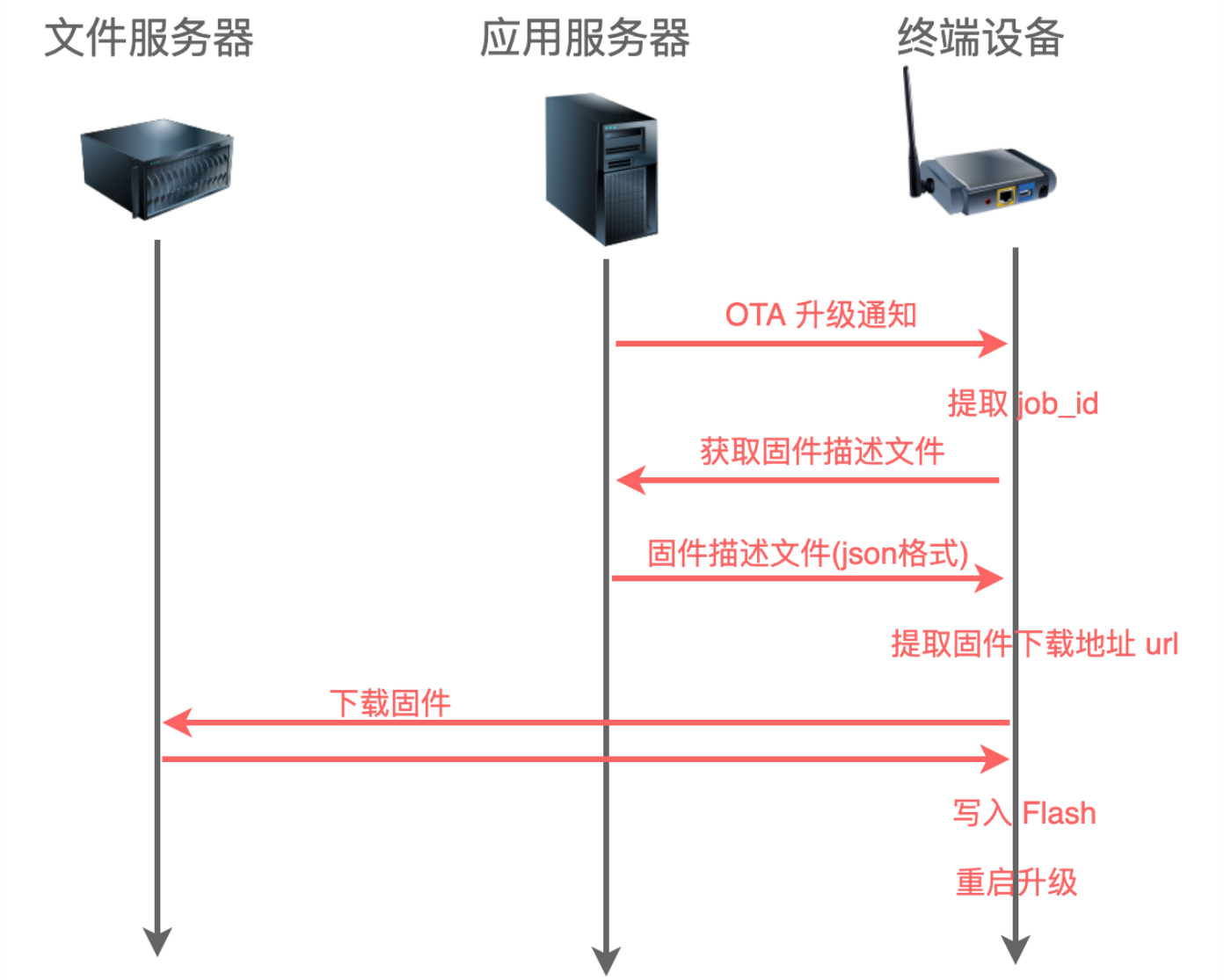
升级通知指令的内容中，一定会包含 OTA 升级的 Job ID，例如：

```
{
  "timestamp": "xxxxxx",
  "job_id": "001"
}
```

当终端设备接收到这个升级通知指令时，提取出 job_id 字段，然后向云平台发起请求：获取与这个 job_id 关联的固件描述信息，也就是之前上传的 Json 格式的文件息。

AWS 平台接收到这个请求后，就会把与这个 job_id 相关联的 OTA 升级任务描述文件(json文件)，发送给终端设备。

设备拿到了固件描述文件，自然也就知道了固件的：版本，下载地址，MD5 值等信息，于是就进入后面的下载环节了。



以上的过程描述，基本上是一个终端设备触发 OTA 升级的最基本的过程。

在实际的项目中，可能会遇到一些稍微复杂的情况。

例如：一个终端设备一直处于断电状态。此时，云平台中已经对固件进行了好几次的升级，但是由于这台设备一直没有运行，因此它的固件已经过时了好几个版本。

有一天，这台设备上电运行了，此时它会从云平台接收到好几个升级任务，这个时候应该如何处理呢?

也许，我们就要对升级通知的指令中，赋予更多详细的内容，让这台设备有足够的信息来判断该如何进行升级。

ESP32 固件下载和本地升级

ESP32 在提取出固件的下载地址(URL)之后，就开始进入下载环节了。

官方文档非常详细的描述了固件的下载过程。

下面这段代码，就是从官方文档中摘抄过来的:

公众号【IOT物联网小镇】

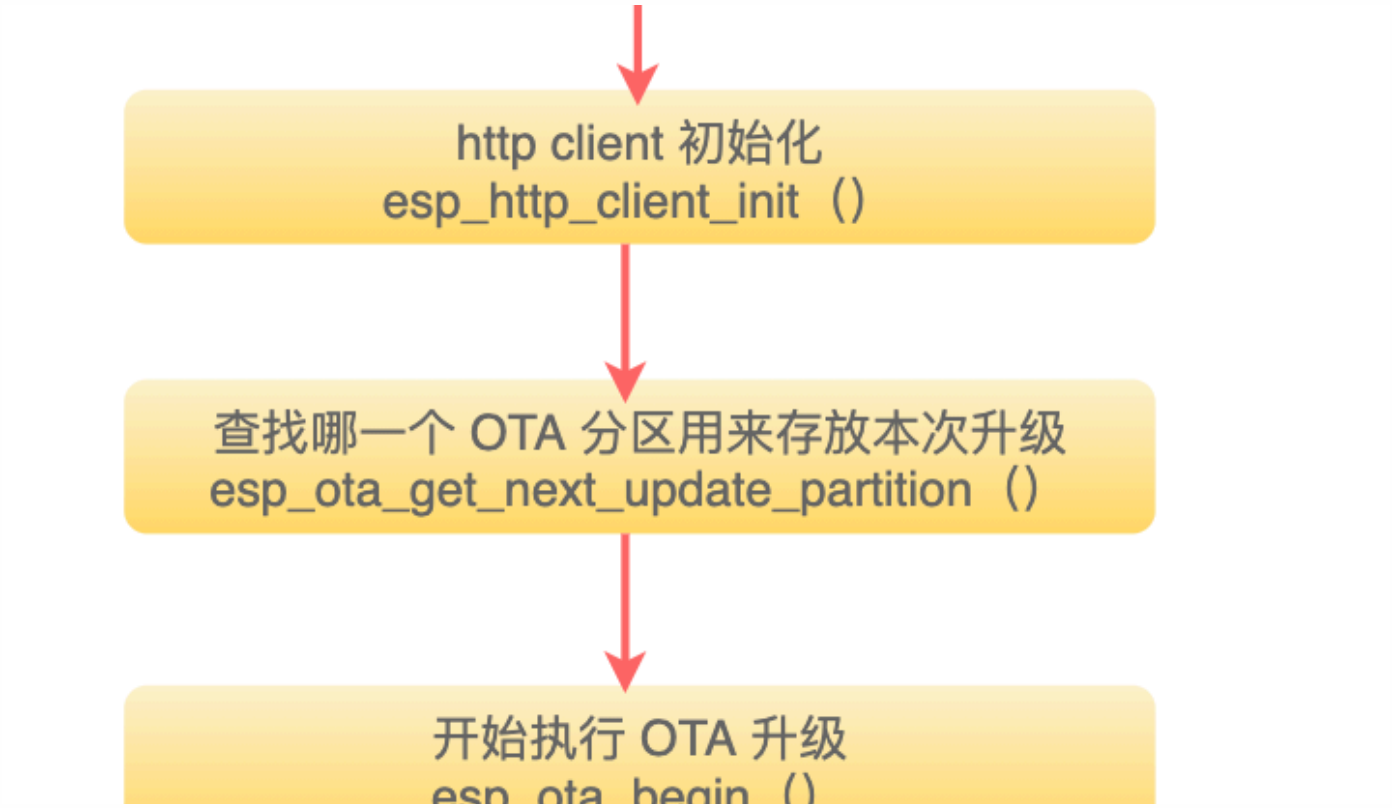
链接地址：https://docs.espressif.com/projects/esp-idf/zh_CN/latest/esp32/api-reference/system/ota.html

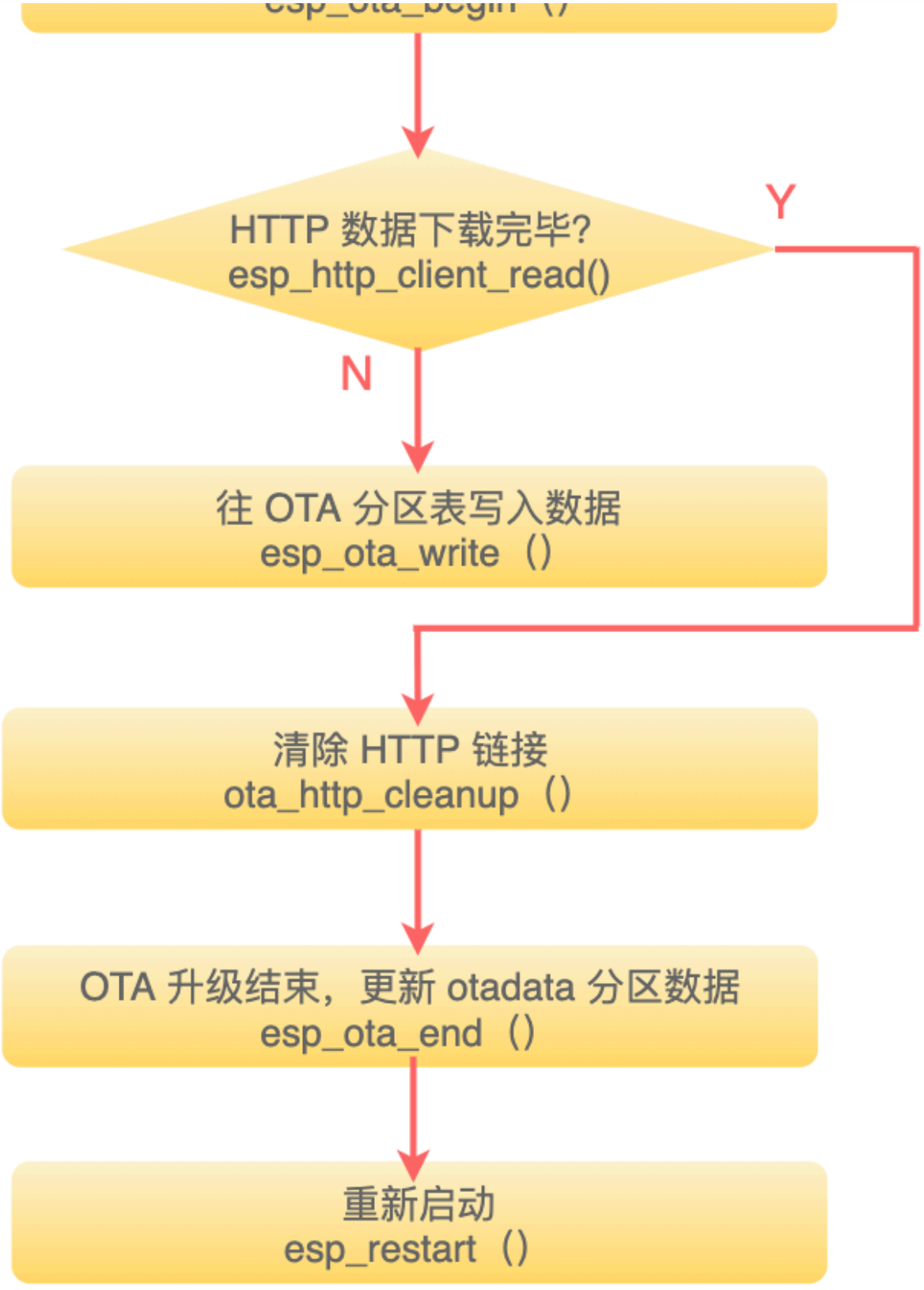
```
bool image_header_was_checked = false;
while (1) {
    int data_read = esp_http_client_read(client, ota_write_data, BUFFSIZE);
    ...
    if (data_read > 0) {
        if (image_header_was_checked == false) {
            esp_app_desc_t new_app_info;
            if (data_read > sizeof(esp_image_header_t) +
sizeof(esp_image_segment_header_t) + sizeof(esp_app_desc_t)) {
                // check current version with downloading
                if (esp_efuse_check_secure_version(new_app_info.secure_version) ==
false) {
                    ESP_LOGE(TAG, "This a new app can not be downloaded due to a
secure version is lower than stored in efuse.");
                    http_cleanup(client);
                    task_fatal_error();
                }

                image_header_was_checked = true;

                esp_ota_begin(update_partition, OTA_SIZE_UNKNOWN, &update_handle);
            }
        }
        esp_ota_write( update_handle, (const void *)ota_write_data, data_read);
    }
}
```

把这个过程画成流程图，就是下面这个样子：





我们假设这次固件升级，存储在 ota_0 这个分区中。

在固件下载完毕之后，esp_ota_end() 函数会在 otadata 分区写入一个标记: 下次启动时，请加载 ota_0 分区中的固件程序。

当 ESP32 重新启动时，启动加载器从 otadata 分区读取数据，得知这一次需要启动 ota_0 分区里的固件。

此时有一件很重要的事情需要做：当 ota_0 分区中的固件启动正确无误后，需要调用函数 esp_ota_mark_app_valid_cancel_rollback() 往 otadata 区写 ESP_OTA_IMG_VALID，标记: 这个分区中的固件是没有问题的!

这样的话，以后每次重启时，都会加载 ota_0 分区里的固件。

相反的情况：如果 ota_0 分区里的固件，在第一次启动后新固件运行有问题，需要调用函数 esp_ota_mark_app_invalid_rollback_and_reboot() 往 otadata 区写 ESP_OTA_IMG_INVALID ，标记：这个分区中的固件有问题！

这样的话，重启之后，启动加载器将会选择之前的 app 分区里的固件，可能是 factory 分区，也可能是 ota_1 分区。

OTA 升级过程中断了，怎么办?

以上描述的过程都是理想的情况，那么如果遇到一些异常情况，该如何处理呢?

例如：从接收到固件描述信息，到固件下载完成。在这期间的任何一个时间点，如果因为断电等原因，导致设备重启了，该如何继续 OTA 升级过程?

我们知道，在程序运行的时候，所有的数据都是保存在内存中的。

重启之后，内存中的数据是一篇空白。

如果希望 OTA 升级过程可以在任何异常情况下都能顺利进行，必须保存一些必要的信息，包括:

1. json 格式的固件描述文件;
2. 固件下载过程中已经完成的每一个阶段;

这些信息可以调用 nvs_write() 函数，保存在非易失性存储设备中。

即使系统因为断电等原因重启了，也可以通过 nvs_read() 函数，读取之前已经完成的步骤，然后继续后续的升级操作。

通过 ESP32，升级 MCU 固件

ESP32 模组，仅仅是一个用来连接网络云平台的无线设备。

对于一个实际的产品而言，发挥实际功能控制作用的，往往是另一片单片机，比如: STM32。

单片机中的固件也有可能需要进行 OTA 升级，此时 ESP32 就要作为中间的一个媒介，先把 MCU 固件下载下来存储在本地，然后再通过串口发送给单片机。

在这种情况下，ESP32 接收到的 OTA 固件描述信息就有可能是下面这个样子:

```
{
  "product": "产品名称",
  "group": "设备分组",
  "firmware":
  [
     {
        "ota_type": "stm32",
        "url": "http://xxx/mcu-v1.2.3.bin",
        "md5": "xxx"
     }
  ]
}
```

从 ota_type 字段，可以知道这次是给 MCU 进行升级，接下来的下载过程就与上述流程很类似了。

唯一的区别就是：下载的时候，需要把固件保存到 Flash 上的一块独立的数据分区中，而不是 ota_0 或 ota_1 分区。

------ End ------

至此，关于 ESP32 模组以及 MCU 的 OTA 升级过程就基本描述完毕了。

以上这些内容，都是是一些结构性的流程节点，剩下部分就是一些细节问题了，按照官方文档的指导步骤，都可以顺利完成开发！

推荐阅读

【1】C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
【2】一步步分析-如何用C实现面向对象编程
【3】原来gdb的底层调试原理这么简单
【4】内联汇编很可怕吗？看完这篇文章，终结它！
【5】都说软件架构要分层、分模块，具体应该怎么做

[这里插入：微信公众号]

公众号【IOT物联网小镇】

扫码添加 道哥 私人微信

C/C++、物联网

Linux 操作系统

Linux 应用开发

喜欢请分享，满意点个赞，最后点在看。