Linker Notes

大头小脑

定义

用来把要执行的程序与库文件或其他已经翻译好的子程序（能完成一种独立功能的程序模块）连接在一起，形成机器能执行的程序。

作用

假如一个程序包含多个文件，在分别对每个源程序进行编译并得到多个目标程序后，要把这些目标程序连接起来，同时还要和系统提供的资源（如函数库）连接成一个整体。如果连接成功，会得到一个后缀名为.exe 的可执行文件。

应当说明的是：如果一个程序只包含一个文件，也必须进行连接，因为还要与系统提供的资源连接。

连接属于编译的最后一个环节，每一个连接都是被一个脚本控制。这个脚本使用连接命令语言写的。

连接脚本的主要目的是来描述输入区域的文件怎么映射到输出文件 ，和控制输出文件的存储布局。

Sections command：

区域命令告诉连接器：

1、 怎么把输入区域映射到输出区域。

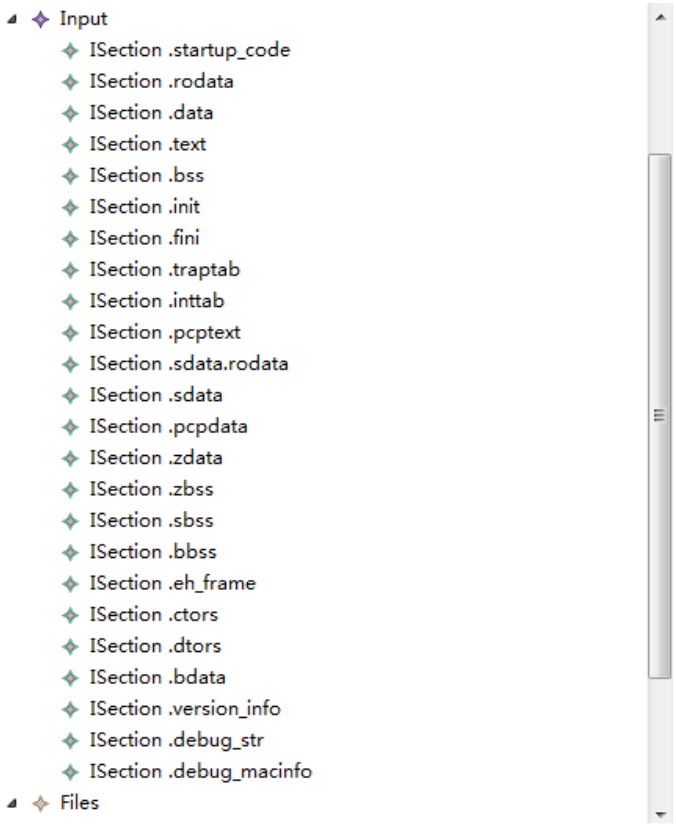
2、 怎么把输出区域放在内存中。

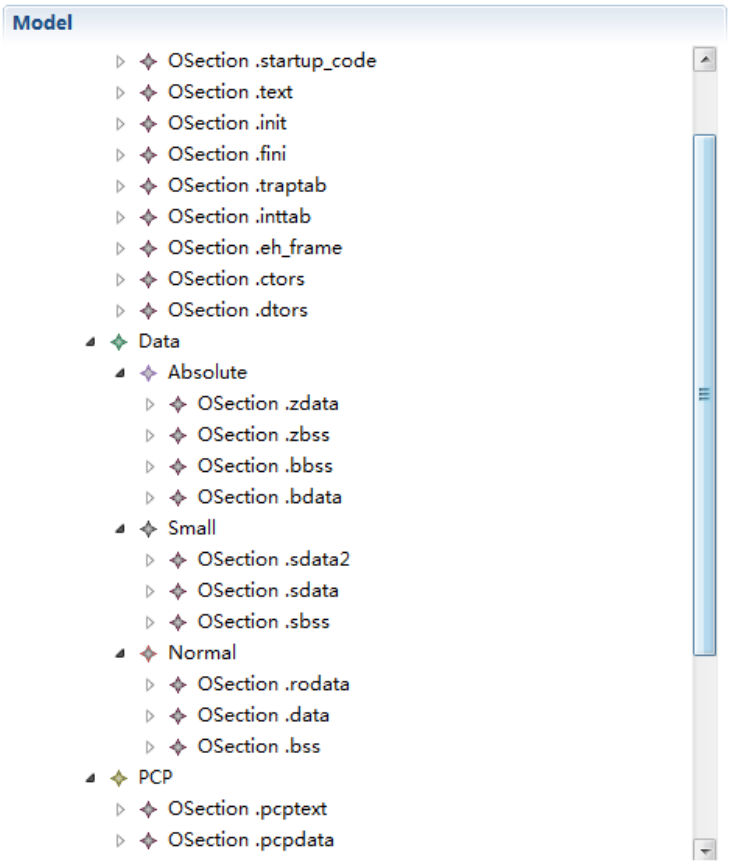
The format of the `SECTIONS` command is:

```
SECTIONS
{
  sections-command
  sections-command
  ...
}
```

Each *sections-command* may of be one of the following:

- an `ENTRY` command (see Section 3.4.1 [Entry command], page 39)
- a symbol assignment (see Section 3.5 [Assignments], page 45)
- an output section description
- an overlay description




Model
OSection .startup_code
OSection .text
OSection .init
OSection .fini
OSection .traptab
OSection .inttab
OSection .eh_frame
OSection .ctors
OSection .dtors
Data
Absolute
OSection .zdata
OSection .zbss
OSection .bbss
OSection .bdata
Small
OSection .sdata2
OSection .sdata
OSection .sbss
Normal
OSection .rodata
OSection .data
OSection .bss
PCP
OSection .pcptext
OSection .pcpdata

输入区域描述是最基本的连接脚本操作。

使用输出区域来告诉连接器如何把你的程序分布在内存中。

使用输入区域说明来告诉连接器如何把输入文件映射到存储区域。

输入区域描述包含一个文件名，可选的跟着一连窜的区域名字，在括号内。

文件名字和区域名字可能是通配符类型。

在输入区域，文件名字或者区域名字，或两者都是通配符类型的。

如果对于输入区域去哪了，比较困惑，使用-M 连接选项来产生一个 MAP 文件，MAP 文件可以准确的展示输入区域怎么和输出区域对应的。

下面的例子非常重要：

This example shows how wildcard patterns might be used to partition files. This linker script directs the linker to place all '.text' sections in '.text' and all '.bss' sections in '.bss'. The linker will place the '.data' section from all files beginning with an upper case character in '.DATA'; for all other files, the linker will place the '.data' section in '.data'.

```
SECTIONS {
  .text : { *(.text) }
  .DATA : { [A-Z]*(.data) }
  .data : { *(.data) }
  .bss : { *(.bss) }
}
```

当 Link-time 垃圾收集在使用的时候，连接选项（'—gc-sections'），标记不应该被取消的区域是有用的。可以使用 KEEP()老处理输入区域。

3.6.4.4 Input Section and Garbage Collection

When link-time garbage collection is in use ('--gc-sections'), it is often useful to mark sections that should not be eliminated. This is accomplished by surrounding an input section's wildcard entry with KEEP(), as in KEEP(*(.init)) or KEEP(SORT_BY_NAME(*)(.ctors)).

完整的输入区域的例子：

3.6.4.5 Input Section Example

The following example is a complete linker script. It tells the linker to read all of the sections from file '`all.o`' and place them at the start of output section '`outputa`' which starts at location '`0x10000`'. All of section '`.input1`' from file '`foo.o`' follows immediately, in the same output section. All of section '`.input2`' from '`foo.o`' goes into output section '`outputb`', followed by section '`.input1`' from '`foo1.o`'. All of the remaining '`.input1`' and '`.input2`' sections from any files are written to output section '`outputc`'.

```
SECTIONS {
  outputa 0x10000 :
    {
    all.o
    foo.o (.input1)
    }
  outputb :
    {
    foo.o (.input2)
    foo1.o (.input1)
    }
  outputc :
    {
    *(.input1)
    *(.input2)
    }
}
```

输出区域类型：

每个输出区域都有一个输出区域类型。类型是括号内的一个关键字。下面的类型被定义：

输出区域的属性根据映射到他的输入区域而定。你可以通过区域类型来覆盖这个。

每一个区域都有一个虚拟地址和一个载入地址。

每一个可载入或者可分配输出区域都含有两个地址。第一个是 VMA，或者叫做虚拟内存地址，第二个是 LMA，叫载入内存地址。

AT 关键字用来之名区域的载入地址。

VMA 和 LMA 的这种特点可以让建立 ROM 存储镜像很简单。

PHDRS　Command

ELF 目标文件格式，使用程序头文件，也叫做段。程序头描述程序应该怎么被载入到内存中。

Linker 将默认产生合理的程序头。但是有时候，你需要具体规定更加精确的制定程序头。

Linker　Script 的表达式：

Linker　script 语言中的表达式语法和 C 语言表达式的相同。

所以的表达式作为整形数据。

在所有的预处理指令中，#Pragma 指令可能是最复杂的了，它的作用是设定编译器的状态或者是指示编译器完成一些特定的动作

FAQ（313-357） userguide　.4.6.3

```
LONG(LOADADDR(.data));  LONG(0 + ADDR(.data));  LONG(SIZEOF(.data));
```

注:第一个 LONG(LOADADDR(.data));是载入到 flash 中的地址。
　第二个 LONG(0 + ADDR(.data));是载入到运行存储中的地址。
　LONG(SIZEOF(.data));是数据输出区域的大小（byte）

通过下面：可以给输出区域分配变量和函数。

```
#pragma section ".internalcode" ax
... user functions
#pragma section
```

存储分配的配置（非常重要，用来分配内存）：

Memory Regions The TriCore contains different memory regions like internal data ram, internal flash etc.

Alias Categories for memory regions. Code located in different output sections can be placed in the internal flash or in the scratch pad ram. To avoid having to assign a memory region to each code output section, so-called ALIAS e.g. CODE is assigned to these code output sections. Using this mechanism you can simply change the assignment of the memory region in the ALIAS CODE e.g. from internal flash to the scratch pad ram; then all code output sections will switch to the new memory region.

O-Section Input sections are collected in output sections (O-Sec). Each output section is assigned to a memory region or ALIAS category.

I-Section Input sections are created via the **#pragma** directive in your source code.

NOLAD Is required for uninitialized data.

LMA, VMA Address of loading and execution. A use case could be: The code of your application is located in the internal flash. Some frequently used functions of your application should be executed fast, therefore they should be located in the internal scratch pad ram (PMI_SPRAM). In such a scenario the code has to be copied from the internal flash (LMA) to scratch pad ram (here VMA) at runtime (see subsection 2.2.7 on page 15).

存储区域： Tricore 含有存储区域，像内部 flash 内存区域 、内部 ram 内存区域。

输出区域： 输入区域被收集在输入区域中。每一个输出区域被分配一个存储区域或者别名。

输入区域： 有两种，一种是目标文件中有的.text .data 段匹配，另一种是通过#pragma 来在用户源程序中添加。

NOLAD： 需要用于没有初始化的数据。

LMA,VMA ： 载入和执行的地址。Flash 中的代码在 flash 中，一些常用的代码当运行的时候，被放在 spram(spram 在 linker script 中怎么没有见到分配区域？？,这种转移是怎么实现的？？)

HighTec compiler 更能完美的支持 Tricore， HighTe Tricore 编译器允许使用不同的地址模式，通过 pragma 指令来控制代码和数据的传播和分配，在合适的存储区域。

这个使用通过 linker 修改来分配内存的过程包括：分配一个输入（I-Sec）给一个输出区域（O-Sec），分配一个存储区域给 O-Sec。

总结：

1、 通过 linker 的学习，了解 linker 工作的基本原理，会使用 GUI 方式配置 data 数据 和 function 存储空间。会直接在.ld 中添加代码。

2、 对于内存方式有了量的认识，以及如何去分配有了初步的认识