

Keil Scatter file 基礎語法

Scatter 檔案基礎介紹

Scatter 分散載入檔案主要在 Keil 連結程序時使用，提供生成 `img file` 時所需要的資訊。它指定了鏡像檔案內部各段的位置，可以為每段程式碼或者資料在 `load` 和 `execute` 時指定不同的儲存位置。

Scatter 中分 2 種域(region):

Load Region (LR, 載入域): 該 `img file` 開始運行前存放的區域，即當系統啟動或載入時，應用程式存放的區域。一般是 ROM 區域(某些場景會要求程序在 RAM 區域運行，掉電不保存程序，這時就在 RAM 區域)。

Execution Region(ER, 執行域): `img file` 執行階段的區域，即系統啟動後，應用程式進行執行和資料訪問的儲存器區域，系統在 `run-time` 執行階段可以有一個或多個執行域，一般是 RAM 區域

Img 裡所有的程式碼和資料都有一個 `loading address(LMA)` 和 `executing address(VMA)`，對於不同的內容，地址可能相同，也可能不同。

- **Code**: 程式碼，載入地址和運行地址相同，一般都處於 ROM
- **RO Data**: 程序定義的常數，載入地址和運行地址相同，一般都處於 ROM
- **RW Data**: 程序中已經初始化的全域變數，有非零 `default` 值，載入地址和運行地址不同，載入地址位於 ROM，運行地址位於 RAM
- **ZI Data**: 程序中未初始化的全域變數，都位於 RAM

上面可以看到 **RW Data** 載入地址和運行地址不相同，那是因為 **RW Data** 不能像 **ZI Data** 那樣沒有初始值，**ZI Data** 只要求其所在的區域全部初始化為零，所以只需要程序根據編譯器給出的 **ZI Data** 基地址及大小，來將相應的 RAM 清零。但 **RW Data** 卻不是，所以編譯器為了完成所有 **RW Data** 賦值，其先將 **RW Data** 的所有非零 `default` 值，先保存到 Flash 中，程序執行時，再從 Flash 中搬運到 RAM 中，所以 **RW Data** 既佔用 Flash 又佔用 RAM，且佔用的空間大小是相等的。

那程序是怎麼實現 **RW Data** 和 **ZI Data** 的初始化操作的呢？Keil `armcc` 在讀入 `scatter` 檔案之後會根據其中的各種地址生成啟動程式碼了，實現對 `img` 的載入，而這一段程式碼是 `__main()` 的一部分，位於 `InRoot$$Sections`。這就是在 `assembly` 啟動程式最後跳轉到 `__main()` 而不是跳向 `main()` 的原因之一。

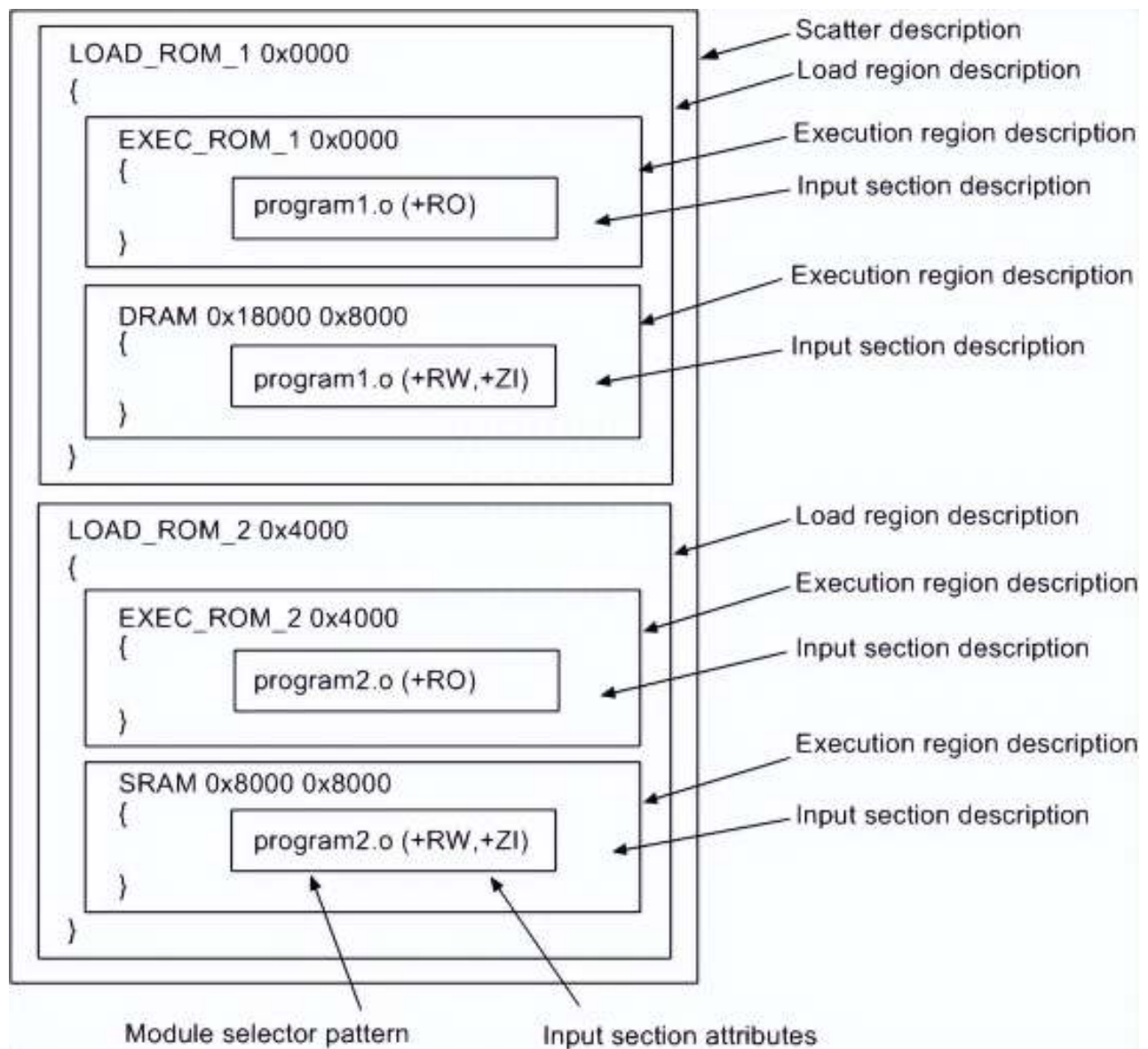
這裡介紹下 **Root region(InRoot\$\$Section)** 的概念。root region 是一個執行區域，其執行地址與其載入地址相同。分散檔案必須至少有一個 root region。

分散載入有個限制：負責建立 **ExecRegion** 的程式碼和資料，不能複製到另一個位置。因此 Keil 自動新增以下內容實現資料的複製，並規定必須包含在 **Root region** 中：

- **__main.o** 和 **__scatter*.o** 實現程式碼和資料從載入地址複製到執行地址
- **__dc*.o** 執行解壓
- **Region\$\$Table** 包含要複製或解壓的程式碼和資料的地址

因為這些 section 被定義為唯讀，所以它們按 ***(+R0)** 萬用字元語法進行分組。因此，如果在非根區域中指定了 ***(+R0)**，則必須使用 **InRoot\$\$Sections** 在 **root region** 中顯式聲明這些 section。

scatter 檔案可以有多個 **load region** 和多個 **execution region**。不過 **load region** 大多數情況下是只有 **1** 個，其大致的結構如下圖所示：



Load Region(ER, 載入域)描述

載入域語法格式如下所示：

```
load_region_name (base_address | (+ offset)) [attribute_list] [max_size]
{
    execution_region_description+
}
```

- **load_region_name**：載入域名稱，可以使用帶引號的名稱。僅當使用與 linker 定義的區域相關符號時，該名稱才區分大小寫。
- **base_address | (+ offset)**：基地址指示符，用來表示 Load Region 的起始地址，可以有下面兩種格式中的一種：
 - **base_address**：表示 Load Region 中的對像在連結時的起始地址，地址必須是 4-bytes 對齊；
 - **+ offset**：表示 Load Region 中的對像，在連結時的起始地址，是在前一個 Load Region 的結束地址後 offset-bytes 處。如果是第一個 Load Region，則它的起始地址即為從 0 地址開始偏移 offset，offset 的值必須能被 4 整除。同時如果使用 +offset，則 Load Region 可能會從先前的載入域繼承某些屬性。
- **attribute_list**：指定 Load Region 內容的屬性 (optional)，包含以下幾種，默認載入域的屬性是 ABSOLUTE。
 - **ABSOLUTE** - 將 Load Region 固定放置於 base designator 指向的起始地址位置(默認屬性，除非使用 PI 或者 RELOC)。
 - **ALIGN alignment** - 為 Load Region 新增 alignment 對齊約束。alignment 必須是 (2^n)對齊。
 - 如果 Load Region 起始地址使用 base_address 指定，那麼 base_address 必須滿足對齊約束。
 - 如果 Load Region 起始地址使用 + offset 指定，則 linker 將計算區域基地址以滿足對齊條件。
 - **NOCOMPRESS** - 默認情況下對 RW 資料壓縮。NOCOMPRESS 關鍵字能夠指定 Load Region 的內容不在最終鏡像中壓縮。
 - **OVERLAY** - 能夠在同一地址擁有多個 Load Region。Keil 不提供覆蓋機制，要在同一地址使用多個 Load Regions，必須自己進行覆蓋管理。如果內容放置在連結後不會更改的固定地址，則這段內容可能與指定為 OVERLAY 區域的其他區域重疊。
 - **PI** - 區域與位置無關。內容不依賴於任何固定地址，連結後可能會被移動，無需任何額外處理。

- **PROTECTED** – 可以防止 Load Region 被覆蓋。
 - **RELOC** – 該區域是可重新定位的。
- **max_size**：指定 Load Region 的最大尺寸。如果 Load Region 的實際尺寸超過了該值，linker 將報錯。
- **execution_region_description**：表示運行域，後面有個+號，表示其可以有一個或者多個 Execution Regions，關於運行域的介紹請看後面。

Execution Region(ER, 運行域)描述

運行域語法格式如下所示：

```
exec_region_name (base_address | +offset) [attribute_list] [max_size | length]
{
    input_section_description*
}
```

- **exec_region_name**：運行域的名稱，它除了唯一的標識一個運行域外，還用來構成 linker 生成的連結符號。
- **(base_address | +offset)**：base 地址指示符，用來表示 exec_region 的起始地址，可以有下面兩種格式中的一種：
 - **base_address**：表示 exec_region 中的對像，在連結時的起始地址，地址必須是 4-bytes 對齊的；
 - **+offset**：表示 exec_region 中的對像，在連結時的**起始地址，是在前一個 exec_region 的結束地址後 offset-bytes 處**。如果是第一個 exec_region，則它的起始地址即為 Load Region 的 base address 偏移 offset，offset 的值必須能被 4 整除。同時如果使用 +offset，則本執行域可能會從父載入域或者前一個執行域繼承某些屬性
- **attribute_list**：指定本 exec_region 內容的屬性(optional)，包含以下幾種：
 - **ABSOLUTE** - 將 exec_region 固定放置於 base designator 指向的起始地址位置。
 - **ALIGN alignment** - 為 exec_region 新增 alignment 對齊約束。alignment 必須是 (2^n) 對齊。
 - 如果 exec_region 起始地址使用 base_address 指定，那麼 base_address 必須滿足對齊約束。
 - 如果執行域起始地址使用 +offset 指定，則 linker 將計算區域基地址以滿足對齊條件。
 - **ALIGNALL value** - 增加 exec_region 內各節的對齊，value 必須是 2^n 並且必須 ≥ 4 。
 - **ANY_SIZE max_size** - 可以填充的 exec_region 內的最大大小。可以使用一個簡單的表示式來指定 max_size。
 - **EMPTY [-] length** - 在 exec_region 中保留給定大小的空記憶體塊，能將任何 section 放置在具有 EMPTY 屬性的區域中。
 - length 表示在記憶體中**向下增長**的堆疊。如果 length 為負值，則將 base_address 視為區域的結束地址。

- **FILL value** - 建立一個固定填充值的區域。如果指定 **FILL**，則必須給出一個值。
 - **NOCOMPRESS** - 默認情況下對 **RW** 資料壓縮。**NOCOMPRESS** 關鍵字能夠指定 **exec_region** 的內容不在最終 **img** 中壓縮。
 - **FIXED** - 固定地址。**linker** 嘗試使執行地址(VMA)等於載入地址(LMA)。如果成功，則該區域為 **Root Region**。如果不成功，則 **linker** 會產生錯誤。
 - **OVERLAY** - 用於具有重疊地址範圍的部分。如果連續的 **exec_region** 具有相同的 **+offset**，則它們被賦予相同的基地址。內容放置在不會更改的固定地址，可能與指定為 **OVERLAY** 的其他區域重疊。
 - **PADVALUE value** - 用於填充的值。如果指定 **PADVALUE**，則必須給出一個 4-bytes 的值。
 - **PI** - 該區域僅包含與位置無關的部分。內容不依賴於任何固定地址，連結後可能會被移動，無需任何額外處理。
 - **UNINIT** - 用於建立包含未初始化資料或記憶體對應 **I/O** 的執行區域。
 - **ZEROPAD** - 只有 **Root Region** 可以使用 **ZEROPAD** 屬性進行零初始化。將 **ZEROPAD** 屬性與非根執行區一起使用會生成警告，並且會忽略該屬性。
- **max_size**：對於標記為 **EMPTY** 或 **FILL** 的 **exec_region**，**max_size** 值被解釋為區域的長度。否則，**max_size** 值被解釋為執行區的最大大小。。
 - **length**：如果指定的長度為負值，則將 **base_address** 作為區結束地址。它通常與 **EMPTY** 一起使用，以表示在記憶體中變小的堆疊。

Input Section(輸入段)描述

輸入段語法描述如下所示：

```
module_select_pattern["("input_section_selector(", "input_section_selector)* ")"] \
("+ " input_section_attr | input_section_pattern | input_symbol_pattern)
```

- **module_select_pattern**：目標檔案濾波器，支援使用萬用字元"*"與"?"，**進行匹配時所有字元不區分大小寫**。
 - 符號"*"代表零個或多個字元，
 - 符號"?"代表單個字元。
- **input_section_attr**：屬性選擇器與輸入段屬性相匹配。每個 **input_section_attr** 的前面有一個"+"號。如果指定一個模式以匹配輸入段名稱，名稱前面必須有一個"+"號。可以省略緊靠"+"號前面的任何逗號。選擇器不區分大小寫(可以識別的為屬性 **First**、**Last**，它們標記執行區域中的第一個和最後一個部分)。

通過使用特殊模組選擇器模式 **.ANY**，可以將輸入段分配給執行區，而無需考慮其父模組。可以使用一個或多個 **.ANY** 模式以任意分配方式填充執行階段域。在大多數情況下，使用單個 **.ANY** 等效於使用*模組選擇器。

```
LR1 0x8000 (2 * 1024)
{
    ER1 +0 (1 * 1024)
    {
        *(+R0)
        ; module_select_pattern : *
        ; input_section_attr : (+R0)
    }
    ER2 +0 (1 * 1024)
    {
        *(+RW +ZI)
        ; module_select_pattern : *
        ; input_section_attr : (+RW +ZI)
        stm32f1xx_hal.o (i.HAL_GetTick) ; specific symbol
    }
}
```