一些工控产品，当系统复位后（非上电复位），可能要求保持住复位前 RAM 中的数据，用来快速恢复现场，或者不至于因瞬间复位而重启现场设备。而 keil mdk 在默认情况下，任何形式的复位都会将 RAM 区的非初始化变量数据清零。如何设置非初始化数据变量不被零初始化，这是本篇文章所要探讨的。

在给出方法之前，先来了解一下代码和数据的存放规则、属性，以及复位后为何默认非初始化变量所在 RAM 都被初始化为零了呢。

什么是初始化数据变量，什么又是非初始化数据变量？（因为我的文字描述不一定准确，所以喜欢举一些例子来辅助理解文字。）

定义一个变量：int nTimerCount=20；变量 nTimerCount 就是初始化变量，也就是已经有初值；

如果定义变量：int nTimerCount；变量 nTimerCount 就是一个非赋值的变量，Keil MDK 默认将它放到属性为 ZI 的输入节。

那么，什么是“ZI”，什么又是“输入节”呢？这要了解一下 ARM 映像文件（image）的组成了，这部分内容略显无聊，但我认为这是非常有必要掌握的。

ARM 映像文件的组成:

- 一个映像文件由一个或多个域（region，也有译为“区”）组成
- 每个域包含一个或多个输出段（section，也有译为“节”）
- 每个输出段包含一个或多个输入段
- 各个输入段包含了目标文件中的代码和数据

输入段中包含了四类内容：代码、已经初始化的数据、未经过初始化的存储区域、内容初始化为零的存储区域。每个输入段有相应的属性：只读的（RO）、可读写的（RW）以及初始化成零的（ZI）。

一个输出段中包含了一些列具有相同的 RO、RW 和 ZI 属性的输入段。输出段属性与其中包含的输入段属性相同。

一个域包含一到三个输出段，各个输出段的属性各不相同：RO 属性、RW 属性和 ZI 属性

到这里我们就可以知道，一般情况下，代码会被放到 RO 属性的输入节，已经初始化的变量会被分配到 RW 属性输入区，而“ZI”属性输入节可以理解为是初始化成零变量的集合。

已经初始化变量的初值，会被放到硬件的哪里呢？（比如定义 int nTimerCount=20；那么初始值 20 被放到哪里呢？），我觉得这是个有趣的问题，比如 keil 在编译完成后，会给出编译文件大小的信息，如下所示：

Total RO Size (Code + RO Data) 54520 (53.24kB)
Total RW Size (RW Data + ZI Data) 6088 (5.95kB)
Total ROM Size (Code + RO Data + RW Data) 54696 (53.41kB)

很多人不知道这是怎么计算的，也不知道究竟放入 ROM/Flash 中的代码有多少。其实，那些已经初始化的变量，是被放入 RW 属性的输入节中，而这些变量的初值，是被放入 ROM/Flash 中的。有时候这些初值的量比较大，Keil 还会将这些初值压缩后再放入 ROM/Flash 以节省存储空间。那这些初值是谁在何时将它们恢复到 RAM 中的？ZI 属性输入节中的变量所在 RAM 又是谁在何时给用零初始化的呢？要了解这些东西，就要看默认设置下，从系统复位，到执行 C 代码中你编写的 main 函数，Keil 帮你做了些什么。

硬件复位后，第一步是执行复位处理程序，这个程序的入口在启动代码里(默认)，摘录一段 cortex-m3 的复位处理入口代码：

```
1: Reset_Handler   PROC        ;PROC 等同于 FUNCTION,表示一个函数的开始,与 ENDP 相对?
2:
3:                 EXPORT  Reset_Handler            [WEAK]
4:                 IMPORT  SystemInit
5:                 IMPORT  __main
6:                 LDR     R0, =SystemInit
7:                 BLX     R0
8:                 LDR     R0, =__main
9:                 BX      R0
10:                ENDP
```

初始化堆栈指针、执行完用户定义的底层初始化代码（SystemInit 函数）后,接下来的代码调用了__main 函数，这里__main 函数会调用一些列的 C 库函数，完成代码和数据的复制、解压缩以及 ZI 数据的零初始化。数据的解压缩和复制，其中就包括将储存在 ROM/Flash 中的已初始化变量的初值复制到相应的 RAM 中去。对于一个变量，它可能有三种属性，用 const 修饰符修饰的变量最可能放在 RO 属性区，已经初始化的变量会放在 RW 属性区，那么剩下的变量就要放到 ZI 属性区了。默认情况下，ZI 数据的零初始化会将所有 ZI 数据区初始化为零，这是每次复位后程序执行 C 代码的 main 函数之前，由编译器“自作

主张”完成的。所以我们要在 C 代码中设置一些变量在复位后不被零初始化，那一定不能任由编译器“胡作非为”，我们要用一些规则，约束一下编译器。

分散加载文件对于连接器来说至关重要，在分散加载文件中，使用 UNINIT 来修饰一个执行节，可以避免__main 对该区节的 ZI 数据进行零初始化。这是要解决非零初始化变量的关键。因此我们可以定义一个 UNINIT 修饰的数据节，然后将希望非零初始化的变量放入这个区域中。于是，就有了第一种方法：

1. 修改分散加载文件，增加一个名为 MYRAM 的执行节，该执行节起始地址为 0x1000A000，长度为 0x2000 字节（8KB），由 UNINIT 修饰：

```
1: LR_IROM1 0x00000000 0x00080000  {    ; load region size_region
2:   ER_IROM1 0x00000000 0x00080000  {  ; load address = execution address
3:    *.o (RESET, +First)
4:    *(InRoot$$Sections)
5:    .ANY (+RO)
6:   }
7:   RW_IRAM1 0x10000000 0x0000A000  {  ; RW data
8:    .ANY (+RW +ZI)
9:   }
10:  MYRAM 0x1000A000 UNINIT 0x00002000  {
11:   .ANY (NO_INIT)
12:  }
13: }
```

那么，如果在程序中有一个数组，你不想让它复位后零初始化，就可以这样来定义变量：

```
unsigned char  plc_eu_backup[PLC_EU_BACKUP_BUF/8] __attribute__((at(0x1000A000)));
```

变量属性修饰符__attribute__((at(adder)))用来将变量强制定位到 adder 所在地址处。由于地址 0x1000A000 开始的 8KB 区域 ZI 变量不会被零初始化，所以处在这一区域的数组 plc_eu_backup 也就不会被零初始化了。

这种方法的缺点是显而易见的：要自己分配变量的地址，如果非零初始化数据比较多，这将是件难以想象的大工程（以后的维护、增加、修改代码等等）。所以要找到一种办法，让编译器去自动分配这一区域的变量。

2. 分散加载文家同方法 1，如果还是定义一个数组，可以用下面方法：

```
   unsigned char  plc_eu_backup[PLC_EU_BACKUP_BUF/8]
__attribute__((section("NO_INIT"),zero_init));
```

变量属性修饰符__attribute__((section(“name”),zero_init))用于将变量强制定义到 name 属性数据节中，zero_init 表示将未初始化的变量放到 ZI 数据节中。因为“NO_INIT”这显性命名的自定义节，具有 UNINIT 属性。

3. 如何将一个模块内的非初始化变量都非零初始化？

假如该模块名字为 test.c，修改分散加载文件如下所示：

```
1: LR_IROM1 0x00000000 0x00080000  {    ; load region size_region
2:   ER_IROM1 0x00000000 0x00080000  {  ; load address = execution address
3:    *.o (RESET, +First)
4:    *(InRoot$$Sections)
5:    .ANY (+RO)
6:   }
7:   RW_IRAM1 0x10000000 0x0000A000  {  ; RW data
8:    .ANY (+RW +ZI)
9:   }
10:  RW_IRAM2 0x1000A000 UNINIT 0x00002000  {
11:   test.o (+ZI)
12:  }
13: }
```

定义时使用如下方法：

```
int uTimerCount __attribute__((zero_init));
```

这里，变量属性修饰符__attribute__((zero_init))用于将未初始化的变量放到 ZI 数据节中变量，其实 keil 默认情况下，未初始化的变量就是放在 ZI 数据区的。

4.将整个程序的非初始化变量都非零初始化 看了上面的，这个已经没有必要说了