**https://blog.csdn.net/ssswwwnnn/article/details/88571947?utm\_medium=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.control&dist\_request\_id=&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.control**

**链接器脚本**

链接器脚本控制每个链接过程，这种脚本使用链接器命令语言。  
链接器的主要目的就是描述输入文件的各个段如何映射到输出文件，并且控制输出文件的内存布局。大多数链接器脚本就只做这些事情。但是，如果必要的话，链接器脚本也可以通过使用如下描述的命令来指挥链接器做许多其他的操作。

链接器总是在使用一个链接器脚本的。如果你没有亲自指定一个，那么链接器就会使用编译到其二进制代码中的默认脚本。你可以使用"–verbose"命令行选项来显示默认的链接器脚本。特定的链接器脚本选项，比如"-r"或者"-N"会影响到默认链接器脚本。

你也可以使用"-T"命令行选项指定自定义的链接器脚本。这时，链接时你指定的链接器脚本就会取代缺省链接器脚本（译注：只是当前取代，不是永久）。

你也可以把链接器脚本作为链接器的输入文件，就好像被链接的文件一样。参阅"隐式链接器脚本"

**基本概念**

为了描述链接器脚本语言我们需要定义一些基本概念和词汇。

链接器把多个输入文件链接为一个输出文件。输出文件和每个输入文件都拥有一种特殊的数据格式，称作目标文件格式。每个文件的都被叫做目标文件，输出文件通常叫做可执行文件，不过这儿出于我们的目的，我们仍然称呼它为目标文件。每个目标文件有一系列的段。我们用输入段来引用输入文件中的段，用输出段来引用输出文件中的段。

目标文件中的每个段都有一个名字和大小。大多数段还有一块关联的数据，称作段内容。一个段可以被标记为可装载的，意思是当输出文件运行时段的内容应该被装载到内存中。没有内容的段可能是可分配的，意思是保留的内存区域。既不是可装载的也不是可分配的段典型的就是包含某类调试信息的段。

每个可装载的或者可分配的输出段有两个地址。第一个地址是VMA（虚拟内存地址），这是当输出文件文件运行时该段所拥有的地址。还有一个段是LMA（装载内存地址），这个是段将被装载到的地址。在大多数情况下，这两个地址是一样的。一个不一样情况的例子就是当一个数据段装载到ROM中的情况，然后再在程序启动的时候拷贝到RAM（这类技术常常用在基于ROM的系统中用来初始化全局变量）。在这种情况下LMA就是ROM地址，VMA就是RAM地址。

你可以使用"objdump"程序的"-h"选项来查看目标文件的各个段。

每个目标文件还含有一系列符号，叫做符号表。一个符号可能是已定义的或者未定义的。每个符号有一个名字，并且每个定义的符号有一个地址，和一些其他信息。

如果你将一个C或者C++程序编译成一个目标文件，对于每个已经定义好的函数、全局或者静态变量你会得到一个已定义的符号。每个在输入文件中被引用到的未定义的函数或者全局变量变为一个未定义的符号。

你可以使用"nm"程序或者使用"objdump"的"-t"选项来查看目标文件的符号表。

**链接器脚本格式**

链接器脚本是文本文件。

你把一系列命令来写成链接器脚本。每个命令可以是一个关键字，通常会跟参数或者一个  
赋值的符号。你可以用分号来分隔命令。空白会被忽略。

字符串比如文件名或者格式名可以直接输入，如果文件名内有特殊字符（比如可能被认为  
用来分隔文件明的逗号）时，那么用双引号将文件名围起来。文件名中不允许出现双引号。

你可以在链接器脚本中使用C风格的注释，由’/*‘和’*/'包围。就像C中一样，注释等同于  
空白。

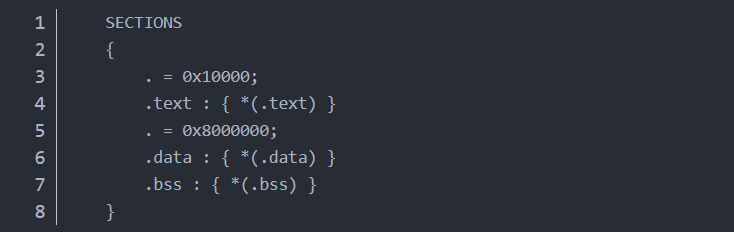
**简单链接器脚本示例**

许多链接器脚本都相当简单。

最简单的链接器脚本就一个命令： “SECTIONS”。你可以使用"SECTIONS"命令来描述输出  
文件的内存布局。

“SECTIONS"命令非常强大。这里我们将会描述一下它的简单用法。假定你的程序里面只有  
代码，已经初始化的数据和为数初始化的数据。那么这些将会分别被安排到”.text"，  
“.data"和”.bss"段中。假定这是你的输入文件中仅有的段。

对于这个例子，我们认为代码应该被装载到地址0x10000处，并且数据应该从地址0x8000000  
开始。下面的链接器脚本描述了这个：



你通过使用关键字"SECTIONS"来表示"SECTIONS"命令，后面是一系列符号分配和输出段描  
述，它们被围在大括号中。

上述例子中"SECTIONS"命令的第一行设置了特殊符号位置计数器’.'的值。如果你不通过  
其他手段指定输出段的地址，那么地址就会被设置为当前位置计数器的值。然后位置计数  
器会根据输出段的大小增加。在"SECTIONS"命令的起始处，位置计数器的值为"0"。

第二行定义了一个输出段".text"，冒号是语法的要求，不过现在可能已经忽略了。在输出  
段名字后的大括号里面，你列出了需要放到这个输出段里的输入段。"*“是一个通配符用来  
指代所有文件名。表达式”*(.text)“表示所有输入文件的所有”.text"输入段。

由于当".text"段被定义以后位置计数器的值是"0x10000"，链接器就会把输出文件的  
“.text"段的地址设置为"0x10000”。

接下来的行定义了输出文件的".data"和".bss"段。链接器会把".data"输出段放到地址  
0x8000000。在链接器放置了".data"输出段之后，位置计数器的值变为0x8000000加上  
“.data"输出段的大小。接下来链接器把”.bss"输出段直接放在".data"输出段之后。

如果必要的话，链接器会通过增加位置计数器以保证每个输出段都符合要求的对齐方式。  
在本例中，为".text"和".text"输出段指定的地址很可能满足任意的对齐要求，但是链接  
器可能会在".data"和".bss"段之间插入一个小的空白。

这就是一个简单而完备的链接器脚本。

**简单链接器脚本命令**

本节我们描述简单的链接器脚本命令

入口点：设置入口地址  
程序执行的第一条指令称之为入口点。你可以使用"ENTRY"命令来设置入口点。它的参数是  
一个符号名：  
ENTRY(symbol)  
存在许多种方式来设置入口点。链接器会按照以下顺序来尝试设置入口点，当其中一个成  
功时停止尝试。

1. "-e"命令行选项
2. 链接器脚本中的ENTRY(symbol)命令
3. 符号"start"的值，如果已定义的话
4. ".text"段的第一个字节的地址，如果设置了的话
5. 地址0

文件命令：处理文件的命令  
有多个命令处理文件。  
INCLUDE filename  
在此处包含filename指定的链接器脚本。会在当前目录以及"-L"选项指定的目录搜索  
该文件。你可以最多嵌套10层"INCLUDE"  
你可以在最外层安排"INCLUDE"指令，也可以在"MEMORY"或者"SECTIONS"命令中，或者  
在输出段描述中。

INPUT(file, file, ...)  
INPUT(file file ...)  
“INPUT"命令指挥链接器链接指定名字的文件，就好像在命令行上指定的文件一样。  
举例来说，如果链接的时候你总是想要包含"subr.o”，但是你又不想每个链接命令行  
都放它，那么你就可以在你的链接器脚本中包含"INPUT(subr.o)"  
事实上，只要你愿意，你可以把所有的输入文件都放到链接器脚本里，然后调用链接  
器时仅需要一个"-T"选项。  
当配置了"sysroot prefix"并且文件名以"/“打头，并且脚本在sysroot prefix内处理  
则"filename"会在sysroot prefix内搜索。否则，链接器会尝试在当前目录打开该文  
件。如果没有找到，链接器会搜索档案搜索目录，查看命令行选项中的”-L"描述。  
如果你使用 “INPUT (-lfile)”，链接器会把该名字转化为libfile.a，就好像使用"-l"  
命令行选项一样  
当你在一个隐式链接器脚本中使用"INPUT"命令时，则那些文件会在该脚本文件被包含  
的时候包含到链接中去。这个会影响到档案包的搜索。

GROUP(file, file, ...)  
GROUP(file file ...)  
“GROUP"命令类似"INPUT”，除了这里的文件都必须是档案包以外，并且它们会被重复  
搜索直到再也没有未定义的引用。参看命令行选项的"-("描述。

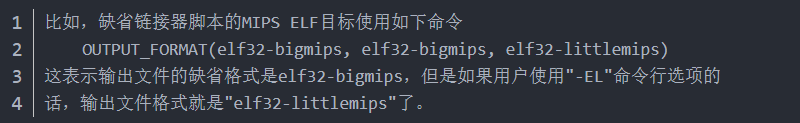
AS\_NEEDED(file, file, ...)  
AS\_NEEDED(file file...)  
这个构造之可能出现在"INPUT"和"GROUP"命令里面，这些文件就像直接出现在"INPUT"  
或者"GROUP"命令中那样被处理，一个例外是ELF共享库，只在它们被用到时才会被添  
加。这个构造会为所有在里面列出的文件启用"–as-needed"选项。并且重新装载前面  
的–as-needed和它后面的–no-as-needed设置。  
OUTPUT(filename)  
“OUTPUT"命令设置输出文件名。使用这个命令就和”-o filename"命令行选项一个效  
果。如果都使用了，命令行选项优先。你可以使用这个命令给输出文件定义一个缺省  
名字，而不是使用"a.out"

SEARCH\_DIR(path)  
SEARCH\_DIR命令添加库搜索路径。它的效果和"-L"命令行选项完全相同。如果都使用  
了，命令行选项优先。

STARTUP(filename)  
"STARTUP"命令类似"INPUT"命令，除了"filename"文件会作为链接的第一个输入文件  
以外，就好像它是命令行上指定的第一个文件一样。这在那种入口点总是位于第一个  
文件的系统中可能有用。

格式命令：处理目标文件格式的命令  
有一组链接器命令来处理目标文件格式

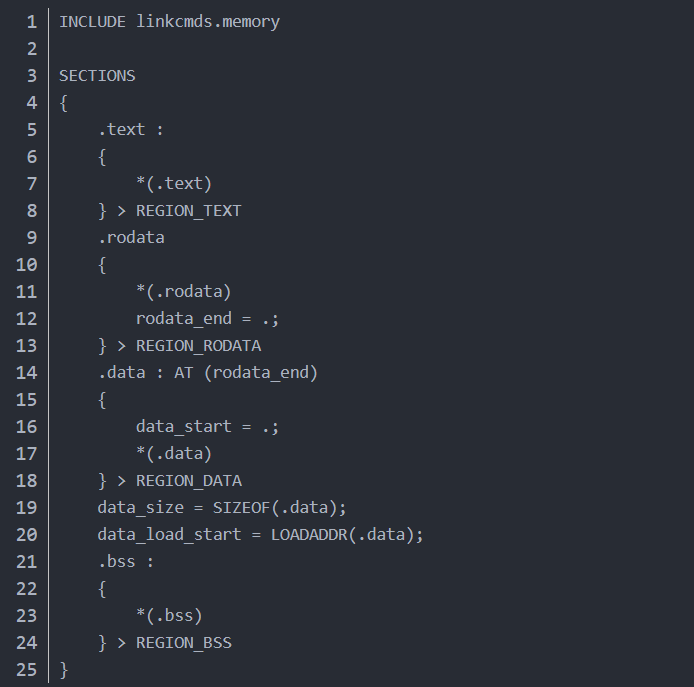
OUTPUT\_FORMAT(bfdname)  
OUTPUT\_FORMAT(default, big, little)  
“OUTPUT\_FORMAT"命令指定输出文件为BFD格式。该命令和”–oformat bfdname"命令行  
选项相同。  
你可以使用三个不同的参数来和"-EB"和"-EL"命令行选项配合使用指定输出文件的大  
小端格式。  
如果既没有使用"-EB"也没有用"-EL"，那么就使用第一个参数指定的大小端格式。



TARGET(bfdname)  
“TARGET"命令命名读入输入文件时将要使用的BFD格式。它会影响接下来的"INPUT"和  
“GROUP"命令。这个命令和使用”-b bfdname"命令行选项效果相同。如果"TARGET"命令  
使用了，但是"OUT\_FORMAT"命令没有使用，那么最后一个"TARGET"命令也会被用来设  
置输出文件的格式，参见"BFD”。

区域别名：赋予内存区域别名  
可以给使用"MEMORY"命令创建的内存区域添加别名。每个名字至多对应一个内存区域。  
REGION\_ALIAS(alias, region)  
“REGION\_ALIAS"函数为内存区域"region"创建一个别名"alias”。这允许输出段灵活的映射  
到内存区域。下面是一个例子。

假定我们有一个拥有多类型存储设备的嵌入式系统上的应用程序。都有一个通用的目的，  
易失性存储器RAM允许执行代码或存储数据。有一些可能是只读，非易失性的ROM存储器允  
许执行代码和只读访问。另外一种是只读，非易失性的ROM2类型存储，不能在上面执行代  
码，只允许只读访问数据。我们有四个输出段：  
“.text” 程序代码段  
“.rodata” 只读数据段  
“.data” 可读可写的已初始化数据段  
“.bss” 可读可写的用0初始化的数据段  
目标是提供一个链接器命令文件包含一个系统无关部分定义输出段和一个系统相关部分映  
射输出段到系统的可用存储区。我们的嵌入式系统有三类不同存储区安排方案：A，B和C。  
Section Variant A Variant B Variant C  
.text RAM ROM ROM  
.rodata RAM ROM ROM2  
.data RAM RAM/ROM RAM/ROM2  
.bss RAM RAM RAM  
标记"RAM/ROM"和"RAM/ROM2"表示这个段分别装在到区域ROM或者ROM2。请注意".data"段的  
装载地址在所有三种方案中都开始于".rodata"段后面。  
处理这个输出段的链接器脚本放在下面。它包含描述内存布局的系统相关的  
"linkcmds.memory"文件。



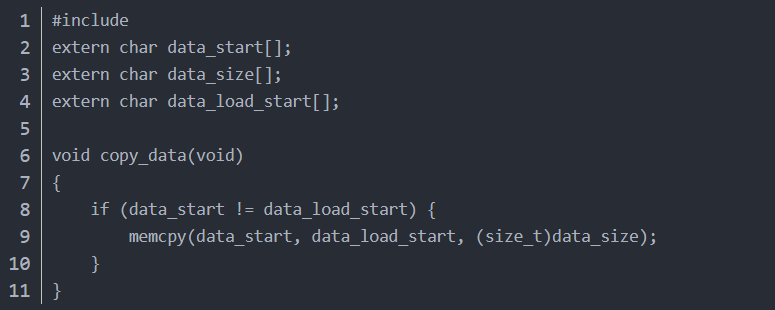
现在我们需要三个不同的linkcmds.memory文件来定义内存区域和别名。下面是三种方案  
A，B和C的linkcmds.memory

方案A：所有的都放到RAM中  
MEMORY  
{  
RAM : ORIGIN = 0, LENGTH = 4M  
}  
REGION\_ALIAS(“REGION\_TEXT”, RAM);  
REGION\_ALIAS(“REGION\_RODATA”, RAM);  
REGION\_ALIAS(“REGION\_DATA”, RAM);  
REGION\_ALIAS(“REGION\_BSS”, RAM);

方案B：程序代码和只读数据放到ROM中。可读写数据放到RAM中。一个已初始化数据的镜像  
放在ROM中并且在系统启动的时候拷贝到RAM中。  
MEMORY  
{  
ROM : ORIGIN = 0, LENGTH = 3M  
RAM : ORIGIN = 0x10000000, LENGTH = 1M  
}  
REGION\_ALIAS(“REGION\_TEXT”, ROM);  
REGION\_ALIAS(“REGION\_RODATA”, ROM);  
REGION\_ALIAS(“REGION\_DATA”, RAM);  
REGION\_ALIAS(“REGION\_BSS”, RAM);

方案C：程序代码放到ROM中，制度数据放到ROM2。可读写数据放到RAM，一个已初始化数  
据的镜像放到ROM2并在系统启动的时候拷贝到RAM中。  
MEMORY  
{  
ROM : ORIGIN = 0, LENGTH = 2M  
ROM2 : ORIGIN = 0x10000000, LENGTH = 1M  
RAM : ORIGIN = 0x20000000, LENGTH = 1M  
}  
REGION\_ALIAS(“REGION\_TEXT”, ROM);  
REGION\_ALIAS(“REGION\_RODATA”, ROM2);  
REGION\_ALIAS(“REGION\_DATA”, RAM);  
REGION\_ALIAS(“REGION\_BSS”, RAM);

可以写一个通用的系统初始化程序来把".data"段从ROM或者ROM2在必要时拷贝到RAM中



杂类命令：其他链接器脚本命令  
还有其他一些链接器脚本命令  
ASSERT(exp, message)  
保证exp非零，如果它为零，则链接器退出并设置一个错误号，打印出错消息message。

EXTERN(symbol symbol …)  
强制"symbol"变成输出文件中的未定义符号。做这个可能，比如说，为了从标准库中  
链接外部模块。你可以用"EXTERN"列出许多"symbols"，也可以多次调用"EXTERN"命令  
这个命令和"-u"命令行选项作用相同。

FORCE\_COMMON\_ALLOCATION  
这个命令和"-d"命令行选项作用相同：让链接器为通用符号分类空间，即便用"-r"指  
定了这是一个可重定位输出文件。

INHIBIT\_COMMON\_ALLOCATION  
这个命令和"–no-define-common"命令行选项作用相同。让链接器省略给通用符号赋  
地址，即便这是一个不可重定位得输出文件。

INSERT [AFTER | BEFORE] output\_section  
这个命令典型的用于由"-T"指定脚本，用来增加缺省段。比如，覆盖。它在前面的链  
接器脚本声明的段前或者后插入"output\_section"，也导致了"-T"选项不会覆盖缺省  
链接器脚本。确切地插入点为孤儿段部分。参见"位置计数器"。插入发生在链接器已  
经为输出段做好了映射之后。在插入前，由于"-T"脚本在缺省链接器脚本之前已经被  
解析，"-T"脚本内的声明在缺省链接器脚本表征之前。特别是，输入段的分配会在缺  
省链接脚本之前被安排到"-T"指定的段。以下是一个使用"-T"脚本时，INSERT的可能  
模样：  
SECTIONS  
{  
OVERLAY :  
{  
.ov1 { ov1\*(.text) }  
.ov2 { ov2\*(.text) }  
}  
}  
INSERT AFTER .text

NOCROSSREFS(section section …)  
这个命令可能被用来告诉链接器发射关于任何特定段之间的引用的错误。  
在特定程序中，特别是在使用覆盖的嵌入式系统中，当一个段被装载到内存，另一个  
段还没有时。任何两个段之间的直接引用都会出错。比如，假如在一个段中的代码调  
用了位于另外一个段中的函数就可能出错。  
这个命令有一个输出段名的列表。如果链接器探测到任何段间引用，就会报告错误并  
且以出错状态退出。注意本命令使用的是输出段名，不是输入段名。

OUTPUT\_ARCH(bfdarch)  
指定特定的机器架构输出。参数是BFD库使用的名字中的一个。你可以使用objdump程  
序的"-f"选项来查看一个目标文件的架构。

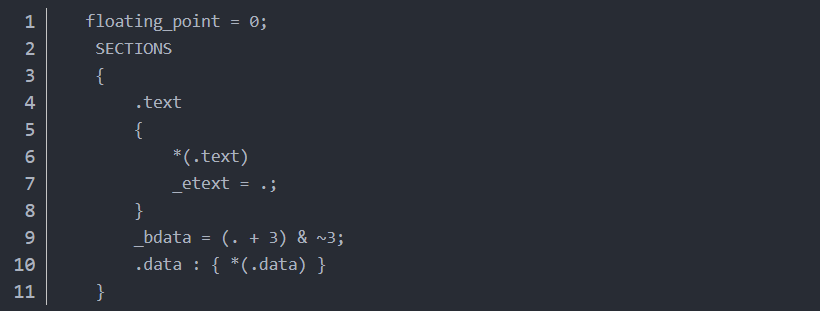
分配，赋值，为符号表赋值

你可以在链接器脚本中给一个符号赋值。这会在全局内定义该符号并且将其以全局形势放  
入到符号表中。

简单赋值  
你可以使用任何C赋值操作符给符号赋值  
symbol = expression;  
symbol += expression;  
symbol -= expression;  
symbol \*= expression;  
symbol /= expression;  
symbol <<= expression;  
symbol >>= expression;  
symbol &= expression;  
symbol |= expression;

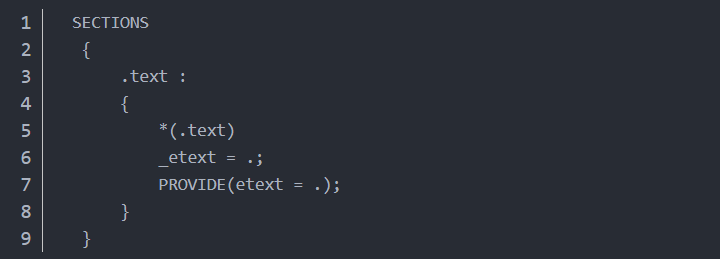
第一种情况以"expression"的值定义了符号"symbol"，在其它的情况下，符号必须已经定  
义过，然后用来调整符号的值。

特殊符号 “.” 表示位置计数器。你只可以在SECTIONS命令中使用它。参见"位置计数器"。  
表达式后的分号是必须的。  
表达式定义如下，参见"表达式"，你可以向命令一样给符号赋值，或者在"SECTIONS"命令  
中像声明一样，或者在"SECTIONS"命令中像段描述一样。  
符号的段会被设置为表达式所在的段，参见"表达式段"  
以下是可能使用符号赋值的三种不同环境的一个例子



在这个例子中，符号"floating\_point"会被定义为0。符号"\_etext"会被定义为紧跟在最  
后一个输入".text"段后面的地址。符号"\_bdata"会被定义为紧跟在".text"输出端后面的  
地址，并且向上以4字节边界对齐。

PROVIDE  
在某些情况下，链接器脚本需要定义一个被引用到但是没有再任何链接的目标文件中定义  
的符号。举例来说，传统的链接器定义了符号"etext"，但是 ANSI C 则要求用户可以使用  
etext作为一个函数名且不能出错。关键字"PROVIDE"被用来定义类似"etext"这样被引用的  
但是没有定义的符号。语法格式是：  
PROVIDE(symbol = expression)  
下面是一个使用"PROVIDE"定义"etext"的例子。



在本例中，如果程序定义了"\_etext"（带下划线的那个），链接器会给出一个多处定义错  
误。假如，程序定义了"etext"（没有下划线），链接器则会悄悄地使用程序的定义。如  
果程序引用了"etext"，但是却没有定义它，那么链接器就是用链接器脚本中的定义。

PROVIDE\_HIDDEN  
类似"PROVIDE"，对于ELF目标，该符号会被隐藏并且不会导出。

源代码参考：如何使用链接器脚本在源代码中定义符号  
在源代码中访问链接器脚本定义的变量是不直观的。特别是链接器脚本符号并不等同于高  
等语言的变量声明，一个符号并不含有值。

在更进一步之前，需要注意的很重要的一点是编译器在把源文件中的名字存到符号表之间  
常常会把它们改成不同的名字。比如说，Fortran编译器通常添加前置或后置下划线，C++  
则会做名字重整（name mangling）。因此在源文件中使用的变量名字和连接器脚本中同一  
个变量的名字可能不同。比如在C中一个链接器脚本变量这样引用：  
extern int foo;  
但是在链接器脚本中它可能被定义为  
\_foo = 1000;  
但是在接下来到例子中，假定不存在名字转换。  
当一个符号在一种高级语言中（比如C）中做了定义。那么发生两件事情，首先编译器在程  
序中为存放该符号的值预留了足够的空间。其次编译器在程序符号表中创建了一个表项来  
存放该符号的地址。也就是符号表包含了保存符号值的内存块的地址。因此下面的C定义，  
在文件范围内：  
int foo = 1000;  
在符号表中创建了一个叫做"foo"的表项。这个表项保存着一个"int"大小内存块的地址，  
在那里初始存放了数值1000。  
当一个程序引用一个符号，编译器产生代码时首先访问符号表找到那个符号的内存块地  
址，然后从那个内存快读取值。所以：  
foo = 1;  
在符号表中查找符号"foo"，找到该符号关连的内存地址，然后向该地址写入值1。但是：  
int \*a = &foo;  
则从符号表中查找符号"foo"，得到它的地址，并且把这个地址拷贝到变量"a"关联的内存  
块中。

相反，链接器脚本声明，则是在符号表中创建一个表项，但是不给它们分配内存。因此它  
们仅仅是一个地址，而没有值。所以比如说在链接器脚本中定义：  
foo = 1000;  
在符号表中创建一个叫做"foo"的表项，该表项保存内存地址位置1000。但是在地址1000处  
没有存放东西。这表示你不能访问一个链接器脚本定义的符号的值，它没有值，所有你能  
做的仅仅是访问链接器脚本定义符号的地址。

因此当你在源代码中使用一个链接器脚本定义的符号时你应当总是使用该符号的地址，并  
且永远不要尝试使用它的值。比如假定你想要从一个叫做".ROM"的内存段中拷贝内容到一  
个叫做".FLASH"的段，并且链接器脚本包含如下声明：  
start\_of\_ROM = .ROM;  
end\_of\_ROM = .ROM + sizeof(.ROM) - 1;  
start\_of\_FLASH = .FLASH;  
那么执行这个拷贝任务的C源代码应该是：  
extern char start\_of\_ROM, end\_of\_ROM, start\_of\_FLASH;  
memcpy(&start\_of\_FLASH, &start\_of\_ROM, &end\_of\_ROM - &start\_of\_ROM);  
注意这里使用了"&"操作符，它们是正确的。

SECTIONS命令（略）

"SECTIONS"命令告诉链接器如何把输入段映射到输出段，以及如何把输出段放到内存中。  
"SECTIONS"命令的格式如下：  
SECTIONS  
{  
sections-command  
sections-command  
…  
}

每个"sections-command"可能是以下一种  
一个 “ENTRY” 命令  
一个符号赋值  
一个输出段描述  
一个覆盖描述

"ENTRY"命令和符号覆盖允许出现在"SECTIONS"内部，为了方便在这些命令中使用位置计数  
器。这也可以使链接器脚本容易理解，因为你可以在一些有意义的输出文件的内存布局位  
置使用这些命令。

输出段描述和覆盖描述在下面。

如果你不在你的链接器脚本中使用"SECTIONS"命令，则链接器会把再输入文件中名字第一  
次碰到的输入段放到同名的输出端中去。如果所有名字在在第一个文件的输入段中都出现  
了，则输出文件中各个段的顺序就和第一个输入文件的顺序相匹配了。第一个段会被安置  
在地址0。

输出段描述

输出段名字

输出段地址

输入段

输入段基础

输入段通配符

输入段通用符号

输入段保留：输入段和垃圾回收

输入段示例

输出段数据

输出段关键字

输出段丢弃

输出段属性

覆盖描述

MEMORY命令（略）

链接器缺省配置允许分配所有可用内存，你可以使用MEMORY命令来重写这个。

PHDRS命令（略）

ELF目标文件格式使用程序头，称为"片断"（segments）。程序头描述了程序如何被装载到  
内存。你可以使用objdump程序的"-p"选项打印它们。

VERSION命令（略）

使用ELF时链接器支持符号版本。符号版本只在使用共享库的时候有用。动态链接器可以  
在它运行一个链接到早期共享库的程序时使用符号版本来选择一个特定版本的函数。

链接器脚本中使用的表达式（略）

链接器脚本语言使用的表达式语法与C语言相同。所有的表达式按照整数计算。所有的表  
达式以相同的大小计算，在主机和目标都是32位时按32位，其它情况按64位计算。

你可以在表达式中使用符号和用表达式设置符号值

链接器为表达式定义了多种特殊目的的内置函数

常量

符号常量

符号

孤儿段

位置计数器  
特殊链接器变量点".“始终包含当前输出位置计数器。由于”.“总是指向一个输出段中的位  
置，它只可能在一个"SECTIONS"命令的表达式中出现。在一个表达式内，”.“符号可以出  
现在任何普通符号能够出现的位置。  
给”."赋值会导致位置计数器移动。这可以被用来在输出段中创建空洞。在一个输出段中，  
位置计数器不可能向后移动，并且除了使用覆盖LMAs外部可能在一个段外向后移动。

操作符

计算

表达式段

内置函数

隐式链接器脚本

如果你向链接器提供了一个它不能认作是目标文件或者库文件的输入文件，那么它会尝试  
把它当作一个链接器脚本文件来读入。如果该文件不能按照链接器脚本文件解析，链接器  
将会报错。

一个隐式链接器脚本文件不会取代默认链接脚本。

典型的，一个隐式链接器脚本只可以包含符号赋值，或者INPUT, GROUP或者VERSION命令。  
在命令行上隐式链接器读入位置之后的输入文件会被影响到，这会影响到档案文件搜索。