通过 CAN 总线 UDS 协议升级 KF32A156 程序

前言

UDS(Unified Diagnostic Services,统一诊断服务)诊断协议是在汽车电子 ECU 环境下的一种诊断通信协议，在 ISO14229 中规定，由 ISO14230-3 和 ISO15765-3 协议衍生出来。

CAN 是控制器局域网络(Controller Area Network, CAN)的简称，是由以研发和生产汽车电子产品著称的德国 BOSCH 公司开发的，并最终成为国际标准（ISO 11898），是国际上应用最广泛的现场总线之一。

本应用笔记介绍了一种通过 CAN 总线 UDS 协议升级 KF32A156 芯片的方法。以 KF32A156MQV 为例，通过 IDE 烧录 boot 程序，再通过 TCANLINPro 软件与芯片通信，传输 App 数据，最后校验完成，跳转 App 程序。

本应用笔记使用的 ChipON IDE 软件可以从 ChipON 官方网站 www.chipon-ic.com 下载，KF32A156 芯片外设固件库及代码例程可以从 ChipON 官方 Gitee 仓库 ChipON (gitee.com) 下载。

本应用笔记使用的 TCANLINPro 软件，相关 CAN 工具可以从 www.toomoss.com 下载和购买。

本应用笔记关联的代码以及相关文档仅做开发参考，不用作商业用途。

目录

1. 流程介绍

1.1boot loader 流程

KF32 的存储空间按照线性地址排列，当芯片下入 boot 程序后，会首先检查是否有 APP 程序，如果已经存在 APP 程序则直接执行跳转，如果没有 APP 程序则会等待通信。当外部通信依照既定协议建立后，FLASH 的数据开始传输，芯片将数据写入指定的地址，同时读出进行校验。当所有数据传输完成后，boot 程序将设置中断向量表偏移，然后程序指针跳转

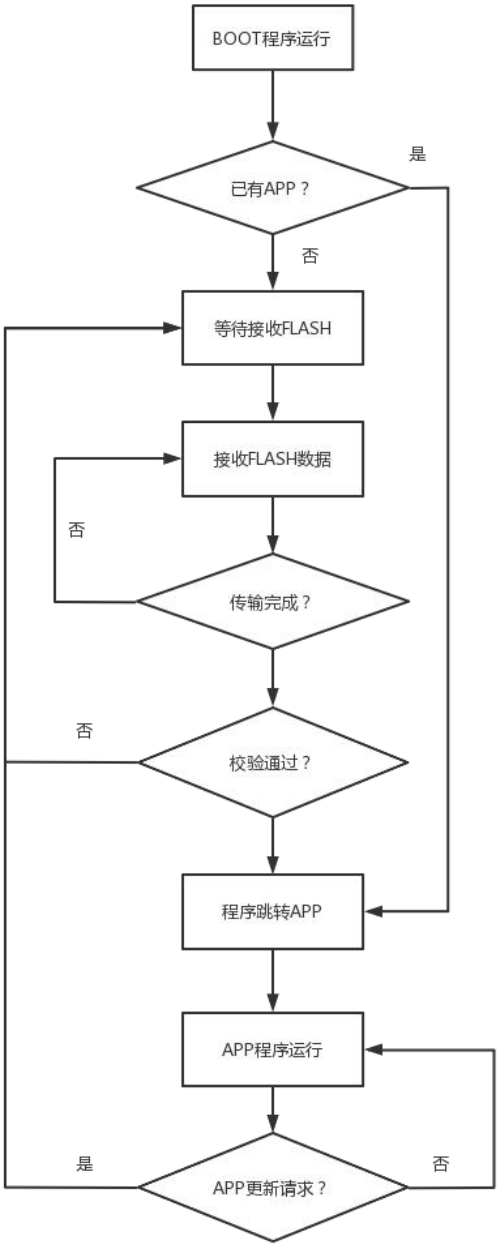

图 1-1bootloader 实现流程

至 APP 程序运行地址。

通过以上这种方式可以实现程序的升级功能。由于通信的方式是由开发者提前约定和固化的，后续软件都可以通过该流程更新，解决了芯片烧写程序需要专有设备，需要现场操作的困扰。

1.2UDS 流程

1.2.1 CAN 总线和 UDS 协议

在介绍 UDS 传输流程前，需要简单介绍一下 CAN 总线。

CAN 是 Controller Area Network 的缩写，是 ISO 国标标准化的串行通信协议，被大量运用在汽车产业中，以满足车身电子间对于安全，舒适，便捷，低功耗，低成本通信的需求。

由于篇幅限制，关于 CAN 总线物理层的介绍请读者参考其他资料，这里只介绍 CAN2.0B 标准帧的数据帧。在 CAN 总线中，可以连接数个节点，而节点与节点通信是通过数据帧实现。CAN 总线并没有直接规定节点的优先级，但是通过仲裁段 ID，可以确定数据帧的优先级，同时 CAN2.0 标准中规定，标准格式的数据帧仲裁 ID 为 11 位，ID 从 0x00~0x7FF。在 CAN 总线中，数据位 1 为隐性电平，数据位 0 为显性电平，当显性电平和隐性电平同时在总线上发送时，总线呈现显性电平。当仲裁过程发生时，依次对比仲裁位，显性电平优先独立出现的 ID 会仲裁成功，继续通过总线发送数据，失败的节点退出本次发送。简单来说，ID 序列越低，越容易获得仲裁成功。CAN 总线数据域最大 64 位，也就是 8 个字节。

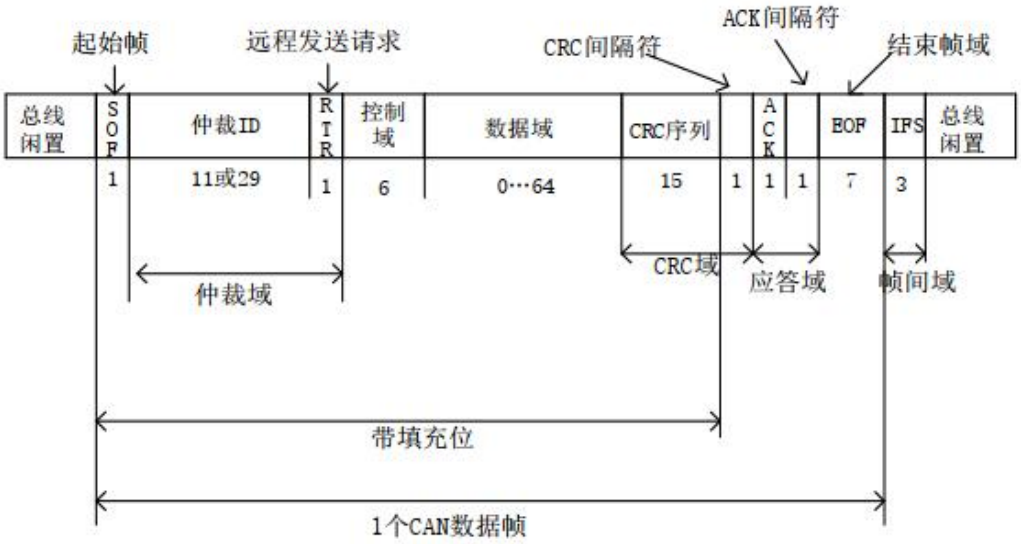

图 1-2 CAN 协议帧格式图

UDS 的本质是一系列服务的集合。UDS 的服务包含 6 个大类，共 26 种，每种服务都有自己独立的 ID，称为 SID。

SID，Service Identifier,诊断服务 ID。UDS 通信本质是一种比较简单的交互协议，一般

流程为诊断方发起，给 ECU 发送请求数据，这个数据中的第一个字节就为 SID。ECU 如果执行肯定的响应，需在首字节回复"SID+0x40"，如果回复否定响应，首字节需为 0x7F，次字节回复对应 SID。部分 SID 包含子功能，则在首字节发送对应服务请求后，在次字节继续传输子功能代码。

例如：　诊断方发送　0x10
ECU 回复　0x7F　0X10
表示诊断方申请 0x10 服务(进入编程模式)，ECU 拒绝响应

诊断方发送 0x10 0x03
ECU 回复　0x50 0x03
表示诊断方申请 0x10 服务，0x03 子功能，ECU 成功响应

UDS 的服务分为 6 大类，本应用笔记会介绍例程代码中运用到的几个服务，其他服务请读者自行查阅其他资料。

大类	SID(hex)	诊断服务名
诊断和通信管理功能单元	10	诊断会话控制
	11	ECU 复位
	27	安全访问
	28	通讯控制
	85	控制 DTC 的设置
例行程序功能单元	31	例行程序控制
上传下载功能单元	34	请求下载
	36	数据传输
	37	请求退出传输

表 1-1 UDS 服务代码

1.2.2 UDS 服务

0x10 诊断会话

该服务包含 3 个子功能，01 Default 默认会话，02 Programming 编程会话，03 Extended 扩展会话，ECU 上电时，进入的是默认会话。一般来说，默认会话权限最小，可操作的服务少；编程模式用于解锁 bootloader 相关操作的诊断服务；扩展模式通常用于解锁高权限诊断服务，例如写入数据、读写诊断码。本例程中代码升级响应的是 0x03 子服务。

0x11 ECU 复位

该服务会控制 ECU 端执行复位，该服务包含 5 个功能。0x01 HardReset 硬重置，模拟服务器重新上电动作，后续所执行的操作视具体情况而定。0x02 KeyOffOnReset 点火钥匙关闭/重置。该值功能类似于驾驶员关闭再重新打开点火钥匙的动作。所执行的操作视具体情况而定。0x03 SoftReset 软重置，可使服务器立即重启应用程序。所执行的操作视具体情

况而定。本例程中代码升级响应的是 0x01 子服务。

0x27 安全访问

此服务提供访问 ECU 内部数据或者出于安全考虑因素被限制的诊断服务的请求权。常见读服务如 0x22 读取非安全信息时能够直接读取，不需要利用此服务。

安全访问序列一共由 4 步组成。

1) 诊断仪发送请求 seed 的报文 0x27 0x01
2) ECU 响应 seed 0x67 0x01 XX XX XX XX
3) 诊断仪根据返回的 seed 按照约定的算法计算 key 值并发送给 ECU 请求验证
   0x27 0x02 XX XX XX XX
4) ECU 收到请求后，按照约定的算法对 key 值校验，并给出相应响应
   肯定响应 0x67 0x02
   否定响应 0x7F 0X27

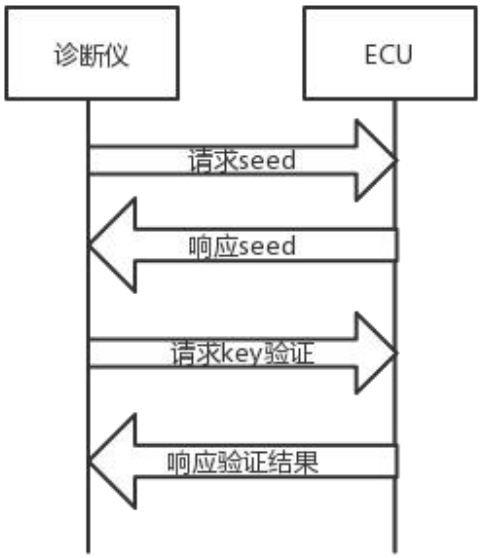

图 1-3 安全访问顺序图

0x28 通信控制

通讯控制服务用于开启或关闭 ECU 对某些报文的发送或接收，如应用报文和网络管理报文等。

0x85 诊断故障码设置控制

诊断故障代码设置控制服务用于停止或重启 ECU 诊断故障代码的记录。

当通过该服务对故障码记录进行抑制操作后，若会话层时序参数超时从而 ECU 进入默认会话，或 ECU 执行复位操作后，诊断故障代码应该恢复记录。

0x31 程序控制

该服务主要用于对特定程序的控制操作，诸如启动程序，停止程序，请求运行结果等。子功能 0x01 starRoutine 启动程序，0x02 stopRoutine，0x03 requestRoutineResults 请求程序运行结果。后面字节的含义一般表示程序 ID 和可选参数，具体协议由用户定义。

0x34 请求下载

Table 393 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	RequestDownload Request SID	M	0x34	RD
#2	dataFormatIdentifier	M	0x00 – 0xFF	DFI_
#3	addressAndLengthFormatIdentifier	M	0x00 – 0xFF	ALFID
#4 : #(m-1)+4	memoryAddress[] = [byte#1 (MSB) : byte#m]	M : C_1	0x00 – 0xFF : 0x00 – 0xFF	MA_ B1 : Bm
#n-(k-1) : #n	memorySize[] = [byte#1 (MSB) : byte#k]	M : C_2	0x00 – 0xFF : 0x00 – 0xFF	MS_ B1 : Bk
C_1: The presence of this parameter depends on address length information parameter of the addressAndLengthFormatIdentifier C_2: The presence of this parameter depends on the memory size length information of the addressAndLengthFormatIdentifier.				

图 1-4 请求下载服务定义

Data1 为 SID，Data2 标识了数据格式的相关信息，例如数据是否有加密，加密信息，加密算法，数据压缩格式等，该信息具体内容由用户约定。Data3 表示后续内存地址和内存大小信息。Data 的低 4 位表示写入地址，高四位表示写入内存大小。后续数据根据 Data3 的内容定义不定长。

0x36 数据传输

当 0x34 服务得到了正确响应，诊断仪会启用此服务启动传输过程。

Table 403 — Request message definition

A_Data byte	Parameter Name	Cvt	Byte Value	Mnemonic
#1	TransferData Request SID	M	0x36	TD
#2	blockSequenceCounter	M	0x00 – 0xFF	BSC
#3 : #n	transferRequestParameterRecord[] = [transferRequestParameter#1 : transferRequestParameter#m]	C : U	0x00 – 0xFF : 0x00 – 0xFF	TRPR_ TRTP_ : TRTP_
C = Conditional: this parameter is mandatory if a download is in progress.				

图 1-5 数据传输服务定义

Data2 blockSequenceCounter 表示传输块计数。当开启一次新的数据传输过程，且 0x34

服务得到正确回应后，Data2 从 0x01 开始计数，后续数据为 UDS 连续帧，具体长度和内容由用户约定。当 Data2 写满为 0xFF 时，下一帧由 0x00 继续。

0x37 请求数据传输退出

该服务用于当数据传输过程完成或者遇到故障，需要退出时，停止与 ECU 之间的数据传输。

1.2.3 UDS 网络层协议

在上文的介绍中不难发现，当数据开始传输时，CAN 的标准帧最大支持 8 字节，而 UDS 协议中最大的设计容量是 4095 字节。当 UDS 应用服务需要发送超过 8 字节的数据信息，在 CAN 总线线上就需要超过 1 帧报文才能接收完毕，这就需要定义网络层协议。

协议数据单元(N_PDU)

名称	缩写	描述
寻址信息	N_AI	源地址，目标地址和寻址方式信息
协议控制信息	N_PCI	单帧、第一帧、连续帧和流控制帧
数据	N_Data	应用层数据

表 1-2

协议控制信息(N_PCI)

N_PCI 类型	字节			
	Byte1		Byte2	Byte3
	Bit7~4	Bit3~0		
单帧 SF	0	SF_DL(0~7)	N_Data	
第一帧 FF	1	FF_DL		N_Data
连续帧 CF	2	SN(0~15 循环)	N_Data	
流控帧 FC	3	FS	BS	STmin

表 1-3

由表 1-2 可得知，同一个 N_PDU 下，只有一个对应的 N_PCI，通过识别 CAN 报文的首字节可以确定其属性。

单帧 SF

由诊断仪到 ECU 发起某个服务或者 ECU 向诊断仪回复某个服务，仅需要一帧即可完成数据的传递。参数 SF_DL 代表后续 N_Data 数据的长度。

第一帧 FF

当发送方发送的数据过长时，需要多帧报文才能发完，通过第一帧传递后续数据长度等信息。通过 Byte1 的低 4 位和 Byte2 组成 12 位的长度信息，FF_DL。

流控帧 FC

流控帧为接收方在收到发送方的传输信息后，根据自身缓存，接收快慢能力等信息回复

给发送方，约定发送过程中的信息，如发现缓存不匹配，申请停止；定义发送连续两帧之间的延时等。

流控帧的首字节低 4 位组成 FS 信息。FS 为 0 表示继续发送，1 表示等待，2 表示溢出。第二个字节 BS 表示定义块大小，即允许连续发送连续帧的次数。第三个字节表示 STmin，约定两个连续帧之间的最小时间间隔。

CF 连续帧

在经过第一帧发送数据信息，经过流控帧确认发送过程后，发送方会连续传输连续帧来发送信息。连续帧首字节的低 4 位表示 SN，数据滚码，从 0x01 开始依次递增，当本次发送的 SN 为 0x0F 时，下一帧从 0x00 继续。

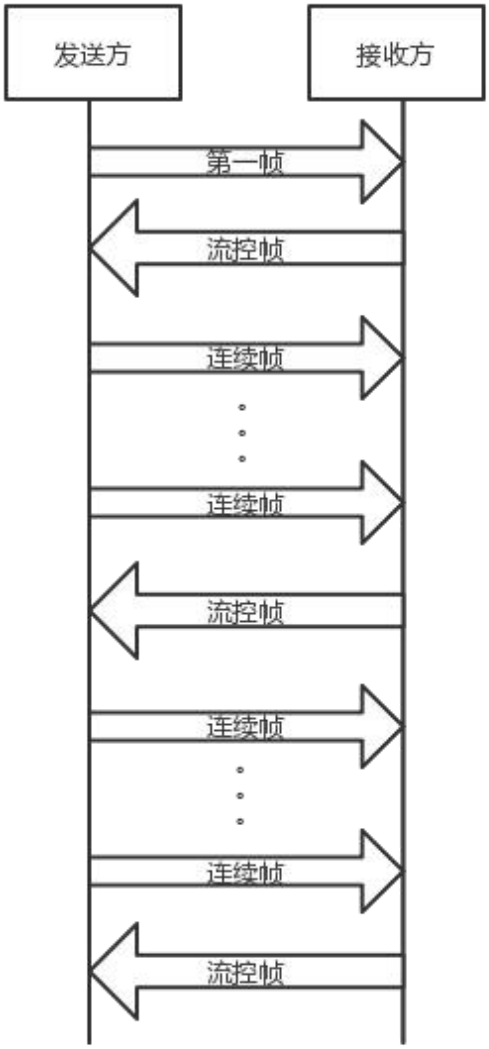

图 1-6 UDS 数据传输时序

根据图 1-6 所示，当 UDS 需要传输长数据的时候：

1. 发送方开启第一帧
2. 接收方确认后发送流控帧
3. 发送方按照约定的块长度，不停发送连续帧
4. 若发送过程未完成，则继续 2.3.两步

2. 升级过程

经过上述板块的介绍，下面结合例程代码实验升级的过程。

2.1 软件下载

打开 ChipON IDE KF32 软件，导入例程目录下的”can_uds_bootloader_v01”和”can_uds_bootloader_app_v01”两个工程，前者为芯片的 boot 程序，在代码中实现 CAN 通信和 UDS 协议解析，提供程序引导下载和跳转，后者为 app 工程，下文简称前者为 boot 程序，后者为 app 程序。

1、打开 boot 程序，点击编译按钮，然后连接硬件电路，正确下载程序。硬件以 KF32A156-MINI-EVB 开发板为例，用 mini-USB 接口的数据线连接开发板，可以在上位机上用串口助手以波特率 115200 观察到数据输出，表示系统运行正常。

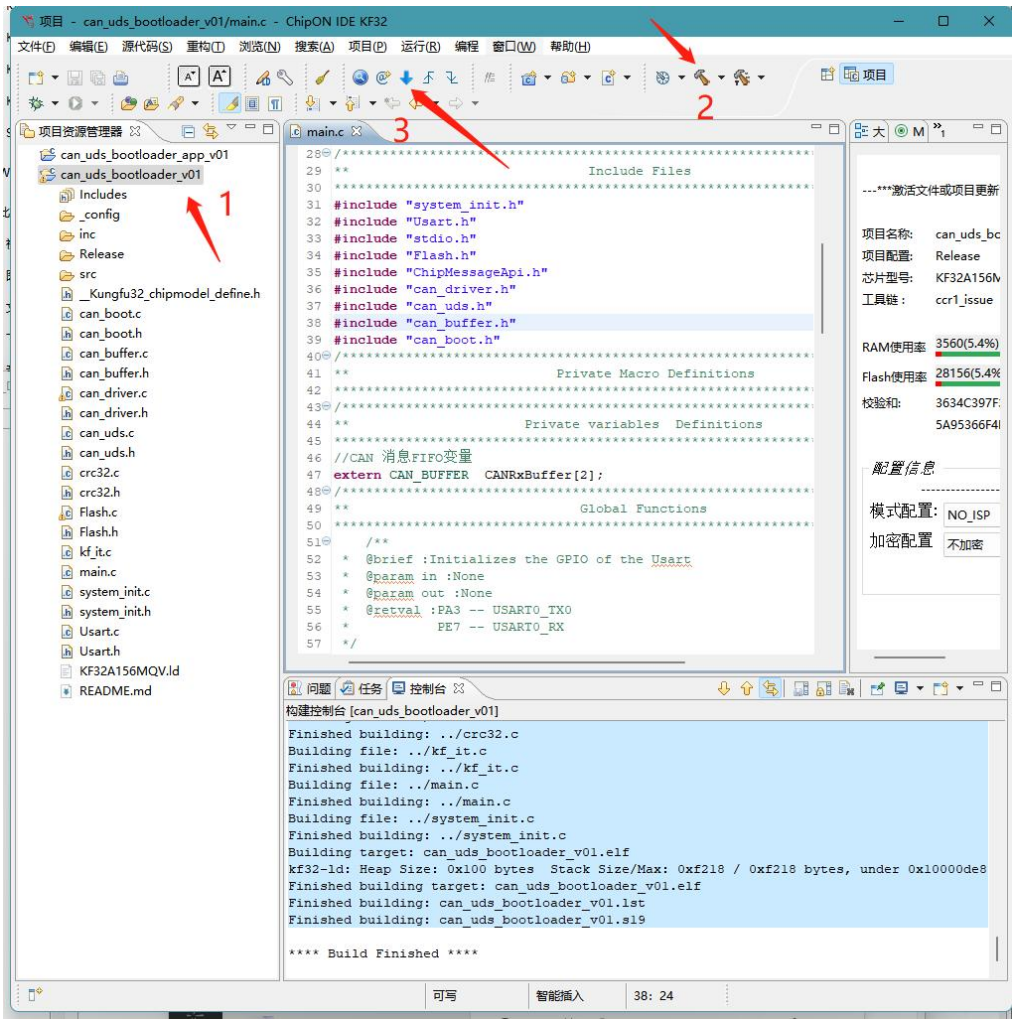

图 2-1

图 2-2

2.2 软件升级

2、打开 TCANLINPro 软件，连接 CAN 调试设备，点击启动按钮并正确配置相应参数。

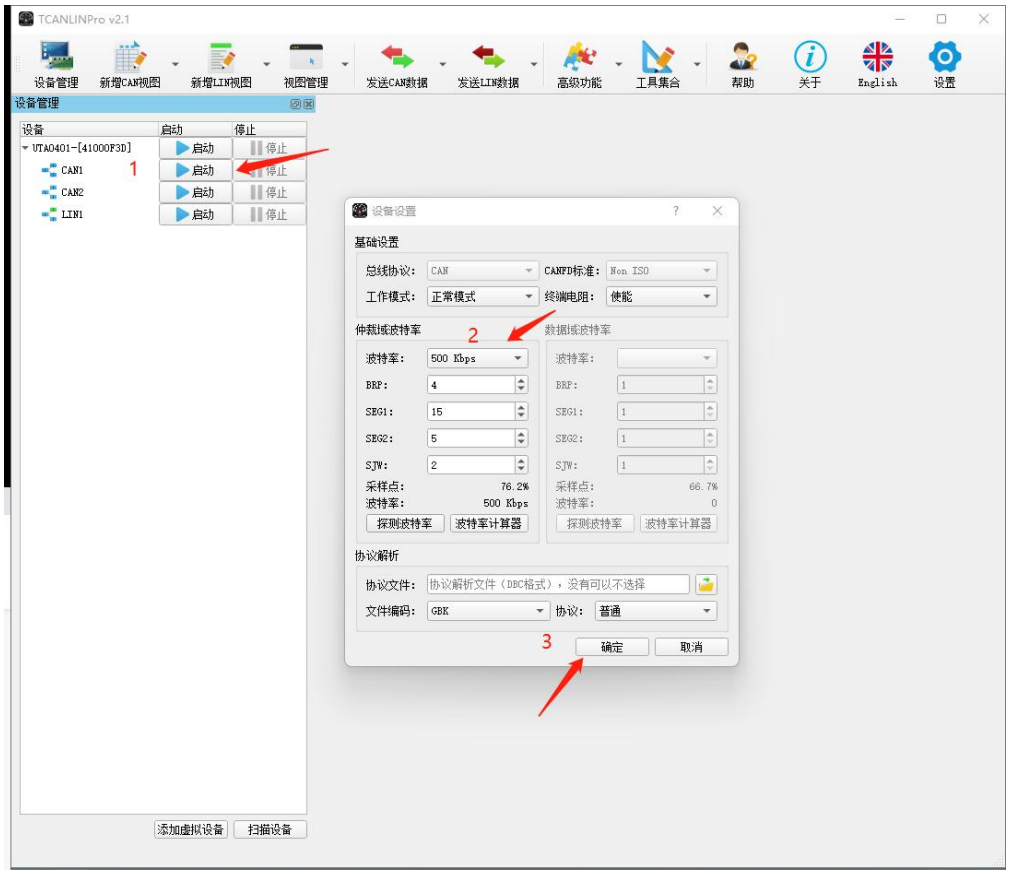

图 2-3

3、点击界面上端的“高级功能”快捷键，在弹出的菜单中选择“CAN UDS 固件升级”按钮。

图 2-4

4、在弹出的对话框中可以选择自定义流程文件，可以选择例程目录下提供的.tflash 文件，然后再点击旁边的“齿轮”按钮，进入升级流程设置界面。如果没有.tflash 文件或者需要自行配置也可以直接点击“齿轮”按钮，自行配置升级流程。

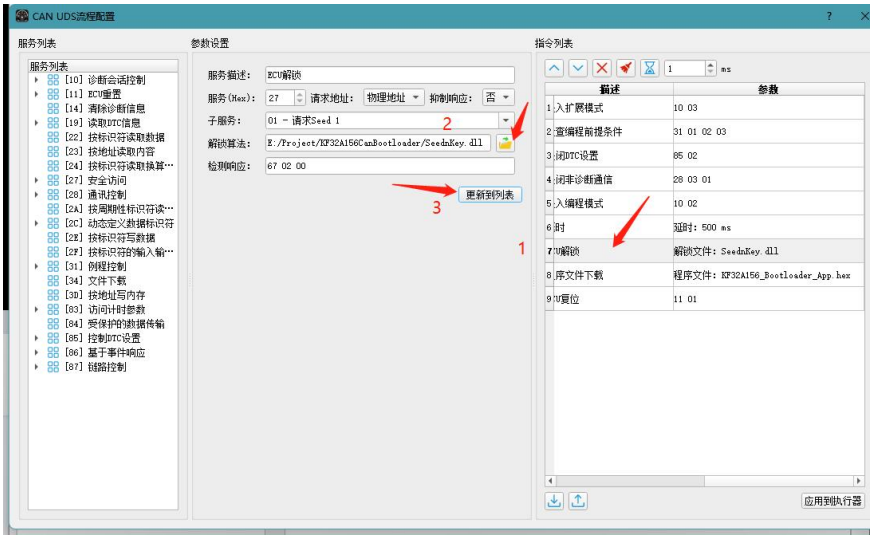

图 2-5

5、点击流程第 7 行，然后再选择解锁算法文件，例程目录下有提供测试用.dll 文件。本例程中为了演示效果，即使解锁失败升级过程也能继续进行，用户可根据开发实际需求自行配置文件。选择结束后点击“更新到列表”，生效本次操作。

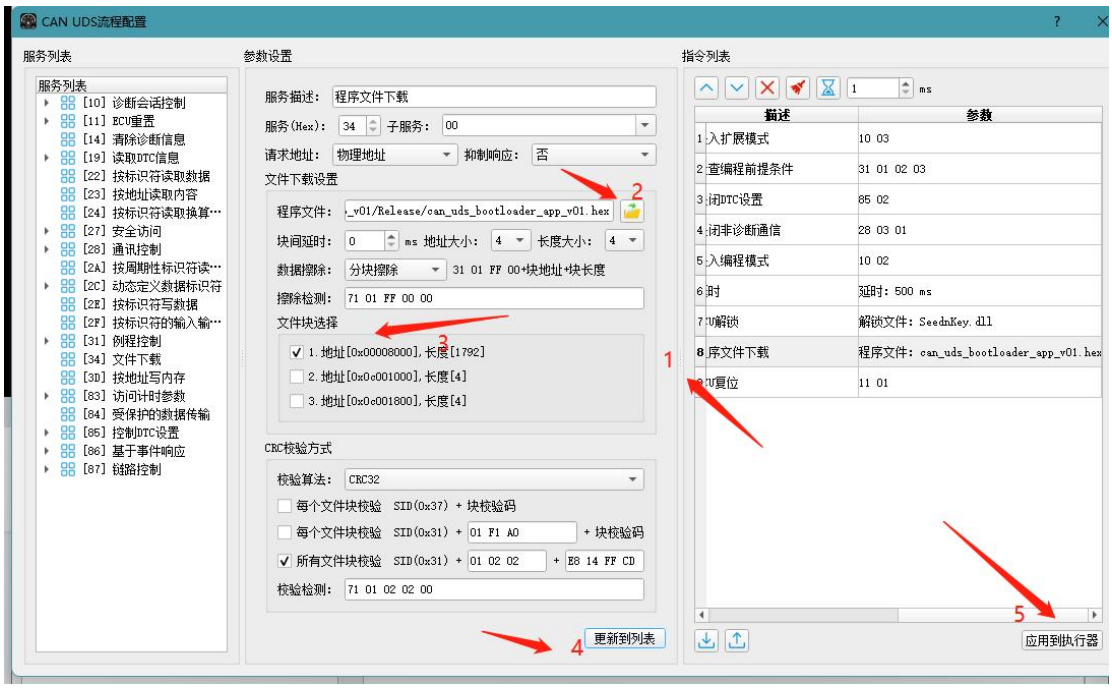

图 2-6

6、再点击流程第 8 行，选择需要下载的 app 程序。注意首次添加时可能会提示解析错误，这是由于之前默认 app 程序地址错误所致，点击确认，然后重新选择 app 程序路径。选择好后继续选择文件块。由于 hex 文件包含地址信息，所以在解析的时候可能出现不同块，根据具体需求选择需要下载的地址块，点击“更新到列表”，再点击“应用到执行器”，完成对升级过程的配置。

7、回到固件升级弹窗，点击“执行”按钮，系统开始自动更新，过程中的信息会显示在该弹窗和串口窗口。

图 2-7

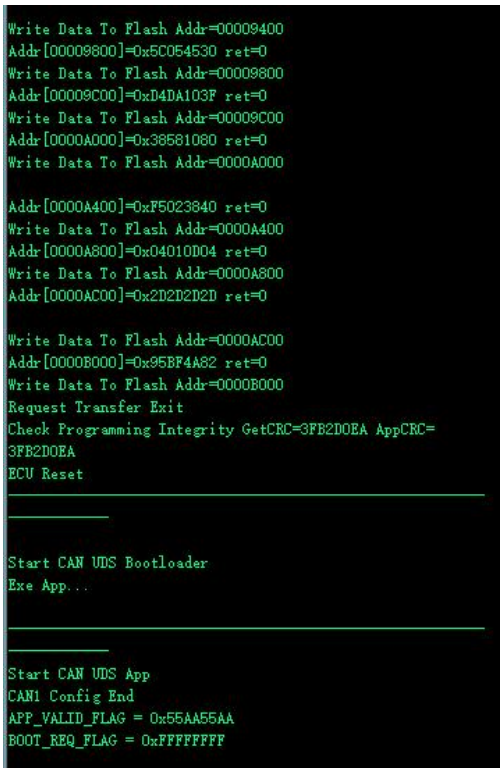

图 2-8

8、如图 2-8 所示，在 app 数据下载完成后，boot 程序执行跳转程序，app 程序成功运行，升级过程成功。

9、由于 app 程序中也包含 can 相关代码和 UDS 协议，再次点击“执行”，app 程序会清除相应状态，然后复位芯片，这样再次从 boot 程序运行，继续执行升级过程。

3. 代码讲解

3.1 相关参数配置

在从 boot 程序跳转到 app 程序过程中，需要使用 app 程序的中断向量表，而当 app 启动后需要将 boot 程序的向量表切换为 app 程序自己的向量表，这个过程成为向量表的重映射。常规程序中，中断产生后，在中断现场压栈工作后，硬件自动寻址对应中断服务函数地址，并跳转程序指针，执行函数内容。硬件寻址中断服务函数的时候，中断入口地址=SYS_VECTOFF(向量表偏移)+中断向量编号*4。SYS_VECTOFF 复位值是 0x0000 0000，所以通过向 SYS_VECTOFF 赋值 app 程序偏移地址的方式实现重映射过程。

同时，在生成 app 程序的过程中需要注意，默认工程编译文件会按照默认目标地址编译，即从地址空间起始开始。在执行升级的过程中，我们已经烧录了 boot 程序，该程序的地址已经是从 flash 起始地址开始。所以，为了使 app 程序编译后的地址，不与 boot 程序区域重叠，需要修改 app 程序的起始地址

记录编译地址等信息的链接文件默认会保存在 ChipON IDE 安装目录下：

ChipON IDE\KungFu32\ChiponCC32\scripting\ccr1_issue 找到对应芯片型号的 xxx.ld 链接文件，将其复制到工程目录下。打开 ld 文件后可以看到对应记录的 flash 起始地址和大小。本例程中将 boot 程序大小限制在 0x8000，所以将 app 的起始地址修改为 0x0000 8000。最后，将修改好的 ld 文件在 IDE 中生效：选中项目->右键->属性->C/C++构建->设置->通用设定->芯片脚本文件写入 -T"../KF32F350MQV.ld" ->应用->确定。

同时，为了避免 boot 区程序超过界限，应当在 boot 程序中配置程序大小为 0x8000。

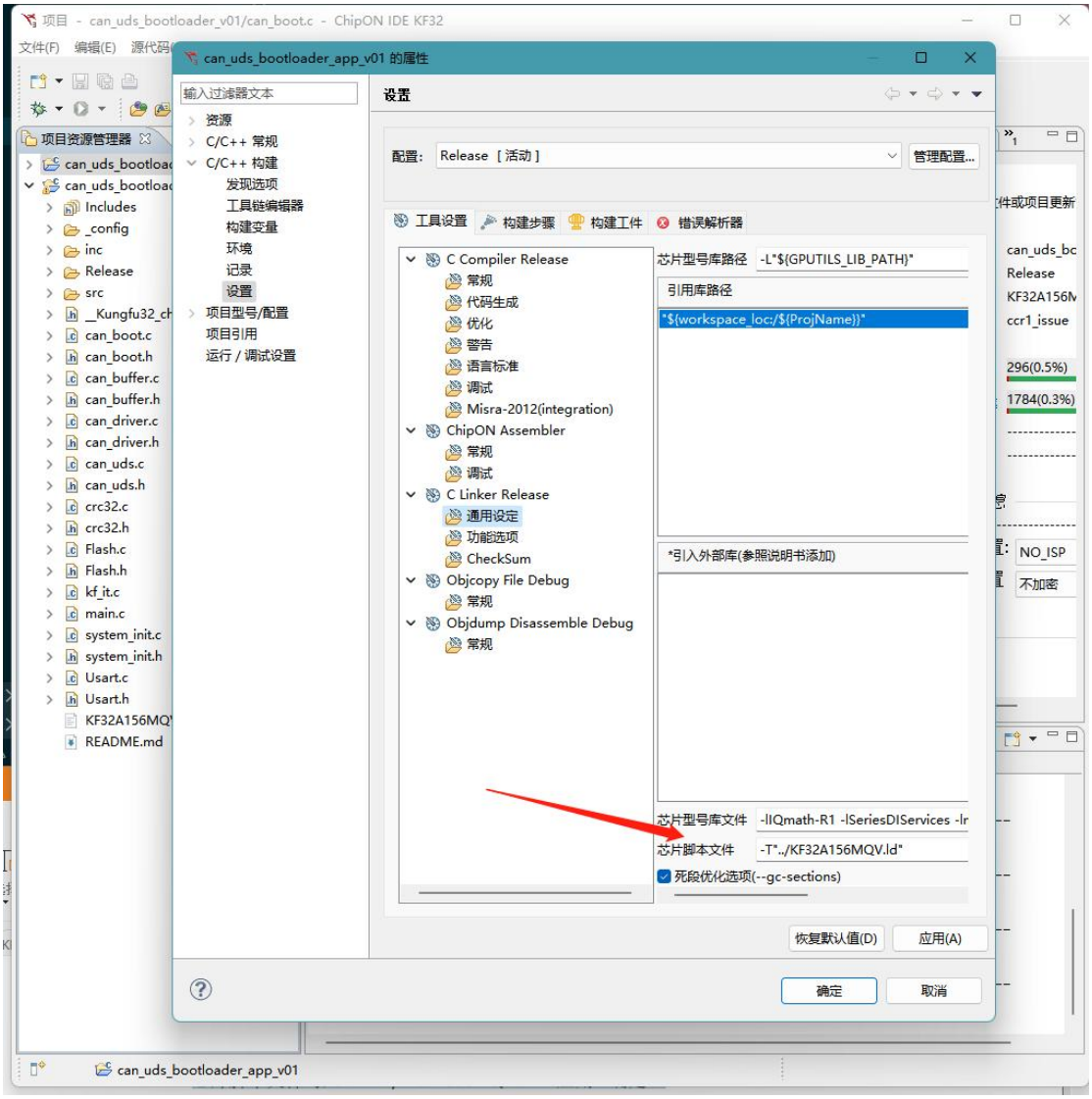

图 3-1

本例程工程中已将相关配置信息配置完善，用户在自行建立 app 程序的时候一定需要注意以上设置。

需要注意的是，当程序从 boot 程序跳转 app 程序的时候，由于正常上电执行过程是会在 start_up()函数中启动，若用户没有修改此执行顺序，app 程序运行至 main()函数时会初始化 RAM 空间，但由于芯片本身没有复位，在 boot 程序中已经初始化的寄存器状态是会保留的，在 app 程序中可以不用重复初始化，以加快启动过程。

3.2 boot 代码

在例程目录下，can_driver.h 头文件中定义了 CAN 总线相关的配置，用户可根据需要自行配置如 CAN 模块的选择，CAN 总线波特率，等参数。

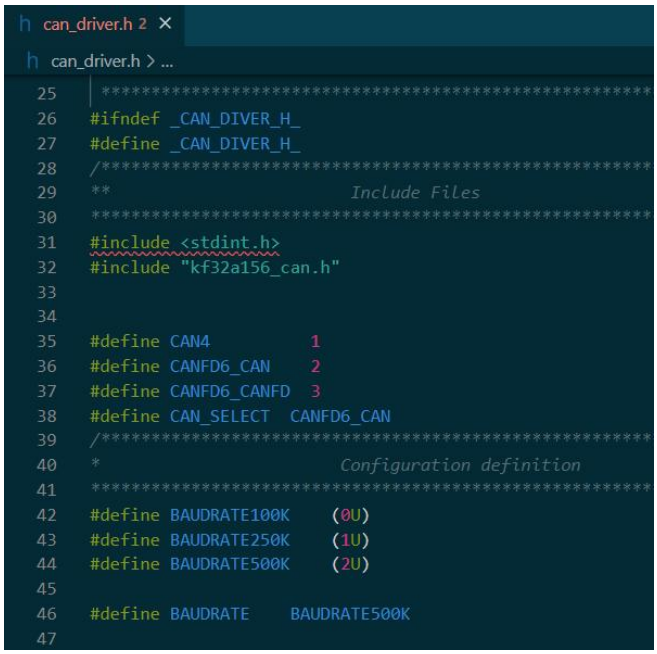

图 3-2

Can_uds.h 头文件下，用户可以配置 USD 协议相关信息，例如将 DEBUG_SHOW_CAN_MSG 宏定义配置为 1，可以使能原始 CAN 数据打印，MCU 接收到的原始 CAN 数据可以通过串口打印出来。节点地址等定义是与上位机软件相匹配的，用户若是修改这些参数，需要在上位机配置升级过程中时修改对应地址，以避免通信异常。

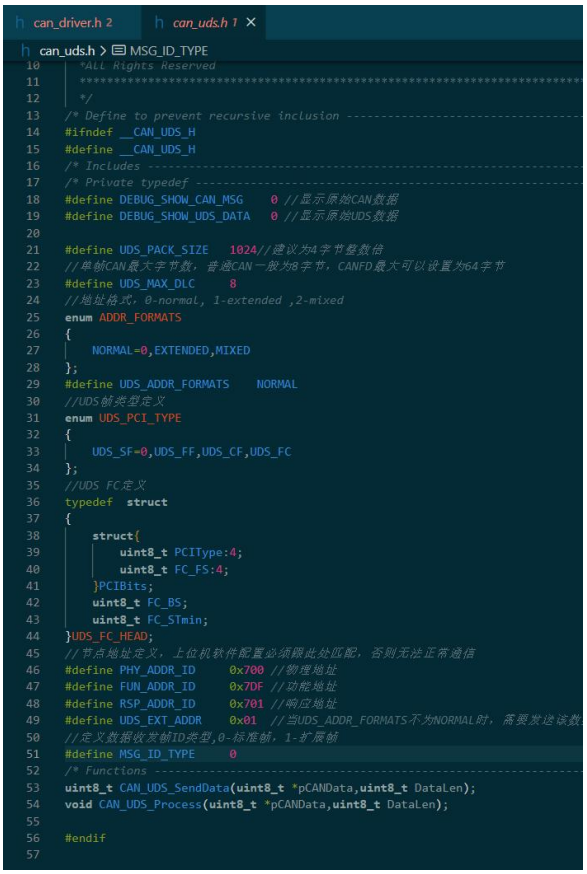
can_driver.h 2
can_uds.h 1
can_uds.h > MSG_ID_TYPE
*All Rights Reserved
*/
/* Define to prevent recursive inclusion -------------------------------
#ifndef __CAN_UDS_H
#define __CAN_UDS_H
/* Includes --
/* Private typedef ---
#define DEBUG_SHOW_CAN_MSG 0 //显示原始CAN数据
#define DEBUG_SHOW_UDS_DATA 0 //显示原始UDS数据
#define UDS_PACK_SIZE 1024//建议为4字节整数倍
//单帧CAN最大字节数，普通CAN一般为8字节，CANFD最大可以设置为64字节
#define UDS_MAX_DLC 8
//地址格式，0-normal, 1-extended ,2-mixed
enum ADDR_FORMATS
{
NORMAL=0,EXTENDED,MIXED
};
#define UDS_ADDR_FORMATS NORMAL
//UDS帧类型定义
enum UDS_PCI_TYPE
{
UDS_SF=0,UDS_FF,UDS_CF,UDS_FC
};
//UDS FC定义
typedef struct
{
struct{
uint8_t PCIType:4;
uint8_t FC_FS:4;
}PCIBits;
uint8_t FC_BS;
uint8_t FC_STmin;
}UDS_FC_HEAD;
//节点地址定义，上位机软件配置必须跟此处匹配，否则无法正常通信
#define PHY_ADDR_ID 0x700 //物理地址
#define FUN_ADDR_ID 0x7DF //功能地址
#define RSP_ADDR_ID 0x701 //响应地址
#define UDS_EXT_ADDR 0x01 //当UDS_ADDR_FORMATS不为NORMAL时，需要发送该数
//定义数据收发帧ID类型,0-标准帧，1-扩展帧
#define MSG_ID_TYPE 0
/* Functions ---
uint8_t CAN_UDS_SendData(uint8_t *pCANData,uint8_t DataLen);
void CAN_UDS_Process(uint8_t *pCANData,uint8_t DataLen);
#endif

图 3-3

```
/**
 * @brief  处理UDS服务，UDS相关功能在此函数实现
 * @param  SID 服务ID
 * @param  pData 服务参数指针
 * @param  DataLen 服务参数长度
 * @return 函数执行状态
 * @retval 1 成功
 * @retval 0 失败
 */
uint8_t CAN_UDS_ServiceProcess(uint8_t SID,uint8_t *pData,uint32_t DataLen)
{
  static uint32_t MemoryAddress;
  static uint32_t MemorySize;
  static uint32_t AppCRC32=0xFFFFFFFF;
  //static uint8_t DataBuffer[1024+2];
  static uint8_t FirstCRC=1;
  const uint32_t BlockSize = UDS_PACK_SIZE+2;//Max Number Of Block Length,需要加上SID和BSC
  static uint8_t BSC=1;//Block Sequence Counter
  switch(SID)
  {
    case 0x10:
      if((DataLen>=1)&&(pData[0]==0x03)){//extendedDiagnosticSession…
      else if((DataLen>=1)&&(pData[0]==0x02)){//Programming Mode…
      break;
    case 0x11:
      if((DataLen>=1)&&(pData[0]==0x01)){//ECU Reset…
      break;
    case 0x31:
      if((DataLen>=3)&&(pData[0]==0x01)&&(pData[1]==0xFF)&&(pData[2]==0x00)){//Erase Flash Memory…
      }else if((DataLen>=3)&&(pData[0]==0x01)&&(pData[1]==0xFF)&&(pData[2]==0x01)){//Check Programmin
      }else if((DataLen>=7)&&(pData[0]==0x01)&&(pData[1]==0x02)&&(pData[2]==0x02)){//Check Programmin
      }else if((DataLen>=3)&&(pData[0]==0x01)&&(pData[1]==0x02)&&(pData[2]==0x03)){//Check Programmin
      }else if((DataLen>=1)&&(pData[0]==0x01)){…
      }else{…
      break;
    case 0x34:
      if((DataLen>=10)&&(pData[0]==0x00)&&(pData[1]==0x44)){//Request Download…
      }else if((DataLen>=6)&&(pData[0]==0x00)&&(pData[1]==0x22)){//Request Download…
      }else{…
      break;
    case 0x36:
      if(DataLen>=2){//Transfer Data…
      }else{…
      break;
    case 0x85://ControlDTCSetting (0x85) service
      if(DataLen>=1){…
      break;
    case 0x28://CommunicationControl (0x28) service
      if((DataLen>=1)&&(pData[0]==0x03)){//禁止非诊断报文的发送和接收…
      break;
    case 0x27://SecurityAccess (0x27) service
      printf("SecurityAccess\r\n");
      if(pData[0]==0x01){…
      }else if(pData[0]==0x02){…
      break;
    case 0x37://Request Transfer Exit
      printf("Request Transfer Exit\r\n");
      CAN_UDS_ServiceRespond(SID|0x40,pData,0);
      break;
    default:
      break;
  }
  return 1;
}
```

图 3-4

在 can_uds.c 文件中包含了关于 UDS 协议的实现。其中如图 3-4 所示，CAN_UDS_ServiceProcess 函数实现了对于 UDS 服务的解析，部分服务的执行和运行逻辑可由用户自行扩展，补充。

4. 总结

通过以上对于 UDS 协议的介绍，结合例程目录下的代码资料，用户可以自行实现芯片通过 CAN 总线 UDS 协议升级程序的过程。同时本应用笔记提供的代码为 DEMO 测试用，用户可以根据自己实际的需求，在已有框架上进行移植，删减，扩展。

5.版本历史

文档版本历史

日期	版本	变更
2022 年 10 月 27 日	v01	初始版本。