GR54xx NVDS用户手册

版本： **1.0**

发布日期： **2024-08-20**

深圳市汇顶科技股份有限公司

商标声明

GOODiX 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗中或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田保税区腾飞工业大厦B座13层

电话：+86-755-33338828　　邮编：518000

网址：www.goodix.com

前言

编写目的

本文档主要介绍GR54xx NVDS系统的存储结构和关键特性、NVDS接口以及使用注意事项等，旨在帮助用户更好地使用NVDS存储数据。

读者对象

本文适用于以下读者：

- 芯片用户
- 开发人员
- 测试人员
- 技术支持工程师

版本说明

本文档为第1次发布，对应的产品系列为GR54xx。

修订记录

版本	日期	修订内容
1.0	2024-08-20	首次发布

目录

1 概述

非易失性数据存储（Non-volatile Data Storage，NVDS）是一个轻量级键值对数据存储系统，其利用Flash硬件抽象层（Flash HAL）提供的Flash读写接口，将数据存储于Flash中，可确保系统掉电后数据也不会丢失。

NVDS系统适用于存储小块数据，如应用程序的配置参数、校准数据、状态信息以及用户信息等。低功耗蓝牙（Bluetooth LE）协议栈也会使用NVDS存储设备绑定等参数。

NVDS系统具有以下特性：

- 每个存储项（tag）都具有唯一的tag ID用于标识。用户程序可以根据tag ID对数据内容进行读取和更改操作，而无需关心数据存储的物理地址。
- 针对Flash存储介质的特性进行了优化，支持数据校验、垃圾回收和擦写平衡。
- 存储区域的大小和起始地址可配置。Flash区域以扇区（Sector）为单位，一个Sector的大小为4 KB。NVDS存储区域可配置为若干个Sector，且配置的起始地址需按4 KB对齐。

GR54xx SDK默认使用Flash最后几个Sector作为NVDS存储区域，其起始地址为Flash结束地址减去NVDS区域大小。开发者可在*custom_config.h*中配置宏NVDS_START_ADDR和NVDS_NUM_SECTOR来指定NVDS起始地址和Sector数。需注意NVDS_NUM_SECTOR不包含NVDS垃圾回收区域，整个NVDS所占用的Sector数为“NVDS_NUM_SECTOR + 1”。

2 NVDS设计实现

本章主要介绍NVDS存储结构和关键特性，以帮助用户了解NVDS实现逻辑。

2.1 存储结构

NVDS在Flash中的存储结构如下图所示。NVDS占用连续的若干个Sector的Flash空间，Sector数在NVDS初始化时由用户指定。每个Sector的第一个Page存储NVDS header，其余Page存储NVDS item。最后一个Sector为垃圾回收（Garbage Collection，GC）区域，用于NVDS垃圾回收时备份item。

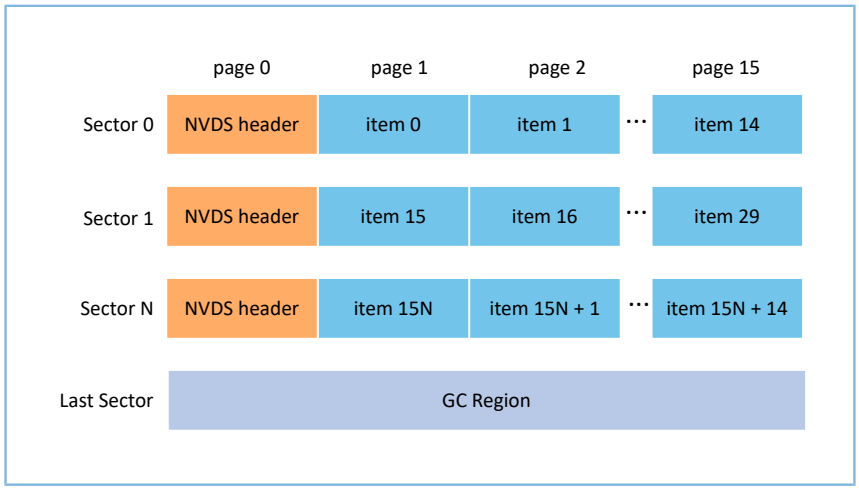

图 2-1 NVDS存储结构

根据NVDS存储结构，NVDS数据存储具有以下特性：

- 一个Flash Sector（4 KB）具有16个Page (256 Bytes)，最多存储15个NVDS item（按Page对齐）。
- 键值对（Key-Value）数据按item存储，最少占用一个item，最多占用15个item。
- 数据长度最大可达3720字节。当长度大于248字节时，将自动分为多个item进行存储。如果长度小于等于248字节，将独占一个item。

2.1.1 NVDS Header

NVDS header用于标识NVDS版本号和扇区信息，由“magic（8 Bytes）+ version（8 Bytes）+ sector info（8 Bytes）”组成，其结构如下图所示。

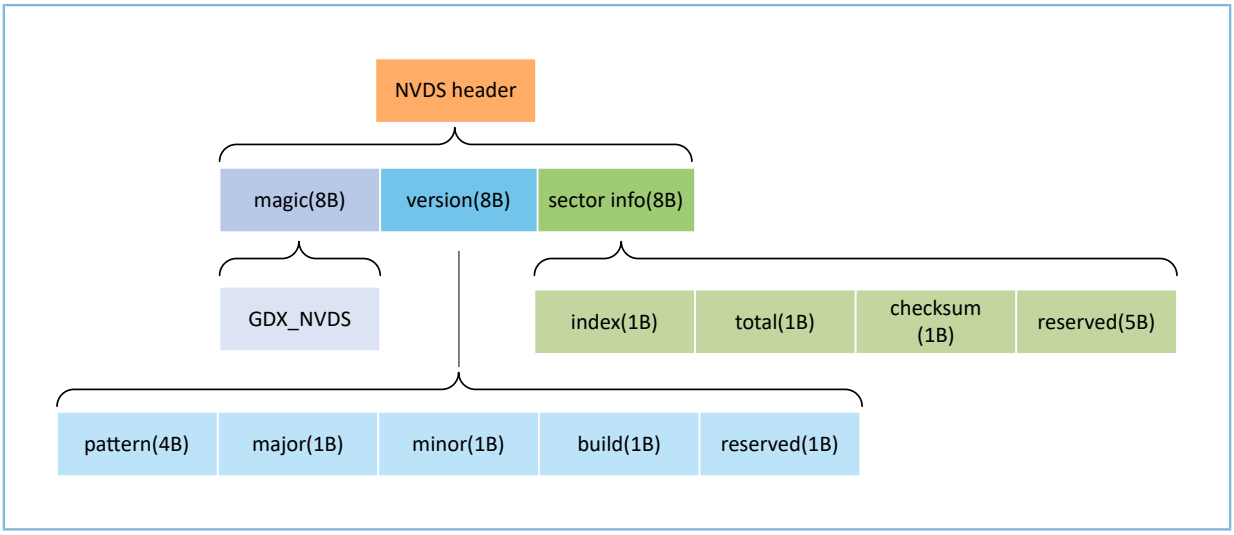

图 2-2 NVDS header结构

NVDS header各字段的具体描述参见下表：

表 2-1 NVDS header字段描述

字段名称	子字段名称	长度（Byte）	描述
magic	/	8	固定为“GDX_NVDS”字符串。
version	pattern	4	标识是否存在版本号。
	major	1	主版本号
	minor	1	子版本号
	build	1	Fix版本号
	reserved	1	保留位
sector info	index	1	当前Sector在所有NVDS Sector中的序号
	total	1	NVDS占用的Sector总数
	checksum	1	“index + total”的校验和
	reserved	5	保留位

2.1.2 NVDS Item

NVDS item由item header和item data组成，其结构如下图所示。

- item header：长度固定为8 Bytes，记录tag信息，包含ID、长度、item分块信息等，各字段的详细描述参见表 2-2 。
- item data：长度为1 ～ 248 Bytes，取决于tag内容。

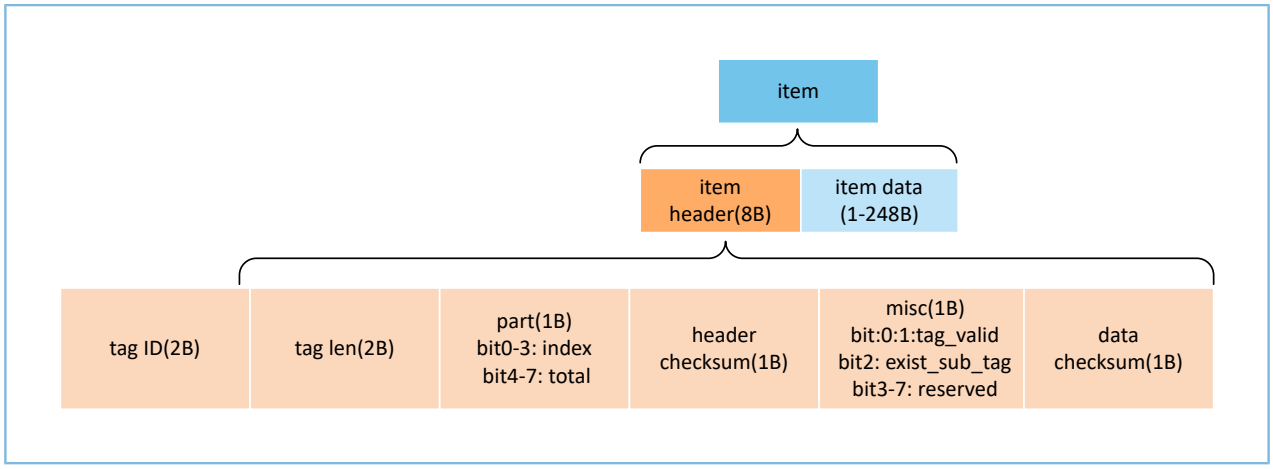

图 2-3 item结构

item header各字段的具体描述参见下表：

表 2-2 item header字段描述

字段名称	长度（Byte）	描述
tag ID	2	数据的tag标识
tag len	2	数据长度 **说明：** 单个tag最多占用15个item，即单个tag最大长度为15 * 248 = 3720 Bytes。
part	1	item分块信息 • bit 0 ～ bit 3：index，当前item序号 • bit 4 ～ bit 7：total，占用的item总数 **说明：** 若tag只占用一个item，则index和total均为1。
header checksum	1	数据头（“tag ID + tag len + part”）的校验和
misc	1	tag信息 • bit 0 ～ bit 1：tag valid，标识数据是否有效。0x02 # 有效；其他值 # 无效 • bit 2：sub tag，标识tag是否存在sub tag。 • bit 3 ～ bit 7：reserved，保留位
data checksum	1	当前item中数据的校验和

为了节省Flash空间，NVDS支持将多个tag存入一个item中，具体的sub tag结构如下图所示。当多个sub tag存入一个item时，由main tag决定sub tag存储于哪个item中。sub tag相邻存储，且不能跨越item。因此，sub tag数据的最大长度为256 - 8 - 8 = 240 Bytes。

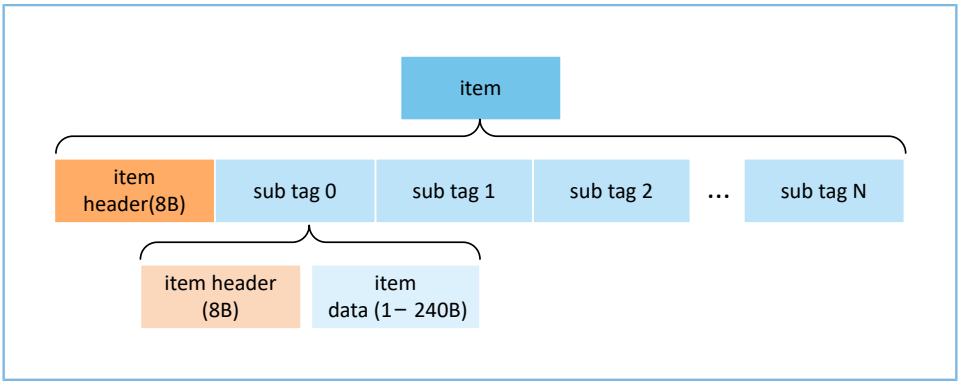

图 2-4 sub tag结构

2.1.3 GC区域

GC区域用于NVDS垃圾回收时备份item，其结构如下图所示：

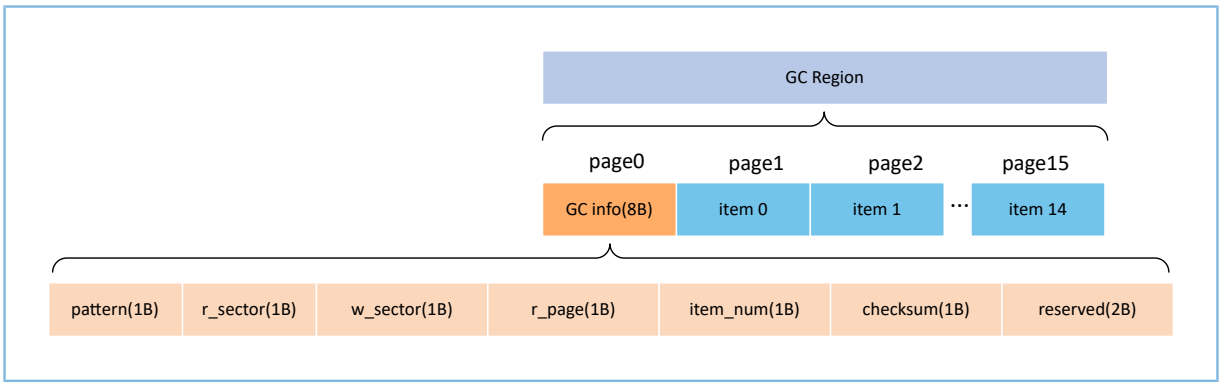

图 2-5 GC结构

GC info各字段具体描述参见下表：

表 2-3 GC信息字段(GC_info)描述

字段名称	长度（Byte）	描述
pattern	1	标识GC info是否有效。 • 0xA5：有效 • 其它值：无效
r_sector	1	指向正在读取的Sector
w_sector	1	指向正在写入的Sector
r_page	1	正在读取的Page的偏移量
item_num	1	GC区域中的有效item数
checksum	1	GC info（从pattern到item_num的数据）的校验和
reserved	2	保留位

2.2 关键特性

2.2.1 数据唯一性

item header中的tag用于标识数据。通过以下操作可确保tag标识的数据唯一性，即一个tag对应一条数据。

- 存储数据时，先在NVDS中查找指定tag。
  - 如果找到tag，则对比数据内容：若内容完全一致，则直接返回操作成功；若内容不一致，则先将新数据添加到NVDS末尾，再将旧数据置为无效，初始化时会检查是否存在相同tag，若存在则删除无效的新数据或者旧数据。
  - 如果未找到tag，则直接将新数据添加到NVDS末尾。
- 删除tag后，该tag被标记为无效，无法再次读取tag内容。
- 垃圾回收时，仅在GC区域有效的情况下，才会先删除NVDS中已拷贝的tag，再回写tag。

2.2.2 数据校验

- 当NVDS 写入或读取数据时，会对数据进行校验。
- 写入数据时，分别按字节累加计算item header和item data的8位校验和。
- 读取数据时，对数据的item header和item data进行校验，检查数据是否损坏。

2.2.3 数据安全

当Flash意外断电或复位时，仅当前操作page中的数据可能会被损坏，NVDS的存储结构根据此特性设计。NVDS数据按item存储，每个item占用一个Flash page。因此，在NVDS写入过程中发生芯片复位，只会影响当前写入的tag，而不会影响其它tag。

在进行GC处理时，会先将tag数据备份至GC区域，然后再擦除NVDS扇区。如果在GC过程中断电，重新上电后可将GC区域中的备份内容恢复至NVDS。因此，不会因为意外复位导致NVDS中已写入的数据丢失，可保证NVDS中数据的安全性。

2.2.4 擦写平衡

在NVDS数据的存储、删除或修改过程中，仅当空闲空间不足时才进行Flash擦除操作，从而最大限度地减少擦除次数，以实现擦写平衡。

- 存储数据：从NVDS起始地址开始，按page顺序依次将数据存储至NVDS末尾位置。
- 删除数据：将NVDS中数据的Tag ID置为无效，并将数据长度加到可用空间长度。
- 修改数据：先将原数据标记为无效，然后将新数据添加至空闲空间。
- GC处理：从第一个无效Item所在Sector开始，先将有效Item备份至GC区域，然后擦除NVDS Sector，最后再将GC区域中的item回写至NVDS Sector区域。

2.2.5 Cache机制

在NVDS中查找数据时，需要从前往后遍历item，直到item header中的tag和所查找的tag相同。并且，Flash读取速度相对于RAM较慢。为了优化数据检索速度，NVDS设计了tag cache机制，缓存item的tag ID、index和地址。

Tag Cache机制包括：

- 储存数据：如果Cache中有空位，则将数据信息添加到Cache中。
- 查找数据：先在Cache中进行查找，若在Cache中未搜索到数据时，才在NVDS中进行查找。
- 删除数据：同步删除Cache中的数据信息（如果Cache中存在该数据信息）。执行删除操作时，将待删除数据后面的数据信息往前移动一个位置进行覆盖。

用户可对Cache功能进行配置，配置文件路径为：`SDK Folder\components\libraries\nvds\gr_nvds_config.h`，其中SDK Folder为SDK包根目录。

- 使能Cache功能：

```
#define ENABLE_TAGS_CACHE          1  //1: enable, 0: disable
```

- 更改Cache数量：

```
#define TAGS_CACHE_SIZE 30U
```

3 NVDS接口说明

GR54xx SDK提供了NVDS初始化、反初始化、数据读取、数据存储以及数据删除等接口，可方便开发者操作Flash中的非易失性数据。

提示:

关于NVDS接口的详细定义，可参考NVDS头文件`SDK_Folder\components\libraries\nvds\gr_nvds.h`。

3.1 初始化

表 3-1 初始化接口函数

函数原型	nvds_err_t nvds_init(uint32_t start_addr, uint8_t sectors)
功能说明	根据指定的地址和Sector数目，分配合适的Flash空间，或者将已初始化的空间进行重新组织。
输入参数	• start_addr：NVDS区域的起始地址，需按Sector（4 KB）对齐。可在custom_config.h中配置宏NVDS_START_ADDR来设置参数值。如果传入0，则NVDS区域默认占用Flash的最后两个或者多个Sector。 • sectors：NVDS占用的Flash Sector数。可在custom_config.h中配置宏NVDS_NUM_SECTOR来设置参数值。
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。
备注	Sector数取决于具体应用需求和蓝牙协议栈NVDS存储情况，实际占用的Sector数为“sectors + 1”。

NVDS初始化流程如下：

1. 检查参数的合法性。
2. 读取GC info并检查是否有效。若GC info有效，则表示在GC过程中发生芯片复位，可根据GC info继续GC处理。
3. 读取所有Sector的NVDS header。
   - 如果NVDS header存在，则表示NVDS已被初始化。此时，需要遍历所有item来初始化Cache，并计算可用空间大小和空闲空间的起始地址。然后，检查是否存在重复tag，若存在则删除无效或者位置在前的tag。
   - 如果NVDS header不存在，则表示NVDS是首次初始化。此时，需要擦除所有NVDS数据，并写入NVDS header。

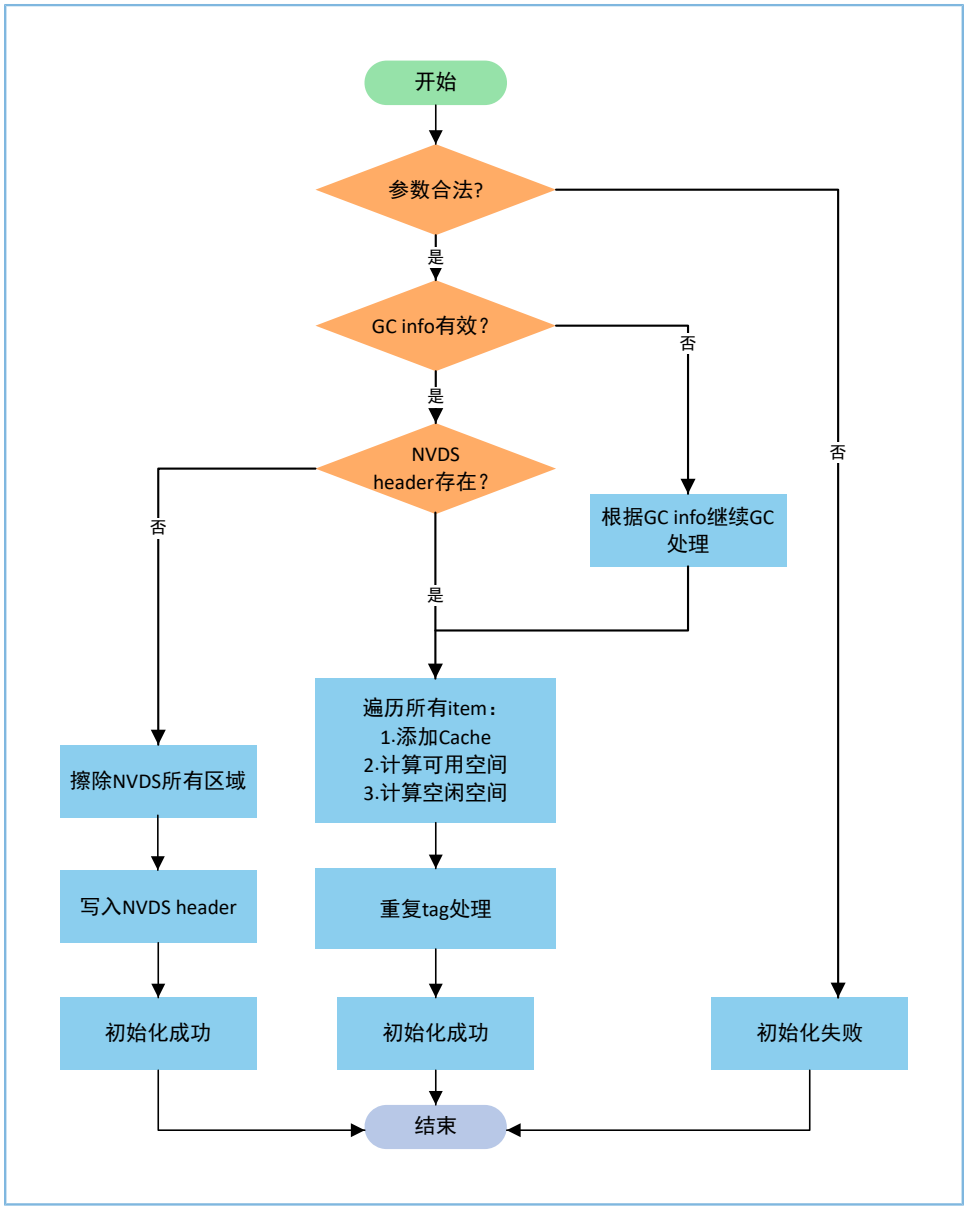

图 3-1 NVDS初始化流程图

3.2 反初始化

表 3-2 反初始化接口函数

函数原型	nvds_err_t nvds_deinit(uint32_t start_addr, uint8_t sectors)
功能说明	NVDS反初始化，擦除NVDS所占用的Flash区域。
输入参数	• start_addr：NVDS的起始地址，需按Sector（4 KB）对齐。可在custom_config.h中配置宏NVDS_START_ADDR来设置参数值。如果传入0，则NVDS区域默认占用Flash的最后两个或者多个Sector。 • sectors：NVDS占用的Flash Sector数。可在custom_config.h中配置宏NVDS_NUM_SECTOR来设置参数值。
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。

备注	NVDS反初始化时会执行Flash擦除操作，将NVDS和GC区域全部擦除，且实际擦除的扇区数为“sectors + 1”。

3.3 读取tag

表 3-3 读取tag接口函数

函数原型	nvds_err_t nvds_get(NvdsTag_t tag, uint16_t *p_len, uint8_t *p_buf)
功能说明	从NVDS中读取tag ID对应的数据。
输入参数	• tag：待查询的tag ID • p_len：p_buf的长度
输出参数	• p_len：实际读取的数据长度 • p_buf：读取数据的Buffer
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。
备注	仅当该函数返回NVDS_SUCCESS时，p_buf中数据才有效。否则，在读取过程中，p_buf可能会存储脏数据。

NVDS tag读取流程为：

1. 检查参数的合法性。
2. 在Cache中查找tag。若Cache中未找到则在NVDS中查找。
3. 在NVDS中查找tag。依次读取item header，并对比header中的tag和所查询的tag，如果一致且header校验通过则返回item data地址。如果遍历完整个NVDS区域都未找到，则说明tag不存在。
4. 读取item data并进行校验。若校验通过，则返回数据。

说明:

如果item data校验不通过，那么p_buf中会存储脏数据。仅当nvds_get返回NVDS_SUCCESS时，p_buf中的数据才有效。

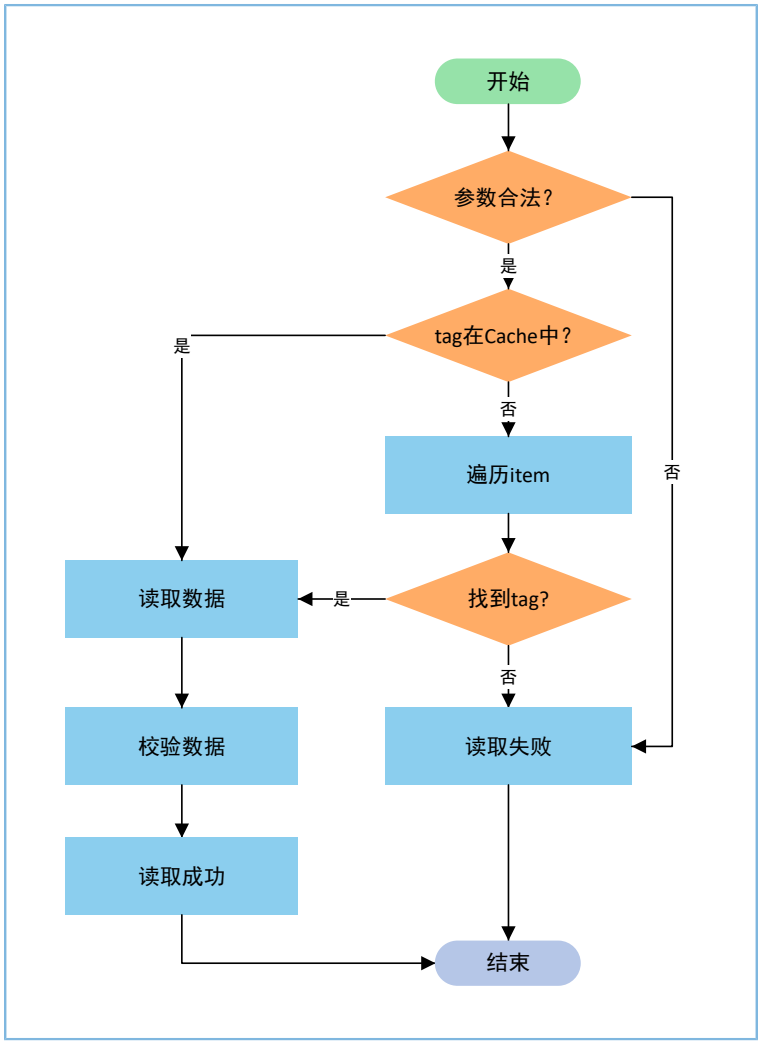

图 3-2 读取tag流程图

3.4 写入tag

表 3-4 写入tag接口函数

函数原型	nvds_err_t nvds_put(NvdsTag_t tag, uint16_t len, const uint8_t *p_buf)
功能说明	将数据写入NVDS，并使用tag ID标识。
输入参数	• tag：待写入数据的tag ID • len：待写入数据长度 • p_buf：待写入数据的Buffer
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。
备注	

NVDS tag写入流程为：

1. 检查参数的合法性。

2. 在NVDS中查询tag。
   - 如果未找到tag，直接执行步骤4。
   - 如果找到相同tag，且数据内容完全一致，则直接返回成功。若数据内容和传入的数据不同（长度或内容不同），跳转至步骤3。

3. 删除Cache中指定tag的数据信息。查找tag在Cache中的位置，若不在Cache的末尾，则将其后面的数据信息项向前移动一个位置，以覆盖删除的数据信息项。

4. 检查NVDS连续空闲空间。
   - 若NVDS连续空闲空间足够容纳新数据，则执行步骤5。
   - 若可用空间充足但连续空闲空间不足，则需进行GC处理。GC处理完成后，再执行步骤5。

5. 将数据写入NVDS。
   (1) 生成item并将item写入Flash。
   (2) 若Cache中还有空位，则将item信息添加到Cache中。
   (3) 更新可用空间（减去item长度），将NVDS空闲空间位置指针指向下一个空闲的Flash page。

6. 删除旧数据。更新数据时，新数据写入成功后，将旧数据标记为“TAG_DEL”，并将旧数据长度添加到可用空间大小。

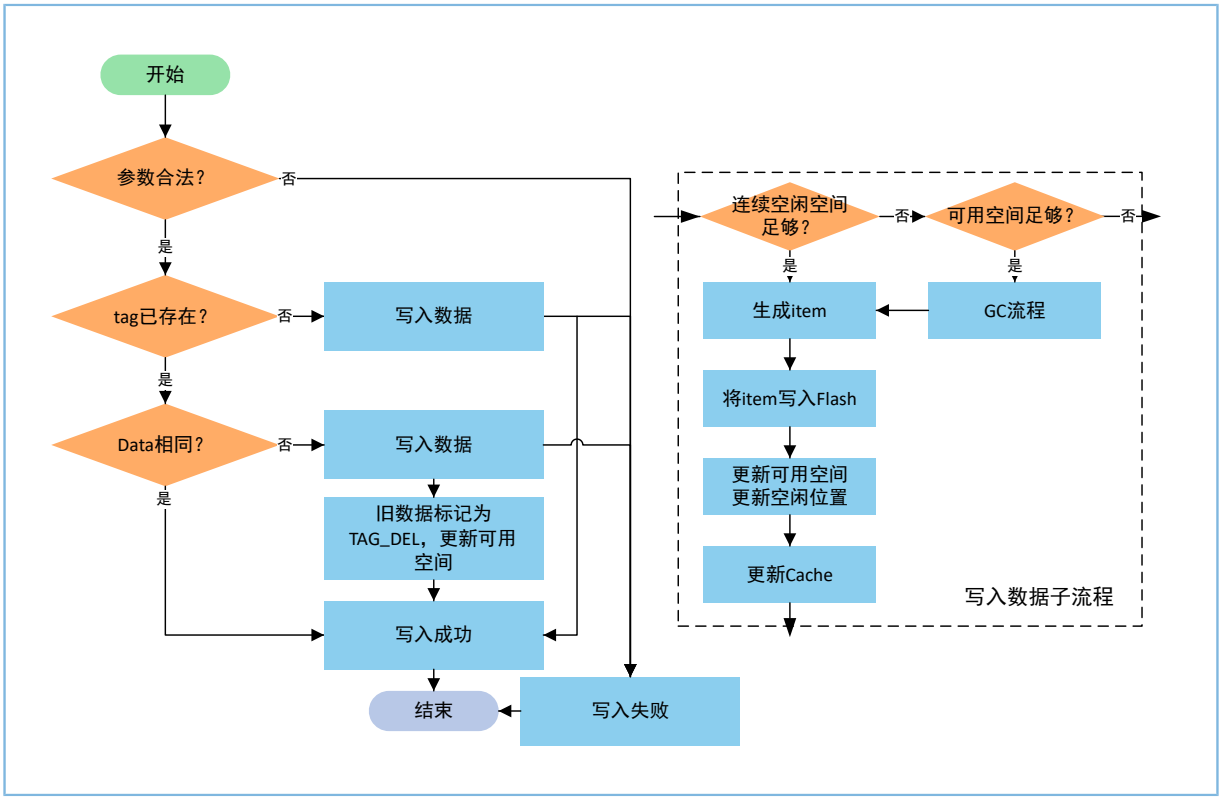

图 3-3 写入tag流程图

3.5 删除tag

表 3-5 删除tag接口函数

函数原型	nvds_err_t nvds_del(NvdsTag_t tag)
功能说明	删除NVDS中tag ID对应的数据。
输入参数	tag：存储数据的tag ID
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。
备注	

NVDS tag删除流程如下：

1. 检查参数的合法性。
2. 查找tag。
   - 如果找到tag，将NVDS中该tag的所有item标记为“TAG_DEL”，并将tag长度添加到可用空间大小，删除Cache中的数据。
   - 如果未找到tag，则返回数据不存在。

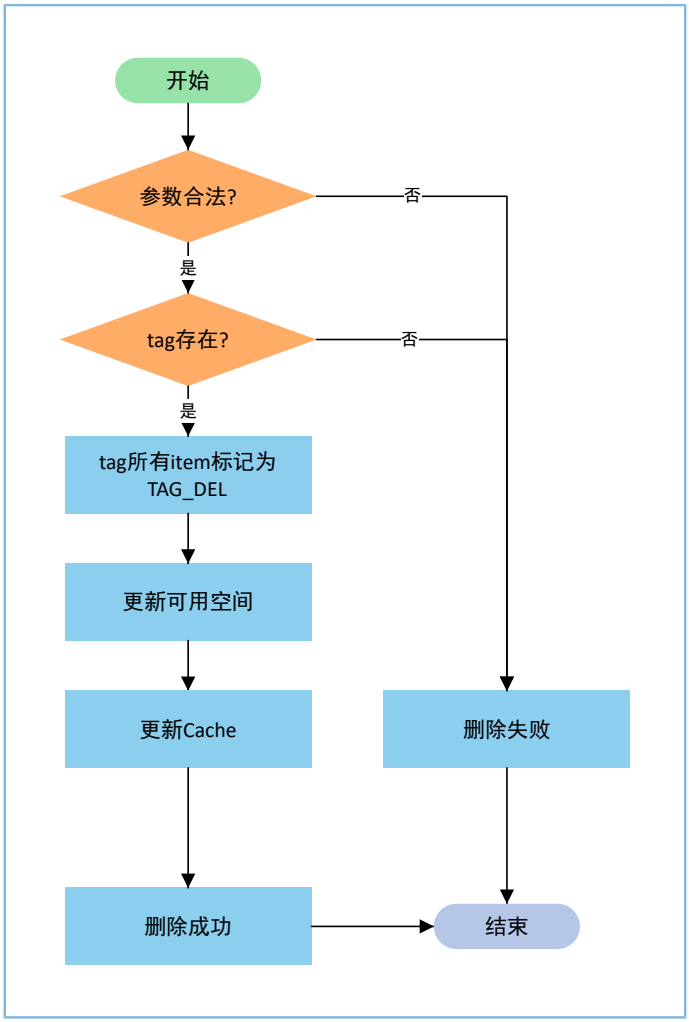

图 3-4 删除tag流程图

3.6 获取tag长度

表 3-6 获取tag长度接口函数

函数原型	uint16_t nvds_tag_length(NvdsTag_t tag)
功能说明	获取指定tag ID的数据长度。
输入参数	tag：存储数据的tag ID
返回值	• tag长度 • 0：tag不存在
备注	

获取tag长度流程如下：

1. 查找tag ID，通过在Cache和NVDS区域查找item header方式查找tag ID。

   若找到具有相同tag ID的有效item header，则返回该item header中的tag长度。在查找过程中，不会进行item内容校验。当tag有多个item时，由于item header中的tag长度都相同，因此从任意一个item header均可得知tag长度。

2. 若未找到tag ID，则返回0。

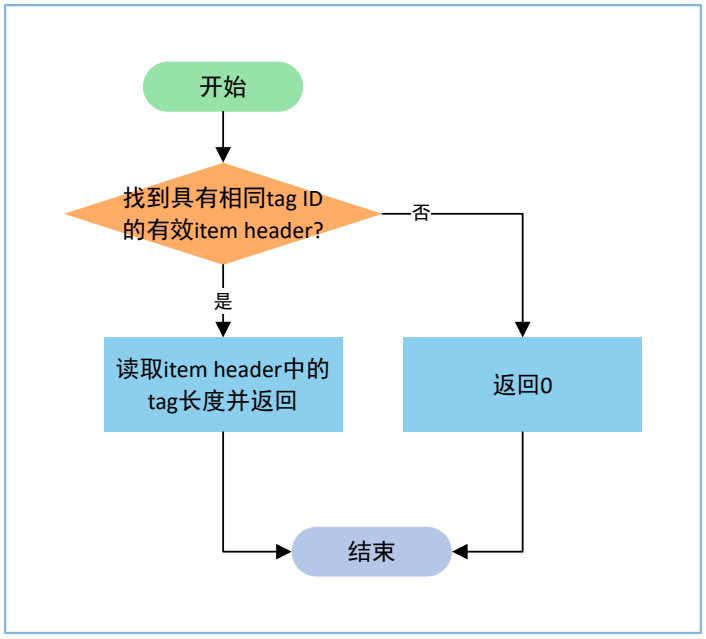

图 3-5 获取tag长度流程图

3.7 读取sub tag

表 3-7 读取sub tag接口函数

函数原型	nvds_err_t nvds_get_sub_tag(NvdsTag_t main_tag, NvdsTag_t sub_tag, uint16_t *p_len, uint8_t *p_buf)
功能说明	读取指定的sub tag。
输入参数	• main_tag：主tag ID • sub_tag：副tag ID • p_len：p_buf长度
输出参数	• p_len：实际读取的数据长度 • p_buf：读取数据的Buffer
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。
备注	

NVDS sub tag读取流程如下：

1. 检查参数的合法性。
2. 查找main tag，以确定哪些sub tag合并在一个item中。首先在Cache中查找main tag，如果未找到再在NVDS区域查找。
   - 若未找到main tag，则表示sub tag不存在，直接返回读取失败。
   - 若找到main tag，则读取main tag所在item，并在item中从头逐个查找sub tag。
3. 找到sub tag后，对sub tag内容进行校验。如果校验通过，将sub tag内容拷贝至p_buf，并返回成功。

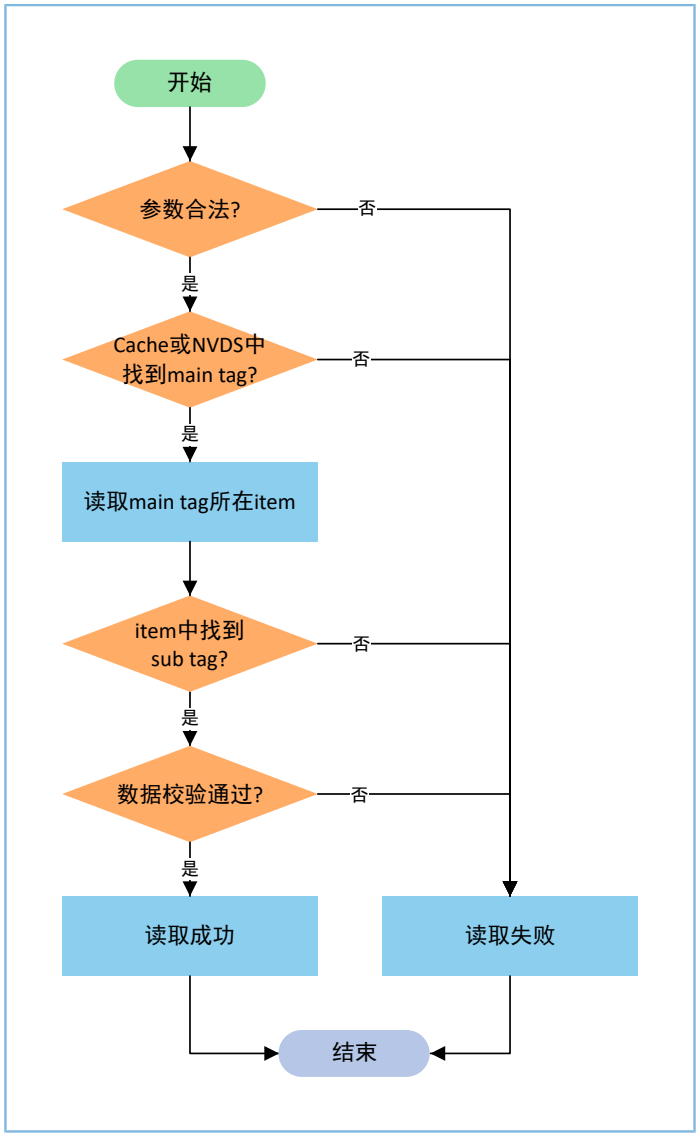

图 3-6 读取sub tag流程

3.8 写入sub tag

表 3-8 写入sub tag接口函数

函数原型	nvds_err_t nvds_put_sub_tag(NvdsTag_t main_tag, NvdsTag_t sub_tag, uint16_t len, const uint8_t *p_buf)
功能说明	写入指定的sub tag。
输入参数	• main_tag：主tag ID • sub_tag：副tag ID • len：待写入sub tag的长度 • p_buf：待写入sub tag的内容
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。
备注	

NVDS sub tag写入流程如下：

1. 检查参数合法性。
2. 查找main tag。
   - 若Cache和NVDS中均未找到main tag，则根据main tag和sub tag生成一个item，并将其写入Flash。
   - 若找到main tag存在，则读取main tag的item。
3. 查找sub tag。
   - 若找到sub tag且数据相同，则返回成功。
   - 若找到sub tag但数据不同，则删除sub tag后跳转至步骤4。
   - 若未找到sub tag，跳转至步骤4。
4. 计算item可用空间大小。如果可用空间足够，则生成新item，并将其写入Flash。否则，写入失败并返回错误信息。
5. 删除旧item。写入新item后，再删除旧item。

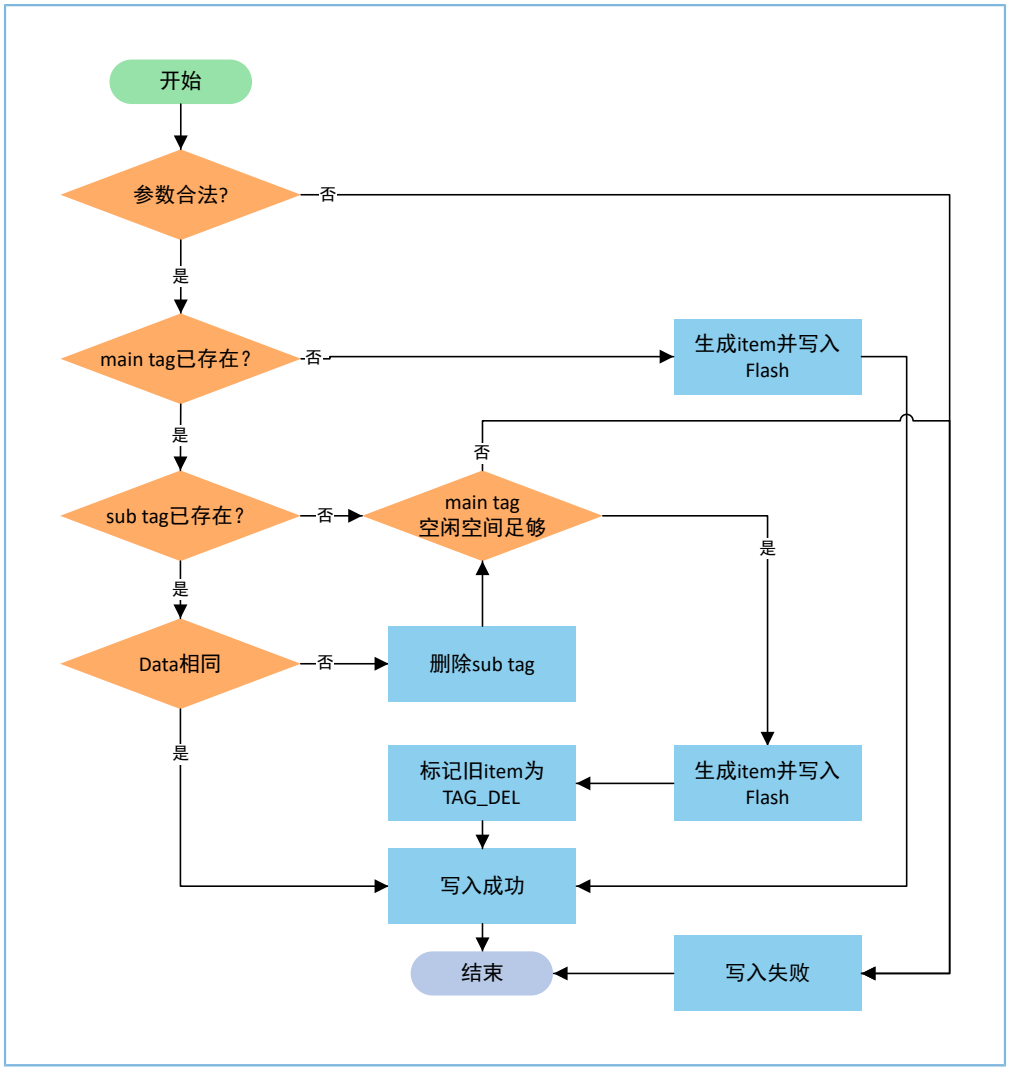

图 3-7 写入sub tag流程图

3.9 删除sub tag

表 3-9 删除sub tag接口函数

函数原型	nvds_err_t nvds_del_sub_tag(NvdsTag_t main_tag, NvdsTag_t sub_tag)
功能说明	删除指定的sub tag。
输入参数	• main_tag：主tag ID • sub_tag：副tag ID
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。
备注	

NVDS sub tag删除流程如下：

1. 检查参数的合法性。
2. 查找main tag。如果Cache和NVDS中均未找到main tag，则直接返回失败。

3. 读取main tag的item后，从item中删除sub tag。如果sub tag不存在，则返回失败。
4. main tag item处理。
   - 若还存在其它sub tag，则按之前的顺序排列这些sub tag，重新生成item并写入Flash，然后删除旧item。
   - 若不存在其它sub tag，则删除main tag。

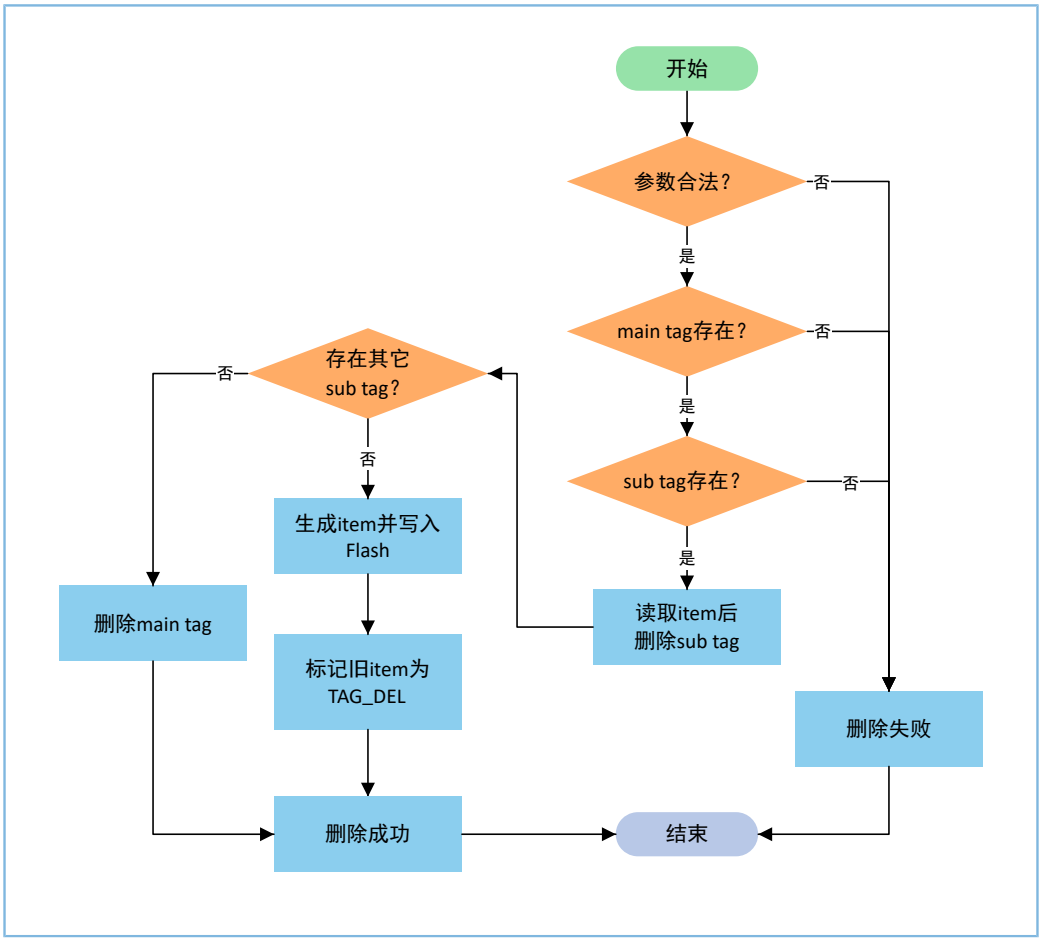

图 3-8 删除sub tag流程图

3.10 获取sub tag长度

表 3-10 获取sub tag长度接口函数

函数原型	uint16_t nvds_sub_tag_length(NvdsTag_t main_tag, NvdsTag_t sub_tag)
功能说明	获取指定的sub tag长度。
输入参数	• main_tag：主tag ID • sub_tag：副tag ID
返回值	• sub tag长度 • 0：不存在sub tag
备注	

获取sub tag长度流程如下：

1. 查找main tag ID，在Cache和NVDS中通过搜索item header方式进行查找。如果未找到，则返回0。
2. 如果找到main tag ID，则读取该item，然后在item中遍历查找sub tag。如果找到sub tag，则返回sub tag长度，否则返回0。查找sub tag时只检查header，不检查tag数据。

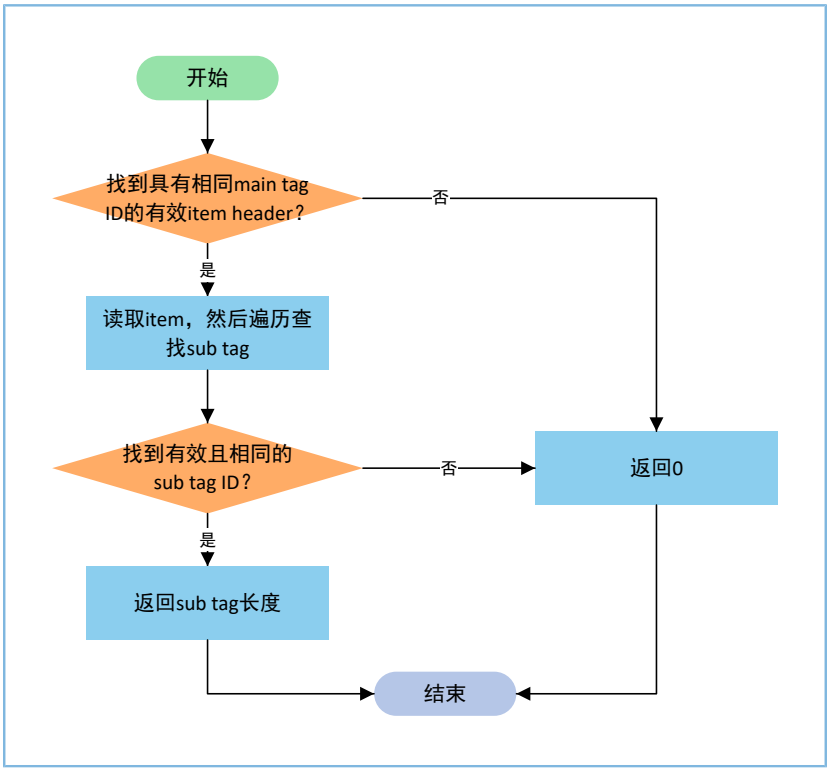

图 3-9 获取sub tag长度流程图

3.11 GC处理

在NVDS写入时，若可用空间足够但空闲空间不足，NVDS将自动进行GC处理。考虑到GC处理时间不确定且较长，为避免在业务繁忙期间插入GC处理，提供如下接口以方便用户在业务空闲期间主动进行GC处理。

表 3-11 GC处理接口函数

函数原型	nvds_err_t nvds_gc(void)
功能说明	执行GC处理。
输入参数	无
返回值	• NVDS_SUCCESS：执行成功 • 其他：执行异常，具体错误参考nvds_err_t定义。
备注	在NVDS自动或用户主动进行GC处理的开始和结束阶段，均会调用void nvds_gc_start_callback(void)和void nvds_gc_end_callback(void)函数。用户可重新实现这两个回调函数，以便在GC开始和结束时执行必要的操作（如看门狗喂狗）。

GC处理流程如下：

1. 擦除GC区域。
2. 从NVDS起始地址查找第一个无效item地址。该item之前的Sector无需处理。
3. 在Sector中查找有效item，并将其拷贝至GC区域。
4. 生成GC info。当GC区域已满或所有Sector均完成处理时，生成GC info。在GC info中，可根据r_sector和r_page计算出GC备份的item地址，还可根据w_sector计算出备份数据待写入的Sector。如果在GC处理过程中发生芯片复位，在芯片重新运行后根据GC info可在NVDS初始化时恢复GC处理。
5. 将已拷贝至GC的item标记为“TAG_DEL”，并擦除无效item所在Sector。
6. 将GC区域备份的item拷贝至被擦除的Sector。
7. 擦除GC区域。
8. 判断是否已经处理完所有Sector。若否，则返回步骤3，直至所有Sector处理完成。

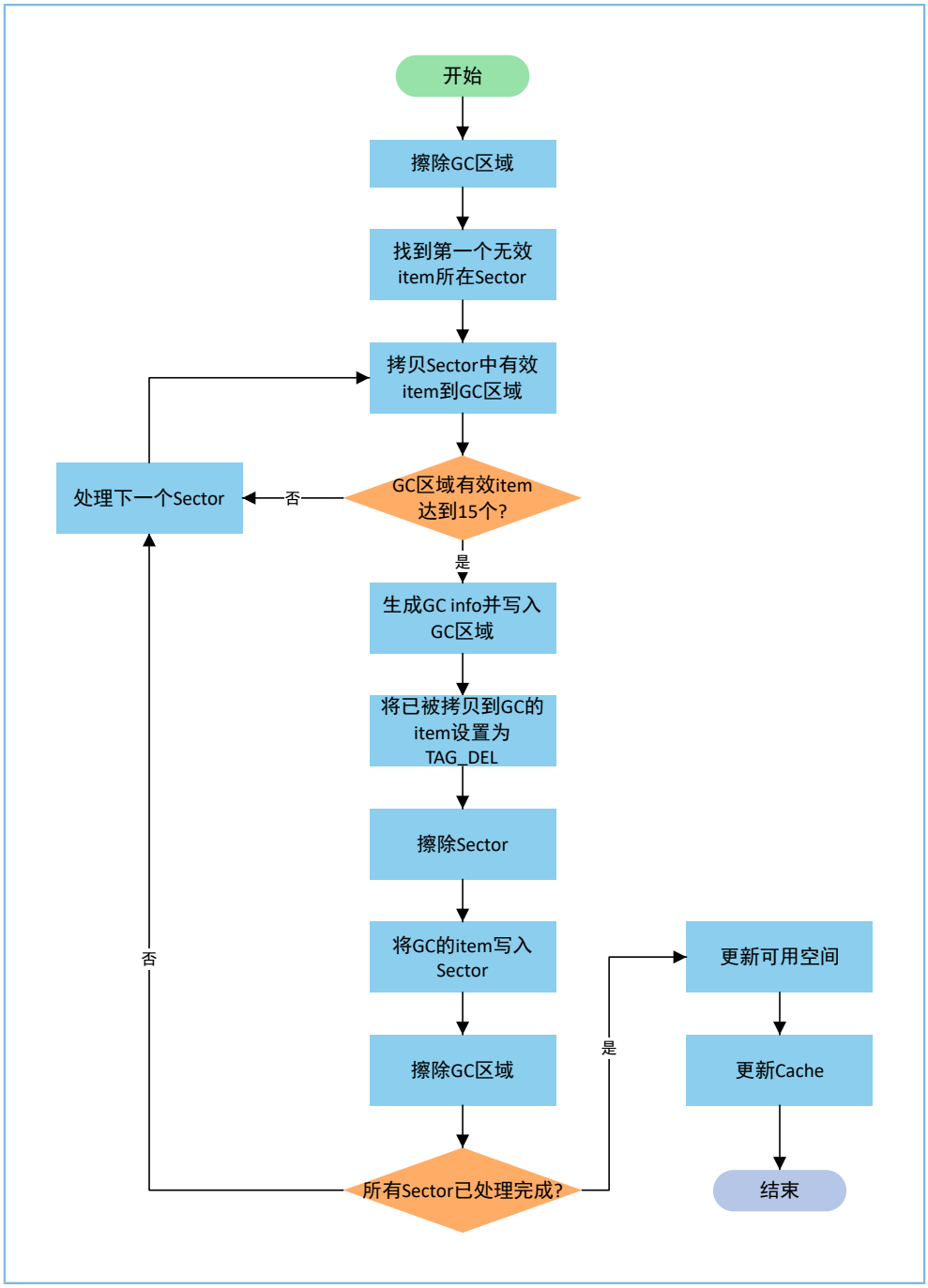

图 3-10 GC处理流程图

3.12 获取可用空间

表 3-12 获取可用空间接口函数

函数原型	uint32_t nvds_get_avail_size(void)
功能说明	获取NVDS当前可使用的最大空间大小，即可用空间大小。
输入参数	无
返回值	可用空间大小
备注	

3.13 获取空闲空间

表 3-13 获取空闲空间接口函数

函数原型	uint32_t nvds_get_empty_size(void)
功能说明	获取NVDS无需GC处理时可使用的空间大小，即空闲空间大小。
输入参数	无
返回值	空闲空间大小
备注	

4 蓝牙协议栈使用NVDS情况

Bluetooth LE协议栈会在NVDS中存储必要信息，为了节省Flash空间，对于在实际应用中只读取的数据或者同时写入的数据会存储在一个NVDS item中，具体使用情况参见下表：

表 4-1 蓝牙协议栈使用NVDS情况

类型	存储内容	长度（bytes）	Tag ID	实际存储方式	备注
Bond Manager	LRU list	10	0x8001	单独一个item存储	绑定设备最近使用的更新列表
	White list（10个）	7	0x8010 ～ 0x801F	每个white list单独一个item存储	设备白名单
	Bond addr info	7	0x8040 ～ 0x8300	每对addr info和pair info合并为一个item存储	每个绑定设备对应一组addr info、pari info和gatt info。最大绑定数由custom_config.h中CFG_MAX_BOND_DEVS配置。
	Bond pair info	140	0x8040 ～ 0x8300		
	Bond gatt info	190	0x8040 ～ 0x8300	每个gatt info单独一个item存储	
	Per adv list (4个)	8	0x8020 ～ 0x802F	adv list合并为一个item存储	周期性广播下使用
Stack	BD_ADDRESS	6	0xC001	合并为一个item存储，main tag为0xC001	蓝牙设备地址
	DEVICE_NAME	197（最大值）	0xC002		蓝牙设备名称 默认情况下，NVDS中可存储的设备名称长度最大为197字节。
	LPCLK_DRIFT	2	0xC003		LPCLK时钟精度
	ACTIVITY_MOVE_CONFIG	1	0xC005		是否支持锚点移动标识。
	RF_XO_OFFSET	2	0xC016		XO时钟校准值
	LE_PRIVATE_KEY_P256	32	0xC018	单独一个item存储	配对中需要强制使用固定的私有key时，所存储的私有key

5 NVDS使用注意事项

- NVDS适用于存储小块数据，不能存储长度大于3720 Bytes的数据。
- 更新NVDS tag数据时，先写入新数据，再删除旧数据。因此，在分配NVDS空间时，需预留足够的可用空间来存储新数据，可用空间大小至少为待更新tag数据长度。例如，当更新tag A数据时，若待更新数据长度为256字节，则NVDS空间应至少预留2个Page的可用空间。
- 在NVDS写入或删除过程中，如果发生断电或芯片复位，当前操作可能失败，导致数据未被写入或删除。未能成功写入的数据将无法读取，即使部分数据已存入NVDS。
- 一个sub tag的最大长度为240字节。当一个main tag存入多个sub tag时，每个sub tag的可用长度将相应减小（每个sub tag都包含8字节的头信息）。
- NVDS sub tag操作需使用对应的nvds_xxx_sub_tag接口。
- NVDS接口不支持重入，请勿在中断或多任务中调用NVDS接口。
- NVDS接口的最大压栈深度通常为896字节。不同编译器和编译选项下可能略有不同。
- NVDS不支持Flash区域重叠。在NVDS已初始化的情况下，对应Flash区域未擦除前不能修改NVDS起始地址和扇区数目。当新的NVDS区域与旧的NVDS区域存在交叉时，由于NVDS无法识别，会导致数据存储错误。因此，在实际应用中，应为NVDS预留足够空间，并避免在固件升级时修改NVDS区域地址。
- 如需实现更完善的数据存储，可自行开发或引入开源文件系统（如littleFS）。