ARM Cortex-M底层技术（十二）KEIL MDK 分散加载-堆栈与预处理器 - weixin_39118482的博客

本文链接：https://blog.csdn.net/weixin_39118482/article/details/80202297
分散加载-堆栈与预处理器

在分散加载中处理堆栈：

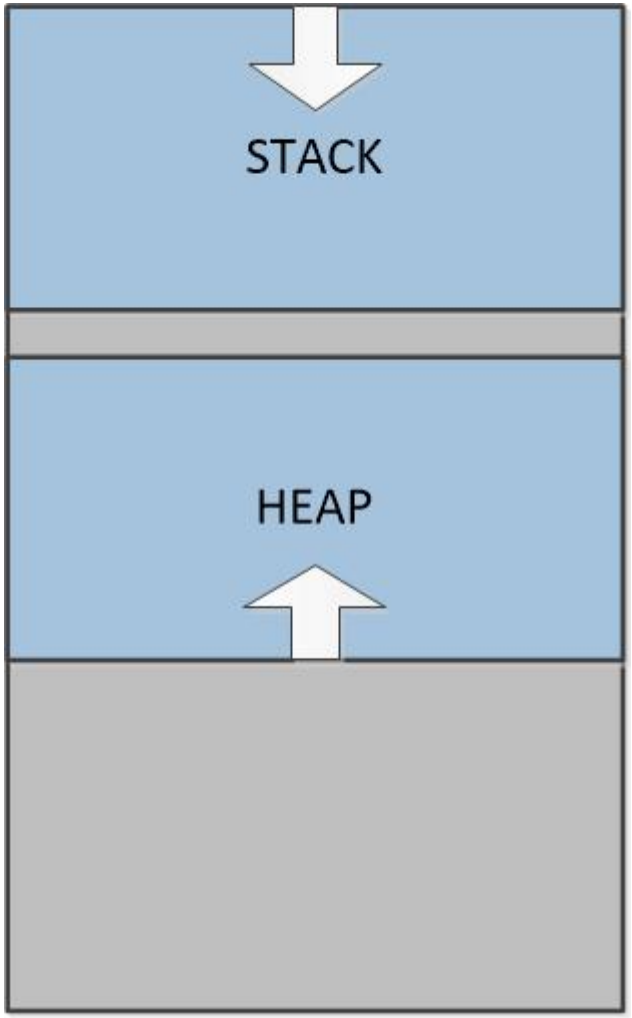
分散加载机制提供了一种方法，用于指定如何在映像中放置代码和静态分配数据。 应用程序的堆栈和堆是在 C 库初始化过程中设置的。 通过使用特别命名的 ARM_LIB_HEAP、ARM_LIB_STACK 或 ARM_LIB_STACKHEAP 执行区，可以调整堆栈和堆的放置。 此外，如果不使用分散加载描述文件，则可以重新实现 __user_initial_stackheap() 函数。

堆栈在分散加载中这样配置：

```
LOAD_FLASH    起始地址    加载域大小
{
   ...
   ARM_LIB_STACK    起始地址    EMPTY    -栈大小{ }
   ARM_LIB_HEAP    起始地址    EMPTY    堆大小{ }
   ...

}
```

我们知道在Cortex-M体系下栈是满递减堆栈（如下图所示），栈的方向是向下生长的，所以要在栈大小的前面加一个减号（注意那个减号）；属性是EMPTY。



在分散加载中使用预处理器：

在分散加载的第一行顶格编写以下语句即可调用预处理器：

“#! armcc -E”

调用预处理器就是说可以再分散加载中使用一些预处理器语句，比如：

#define等

```
#define m_interrupts_start              0x00000000
#define m_interrupts_size               0x00000400
#define m_text_start                    0x00000400

#define m_text_size                     0x0007FC00
```

然后在分散加载中使用m_interrupts_start或者m_interrupts_size来代替具体的地址，如下：

```
LR_m_text m_interrupts_start m_text_start+m_text_size-m_interrupts_start {
  VECTOR_ROM m_interrupts_start m_interrupts_size {
    * (RESET,+FIRST)
  }
  ER_m_text m_text_start FIXED m_text_size {
    * (InRoot$$Sections)
    .ANY (+RO)

  }

}
```

那么下面我贴出官方的一个工程上的分散加载，大家一起参考下，根据我们之前讲过的分散加载的文档，分析下这个实际中使用的分散加载：

```
#! armcc -E

#if (defined(__ram_vector_table__))
  #define __ram_vector_table_size__    0x00000400
#else
  #define __ram_vector_table_size__    0x00000000
#endif



#define m_interrupts_start            0x00000000
#define m_interrupts_size             0x00000400
```

```
#define m_text_start                0x00000400
#define m_text_size                 0x0007FC00


#define m_interrupts_ram_start      0x20000000
#define m_interrupts_ram_size       __ram_vector_table_size__


#define m_data_start                (m_interrupts_ram_start + m_interrupts_ram_size)
#define m_data_size                 (0x00028000 - m_interrupts_ram_size)


#define m_usb_sram_start            0x40100000
#define m_usb_sram_size             0x00002000


/* USB BDT size */
#define usb_bdt_size                0x0
/* Sizes */
#if (defined(__stack_size__))
  #define Stack_Size                __stack_size__
#else
  #define Stack_Size                0x0400
#endif


#if (defined(__heap_size__))
  #define Heap_Size                 __heap_size__
#else
```

```
  #define Heap_Size                 0x0400
#endif


LR_m_text m_interrupts_start m_text_start+m_text_size-m_interrupts_start { ; load region size_region
  VECTOR_ROM m_interrupts_start m_interrupts_size { ; load address = execution address
    * (RESET,+FIRST)
  }
  ER_m_text m_text_start FIXED m_text_size { ; load address = execution address
    * (InRoot$$Sections)
    .ANY (+RO)
  }


#if (defined(__ram_vector_table__))
  VECTOR_RAM m_interrupts_ram_start EMPTY m_interrupts_ram_size {
  }
#else
  VECTOR_RAM m_interrupts_start EMPTY 0 {
  }
#endif
  RW_m_data m_data_start m_data_size-Stack_Size-Heap_Size { ; RW data
    .ANY (+RW +ZI)
  }
  ARM_LIB_HEAP +0 EMPTY Heap_Size {    ; Heap region growing up
  }
  ARM_LIB_STACK m_data_start+m_data_size EMPTY -Stack_Size { ; Stack region growing down
  }
}
```

```
LR_m_usb_bdt m_usb_sram_start usb_bdt_size {
  ER_m_usb_bdt m_usb_sram_start UNINIT usb_bdt_size {
    * (m_usb_bdt)
  }
}


LR_m_usb_ram (m_usb_sram_start + usb_bdt_size) (m_usb_sram_size - usb_bdt_size) {
  ER_m_usb_ram (m_usb_sram_start + usb_bdt_size) UNINIT (m_usb_sram_size - usb_bdt_size) {
    * (m_usb_global)
  }
}
```