

(LK) 链接脚本

手稿

封面

本节简单介绍链接脚本的一般格式和常用命令。

概述

通常，编译程序的最后一步是运行 `ld` 进行链接。`ld` 将许多目标文件和归档文件合并，重定位它们中的数据并绑定符号引用。

`ld` 接受遵循 AT&T 的链接编辑器命令语言 (Link Editor Command Language) 语法超集编写的链接器命令语言 (Linker Command Language) 文件，从而为链接过程显式地提供整体控制。

`ld (version 2)` 使用通用目的 BFD 库来操作目标文件，这让它能够以多种格式读取、合并和写入目标文件 —— 例如 COFF、`a.out` 等。不同格式可以链接到一起，生成任何可用类型的目标文件。

链接器脚本

ld 命令语言是语句的集合。它们中一些只是简单的关键字，用于设置某个选项；而另一些则用于选择和分组输入文件，或者命名输出文件。两种语句在链接过程中都有着基础而普遍的影响。

ld 命令语言最基础的命令是 SECTIONS 命令。每个有意义的命令脚本都必须有一个 SECTIONS 命令：它指定输出文件的布局图，包含各种程度的细节。任何情况下其他命令都不是必须的。

MEMORY 命令通过描述目标架构的可用内存来补充 SECTIONS 命令。该命令是可选的；如果你不使用 MEMORY 命令，ld 会假设所有输出在连续内存块中有足够的内存。

注释

链接器脚本中可以有注释，格式和 C 语言中的注释一样，即，用 `/*` 和 `*/` 包围起来。和 C 一样，它们被当做空白来处理。

例如，下面是默认链接器脚本开头的一段注释。

```
/* Script for -z combreloc: combine and sort reloc sections */  
/* Copyright (C) 2014-2015 Free Software Foundation, Inc.  
   Copying and distribution of this script, with or without modification,  
   are permitted in any medium without royalty provided the copyright  
   notice and this notice are preserved.  */
```

表达式

许多有用的命令都会涉及到算术表达式。链接器脚本中表达式的语法和 C 语言完全一样，具有以下属性：

1. 所有表达式都被当做整数，且具有 long 或 unsigned long 类型；
2. 所有常量都是整数；
3. 支持所有 C 语言算术运算符；
4. 可以创建、定义和引用全局变量；
5. 可以调用特殊目的的内置函数；

下面是一个表达式的例子：

```
. = SEGMENT_START("text-segment", 0x400000) + SIZEOF_HEADERS;
```

整数

整数有多种表现形式，包括：

八进制整数 -- 以 `0` 开始，后面是零或多个八进制数字（`01234567`），例如 `0127`

十进制整数 -- 以非零开始，后面是零或多个十进制数字（`0123456789`），例如 `2546`

十六进制整数 -- 以 `0x` 或 `0X` 开始，后面是一或多个十六进制数字（`0123456789abcdefABCDEF`），例如 `0x93`

负整数 -- 以短线（`-`）为前缀，例如 `-384`

此外，`K` 和 `M` 可用作常量后缀。举例来说，下面是相等的数值：

```
_val_1 = 2K;  
_val_2 = 2048;  
_val_3 = 0x800;
```

符号名称

如果不带双引号，符号名称必须以字母、下划线或点（.）开头，可以包含任何字母、下划线、数字、点和短线。不带双引号的符号不能与关键字冲突。

如果想要在符号名称中包含一些特殊字符或者与关键字同名，你只能把它们放到双引号中。举例来说，下面这样的符号名称是有效的，尽管它们看起来有点怪：

```
"SECTIONS"
```

```
".data and .rodata"
```

位置计数器

点 (.) 是一个特殊的链接器变量，它包含的是当前输出位置计数器。由于 . 引用的总是输出段 (section) 中的位置，因此它必须出现在 SECTIONS 命令中。一个普通符号可以出现在表达式中的任何地方都可以出现 .，但对它赋值有副作用。向 . 赋值会导致位置计数器移动，这种操作可用于在输出段中创造洞 (hole)。注意，位置计数器不能向后移动。

```
SECTIONS
{
    output :
    {
        f1(.text)
        . = . + 1000;
        f2(.text)
    } = 0x1234;
}
```

在上面的例子中，f1 位于输出段的开始，然后是 1000 字节的洞，再后面是 f2。最后的 = 0x1234 表示在洞中填充字节 0x12 和 0x34。

表达式的计算

对于表达式，链接器使用“惰性计算”，也就是说，只有在完全必要时才会计算。为了能够做任何链接，链接器需要知道起始地址的值以及内存区域的长度；因此，当链接器读入命令文件后，这些值会尽可能早地被计算出来。

不过，其他值（比如符号的值）只有在存储分配完成后才可知或需要。这样的值会延迟到其他信息（比如输出段大小）在符号赋值表达式中可用时才被计算。

赋值语句

你可以使用任何 C 赋值操作符来创建全局符号并赋值。形如：

```
symbol = expression ;  
symbol &= expression ;  
symbol += expression ;  
symbol -= expression ;  
symbol *= expression ;  
symbol /= expression ;
```

以下两件事使得赋值不同于 ld 表达式中的其他操作符：

1. 赋值只能用于表达式末端，`a=b+3;` 是允许的，而 `a+b=3;` 则是错误的；
2. 必须在赋值语句结尾放置一个分号（`;`）；

赋值语句可出现在：

- ld 脚本中作为命令；
- SECTIONS 命令中作为独立语句；
- SECTIONS 命令中作为段定义内容的一部分；

表达式类型

当一个链接器表达式被计算并赋值给变量之后，它要么是绝对表达式类型，要么是可重定位表达式类型。

对于绝对表达式类型，符号中包含的值即输出文件中的值；而对于可重定位表达式类型来说，符号值是相对于段起始地址处的固定偏移量。

表达式的类型取决于它们在脚本中的位置。段定义中分配的符号在段起始地址相对位置处创建，任何其他位置分配的符号被创建为绝对符号。

函数 `ABSOLUTE` 会让符号成为绝对符号，即使位于段定义中。

如果你希望只有在符号被引用、并且没有其他目标文件定义这个符号时才创建符号，可以使用关键字 `PROVIDE`。

内置函数

下表列出了链接脚本表达式中可以使用的一些内置函数。

函数	描述
ABSOLUTE(exp)	返回表达式 <code>exp</code> 的绝对值（对应于可重定位值，而不是非负值）
ADDR(section)	返回 <code>section</code> 的绝对地址
LOADADDR(section)	返回 <code>section</code> 的绝对加载地址（如果没有 <code>AT</code> 关键字，同 <code>ADDR</code> ）
ALIGN(exp)	返回对齐到下一个 <code>exp</code> 边界的当前位置。 <code>exp</code> 必须是 2 的倍数 (2^n)。 相当于: $(. + exp - 1) \& \sim(exp - 1)$
DEFINED(symbol)	如果在链接器全局符号表中并且有定义，返回 1；否则，返回 0。
NEXT(exp)	返回下一个未分配地址（ <code>exp</code> 整数倍）。 如果没使用 <code>MEMORY</code> 命令为输出文件定义不连续内存，则该函数与 <code>ALIGN(exp)</code> ；一样。
SIZEOF(section)	如果 <code>section</code> 已分配，返回它的大小（字节数）。
SIZEOF_HEADERS sizeof_headers	返回输出文件头的大小（字节数）。如果想要管理页，可以将它作为首个段的起始地址。
MAX(exp1, exp2)	返回 <code>exp1</code> 和 <code>exp2</code> 之间的最大值
MIN(exp1, exp2)	返回 <code>exp1</code> 和 <code>exp2</code> 之间的最小值

分号

以下这些地方必须使用分号（;），其他地方可为了保持美观而存在，但会被忽略。

赋值

分号必须出现在赋值表达式的末尾

PHDRS

分号必须出现在 PHDRS 语句的末尾

入口点

链接器命令语言有一条命令专门用来定义输出文件中要执行的第一条指令，即入口点（entry point）。形如：

`ENTRY(symbol)`

定义入口点不止有 `ENTRY`，还有其他方式（优先级由高到低，高优先级覆盖低优先级设置）：

- 命令行选项 `-e entry`
- 链接器脚本中的 `ENTRY(symbol)` 命令
- 符号 `start` 的值（如果存在的话）
- `.text` 段首字节的地址（如果存在的话）
- 地址 `0`

指定输出段

SECTIONS 命令精确控制着输入段如何放入输出段、在输出文件中的顺序以及分配到哪个输出段。

一个脚本文件中最多只能有一个 **SECTIONS** 命令，但它里面可以有任意多条语句。**SECTIONS** 命令中的语句可以做以下 3 件事：

- 定义入口点
- 为符号分配值
- 描述输出段的布置，以及哪个输入段会进入

如果不使用 **SECTIONS** 命令，链接器将每个输入段放到同名输出段中，按照它们在输入文件中首次出现的顺序进行排列。举例来说，如果第一个输入文件中出现了所有输入段，那么输出文件中段的顺序与第一个输入文件中它们出现的顺序相同。

段定义

SECTIONS 命令中最常用的语句就是段定义，它们用来指定输出段的属性，包括：位置、对齐、内容、填装模式以及目标内存区。大多数这些选项是可选的，因此段定义最简单的形式是下面这样。

```
SECTIONS { ...  
    secname : {  
        contents  
    }  
    ... }
```

secname 是输出段名称，contents 则描述了里面的具体内容。为了表述明晰，secname 周围的空白是必须的；其他空白是可以省略的，但 : 和 {} 是不可少的。

secname 必须符合输出模式的约束。例如，a.out 只允许使用 .text, .data 和 .bss。

特殊名称 /DISCARD/ 可用于将输入段排除在最终的链接输出之外。

如果某个段没有任何内容，链接器也不会创建这个输出段。

段的布置

在段定义中，你可以通过罗列某些输入文件、输入文件段或二者的组合来指定输出段的内容。你还可以在段中放置任何数据，定义相对于段起始处的符号。段定义中的内容可以包含下表中任何种类的语句，此类语句可以在单个段定义中出现任意多次，用空白分隔。

filename	输出段中只含输入文件名，文件中所有段都放入当前段定义。
示例	<code>.data : { fa.o fb.o fc.o }</code>
filename(section) filename(section, section, ...) filename(section section ...)	列出输入文件的一或多个段名，插入当前输出段； 如果括号内有多输入文件段，用逗号 (,) 或空白分隔。
示例	<code>fa(.data)</code> <code>fb(.data, .rodata)</code> <code>fc(.data .rodata)</code>
*(section) *(section, section, ...) *(section section ...)	用通配符 * 表示所有文件（其他通配符的用法类似，如 ? []）。 如果前面已经显式指定了某些文件，那么 * 指的是所有剩余文件（在输出文件中的位置还未定义的那些文件）。
示例	<code>SECTIONS {</code> <code>.text :{</code> <code>*(.text .stub .text.* .gnu.linkonce.t.*)</code> <code>}</code> <code>}</code>
filename(COMMON) *(COMMON)	指定于输出文件何处放置未初始化数据。

段定义的完整语法

段定义的完整语法描述如下所示。

```
SECTIONS {  
    ...  
    secname start BLOCK(align) (NOLOAD) : AT ( ldadr )  
        { contents } >region :phdr =fill  
    ...  
}
```

其中，只有 *secname* 和 *contents* 是必须的，其他都是可选项。

start -- 强制输出段在指定地址处加载。

BLOCK(*align*) -- 在段开始前移动位置计数器，用来段对齐。

(NOLOAD) -- 标记段，让程序加载器在运行时不将它加载到内存。

AT(*ldadr*) -- 指定段的加载地址；默认（没有 AT 时）与重定位地址一样；常用于构建 ROM 镜像。

>*region* -- 将该段分配给之前定义的内存区域。

:*phdr* -- 将该段分配给程序头描述的段（segment）。

=*fill* -- 指定该段的初始填充值；任何未分配的洞均用 *fill* 表达式的最后 2 个字节进行填充。

特殊用途命令

下面是特殊用途的一些其他命令。

命令	用途
CONSTRUCTORS	告诉链接器于何处放置构造器和解析器信息
FLOAT NOFLOAT	用于早期链接器，请求某个特殊数学子函数库。
FORCE_COMMON_ALLOCATION	让 ld 为 COMMON 符号分配空间，即使指定了可重定向输出文件。
INCLUDE filename	包含链接器脚本；先搜索当前目录，然后 -L 指定的目录。最多嵌套 10 层。
INPUT(file, file, ...) INPUT(file file ...)	在链接中包含二进制文件，不放入段定义。
GROUP(file, file, ...) GROUP(file file ...)	同上，但文件是归档文件。
OUTPUT(filename)	指定输出文件名
OUTPUT_FORMAT(bfdname)	如果支持多种目标代码格式，指定输出格式为其中之一。
SEARCH_DIR(path)	添加归档库文件的搜索路径
STARTUP(filename)	确保 filename 是链接过程使用的首个文件
TARGET(format)	如果支持多种目标代码格式，指定输入文件格式为其中之一。
NOCROSSREFS(section section ...)	当某些段之间发生引用时，ld 发出错误。

默认链接脚本

最后，我们给出 ld 默认链接脚本的片段，你可以在命令行上运行 `ld --verbose` 来查看完整代码。

```
OUTPUT_FORMAT("elf64-x86-64", "elf64-x86-64", "elf64-x86-64")
OUTPUT_ARCH(i386:x86-64)
ENTRY(_start)
SECTIONS
{
    /* Read-only sections, merged into text segment: */
    PROVIDE (__executable_start = SEGMENT_START("text-segment", 0x400000));
    . = SEGMENT_START("text-segment", 0x400000) + SIZEOF_HEADERS;

    .text :
    {
        *(.text .stub .text.* .gnu.linkonce.t.*)
    }

    .data :
    {
        *(.data .data.* .gnu.linkonce.d.*)
    }

    .bss :
    {
        *(.bss .bss.* .gnu.linkonce.b.*)
        *(COMMON)
    }
}
```



参考资料

参考资料：

https://ftp.gnu.org/old-gnu/Manuals/ld-2.9.1/html_mono/ld.html