第一章 ESP32 的 NVS 测试

1. 学习目的及目标

- 掌握 NVS 概念
- 掌握 NVS 库函数接口
- 编写 NVS 存储多种类型

2. 乐鑫 NVS 简介（原文）

非易失性存储（Non-volatile storage）简称 NVS，乐鑫使用一套 NVS 库将键值对保存在 SPI flash 中。

NVS 库可以使用 read、write、erase 的 API 操作 flash 的一部分，该库使用 data 类型和 nvs 子类型的所有分区。应用程序可以使用 nvs_openAPI 选用 nvs 表中的分区或通过 nvs_open_from_part API 指定其名称后使用其他分区。

2.1. NVS 可以保存的类型

- 整型: uint8_t, int8_t, uint16_t, int16_t, uint32_t, int32_t, uint64_t, int64_t
- 字符串：必须以 0 结尾，因为需要知道字符串的长度，以便保存。
- 二进制数据：可变长。
- 暂不支持浮点数保存
- 字符串和二进制数据目前仅限于 1984 字节。对于字符串，这包括空终止符。

2.2. NVS 的命名空间

为了缓解不同组件之间的密钥名称之间的潜在冲突，NVS 将每个键值对分配给一个名称空间，类似数据库中的表。名称空间名称遵循与键名相同的规则，即最多 15 个字符。命名空间名称在 nvs_open 或 nvs_open_from_part 调用中指定。随后调用的 nvs_read_*，nvs_write_*和 nvs_commit 将返回不透明句柄。这样，句柄与名称空间相关联，并且键名不会与其他名称空间中的相同名称相冲突。

在不同 NVS 分区中具有相同名称的名称空间被视为单独的名称空间。

2.3. 存储结构

详细了解请看上面原文。

一般存储配置信息不需要，当存储大量数据时，需要考虑分区等问题。

3. NVS 优势（原文）

3.1. 接口更加安全

相比较于 spi_flash_read 和 spi_flash_write 等接口, NVS 不直接操作 address. 对于终端用户而已, 更加安全.

例如: 应用复杂一点, 容易 spi_flash_write(address, src, size) 不小心写到同一个地址, 或地址写覆盖, 而导致长时间 debug

3.2. 接口使用接近用户习惯

NVS 接口类似于电脑上操作文件一样:

打开文件(nvs_open), 写文件(nvs_set_xxx), 保存文件(nvs_commit), 关闭文件(nvs_close)

打开文件(nvs_open), 读取文件(nvs_get_xxx), 关闭文件(nvs_close)

3.3. 擦写均衡，使 flash 寿命更长

NVS 在操作少量数据上，NVS 分区更大时，擦写均衡表现的更为明显.

例如：flash 一个 sector 为 4KB，NVS 分配大小为一个 sector，写同一个 64 Bytes 数据到 flash，分别比较 spi_flash_xxx 和 nvs 写 64 次

spi_flash_write：每次写 flash 前，需擦除 flash. 对应：64 次擦除 flash，64 次写 flash

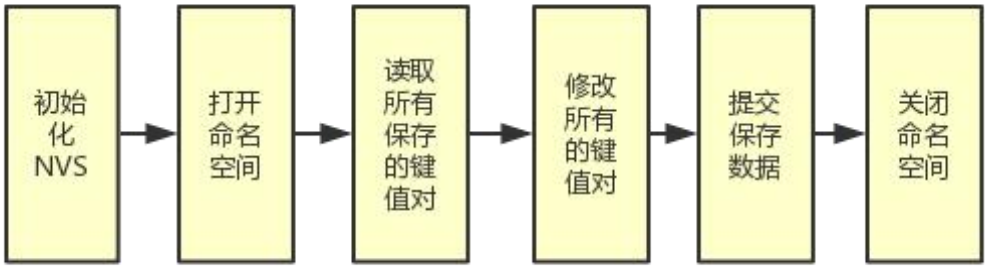
nvs：nvs 内部有擦写均衡，有标志位记录当前有效存储. 如第一次擦除 sector，再写 sector 0-63 Byte，第二次写 sector 64-127 Bytes，第 64 次(4KB/64Bytes) 写完 sector 最后一个 64 Byte. 对应：1 次擦除 flash，64 次写 flash

这样 NVS 减少 64 倍擦除操作，对 flash 寿命有较大提升.

在 NVS 分区更大，存储信息少时，表现的更为明显.

4. 软件设计

1.1. NVS 使用逻辑



1.2. ESP32 NVS 接口介绍

- NVS 初始化函数：nvs_flash_init(void);
- NVS 擦除函数：nvs_flash_erase(void);
- NVS 打开命名空间函数：nvs_open();

参数原型	esp_err_t nvs_open (const char* name, nvs_open_mode open_mode, nvs_handle *out_handle)
参数功能	NVS 打开命名空间函数
参数	[in] name：命名空间名称，最长 15 字符 [in] open_mode：打开模式 NVS_READWRITE ：读写模式 NVS_READONLY：只读模式 [in] out_handle：NVS 句柄
返回值	ESP_OK：打开成功

- NVS 写键值对函数：
  - esp_err_t nvs_set_i8 (nvs_handle handle, const char* key, int8_t value);
  - esp_err_t nvs_set_u8 (nvs_handle handle, const char* key, uint8_t value);
  - esp_err_t nvs_set_i16 (nvs_handle handle, const char* key, int16_t value);
  - esp_err_t nvs_set_u16 (nvs_handle handle, const char* key, uint16_t value);
  - esp_err_t nvs_set_i32 (nvs_handle handle, const char* key, int32_t value);
  - esp_err_t nvs_set_u32 (nvs_handle handle, const char* key, uint32_t value);
  - esp_err_t nvs_set_i64 (nvs_handle handle, const char* key, int64_t value);
  - esp_err_t nvs_set_u64 (nvs_handle handle, const char* key, uint64_t value);
  - esp_err_t nvs_set_str (nvs_handle handle, const char* key, const char* value);
- NVS 读键值对函数：
  - esp_err_t nvs_get_i8 (nvs_handle handle, const char* key, int8_t* out_value);
  - esp_err_t nvs_get_u8 (nvs_handle handle, const char* key, uint8_t* out_value);
  - esp_err_t nvs_get_i16 (nvs_handle handle, const char* key, int16_t* out_value);
  - esp_err_t nvs_get_u16 (nvs_handle handle, const char* key, uint16_t* out_value);
  - esp_err_t nvs_get_i32 (nvs_handle handle, const char* key, int32_t* out_value);
  - esp_err_t nvs_get_u32 (nvs_handle handle, const char* key, uint32_t* out_value);
  - esp_err_t nvs_get_i64 (nvs_handle handle, const char* key, int64_t* out_value);
  - esp_err_t nvs_get_u64 (nvs_handle handle, const char* key, uint64_t* out_value);
  - esp_err_t nvs_get_str (nvs_handle handle, const char* key, char* out_value, size_t* length);
- 擦除键值对函数：esp_err_t nvs_erase_key(nvs_handle handle, const char* key);
- 擦除整个命名空间函数：esp_err_t nvs_erase_all(nvs_handle handle);
- 关闭命名空间函数：void nvs_close(nvs_handle handle);

更多更详细接口请参考官方指南。

1.3. 代码编写

- NVS 读写

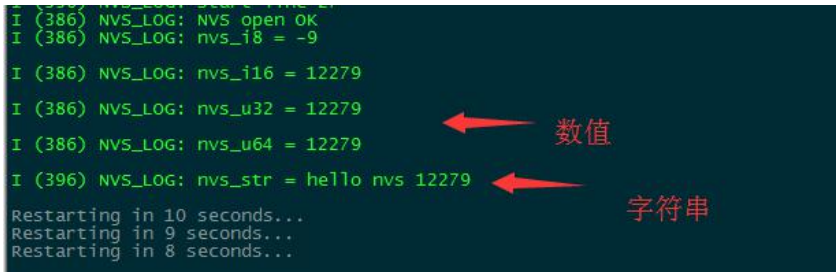
```
1  void app_main()
2  {
3       ESP_LOGI(TAG, "start line %d",__LINE__);//打印行号
4      //初始化 NVS
5      esp_err_t err = nvs_flash_init();
6      if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
7          //发现新版本
8          //擦除
9          ESP_ERROR_CHECK(nvs_flash_erase());
10         err = nvs_flash_init();
```

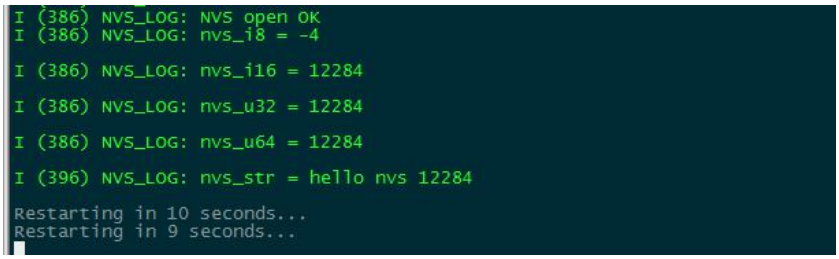
```
    }
    //打开,类似数据库的表
    err = nvs_open("hx_list", NVS_READWRITE, &my_handle);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "opening NVS Error (%s)!\n", esp_err_to_name(err));
    } else {
           ESP_LOGI(TAG, "NVS open OK");

        //读取，类似数据读字段对应的值
        err = nvs_get_i8(my_handle, "nvs_i8", &nvs_i8);
        if(err==ESP_OK) ESP_LOGI(TAG, "nvs_i8 = %d\n", nvs_i8);
        err = nvs_get_i16(my_handle, "nvs_i16", &nvs_i16);
        if(err==ESP_OK) ESP_LOGI(TAG, "nvs_i16 = %d\n", nvs_i16);
        err = nvs_get_u32(my_handle, "nvs_u32", &nvs_u32);
        if(err==ESP_OK) ESP_LOGI(TAG, "nvs_u32 = %d\n", nvs_u32);
        err = nvs_get_u64(my_handle, "nvs_u64", &nvs_u64);
        if(err==ESP_OK) ESP_LOGI(TAG, "nvs_u64 = %llu\n", nvs_u64);
        //字符串
        err = nvs_get_str (my_handle, str_key, str_value, &len);//100 是读取最大长度
        if(err==ESP_OK) ESP_LOGI(TAG, "nvs_str = %s\n", str_value);

        //修改字段的值
        //写数值
        nvs_i8+=1;
        nvs_i16+=1;
        nvs_u32+=1;
        nvs_u64+=1;
        err = nvs_set_i8(my_handle, "nvs_i8", nvs_i8);
        if(err!=ESP_OK) ESP_LOGE(TAG, "nvs_i8 Error");
        err = nvs_set_i16(my_handle, "nvs_i16", nvs_i16);
        if(err!=ESP_OK) ESP_LOGE(TAG, "nvs_i16 Error");
        err = nvs_set_u32(my_handle, "nvs_u32", nvs_u32);
        if(err!=ESP_OK) ESP_LOGE(TAG, "nvs_u32 Error");
        err = nvs_set_u64(my_handle, "nvs_u64", nvs_u64);
        if(err!=ESP_OK) ESP_LOGE(TAG, "nvs_u64 Error");
        //写字符串
        sprintf(str_value,"hello nvs %llu",nvs_u64);
        err = nvs_set_str (my_handle, str_key, str_value);

        //提交，必须提交才能写入 NVS
        err = nvs_commit(my_handle);
        if(err!=ESP_OK) ESP_LOGE(TAG, "nvs_commit Error");

        //关闭
        nvs_close(my_handle);
     }
```

```
57      //复位查看值。。。。。
58      esp_restart();
59  }
```

5. 效果展示

- 10 秒定时操作一次 NVS





6. NVS 总结

- 主要学习 ESP32 NVS 的接口测试，方法很简单。
- 源码地址：https://github.com/xiaolongba/wireless-tech