使用ld  
\*\*\*\*\*\*\*\*  
本文档介绍GNU连接器ld的2.14版本.  
  
本文档在GNU自由文档许可证下发行.在"GNU自由文档许可证"一章中有关于本许可证的一份拷贝.  
  
概述  
\*\*\*\*\*\*\*\*  
  
'ld'把一定量的目标文件跟档案文件连接起来,并重定位它们的数据,连接符号引用.一般,在编译一个程序  
时,最后一步就是运行'ld'.  
  
'ld'能接受连接命令语言文件,这是一种用AT&T的连接编辑命令语言的超集写成的文件,用来在连接的整个  
过程中提供显式的,全局的控制.  
  
本版本的'ld'使用通用BFD库来操作目标文件.这就允许'ld'读取,合并,写入目标文件时,可以使用各种不同  
的格式,比如,COFF或'a.out'. 不同的格式可以被连接到一起产生一个有效的目标文件.  
  
除了它的灵活性,GNU连接器比其它连接器更有用的地方在于它提供了诊断信息. 许多连接器在碰到一个错误  
的时候立即放弃执行;但'ld'却能够继续执行,以让你发现其他的错误(或者,在某些情况下,得到一个带有错误  
的输出文件)  
  
引用  
\*\*\*\*\*\*\*\*\*\*  
  
GNU连接器'ld'能够处理大量的不同情况,并且跟其他的连接器保持尽可能的兼容.这样,你就拥有更多的选择来  
控制它的行为.  
  
命令行选项  
====================  
  
连接器提供大量的命令行选项,但是,在实际使用中,只有少数被经常使用.比如,'ld'的一个经常的使用场合是在  
一个标准的Unix系统上连接标准的Unix目标文件.在这样的一个系统上,连接文件'hello.o'如下:  
  
ld -o OUTPUT /lib/crt0.o hello.o -lc  
  
这告诉'ld'产生一个叫OUTPUT的文件,作为连接文件'/lib/crt0.o'和'hello.o'和库'libc.a'的结果.'libc.a'  
来自标准的搜索路径.(参阅下文的关于'-l'选项的讨论).  
  
有些命令行选项可以在命令行的任何位置出现.但是,那些带有文件名的选项,比如'-l'或者'-T',会让文件在选  
项出现的位置上被读取. 对于非文件选项,以带不同的参数重复它,不会有进一步的效果,或者覆盖掉前面的相同  
项.那些多次出现时具有特殊含义的选项会在下文的描述中指出.  
  
无参数选项是那些被连接的目标文件和档案文件.它们可能紧随命令行选项,或在它们前面,或者跟它们夹杂在一  
起,但是一个目标文件参数是不会出现在一个选项跟它的参数之间的.  
  
通常,连接器至少引用一个目标文件,但是你可指定其它形式的二进制输入文件,这可以通过'-l','-R'或者脚本  
命令语言来实现.如果没有任何二进制文件被指定,连接器不会产生任何输出,并给出信息:"缺少输入文件."  
  
如果连接器不能识别目标文件的格式,它会假设这些只是连接脚本.以这种方式指定的脚本增加了连接用的主连  
接脚本的内容(主连接脚本即缺省连接脚本或使用'-T'指定的脚本). 这个特性可以允许连接器连接一些文件,  
它们看上去既像目标文件,又像档案文件,但实际上只是定义了一些符号值,或者使用'INPUT'或'GROUP'来载入其  
它的目标文件.需要注意的是,用这种方式指定一个脚本只是增加了主连接脚本的内容;要完全替换掉主连接脚本  
,需要使用'-T'.  
  
对于名称是单个字符的选项,选项参数必须紧跟在选项字母后面,中间不留空,或者也可留有一个空格.  
  
对于名称是多个字符的选项,选项前可以有一个或两个破折号;比如,'-trace-symbol'和`--trace-symbol'是等价  
的. 注意,对于这条规则有一个例外.那些以小写字母'o'开头的多字符选项前面只能是两个破折号,这是为了避免  
跟选项'-o'混淆. 比如'-omagic'把输出文件的名字定为'magic',而'--omagic'在输出文件中设置NMAGIC标志.  
  
多字符选项的参数必须跟选项名间以一个等于号分开,或者以一个空格分开.比如:`--trace-symbol foo'和  
`--trace-symbol=foo'是等价的. 多字符选项的名字唯一缩写符也是可以被接受的.  
  
注意,如果连接器通过被编译器驱动来间接引用(比如gcc), 那所有的连接器命令行选项前必须加上前缀'-Wl'  
(或者能被特定编译器驱动接受的其他前缀),就像下面这样:  
  
gcc -Wl,--startgroup foo.o bar.o -Wl,--endgroup  
  
这很重要,因为否则的话,编译器驱动程序会默认丢掉这些连接选项,产生一个错误的连接.  
  
下面是关于被GNU连接器接受的常用命令行开关的一个列表:  
  
`-aKEYWORD'  
这个选项在HP/UX兼容系统上被支持. 参数KEYWORD必须是下面字符串中的一个:`archive',  
`shared', or `default'. `-aarchive'在功能上跟`-Bstatic'相同,而另外两个关键字功能上跟  
`-Bdynamic'相同. 这个选项可被多次使用.  
  
`-AARCHITECTURE'  
`--architecture=ARCHITECTURE'  
在最近发行版本的'ld'中,这个选项只在Intel 960系列架构上有用. 在那种'ld'配置中,参数  
ARCHITECTURE确定960系列的某一特定架构,启用某些安全措施,并修改档案库的搜索路径.  
  
将来的'ld'发行版可能为其它架构系列支持相似的功能.  
  
`-b INPUT-formAT'  
`--format=INPUT-formAT'  
'ld'可以被配置为支持多于一种的目标文件.如果你的'ld'以这种方式被配置,你可以使用'-b'选  
项为输入目标文件指定二进制格式. 就算'ld'被配置为支持可选目标格式,你不必经常指定这一项,  
因为'ld'被配置为在每一台机子上把最常用的格式作为默认输入格式. INPUT-formAT是一个字符串,  
  
你可能在连接一个不常用的二进制格式文件时需要这个参数.你也可使用'-b'来显式切换格式(在连接  
不同格式的目标文件时),方法是在每一组特定格式的目标前使用'-b INPUT-formAT'.  
  
缺省的格式是从环境变量'GNUTARGET'中得到的.你也可以从一个脚本中定义输入格式,使用的命令是  
'TARGET'.  
  
`-c MRI-COMMANDFILE'  
`--mri-script=MRI-COMMANDFILE'  
为了跟MRI生产的连接器兼容,'ld'接受另一种用受限命令语言写成的脚本文件,通过选项'-c'引入MRI  
脚本文件;使用'-T'选项是运行用普通'ld'脚本语言写的连接脚本.如果MRI-CMDFILE不存在,'ld'在'-L'  
指定的目录中寻找.  
  
`-d'  
`-dc'  
`-dp'  
这三个选项是等价的; 多字符形式是为了跟其他连接器兼容才被支持的.它们给普通符号分配空间,即  
使一个重定位输出文件已经被指定(通过'-r'). 脚本命令`FORCE\_COMMON\_ALLOCATION'具有同样的效果.  
  
`-e ENTRY'  
`--entry=ENTRY'  
使用符号ENTRY作为你的程序的开始执行点,而不是使用缺省的进入点.如果没有叫做ENTRY的符号,连接器  
会企图把ENTRY作为一个数字进行分析,并使用它作为入口地址(数字会被解释为10进制的;你可以使用前  
导的'0x'强制为16进制,或'0'作为8进制.)  
  
`-E'  
`--export-dynamic'  
当创建一个动态连接的可执行程序时, 把所有的符号加到动态符号表中.动态符号表是一个符号集,这  
些符号对于运行时的动态对象是可见的.  
  
如果你不使用这个选项,动态符号表中就会只含有那些连接进来的动态对象中用到的符号  
  
如果你使用'dlopen'来载入动态对象,它需要引用程序中的符号,那你可能需要在连接程序时用到这个  
选项.  
  
你也可以使用版本脚本来控制哪些符号应当被加到动态符号表中.  
  
`-EB'  
连接big-endian对象. 这会影响缺省输出格式.  
  
`-EL'  
连接little-endian对象. 这会影响缺省输出格式.  
  
`-g'  
忽略. 为了跟其它工具兼容而提供.  
  
`-i'  
执行一个增量连接(跟'-r'等同)  
  
`-init NAME'  
当创建一个ELF可执行文件或共享对象时,当可执行文件或共享对象被加载时,调用NAME, 这是通过把  
DT\_INIT设置成函数的地址实现的. 缺省情况下,连接器使用'\_init'作为调用的函数.  
  
`-lARCHIVE'  
`--library=ARCHIVE'  
增加一个档案文件ARCHIVE到连接的文件列表中.这个选项可以被多次使用. 'ld'会为每一个指定的  
ARCHIVE搜索它的路径列表,寻找`libARCHIVE.a'  
  
对于支持共享库的系统, 'ld'可能还会搜索扩展名不是'.a'库.特别的,在ELF和SunOS系统上,'ld'会  
在搜索带有'.a'扩展名的库前搜索带'.so'扩展名的库.  
  
`-M'  
`--print-map'  
打印一个连接位图到标准输出.一个连接位图提供的关于连接的信息有如下一些:  
  
\* 目标文件和符号被映射到内存的哪些地方.  
  
\* 普通符号如何被分配空间.  
  
\* 所有被连接进来的档案文件,还有导致档案文件被包含进来的那个符号.  
  
`-n'  
`--nmagic'  
关闭所有节的页对齐,如果可能,把输出格式标识为'NMAGIC'.  
  
`-N'  
`--omagic'  
把text和data节设置为可读写.同时,取消数据节的页对齐,同时,取消对共享库的连接.如果输出格式  
支持Unix风格的magic number, 把输出标志为'OMAGIC'.  
  
`--no-omagic'  
这个选项执行的操作大部分正好跟'-N'相反.它设置text节只读,强制data节页对齐. 但是,这个选项  
并不开启连接共享库的功能. 使用'-Bdynamic'开启这个功能.  
  
`-o OUTPUT'  
`--output=OUTPUT'  
使用OUTPUT作为'ld'产生的程序的名字;如果这个选项没有指定,缺省的输出文件名是'a.out'.脚本命  
令'OUTPUT'也可以被用来指定输出文件的文件名.  
  
`-O LEVEL'  
如果LEVEL是一个比0大的数值, 'ld'优化输出.这可能会明显多占用时间,所以只有在生成最后的文件  
时使用.  
  
`-q'  
`--emit-relocs'  
把重定位节和内容留在完全连接后的可执行文件中. 连接分析和优化工具可能需要这些信息用来进行  
正确的修改与执行. 这在大的可执行文件中有用.  
  
这个选项目前只支持ELF平台.  
`-r'  
`--relocateable'  
产生可重定位的输出, 比如,产生一个输出文件它可再次作为'ld'的输入.这经常被叫做"部分连接".  
作为一个副作用,在支持标准Unix魔数的环境中,这个选项会把输出文件的魔数设置为'OMAGIC'. 如  
果这个选项没有被指定,一个绝对文件就会被产生.当连接C++程序时,这个选项就不会解析构造函数的  
引用;要解析,必须使用'-Ur'  
  
如果输入文件跟输出文件的格式不同,只有在输入文件不含有重定位信息的时候部分连接才被支持.输  
出格式不同的时候会有更多的限制.比如,有些'a.out'的格式在输入文件是其他格式的时候完全不支  
持部分连接.  
  
这个选项跟'-i'等效.  
  
`-R FILENAME'  
`--just-symbols=FILENAME'  
从FILENAME中读取符号名跟它们的值,但不重位这个文件,或者根本不把它包含在输出文件中.这就允  
许你的输出文件引用其它程序中定义的绝对内存地址.你可以多次使用这个选项.  
  
为了跟其他ELF连接器兼容,如果'-R'选项后面跟有一个目录名,而不是一个文件名,它会被处理成  
'-rpath'选项.  
  
`-s'  
`--strip-all'  
忽略输出文件中所有的符号信息.  
  
`-S'  
`--strip-debug'  
忽略输出文件中所有的调试符号信息(但不是所有符号).  
  
`-t'  
`--trace'  
打印'ld'处理的所有输入文件的名字.  
  
`-T SCRIPTFILE'  
`--script=SCRIPTFILE'  
把SCRIPTFILE作为连接脚本使用. 这个脚本会替代'ld'的缺省连接脚本(而不是增加它的内容),所以  
命令文件必须指定所有需要的东西以精确描述输出文件. 如果SCRIPTFILE在当前目录下不存在,'ld'  
会在'-L'选项指定的所有目录下去寻找.多个'-T'选项会使内容累积.  
  
`-u SYMBOL'  
`--undefined=SYMBOL'  
强制SYMBOL在输出文件中作为一个无定义的符号被输入.这样做会有一些效果,比如,会引发从标准库  
中连接更多的模块. '-u'可以以不同的参数反复使用,以输入多个无定义的符号.这个选项跟连接脚  
本命令中的'EXTERN'是等效的.  
  
`-Ur'  
对于不是C++的程序,这个选项跟'-r'是等效的: 它产生可重定位的输出,比如,一个输出文件它可以再  
次作为'ld'的输入. 当连接C++程序时,'-Ur'解析构造函数的引用,跟'-r'不同. 但如果在一些用'-Ur'  
连接过的文件上再次使用'-Ur',它不会工作,因为一旦构造函数表被建立,它不能被添加内容.请只在  
最后一遍连接的时候使用'-Ur', 对其它的,只使用'-r'.  
  
`--unique[=SECTION]'  
对于所有匹配SECTION的输入节,在输出文件中都各自创建单独的节,或者,如果可选的通配符SECTION  
参数丢失了,为每一个孤儿输入节创建一个输出节. 一个孤儿节是一个连接脚本中没有指定的节.你  
可以在命令行上多次使用这个选项; 它阻止对同名输入节的合并,在连接脚本中重载输出节分配.  
  
`-v'  
`--version'  
`-V'  
显示'ld'的版本. '-V'选项同时会列出支持的模拟器.  
  
`-x'  
`--discard-all'  
删除所有的本地符号.  
  
`-X'  
`--discard-locals'  
删除所有的临时本地符号.对于大多数目标平台,就是所有的名字以'L'开头的本地符号.  
  
`-y SYMBOL'  
`--trace-symbol=SYMBOL'  
打印出所有SYMBOL出现的被连接文件的名字. 这个选项可以被多次使用. 在很多系统中,这在预先确定底  
线时很有必要.  
  
当你拥有一个未定义的符号,但不知道这个引用出自哪里的时候,这个选项很有用.  
  
`-Y PATH'  
为缺省的库搜索路径增加一条路径.这个选项是为了跟Solaris兼容.  
  
`-z KEYWORD'  
能被识别的关键字包括'initfirst', 'interpose', 'loadfltr',`nodefaultlib', `nodelete',  
`nodlopen', `nodump', `now', `origin',`combreloc', `nocombreloc' and `nocopyreloc'. 为了跟  
Solaris兼容,所有其它的关键字都被忽略. 'initfirst'标志一个对象,使它在运行时,在所有其他对象之  
前被初始化. 'interpose'标志一个对象,使它的符号表放在所有其他符号之前,作为主要的执行者.  
'loadfltr'标志一个对象, 使它的过滤器在运行时立即被处理.'nodefaultlib'标志一个对象,使在搜索  
本对象所依赖的库时,忽略所有缺省库搜索路径. 'nodelete'标志一个对象,使它在运行时不会被从内存  
中删除.'nodlopen'标志一个对象,使这个对象不可以通过'dlopen'载入.'nodump'标志一个对象,使它不能  
被'dldump'转储. 'now'标志一个对象,使它成为非懒惰运行时绑定对象. 'origin'标志一些可能含有  
$ORIGIN的对象,'defs'不允许无定义符号. 'muldefs'允许重定义. 'comberloc'组合多个重定位节,重新  
排布它们,让动态符号可见. 'nocomberloc'使多个重定位节组合无效. 'nocopyreloc'使重定位拷贝后的  
结果无效.  
  
`-( ARCHIVES -)'  
`--start-group ARCHIVES --end-group'  
ARCHIVES应当是一个关于档案文件的列表. 它们可以是显式的文件名,或者'-l'选项.  
  
这些指定的档案文件会被多遍搜索,直到没有新的无定义引用被创建. 通常,一个档案文件只会被搜索一  
次. 但如果这个档案文件中的一个符号需要被用来解析一个档案中的目标引用到的无定义的符号,而这个  
符号在命令行上的后面某个档案文件中出现, 连接器不能解析这个引用. 把这些档案文件分组后,它们都  
可被反复搜索直到所有可能的引用都被解析了为止.  
  
使用这个选项有一个很大的运行开销. 只有在无法避免在多个档案文件中使用循环引用时才用它.  
  
`--accept-unknown-input-arch'  
`--no-accept-unknown-input-arch'  
告诉连接器接受那些架构不能被识别的输入文件. 但前提假设是用户知道他们在做什么,并且是故意要连  
接这些未知的输入文件. 在版本2.14之前,这个是连接器的缺省行为. 从版本2.14以后的,缺省行为是拒  
绝这类输入文件, 所以`--accept-unknown-input-arch'选项被用来恢复旧的行为.  
  
`-assert KEYWORD'  
这个选项被忽略,只是用来跟SunOS保持兼容.  
  
`-Bdynamic'  
`-dy'  
`-call\_shared'  
连接动态链接库. 这个仅仅在支持共享库的平台上有用.在这些平台上,这个选项通常是默认行为. 这个选  
项的不同形式是为了跟不同的系统保持兼容. 你可以在命令行上多次使用这个选项:它影响紧随其后的'-l'  
选项的库搜索.  
  
`-Bgroup'  
在动态节的'DT\_FLAGS\_1'入口上设置'DF\_1\_GROUP'标志.这会让运行时连接器在处理在这个对象和它的相  
关部分搜索时只在组中. '--no-undefined'是隐式的. 这个选项只在支持共享库的ELF平台上有用.  
  
`-Bstatic'  
`-dn'  
`-non\_shared'  
`-static'  
不连接共享库. 这个仅仅在支持共享库的平台上有用. 这个选项的不同形式是为了跟不同的系统保持兼  
容. 你可以在命令行上多次使用这个选项:它影响紧随其后的'-l'选项的库搜索.  
  
`-Bsymbolic'  
当创建一个共享库时, 把对全局符号的引用绑定到共享库中的定义(如果有), 通常, 一个连接共享库的程  
序重载共享库中的定义是可能的. 这个选项只在支持共享库的ELF平台上有用.  
  
`--check-sections'  
`--no-check-sections'  
让连接器在节地址被分配后不要去检查节地址是否重叠.通常,连接器会执行这种检查,如果它发现了任何  
重叠,它会产生相应的错误信息. 连接器知道也允许节的重叠. 缺省的行为可以使用命令行开关  
`--check-sections'来恢复.  
  
`--cref'  
输出一个交叉引用表. 如果一个连接器位图文件被产生, 交叉引用表被打印到位图文件. 否则, 它被打印  
到标准输出.  
  
表的格式相当的简单, 所以,如果需要,可以通过一个脚本很轻易地处理它. 符号是以名字被打印输出,存  
储. 对于每一个符号,给出一个文件名列表. 如果符号被定义了, 列出的第一个文件是符号定义的所在.  
接下来的文件包含符号的引用.  
  
`--no-define-common'  
这个选项限制对普通符号的地址分配. 脚本命令`INHIBIT\_COMMON\_ALLOCATION'具有同等的效果.  
  
`--no-define-common'选项允许从输出文件的类型选择中确定对普通符号的地址分配; 否则, 一个非重定  
位输出类型强制为普通符号分配地址. 使用'--no-define-common'允许那些从共享库中引用的普通符号只  
在主程序中被分配地址. 这会消除在共享库中的无用的副本的空间, 同时,也防止了在有多个指定了搜索  
路径的动态模块在进行运行时符号解析时引起的混乱.  
  
`--defsym SYMBOL=EXPRESSION'  
在输出文件中建立一个全局符号,这个符号拥有一个EXPRESSION指定的绝对地址. 你可以多次使用这个选  
项定义多个符号. EXPRESSION支持一个受限形式的算术运算:你可以给出一个十六进制常数或者一个已存  
在符号的名字,或者使用'+'和'-'来加或减十六进制常数或符号. 如果你需要更多的表达式,可以考虑在脚  
本中使用连接器命令语言, 注意在SYMBOL,=和EXPRESSION之间不允许有空格.  
  
`--demangle[=style]'  
`--no-demangle'  
这些选项控制是否在错误信息和其它的输出中重组符号名. 当连接器被告知要重组, 它会试图把符号名以  
一种可读的形式的展现: 如果符号被以目标文件格式使用,它剥去前导的下划线,并且把C++形式的符号名  
转换成用户可读的名字. 不同的编译器有不同的重组形式. 可选的重组形式参数可以被用来为你的编译器  
选择一个相应的重组形式. 连接器会以缺省形式重组直至环境变量`COLLECT\_NO\_DEMANGLE'被设置. 这些  
选项可以被用来重载缺省的设置.  
  
`--dynamic-linker FILE'  
设置动态连接器的名字. 这个只在产生动态连接的ELF可执行文件时有效. 缺省的动态连接器通常是正确  
的; 除非你知道你在干什么,不要使用这个选项.  
  
`--embedded-relocs'  
这个选项只在连接MIPS嵌入式PIC代码时有效, 这些代码必须是由GNU的编译器跟汇编器通过-membedded-pic  
选项生成的. 它导致连接器产生一个表,这个表被用来在运行时重定位所有的被静态初始化为指针值的数  
据.  
  
`--fatal-warnings'  
把所有的警告视为错误.  
  
`--force-exe-suffix'  
确保输出文件有一个.exe后缀.  
  
如果一个被成功完整连接的输出文件不带有一个'.exe'或'.dll'后缀, 这个选项确保连接器把输出文件  
拷贝成带有'.exe'后缀的同名文件. 这个选项在使用微软系统来编译未经修改的Unix的makefile时很有  
用, 因为有些版本的windows不会运行一个不带有'.exe'后缀的映像.  
  
`--no-gc-sections'  
`--gc-sections'  
允许对未使用的输入节的碎片收集. 在不支持这个选项的平台上,被忽略. 这个选项不能跟 '-r'选项共存  
也不能被用来进行动态连接. 缺省行为可以用`--no-gc-sections'进行恢复.  
  
`--help'  
在标准输出上打印一个命令行选项概要,然后退出.  
  
`--target-help'  
打印一个所有目标平台相关的选项的概要,然后退出.  
  
`-Map MAPFILE'  
打印一个连接位图到文件MAPFILE中. 参阅上面关于'-M'选项的描述.  
  
`--no-keep-memory'  
'ld'通常会以速度优先于内存使用的方式优化程序,这是通过把输入文件的符号表放在内存缓冲中实现的,  
这个选项告诉'ld'以内存使用优先来优化, 尽可能的减小符号表的重读. 这在'ld'在连接一个大文件时  
超出内存限制时有用.  
  
`--no-undefined'  
`-z defs'  
通常,当创建一个非符号共享库时, 无定义的符号允许出现,并留待运行时连接器去解决. 这个选项关闭这  
样的无定义符号的使用. 开关`--no-allow-shlib-undefined'控制共享对象被连接进共享库时的行为.  
  
`--allow-multiple-definition'  
`-z muldefs'  
通常,当一个符号被定义多次时, 连接器会报告一个致命错误. 这些选项允许重定义并且第一个定义被使  
用  
  
  
`--allow-shlib-undefined'  
`--no-allow-shlib-undefined'  
允许(缺省)或不允许无定义符号存在于共享对象中. 这个开关的设置会重载'--no-undefined',这里只关  
注共享对象. 这样,如果'--no-undefined'被设置,但'--no-allow-shlib-undefined'未被设置, 连锁反应  
是存在于规则对象文件中的无定义的符号会引起一个错误,但是在共享对象中的未定义的符号会被忽略.  
  
把`--allow-shlib-undefined'设置为缺省的原因是在连接时指定的共享对象并不一定是载入时可载入的  
那个,所以,符号可能要到载入时间才被解析.  
  
`--no-undefined-version'  
通常当一个符号有一个未定义的版本时,连接器会忽略它. 这个选项不允许符号有未定义的版本,并且碰  
到这种情况,会报告一个严重错误.  
  
`--no-warn-mismatch'  
通常, 如果你因为一些原因,企图把一些不匹配的输入文件连接起来的时候,'ld'会给出一个错误,可能这  
些文件是因为由不同的处理器编译. 这个选项告诉'ld'应当对这样的错误默认允许. 这个选项必须小心  
使用.  
  
`--no-whole-archive'  
为后面的档案文件关闭'--whole-archive'选项的影响.  
  
`--noinhibit-exec'  
当一个可执行文件还可以使用时,就保留它. 通常,连接器如果在连接过程中遇到了错误,就不会产生输出  
文件;当它遇上错误时,它会退出而不写输出文件.  
  
`-nostdlib'  
仅搜索那些在命令行上显式指定的库路径. 在连接脚本中(包含在命令行上指定的连接脚本)指定的库路  
径都被忽略.  
  
`--oformat OUTPUT-formAT'  
'ld'可以被配置为支持多于一种的目标文件. 如果你的'ld'以这种方式被配置,你可以使用'--oformat'  
选项来指定输出目标文件的二进制格式.就算'ld'被配置为支持多种目标格式,你也不必指定这个项,因  
为'ld'应当被配置为把最常用的输出格式作为默认格式. OUTPUT-formAT是一个文本串,是被BFD库支持  
的一个特定格式的名字.脚本命令'OUTPUT\_formAT'也可以指定输出格式,但这个选项可以覆盖它.  
  
`-qmagic'  
这个选项被忽略,只是为了跟Linux保持兼容.  
  
`-Qy'  
这个选项被忽略,只是为了跟SVR4保持兼容.  
  
`--relax'  
一个机器相关的选项. 只有在少数平台上,这个选项被支持.  
  
在某些平台上,'--relax'选项在连接器解析程序中的地址时执行可能的全局优化, 比如松散地址模式和在输出文件  
中合成新的指令.  
  
在某些平台上,连接时全局优化会进行符号调试导致程序不能运行.  
  
在不支持这个选项的平台上,'--relax'被接受,但被忽略.  
  
`--retain-symbols-file FILENAME'  
只保留在FILENAME中列出的那些符号,丢弃所有其他的. FILENAME是一个简单地平坦模式文件, 一个符号占一行.  
这个选项在那些会逐步积累起一个大的全局符号表的系统中(比如 VxWorks)会很有用,它能有效地节约内存空间.  
  
'--retain-symbols-file'不丢弃未定义的符号,和需要重定位的符号.  
  
你可能在命令行上只指定'--retain-symbol-file'一次, 它覆盖'-s'和'-S'的功能.  
  
`-rpath DIR'  
为运行时库的搜索路径增加一个目录. 这个在连接带有共享库的ELF可执行文件时有用. '-rpath'的所有参数会被  
连接起来传递给运行时连接器, 运行时连接器在运行时用它们定位共享对象. '-rpath'选项在定位那些在连接参数  
指定的共享对象需要的共享对象时也很有用; 参阅关于'-rpath-link'选项的描述, 如果在连接一个ELF可执行文件  
时不使用'-rpath'选项,那些环境变量'LD\_RUN\_PATH'选项就会被使用.  
  
'-rptah'选项也可以使用在SunOS上. 缺省地,在SunOS上,连接器会从所有的'-L'选项中形成一个运行时搜索路径.  
如果使用了'-rpath'选项, 那运行时搜索路径就只从'-rpath'选项中得到, 忽略'-L'选项. 这在使用GCC时非常有  
用, 它会用上很多的'-L'选项,而这些路径很可能就是NFS挂上去的文件系统中.  
  
为了同ELF的连接器兼容, 如果'-R'选面后面跟有一个目录名, 而不是一个文件名,那它也会被处理成'-rpath'选  
项.  
  
`-rpath-link DIR'  
当在SunOS上使用ELF时,一个共享库可能会用到另一个共享库. 当'ld -share'把一个共享库作为一个输入文件连接  
时就有可能发生这种情况.  
  
当一个连接器在作非共享,不可重定位连接时,如果遇上这种依赖情况,它会自动定位需要的共享库,然后把它包含在  
连接中, 如果在这种情况中,它没有被显式包含, 那'-rpath-link'选项指定优先搜索的一组路径名.  
  
这个选项必须小心使用,因为它会覆盖那些可能已经被编译进共享库中的搜索路径. 在这种情况下,它就有可能使用  
一个非内部的不同的搜索路径.  
  
连接器使用下面的搜索路径来定位需要的共享库:  
  
1. 所有由'-rpath-link'选项指定的搜索路径.  
  
2. 所有由'-rpath'指定的搜索路径. '-rpath'跟'-rpath\_link'的不同之处在于,由'-rpath'指定的路径被包含在可  
执行文件中,并在运行时使用, 而'-rpath-link'选项仅仅在连接时起作用. 它只用于本地连接器.  
  
3. 在一个ELF系统中, 如果'-rpath'和'rpath-link'选项没有被使用, 会搜索环境变量'LD\_RUN\_PATH'的内容.它也只  
对本地连接器起作用.  
  
4. 在SunOS上, '-rpath'选项不使用, 只搜索所有由'-L'指定的目录.  
  
5. 对于一个本地连接器,环境变量'LD\_LIBRARY\_PATH'的内容被搜索.  
  
6. 对于一个本地ELF连接器,共享库中的`DT\_RUNPATH'和`DT\_RPATH'操作符会被需要它的共享库搜索. 如果'DT\_RUNPATH'  
存在了, 那'DT\_RPATH'就会被忽略.  
  
7. 缺省目录, 常规的,如'/lib'和'/usr/lib'.  
  
8. 对于ELF系统上的本地连接器, 如果文件'/etc/ld.so.conf'存在, 这个文件中有的目录会被搜索.  
  
如果需要的共享库没有被找到, 那连接器会发出一条警告信息,并继续执行连接.  
  
`-shared'  
`-Bshareable'  
创建一个共享库. 这个选项只在ELF, XCOFF和SunOS平台上有用。 在SunOS上，如果'-e'选项没有被使用，并在连接  
中有未定义的符号，连接器会自动创建一个共享库，  
  
`--sort-common'  
这个选项告诉'ld'当它把普通符号放到相应的输出节中时按大小进行排序。排在最前面的是所有的一字节符号，然  
后是所有的二字节，然后是所有的四字节， 然后是其它的。 这是为了避免因为对齐约束而在符号间产生的断裂  
  
`--split-by-file [SIZE]'  
跟'--split-by-reloc'相似，但在SIZE达到时，为每一个输入文件创建一个新的输出节。如果没有给出，SIZE缺省  
地设置为1  
  
`--split-by-reloc [COUNT]'  
试图在输出文件中创建节外的节，这样就没有单个的输出节含有多于COUNT个重定位符。这在产生巨大的用于COFF格  
式的实时内核的可重定位文件时非常有用；因为COFF不能在一个节中表示多于65535个重定位。 注意，这在不支持  
专有节的目标文件格式中会失败，连接器不会把单个输入节分割进行重分配， 所以，如果单个输入节含有多于COUNT  
个重定位符， 那一个输出节会含有同样多的可重定位符。COUNT缺省被设为32768.  
  
`--stats'  
计算并显示关于连接器操作的统计信息, 比如执行时间,内存占用等.  
  
`--traditional-format'  
对于某些目标平台, 'ld'的输出会跟某些面有的连接器的输出有所不同. 这个开关要求'ld'使用传  
统的格式.  
  
比如, 在SunOS上, 'ld'会把符号串表中的两上完全相同的入口合并起来. 这可以把一个带有调试信息  
的输出文件的大小减小百发之三十. 不幸地是, SunOS的'dbx'程序不能读取这个输出的程序(gdb就没  
有问题).'--trafitinal-format'开关告诉'ld'不要把相同的入口合并起来.  
  
`--section-start SECTIONNAME=ORG'  
通过指定ORG, 指定节在输出文件中的绝对地址. 你可以多次使用这个选项来定位多个节. ORG必须是  
一个十六进制整数; 为了跟基他的连接器兼容,你可以忽略前导'0x'. 注意,在SECTIONNAME,等号,ORG  
之间不允许有空格出现.  
  
`-Tbss ORG'  
`-Tdata ORG'  
`-Ttext ORG'  
跟-section-start同义, 不过把SECTIONNAME替换为'.bss', '.data'或'.text'.  
  
`--dll-verbose'  
`--verbose'  
显示'ld'的版本号,并列出支持的连接器模拟. 显示哪些输入文件能被打开,而哪些不能. 显示连接器  
使用的连接脚本.  
  
`--version-script=VERSION-SCRIPTFILE'  
指定连接器的脚本的版本名. 这个常在创建一个需要指定附加的关于版本层次的信息的共享库时使用,  
这个选项只有支持共享库的ELF平台上有效.  
  
`--warn-common'  
当一个普通符号跟另一个普通符号或会号定义合并起来时,警告. 类Unix连接器允许这个选项,有时比  
较实用, 但是在其他的操作系统上的连接器不允许这个. 这个选项可以让你在合并全局符号时发现某  
些潜在的问题. 不幸的是,有些C库使用这项特性,所以你可能会像在你的程序中一样,在库中得到一些  
警告信息.  
  
这里给出三种类型的全局符号的解释(用C语言):  
  
`int i = 1;'  
一个定义, 它会存在于输出文件中的已初始化数据节.  
  
`extern int i;'  
一个未定义符号,它不占用空间. 必须在另外某一处对它有一个定义,或一个普通符号  
  
`int i;'  
一个普通符号.如果对于一个变量只有(一个或多个)普通符号, 它进入输出文件的未初始化数据域. 连  
接器会把同一变量的多个普通符号合并成一个单一的符号. 如果他们有不同的大小, 它采用最大的一  
个. 如果是对同一变量的定义,连接器把一个普通符号转化为一个声明.  
  
'--warn-common'选项可以产生五种类型的警告. 每种警告由两行组成: 第一行描述遇到的符号, 第二  
行描述遇到的前一个具有相同名字的符号. 一个或两个都可能成为普通符号.  
  
1. 把一个普通符号转化为一个引用, 因为这个符号已经有一个定义了.  
FILE(SECTION): warning: common of `SYMBOL'  
overridden by definition  
FILE(SECTION): warning: defined here  
  
2. 把一个普通符号转化为一个引用,因为遇到了第二个关于符号的定义. 这跟前一种情况相同,除了符  
号遇到的顺序相反.  
FILE(SECTION): warning: definition of `SYMBOL'  
overriding common  
FILE(SECTION): warning: common is here  
  
3. 把一个普通符号跟前一个相同大小的普通符号合并.  
FILE(SECTION): warning: multiple common  
of `SYMBOL'  
FILE(SECTION): warning: previous common is here  
  
4. 把一个普通符号跟前一个更大的普通符号合并.  
FILE(SECTION): warning: common of `SYMBOL'  
overridden by larger common  
FILE(SECTION): warning: larger common is here  
  
5. 把一个普通符号跟前一个更小的普通符号合并. 这跟前一种情况相同, 除了遇到的符号的顺序不同.  
FILE(SECTION): warning: common of `SYMBOL'  
overriding smaller common  
FILE(SECTION): warning: smaller common is here  
  
`--warn-constructors'  
如果有全局结构体被使用到了,警告. 这只对很少的一些目标文件格式有用. 对于COFF或ELF格式, 连  
接器不同探测到全局结构体的使用.  
  
`--warn-multiple-gp'  
如果在输出文件中,需要多个全局指针值,警告. 这只对特定的处理器有意义, 比如Alpha. 特别的,有  
些处理器在特定的节中放入很大的常数值. 一个特殊的寄存器(全局指针)指向这个节的中间部分, 所  
以通过一个基地址寄存器相关的地址模式,这个常数可以很容易地被载入. 因为这个基寄存器相关模式  
的偏移值是固定的而且很小(比如,16位), 这会限制常量池的最大尺寸. 所以,一个很大的问题是,为了  
能够定位所有可能的常数,经常需要使用多个全局指针值. 这个选项在这种情况发生时产生一条警告.  
  
`--warn-once'  
对于每一个未定义符号只警告一次, 而不是在每一个用到它的模块中警告一次.  
  
`--warn-section-align'  
如果输出节的地址因为对齐被改变了,警告. 通常, 对齐会被输入节设置. 如果'SECTION'命令没有指  
定节的起始地址, 地址就会被隐式改变.  
  
`--whole-archive'  
对于每一个在命令行中'--whole-archive'选项后面出现的档案文件, 在连接中包含档案文件中的所有  
目标文件, 而不是为需要的目标文件搜索档案文件. 这在把一个档案文件转化为一个共享库时使用, 把  
所有的目标放到最终的共享库中. 这个选项可以被多次使用.  
  
在GCC中使用这个选项需要注意两点: 首先,GCC不知道这个选项, 所以,你必须使用'-Wl, -whole-archive'.  
第二, 不要忘了在你的档案文件列表的后面使用'-Wl, -no-whole-archive',因为GCC会把它自己的档  
案列表加到你的连接后面, 而这可能并不是你所预期的.  
  
`--wrap SYMBOL'  
对SYMBOL符号使用包装函数. 任何未定义的对SYMBOL符号的引用会被解析成'\_wrap\_SYMBOL'. 而任何  
未定义的对'\_real\_SYMBOL'的引用会被解析成SYMBOL.  
  
这可以用来为系统函灵敏提供一个包装. 包装函灵敏应当被叫做'\_\_wrap\_SYMBOL'. 如果需要调用这个  
函数, 那就应该调用'\_\_real\_SYMBOL'  
  
这里是一个没什么实用价值的例子:  
  
void \*  
\_\_wrap\_malloc (int c)  
{  
printf ("malloc called with %ld\n", c);  
return \_\_real\_malloc ©;  
}  
  
如果你使用'--wrap malloc'把这节代码跟其他的代码连接, 那所有的对'malloc'的调用都会调用  
'\_\_wrap\_malloc'函数. 而在'\_\_wrap\_malloc'中的'\_\_real\_malloc'会调用真正的'malloc'函数.  
  
你有可能也希望提供一个'\_\_real\_malloc'函数, 这样,不带有'--wrap'的连接器也会成功连接.如果  
你这样做了, 你不能把'\_\_real\_malloc'的定义跟'\_\_wrap\_malloc'放到同一个文件中;如果放在一起  
汇编器会在连接器之前把调用解析成真正的'malloc'.  
  
`--enable-new-dtags'  
`--disable-new-dtags'  
连接器可以在ELF中创建一个新的动态标签. 但是旧的ELF系统可能不理解这个. 如果你指定了  
'--enable-new-dtags',动态标签会按需要被创建. 如果你指定了'--disable-new-dtags',那不会有  
新的动态标签被创建. 缺省地,新的动态标签不会被创建. 注意这些选项只在ELF系统中有效.  
  
i386 PE平台的特定选项.  
-----------------------------------  
  
i386 PE连接器支持'-shared'选项, 它使输出文件为一个动态链接库(DLL),而不是一个普通的可执行文件. 在  
使用这个选项的时候,你应当为输出文件取名'\*.dll',另外, 连接器完全支持标准的'\*.def'文件, 这类文件可  
以在连接器命令行上象一个目标文件一样被指定(实际上, 它应当被放在它从中导出符号的那个档案文件前面,  
以保证它们象一个普通的目标文件一样被连接进去.)  
  
除了对所有平台通用的那些选项外,i386 PE连接器支持一些只对i386平台专有的命令行选面. 带有值的选项应  
当用空格或等号把它跟值分隔开.  
  
`--add-stdcall-alias'  
如果给出这个选项, 带有标准调用后缀(@NN)的符号会被剥掉后缀后导出.  
  
`--base-file FILE'  
使用FILE作为文件名,该文件是存放用'dlltool'产生 DLL文件时所需的所有重定位符的基地址的.(这  
个选面是i386 PE平台所专有的]  
  
`--dll'  
创建一个DLL文件而不是一个常规可执行文件. 你可能在一个给出的'.def'文件中使用'-shared'或指  
定'LIBRARY'.  
  
`--enable-stdcall-fixup'  
`--disable-stdcall-fixup'  
如果连接器发现有符号不能解析, 它会试图进行'失真连接',即寻找另一个定义的符号,它们只是在  
符号名的格式上不同(cdecl vs stdcall),并把符号解析为找到的这个符号. 比如, 一个未定义的符  
号'\_foo'可能被连接到函数'\_foo@12', 或者一个未定义的符号'\_bar@16'可能被连接到函数'\_bar'.  
如果连接器这么做了, 它会打印出一条警告信息, 因为在正常情况下,这会连接失败, 但有时,由第三  
方库产生的导入库可能需要这个特性. 如果你指定了'--enable-stdcall-fixup', 这个特性会被完全  
开启,警告信息也不会打印出来. 如果你指定了'--disable-stdcall-fixup',这个特性被关闭,而且这  
样的错误匹配会被认为是个错误.  
  
`--export-all-symbols'  
如果给出这个选项,目标中所有由DLL建立的全局符号会被DLL导出. 注意这是缺省情况,否则没有任何  
符号被导出. 如果符号由DEF文件显式地导出,或由函数本身的属性隐式地导出, 缺省情况是除非选项  
给出,否则不导出任何其他的符号. 注意符号`DllMain@12',`DllEntryPoint@0',  
`DllMainCRTStartup@12'和`impure\_ptr'不会自动被导出.而且,由其他的DLL导入的符号也不会被再  
次导出, 还有指定DLL内部布局的符号,比如那些以'\_head\_'开头,或者以'\_iname'结尾的符号也不会  
被导出.还有,'libgcc','libstd++','libmingw32'或'crtX.o'中的符号也不会被导出. ......  
  
环境变量  
=====================  
  
你可以通过环境变量`GNUTARGET', `LDEMULATION'和`COLLECT\_NO\_DEMANGLE'改变'ld'的行为.  
  
`GNUTARGET'在你没有使用'-b'(或者它的同义词'--format')的时候,决定输入文件的格式. 它的值应当是BFD  
中关于输入格式的一个名字. 如果环境中没有'GNUTARGET'变量, 'ld'使用目标平台的缺省格式. 如果  
'GNUTARGET'被设为'default', 那BFD就会通过检查二进制的输入文件来找到输入格式; 这个方法通常会成功,  
但会有潜在的不明确性, 因为没有办法保证指定一个目标文件格式的魔数总是唯一的. 但是, 在每一个系统上  
的BFD配置程序会把这个系统的常规格式放在搜索列表的首位, 所以不明确性可以通过这种惯列来解决.  
  
`LDEMULATION'在你没有使用'-m'选项的时候决定缺省的模拟器. 模拟器可以影响到连接器行为的很多方面,  
特别是连接器的缺省连接脚本. 你可以通过'--verbose'或'-V'选项列出所有可用的模拟器. 如果'-m'选项没  
有使用, 而且`LDEMULATION'环境变量没有定义, 缺省的模拟器跟连接器如何被配置有关.  
  
一般地,连接器缺省状况下会重构符号.但是,如果在环境中设置了`COLLECT\_NO\_DEMANGLE', 那缺省状态下就不  
会重构符号.这个环境变量在GCC的连接包装程序中会以相似的方式被使用. 这个缺省的行为可以被'--demangle'  
或'--no-demangle'选项覆盖.  
  
连接脚本  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
每个连接都被一个'连接脚本'所控制. 这个脚本是用连接命令语言书写的.  
  
连接脚本的一个主要目的是描述输入文件中的节如何被映射到输出文件中,并控制输出文件的内存排布. 几乎  
所有的连接脚本只做这两件事情. 但是,在需要的时候,连接器脚本还可以指示连接器执行很多其他的操作.这  
通过下面描述的命令实现.  
  
连接器总是使用连接器脚本的.如果你自己不提供, 连接器会使用一个缺省的脚本,这个脚本是被编译进连接器  
可执行文件的. 你可以使用'--verbose'命令行选项来显示缺省的连接器脚本的内容. 某些命令行选项,比如  
'-r'或'-N', 会影响缺省的连接脚本.  
  
你可以过使用'-T'命令行选项来提供你自己的连接脚本. 当你这么做的时候, 你的连接脚本会替换缺省的连  
接脚本.  
  
你也可以通过把连接脚本作为一个连接器的输入文件来隐式地使用它,就象它们是一个被连接的文件一样.  
  
基本的连接脚本的概念  
============================  
  
我们需要定义一些基本的概念与词汇以描述连接脚本语言.  
  
连接器把多个输入文件合并成单个输出文件. 输出文件和输入文件都以一种叫做'目标文件格式'的数据格式形  
式存在. 每一个文件被叫做'目标文件'. 输出文件经常被叫做'可执行文件',但是由于需要,我们也把它叫做目  
标文件. 每一个目标文件中,在其它东西之间,有一个节列表.我们有时把输入文件的节叫做输入节; 相似的,输  
出文件中的一个节经常被叫做输出节.  
  
一个目标文件中的每一个节都有一个名字和一个大小尺寸. 大多数节还有一个相关的数据块, 称为节内容. 某  
一个节可能被标式讵'loadable',含义是在输出文件被执行时,这个节应当被载入到内存中去. 一个没有内容的  
节可能是'allocatable', 含义是内存中必须为这个节开辟一块空间,但是没有实际的内容载入到这里(在某些  
情况下,这块内存必须被标式讵零). 一个既不是loadable也不是allocatable的节一般含有一些调试信息.  
  
每一个loadable或allocatable的输出节有两个地址. 第一个是'VMA'或称为虚拟内存地址. 这是当输出文件运  
行时节所拥有的地址. 第二个是"LMA', 或称为载入内存地址. 这个节即将要载入的内存地址. 这大多数情况下  
这两个地址是相同的. 它们两个有可能不同的一个例子是当一个数据节在ROM中时, 当程序启动时,被拷贝到RAM  
中(这个技术经常被用在基于ROM的系统中进行全局变量的初始化). 在这种情况下, ROM地址就是LMA, 而RAM地  
址就是VMA.  
  
你可以通过使用带有'-h'选项的'objdump'来察看目标文件中的节.  
  
每一个目标文件还有一个关于符号的列表, 被称为'符号表'. 一个符号可能是定义过了的,也可能是未定义的.  
每一个符号有一个名字, 而且每一个定义的符号有一个地址. 如果你把一个C/C++程序编译为一个目标文件,对  
于每一个定义的函数和全局或静态变量,你为得到一个定义的符号. 每一个在输入文件中只是一个引用而未定义  
的函数或全局变量会变成一个未定义的符号.  
  
你可以使用'nm'程序来看一个目标文件中的符号, 或者使用'objdump'程序带有'-t'选项.  
  
连接脚本的格式  
====================  
  
连接脚本是文本文件.  
  
你写了一系列的命令作为一个连接脚本. 每一个命令是一个带有参数的关键字,或者是一个对符号的赋值. 你可  
以用分号分隔命令. 空格一般被忽略.  
  
文件名或格式名之类的字符串一般可以被直接键入. 如果文件名含有特殊字符,比如一般作为分隔文件名用的逗  
号, 你可以把文件名放到双引号中. 文件名中间无法使用双引号.  
  
你可以象在C语言中一样,在连接脚本中使用注释, 用'/\*'和'\*/'隔开. 就像在C中,注释在语法上等同于空格.  
  
简单的连接脚本示例  
============================  
  
许多脚本是相当的简单的.  
  
可能的最简单的脚本只含有一个命令: 'SECTIONS'. 你可以使用'SECTIONS'来描述输出文件的内存布局.  
  
'SECTIONS'是一个功能很强大的命令. 这里这们会描述一个很简单的使用. 让我们假设你的程序只有代码节,  
初始化过的数据节, 和未初始化过的数据节. 这些会存在于'.text','.data'和'.bss'节, 另外, 让我们进一  
步假设在你的输入文件中只有这些节.  
  
对于这个例子, 我们说代码应当被载入到地址'0x10000'处, 而数据应当从0x8000000处开始. 下面是一个实现  
这个功能的脚本:  
  
SECTIONS  
{  
. = 0x10000;  
.text : { \*(.text) }  
. = 0x8000000;  
.data : { \*(.data) }  
.bss : { \*(.bss) }  
}  
  
你使用关键字'SECTIONS'写了这个SECTIONS命令, 后面跟有一串放在花括号中的符号赋值和输出节描述的内容.  
  
上例中, 在'SECTIONS'命令中的第一行是对一个特殊的符号'.'赋值, 这是一个定位计数器. 如果你没有以其  
它的方式指定输出节的地址(其他方式在后面会描述), 那地址值就会被设为定位计数器的现有值. 定位计数器  
然后被加上输出节的尺寸. 在'SECTIONS'命令的开始处, 定位计数器拥有值'0'.  
  
第二行定义一个输出节,'.text'. 冒号是语法需要,现在可以被忽略. 节名后面的花括号中,你列出所有应当被  
放入到这个输出节中的输入节的名字. '\*'是一个通配符,匹配任何文件名. 表达式'\*(.text)'意思是所有的输  
入文件中的'.text'输入节.  
  
因为当输出节'.text'定义的时候, 定位计数器的值是'0x10000',连接器会把输出文件中的'.text'节的地址设  
为'0x10000'.  
  
余下的内容定义了输出文件中的'.data'节和'.bss'节. 连接器会把'.data'输出节放到地址'0x8000000'处. 连接  
器放好'.data'输出节之后, 定位计数器的值是'0x8000000'加上'.data'输出节的长度. 得到的结果是连接器会  
把'.bss'输出节放到紧接'.data'节后面的位置.  
  
连接器会通过在必要时增加定位计数器的值来保证每一个输出节具有它所需的对齐. 在这个例子中, 为'.text'  
和'.data'节指定的地址会满足对齐约束, 但是连接器可能会需要在'.data'和'.bss'节之间创建一个小的缺口.  
  
就这样,这是一个简单但完整的连接脚本.  
  
简单的连接脚本命令.  
=============================  
  
在本章中,我们会描述一些简单的脚本命令.  
  
设置入口点.  
-----------------------  
  
在运行一个程序时第一个被执行到的指令称为"入口点". 你可以使用'ENTRY'连接脚本命令来设置入口点.参数  
是一个符号名:  
ENTRY(SYMBOL)  
  
有多种不同的方法来设置入口点.连接器会通过按顺序尝试以下的方法来设置入口点, 如果成功了,就会停止.  
  
\* `-e'入口命令行选项;  
  
\* 连接脚本中的`ENTRY(SYMBOL)'命令;  
  
\* 如果定义了start, 就使用start的值;  
  
\* 如果存在,就使用'.text'节的首地址;  
  
\* 地址`0'.  
  
处理文件的命令.  
---------------------------  
  
有几个处理文件的连接脚本命令.  
  
`INCLUDE FILENAME'  
在当前点包含连接脚本文件FILENAME. 在当前路径下或用'-L'选项指定的所有路径下搜索这个文件,  
你可以嵌套使用'INCLUDE'达10层.  
  
`INPUT(FILE, FILE, ...)'  
`INPUT(FILE FILE ...)'  
'INPUT'命令指示连接器在连接时包含文件, 就像它们是在命令行上指定的一样.  
  
比如,如果你在连接的时候总是要包含文件'subr.o',但是你对每次连接时要在命令行上输入感到厌烦  
, 你就可以在你的连接脚本中输入'INPUT (subr.o).  
  
事实上,如果你喜欢,你可以把你所有的输入文件列在连接脚本中, 然后在连接的时候什么也不需要,  
只要一个'-T'选项就够了.  
  
在一个'系统根前缀'被配置的情况下, 一个文件名如果以'/'字符打头, 并且脚本也存放在系统根  
前缀的某个子目录下, 文件名就会被在系统根前缀下搜索. 否则连接器就会企图打开当前目录下的文  
件. 如果没有发现, 连接器会通过档案库搜索路径进行搜索.  
  
如果你使用了'INPUT (-lFILE)', 'ld'会把文件名转换为'libFILE.a', 就象命令行参数'-l'一样.  
  
当你在一个隐式连接脚本中使用'INPUT'命令的时候, 文件就会在连接时连接脚本文件被包含的点上  
被包含进来. 这会影响到档案搜索.  
  
`GROUP(FILE, FILE, ...)'  
`GROUP(FILE FILE ...)'  
除了文件必须全是档案文件之外, 'GROUP'命令跟'INPUT'相似, 它们会被反复搜索,直至没有未定义  
的引用被创建.  
  
`OUTPUT(FILENAME)'  
'OUTPUT'命令命名输出文件. 在连接脚本中使用'OUTPUT(FILENAME)'命令跟在命令行中使用'-o  
FILENAME'命令是完全等效的. 如果两个都使用了, 那命令行选项优先.  
  
你可以使用'OUTPUT'命令为输出文件创建一个缺省的文件名,而不是常用的'a.out'.  
  
`SEARCH\_DIR(PATH)'  
`SEARCH\_DIR'命令给'ld'用于搜索档案文件的路径中再增加新的路径. 使用`SEARCH\_DIR(PATH)'跟在  
命令行上使用'-L PATH'选项是完全等效的. 如果两个都使用了, 那连接器会两个路径都搜索. 用命  
令行选项指定的路径首先被搜索.  
  
`STARTUP(FILENAME)'  
除了FILENAME会成为第一个被连接的输入文件, 'STARTUP'命令跟'INPUT'命令完全相似, 就象这个文  
件是在命令行上第一个被指定的文件一样. 如果在一个系统中, 入口点总是存在于第一个文件中,那  
这个就很有用.  
  
处理目标文件格式的命令.  
-----------------------------------------  
  
有两个处理目标文件格式的连接脚本命令.  
  
`OUTPUT\_formAT(BFDNAME)'  
`OUTPUT\_formAT(DEFAULT, BIG, LITTLE)'  
`OUTPUT\_formAT'命令为输出文件使用的BFD格式命名. 使用`OUTPUT\_formAT(BFDNAME)'跟在命令行上  
使用'-oformat BFDNAME'是完全等效的. 如果两个都使用了, 命令行选项优先.  
  
你可在使用`OUTPUT\_formAT'时带有三个参数以使用不同的基于'-EB'和'-EL'的命令行选项的格式.  
  
如果'-EB'和'-EL'都没有使用, 那输出格式会是第一个参数DEFAULT, 如果使用了'-EB',输出格式会是  
第二个参数BIG, 如果使用了'-EL', 输出格式会是第三个参数, LITTLE.  
  
比如, 缺省的基于MIPS ELF平台连接脚本使用如下命令:  
  
OUTPUT\_formAT(elf32-bigmips, elf32-bigmips, elf32-littlemips)  
这表示缺省的输出文件格式是'elf32-bigmips', 但是当用户使用'-EL'命令行选项的时候, 输出文件就会  
被以`elf32-littlemips'格式创建.  
  
`TARGET(BFDNAME)'  
'TARGET'命令在读取输入文件时命名BFD格式. 它会影响到后来的'INPUT'和'GROUP'命令. 这个命令跟  
在命令行上使用`-b BFDNAME'相似. 如果使用了'TARGET'命令但`OUTPUT\_formAT'没有指定, 最后的  
'TARGET'命令也被用来设置输出文件的格式.  
  
其它的连接脚本命令.  
----------------------------  
  
还有一些其它的连接脚本命令.  
  
`ASSERT(EXP, MESSAGE)'  
确保EXP不等于零,如果等于零, 连接器就会返回一个错误码退出,并打印出MESSAGE.  
  
`EXTERN(SYMBOL SYMBOL ...)'  
强制SYMBOL作为一个无定义的符号输入到输出文件中去. 这样做了,可能会引发从标准库中连接一些  
节外的库. 你可以为每一个EXTERN'列出几个符号, 而且你可以多次使用'EXTERN'. 这个命令跟'-u'  
命令行选项具有相同的效果.  
  
`FORCE\_COMMON\_ALLOCATION'  
这个命令跟命令行选项'-d'具有相同的效果: 就算指定了一个可重定位的输出文件('-r'),也让'ld'  
为普通符号分配空间.  
  
`INHIBIT\_COMMON\_ALLOCATION'  
这个命令跟命令行选项`--no-define-common'具有相同的效果: 就算是一个不可重位输出文件, 也让  
'ld'忽略为普通符号分配的空间.  
  
`NOCROSSREFS(SECTION SECTION ...)'  
这个命令在遇到在某些特定的节之间引用的时候会产生一条错误信息.  
  
在某些特定的程序中, 特别是在使用覆盖技术的嵌入式系统中, 当一个节被载入内存时,另外一个节  
就不会在内存中. 任何在两个节之间的直接引用都会是一个错误. 比如, 如果节1中的代码调用了另  
一个节中的一个函数,这就会产生一个错误.  
  
`NOCROSSREFS'命令带有一个输出节名字的列表. 如果'ld'遇到任何在这些节之间的交叉引用, 它就  
会报告一个错误,并返回一个非零退出码. 注意, `NOCROSSREFS'命令使用输出节名,而不是输入节名.  
  
`OUTPUT\_ARCH(BFDARCH)'  
指定一个特定的输出机器架构. 这个参数是BFD库中使用的一个名字. 你可以通过使用带有'-f'选项  
的'objdump'程序来查看一个目标文件的架构.  
  
为符号赋值.  
===========================  
  
你可以在一个连接脚本中为一个符号赋一个值. 这会把一个符号定义为一个全局符号.  
  
简单的赋值.  
------------------  
  
你可以使用所有的C赋值符号为一个符号赋值.  
  
`SYMBOL = EXPRESSION ;'  
`SYMBOL += EXPRESSION ;'  
`SYMBOL -= EXPRESSION ;'  
`SYMBOL \*= EXPRESSION ;'  
`SYMBOL /= EXPRESSION ;'  
`SYMBOL <<= EXPRESSION ;'  
`SYMBOL >>= EXPRESSION ;'  
`SYMBOL &= EXPRESSION ;'  
`SYMBOL |= EXPRESSION ;'  
  
第一个情况会把SYMBOL定义为值EXPRESSION. 其它情况下, SYMBOL必须是已经定义了的, 而值会作出相应的调  
整.  
  
特殊符号名'.'表示定位计数器. 你只可以在'SECTIONS'命令中使用它.  
  
EXPRESSION后面的分号是必须的.  
  
表达式下面会定义.  
  
你在写表达式赋值的时候,可以把它们作为单独的部分,也可以作为'SECTIONS'命令中的一个语句,或者作为  
'SECTIONS'命令中输出节描述的一个部分.  
  
符号所在的节会被设置成表达式所在的节.  
  
下面是一个关于在三处地方使用符号赋值的例子:  
  
floating\_point = 0;  
SECTIONS  
{  
.text :  
{  
\*(.text)  
\_etext = .;  
}  
\_bdata = (. + 3) & ~ 3;  
.data : { \*(.data) }  
}  
  
在这个例子中, 符号`floating\_point'被定义为零. 符号'-etext'会被定义为前面一个'.text'节尾部的地址.  
而符号'\_bdata'会被定义为'.text'输出节后面的一个向上对齐到4字节边界的一个地址值.  
  
PROVIDE  
-------  
  
在某些情况下, 一个符号被引用到的时候只在连接脚本中定义,而不在任何一个被连接进来的目标文件中定  
义. 这种做法是比较明智的. 比如, 传统的连接器定义了一个符号'etext'. 但是, ANSI C需要用户能够把  
'etext'作为一个函数使用而不会产生错误. 'PROVIDE'关键字可以被用来定义一个符号, 比如'etext', 这个  
定义只在它被引用到的时候有效,而在它被定义的时候无效.语法是 `PROVIDE(SYMBOL = EXPRESSION)'.  
  
下面是一个关于使用'PROVIDE'定义'etext'的例子:  
  
SECTIONS  
{  
.text :  
{  
\*(.text)  
\_etext = .;  
PROVIDE(etext = .);  
}  
}  
  
在这个例子中, 如果程序定义了一个'\_etext'(带有一个前导下划线), 连接器会给出一个重定义错误. 如果,  
程序定义了一个'etext'(不带前导下划线), 连接器会默认使用程序中的定义. 如果程序引用了'etext'但不  
定义它, 连接器会使用连接脚本中的定义.  
  
SECTIONS命令  
================  
  
'SECTIONS'命令告诉连接器如何把输入节映射到输出节, 并如何把输出节放入到内存中.  
  
'SECTIONS'命令的格式如下:  
  
SECTIONS  
{  
SECTIONS-COMMAND  
SECTIONS-COMMAND  
...  
}  
  
每一个SECTIONS-COMMAND可能是如下的一种:  
  
\* 一个'ENTRY'命令.  
  
\* 一个符号赋值.  
  
\* 一个输出节描述.  
  
\* 一个重叠描述.  
  
'ENTRY'命令和符号赋值在'SECTIONS'命令中是允许的, 这是为了方便在这些命令中使用定位计数器. 这也可  
以让连接脚本更容易理解, 因为你可以在更有意义的地方使用这些命令来控制输出文件的布局.  
  
输出节描述和重叠描述在下面描述.  
  
如果你在连接脚本中不使用'SECTIONS'命令, 连接器会按在输入文件中遇到的节的顺序把每一个输入节放到同  
名的输出节中. 如果所有的输入节都在第一个文件中存在,那输出文件中的节的顺序会匹配第一个输入文件中  
的节的顺序. 第一个节会在地址零处.  
  
输出节描述  
--------------------------  
  
一个完整的输出节的描述应该是这个样子的:  
  
SECTION [ADDRESS] [(TYPE)] : [AT(LMA)]  
{  
OUTPUT-SECTION-COMMAND  
OUTPUT-SECTION-COMMAND  
...  
} [>REGION] [AT>LMA\_REGION] [:PHDR :PHDR ...] [=FILLEXP]  
  
大多数输出节不使用这里的可选节属性.  
  
SECTION边上的空格是必须的, 所以节名是明确的. 冒号跟花括号也是必须的. 断行和其他的空格是可选的.  
  
每一个OUTPUT-SECTION-COMMAND可能是如下的情况:  
  
\* 一个符号赋值.  
  
\* 一个输入节描述.  
  
\* 直接包含的数据值.  
  
\* 一个特定的输出节关键字.  
  
输出节名.  
-------------------  
  
输出节的名字是SECTION. SECTION必须满足你的输出格式的约束. 在一个只支持限制数量的节的格式中,比如  
'a.out',这个名字必须是格式支持的节名中的一个(比如, 'a.out'只允许'.text', '.data'或'.bss').如果  
输出格式支持任意数量的节, 但是只支持数字,而没有名字(就像Oasys中的情况), 名字应当以一个双引号中的  
数值串的形式提供.一个节名可以由任意数量的字符组成,但是一个含有任意非常用字符(比如逗号)的字句必须  
用双引号引起来.  
  
输出节描述  
--------------------------  
  
ADDRESS是关于输出节中VMS的一个表达式. 如果你不提供ADDRESS, 连接器会基于REGION(如果存在)设置它,或  
者基于定位计数器的当前值.  
  
如果你提供了ADDRESS, 那输出节的地址会被精确地设为这个值. 如果你既不提供ADDRESS也不提供REGION, 那  
输出节的地址会被设为当前的定位计数器向上对齐到输出节需要的对齐边界的值. 输出节的对齐要求是所有输  
入节中含有的对齐要求中最严格的一个.  
  
比如:  
.text . : { \*(.text) }  
  
和  
.text : { \*(.text) }  
  
有细微的不同. 第一个会把'.text'输出节的地址设为当前定位计数器的值. 第二个会把它设为定位计数器的  
当前值向上对齐到'.text'输入节中对齐要求最严格的一个边界.  
  
ADDRESS可以是任意表达式; 比如,如果你需要把节对齐对0x10字节边界,这样就可以让低四字节的节地址值为  
零, 你可以这样做:  
  
.text ALIGN(0x10) : { \*(.text) }  
  
这个语句可以正常工作,因为'ALIGN'返回当前的定位计数器,并向上对齐到指定的值.  
  
指定一个节的地址会改变定位计数器的值.  
  
输入节描述  
-------------------------  
  
最常用的输出节命令是输入节描述.  
  
输入节描述是最基本的连接脚本操作. 你使用输出节来告诉连接器在内存中如何布局你的程序. 你使用输入节  
来告诉连接器如何把输入文件映射到你的内存中.  
  
输入节基础  
---------------------------  
  
一个输入节描述由一个文件名后跟有可选的括号中的节名列表组成.  
  
文件名和节名可以通配符形式出现, 这个我们以后再介绍.  
  
最常用的输入节描述是包含在输出节中的所有具有特定名字的输入节. 比如, 包含所有输入'.text'节,你可以  
这样写:  
  
\*(.text)  
  
这里,'\*'是一个通配符,匹配所有的文件名. 为把一部分文件排除在匹配的名字通配符之外, EXCLUDE\_FILE可  
以用来匹配所有的除了在EXCLUDE\_FILE列表中指定的文件.比如:  
  
(\*(EXCLUDE\_FILE (\*crtend.o \*otherfile.o) .ctors))  
  
会让除了`crtend.o'文件和`otherfile.o'文件之外的所有的文件中的所有的.ctors节被包含进来.  
  
有两种方法包含多于一个的节:  
  
\*(.text .rdata)  
\*(.text) \*(.rdata)  
  
上面两句的区别在于'.text'和'.rdata'输入节的输出节中出现的顺序不同. 在第一个例子中, 两种节会交替  
出现,并以连接器的输入顺序排布. 在第二个例子中,所有的'.text'输入节会先出现,然后是所有的'.rdata'节.  
  
你可以指定文件名,以从一个特定的文件中包含节. 如果一个或多个你的文件含有特殊的数据在内存中需要特  
殊的定位,你可以这样做. 比如:  
  
data.o(.data)  
  
如果你使用一个不带有节列表的文件名, 那输入文件中的所有的节会被包含到输出节中. 通常不会这样做, 但  
是在某些场合下这个可能非常有用. 比如:  
  
data.o  
  
当你使用一个不含有任何通配符的文件名时, 连接器首先会查看你是否在连接命令行上指定了文件名或者在  
'INPUT'命令中. 如果你没有, 连接器会试图把这个文件作为一个输入文件打开, 就像它在命令行上出现一样.  
注意这跟'INPUT'命令不一样, 因为连接器会在档案搜索路径中搜索文件.  
  
输入节通配符  
---------------------------------  
  
在一个输入节描述中, 文件名或者节名,或者两者同时都可以是通配符形式.  
  
文件名通配符'\*'在很多例子中都可以看到,这是一个简单的文件名通配符形式.  
  
通配符形式跟Unix Shell中使用的一样.  
  
`\*'  
匹配任意数量的字符.  
  
`?'  
匹配单个字符.  
  
`[CHARS]'  
匹配CHARS中的任意单个字符; 字符'-'可以被用来指定字符的方讧, 比如[a-z]匹配任意小字字符.  
  
`\'  
转义其后的字符.  
  
当一个文件名跟一个通配符匹配时, 通配符字符不会匹配一个'/'字符(在UNIX系统中用来分隔目录名), 一个  
含有单个'\*'字符的形式是个例外; 它总是匹配任意文件名, 不管它是否含有'/'. 在一个节名中, 通配符字  
符会匹配'/'字符.  
  
文件名通配符只匹配那些在命令行或在'INPUT'命令上显式指定的文件. 连接器不会通过搜索目录来展开通配  
符.  
  
如果一个文件名匹配多于一个通配符, 或者如果一个文件名显式出现同时又匹配了一个通配符, 连接器会使用  
第一次匹配到的连接脚本. 比如, 下面的输入节描述序列很可能就是错误的,因为'data.o'规则没有被使用:  
  
.data : { \*(.data) }  
.data1 : { data.o(.data) }  
  
通常, 连接器会把匹配通配符的文件和节按在连接中被看到的顺序放置. 你可以通过'SORT'关键字改变它, 它  
出现在括号中的通配符之前(比如, 'SORT(.text\*)'). 当'SORT'关键字被使用时, 连接器会在把文件和节放到  
输出文件中之前按名字顺序重新排列它们.  
  
如果你对于输入节被放置到哪里去了感到很困惑, 那可以使用'-M'连接选项来产生一个位图文件. 位图文件会  
精确显示输入节是如何被映射到输出节中的.  
  
这个例子显示了通配符是如何被用来区分文件的. 这个连接脚本指示连接器把所有的'.text'节放到'.text'中, 把所有的'.bss'节放到'.bss'. 连接器会把所有的来自文件名以一个大写字母开始的文件中的'.data'节放进'.DATA'节中; 对于所有其他文件, 连接器会把'.data'节放进'.data'节中.  
  
SECTIONS {  
.text : { \*(.text) }  
.DATA : { [A-Z]\*(.data) }  
.data : { \*(.data) }  
.bss : { \*(.bss) }  
}  
  
输入节中的普通符号.  
-----------------------------------  
  
对于普通符号,需要一个特殊的标识, 因为在很多目标格式中, 普通符号没有一个特定的输入节. 连接器会把  
普通符号处理成好像它们在一个叫做'COMMON'的节中.  
  
你可能像使用带有其他输入节的文件名一样使用带有'COMMON'节的文件名。你可以通过这个把来自一个特定输  
入文件的普通符号放入一个节中，同时把来自其它输入文件的普通符号放入另一个节中。  
  
在大多数情况下，输入文件中的普通符号会被放到输出文件的'.bss'节中。比如：  
  
.bss { \*(.bss) \*(COMMON) }  
  
有些目标文件格式具有多于一个的普通符号。比如，MIPS ELF目标文件格式区分标准普通符号和小普通符号。  
在这种情况下，连接器会为其他类型的普通符号使用一个不同的特殊节名。 在MIPS ELF的情况中， 连接器  
为标准普通符号使用'COMMON'，并且为小普通符号使用'.common'。这就允许你把不同类型的普通符号映射到  
内存的不同位置。  
  
在一些老的连接脚本上，你有时会看到'[COMMON]'。这个符号现在已经过时了， 它等效于'\*(COMMON)'。  
  
输入节和垃圾收集  
---------------------------------------  
  
当连接时垃圾收集正在使用中时（'--gc-sections')，这在标识那些不应该被排除在外的节时非常有用。这  
是通过在输入节的通配符入口外面加上'KEEP()'实现的，比如'KEEP(\*(.init))'或者'KEEP(SORT(\*)(.sorts))  
'。  
  
输入节示例  
---------------------  
  
接下来的例子是一个完整的连接脚本。它告诉连接器去读取文件'all.o'中的所有节，并把它们放到输出节  
'outputa'的开始位置处， 该输出节是从位置'0x10000'处开始的。 从文件'foo.o'中来的所有节'.input1'  
在同一个输出节中紧密排列。 从文件'foo.o'中来的所有节'.input2'全部放入到输出节'outputb'中，后面  
跟上从'foo1.o'中来的节'.input1'。来自所有文件的所有余下的'.input1'和'.input2'节被写入到输出节  
'outputc'中。  
  
SECTIONS {  
outputa 0x10000 :  
{  
all.o  
foo.o (.input1)  
}  
outputb :  
{  
foo.o (.input2)  
foo1.o (.input1)  
}  
outputc :  
{  
\*(.input1)  
\*(.input2)  
}  
}  
  
输出节数据  
-------------------  
  
你可以通过使用输出节命令'BYTE','SHORT','LONG','QUAD',或者'SQUAD'在输出节中显式包含几个字节的数据  
每一个关键字后面都跟上一个圆括号中的要存入的值。表达式的值被存在当前的定位计数器的值处。  
  
‘BYTE’，‘SHORT’，‘LONG’‘QUAD’命令分别存储一个，两个，四个，八个字节。存入字节后，定位计  
数器的值加上被存入的字节数。  
  
比如，下面的命令会存入一字节的内容1,后面跟上四字节，其内容是符号'addr'的值。  
  
BYTE(1)  
LONG(addr)  
  
当使用64位系统时，‘QUAD’和‘SQUAD’是相同的；它们都会存储8字节，或者说是64位的值。而如果软硬件  
系统都是32位的，一个表达式就会被作为32位计算。在这种情况下，‘QUAD’存储一个32位值，并把它零扩展  
到64位， 而‘SQUAD’会把32位值符号扩展到64位。  
  
如果输出文件的目标文件格式有一个显式的endianness，它在正常的情况下，值就会被以这种endianness存储  
当一个目标文件格式没有一个显式的endianness时, 值就会被以第一个输入目标文件的endianness存储。  
  
注意， 这些命令只在一个节描述内部才有效，而不是在它们之间， 所以，下面的代码会使连接器产生一个错  
误信息：  
  
SECTIONS { .text : { \*(.text) } LONG(1) .data : { \*(.data) } }  
  
而这个才是有效的：  
  
SECTIONS { .text : { \*(.text) ; LONG(1) } .data : { \*(.data) } }  
  
你可能使用‘FILL’命令来为当前节设置填充样式。它后面跟有一个括号中的表达式。任何未指定的节内内存  
区域（比如，因为输入节的对齐要求而造成的裂缝）会以这个表达式的值进行填充。一个'FILL'语句会覆盖到  
它本身在节定义中出现的位置后面的所有内存区域；通过引入多个‘FILL’语句，你可以在输出节的不同位置  
拥有不同的填充样式。  
  
这个例子显示如何在未被指定的内存区域填充'0x90'：  
  
FILL(0x90909090)  
  
‘FILL’命令跟输出节的‘=FILLEXP’属性相似，但它只影响到节内跟在‘FILL’命令后面的部分，而不是  
整个节。如果两个都用到了，那‘FILL’命令优先。  
  
输出节关键字  
-----------------------  
  
有两个关键字作为输出节命令的形式出现。  
  
`CREATE\_OBJECT\_SYMBOLS'  
这个命令告诉连接器为每一个输入文件创建一个符号。而符号的名字正好就是相关输入文件的名字。  
而每一个符号的节就是`CREATE\_OBJECT\_SYMBOLS'命令出现的那个节。  
  
这个命令一直是a.out目标文件格式特有的。 它一般不为其它的目标文件格式所使用。  
  
`CONSTRUCTORS'  
当使用a.out目标文件格式进行连接的时候， 连接器使用一组不常用的结构以支持C++的全局构造函  
数和析构函数。当连接不支持专有节的目标文件格式时， 比如ECOFF和XCOFF，连接器会自动辩识C++  
全局构造函数和析构函数的名字。对于这些目标文件格式，‘CONSTRUCTORS’命令告诉连接器把构造  
函数信息放到‘CONSTRUCTORS’命令出现的那个输出节中。对于其它目标文件格式，‘CONSTRUCTORS’  
命令被忽略。  
  
符号`\_\_CTOR\_LIST\_\_'标识全局构造函数的开始，而符号`\_\_DTOR\_LIST'标识结束。这个列表的第一个  
WORD是入口的数量，紧跟在后面的是每一个构造函数和析构函数的地址，再然后是一个零WORD。编译  
器必须安排如何实际运行代码。对于这些目标文件格式，GNU C++通常从一个`\_\_main'子程序中调用  
构造函数，而对`\_\_main'的调用自动被插入到`main'的启动代码中。GNU C++通常使用'atexit'运行  
析构函数，或者直接从函数'exit'中运行。  
  
对于像‘COFF’或‘ELF’这样支持专有节名的目标文件格式，GNU C++通常会把全局构造函数与析构  
函数的地址值放到'.ctors'和'.dtors'节中。把下面的代码序列放到你的连接脚本中去，这样会构建  
出GNU C++运行时代码希望见到的表类型。  
  
\_\_CTOR\_LIST\_\_ = .;  
LONG((\_\_CTOR\_END\_\_ - \_\_CTOR\_LIST\_\_) / 4 - 2)  
\*(.ctors)  
LONG(0)  
\_\_CTOR\_END\_\_ = .;  
\_\_DTOR\_LIST\_\_ = .;  
LONG((\_\_DTOR\_END\_\_ - \_\_DTOR\_LIST\_\_) / 4 - 2)  
\*(.dtors)  
LONG(0)  
\_\_DTOR\_END\_\_ = .;  
  
如果你正使用GNU C++支持来进行优先初始化，那它提供一些可以控制全局构造函数运行顺序的功能，  
你必须在连接时给构造函数排好序以保证它们以正确的顺序被执行。当使用'CONSTRUCTORS'命令时，  
替代为`SORT(CONSTRUCTORS)'。当使用'.ctors'和'dtors'节时，使用`\*(SORT(.ctors))'和  
`\*(SORT(.dtors))' 而不是`\*(.ctors)'和`\*(.dtors)'。  
  
通常，编译器和连接器会自动处理这些事情，并且你不必亲自关心这些事情。但是，当你正在使用  
C++，并自己编写连接脚本时，你可能就要考虑这些事情了。  
  
输出节的丢弃。  
-------------------------  
  
连接器不会创建那些不含有任何内容的输出节。这是为了引用那些可能出现或不出现在任何输入文件中的输入  
节时方便。比如：  
  
.foo { \*(.foo) }  
  
如果至少在一个输入文件中有'.foo'节，它才会在输出文件中创建一个'.foo'节  
  
如果你使用了其它的而不是一个输入节描述作为一个输出节命令，比如一个符号赋值，那这个输出节总是被  
创建，即使没有匹配的输入节也会被创建。  
  
一个特殊的输出节名`/DISCARD/'可以被用来丢弃输入节。任何被分配到名为`/DISCARD/'的输出节中的输入  
节不包含在输出文件中。  
  
输出节属性  
-------------------------  
  
上面，我们已经展示了一个完整的输出节描述，看下去就象这样：  
  
SECTION [ADDRESS] [(TYPE)] : [AT(LMA)]  
{  
OUTPUT-SECTION-COMMAND  
OUTPUT-SECTION-COMMAND  
...  
} [>REGION] [AT>LMA\_REGION] [:PHDR :PHDR ...] [=FILLEXP]  
  
我们已经介绍了SECTION, ADDRESS, 和OUTPUT-SECTION-COMMAND. 在这一节中，我们将介绍余下的节属性。  
  
输出节类型  
...................  
  
每一个输出节可以有一个类型。类型是一个放在括号中的关键字，已定义的类型如下所示：  
  
`NOLOAD'  
这个节应当被标式讵不可载入，所以当程序运行时，它不会被载入到内存中。  
  
`DSECT'  
`COPY'  
`INFO'  
`OVERLAY'  
支持这些类型名只是为了向下兼容，它们很少使用。它们都具有相同的效果：这个节应当被标式讵不  
可分配，所以当程序运行时，没有内存为这个节分配。  
  
连接器通常基于映射到输出节的输入节来设置输出节的属性。你可以通过使用节类型来重设这个属性，  
比如，在下面的脚本例子中，‘ROM’节被定址在内存地址零处，并且在程序运行时不需要被载入。  
‘ROM’节的内容会正常出现在连接输出文件中。  
  
SECTIONS {  
ROM 0 (NOLOAD) : { ... }  
...  
}  
  
输出节LMA  
..................  
  
每一个节有一个虚地址（VMA）和一个载入地址（LMA）；出现在输出节描述中的地址表达式设置VMS  
  
连接器通常把LMA跟VMA设成相等。你可以通过使用‘AT’关键字改变这个。跟在关键字‘AT’后面的表达式  
LMA指定节的载入地址。或者，通过`AT>LMA\_REGION'表达式， 你可以为节的载入地址指定一个内存区域。  
  
这个特性是为了便于建立ROM映像而设计的。比如，下面的连接脚本创建了三个输出节：一个叫做‘.text’  
从地址‘0x1000’处开始，一个叫‘.mdata’，尽管它的VMA是'0x2000'，它会被载入到'.text'节的后面，最  
后一个叫做‘.bss’是用来放置未初始化的数据的，其地址从'0x3000'处开始。符号'\_data'被定义为值  
'0x2000', 它表示定位计数器的值是VMA的值，而不是LMA。  
  
SECTIONS  
{  
.text 0x1000 : { \*(.text) \_etext = . ; }  
.mdata 0x2000 :  
AT ( ADDR (.text) + SIZEOF (.text) )  
{ \_data = . ; \*(.data); \_edata = . ; }  
.bss 0x3000 :  
{ \_bstart = . ; \*(.bss) \*(COMMON) ; \_bend = . ;}  
}  
  
这个连接脚本产生的程序使用的运行时初始化代码会包含象下面所示的一些东西，以把初始化后的数据从ROM  
映像中拷贝到它的运行时地址中去。注意这节代码是如何利用好连接脚本定义的符号的。  
  
extern char \_etext, \_data, \_edata, \_bstart, \_bend;  
char \*src = &\_etext;  
char \*dst = &\_data;  
  
/\* ROM has data at end of text; copy it. \*/  
while (dst < &\_edata) {  
\*dst++ = \*src++;  
}  
  
/\* Zero bss \*/  
for (dst = &\_bstart; dst< &\_bend; dst++)  
\*dst = 0;  
  
输出节区域  
.....................  
  
你可以通过使用`>REGION'把一个节赋给前面已经定义的一个内存区域。  
  
这里有一个简单的例子:  
  
MEMORY { rom : ORIGIN = 0x1000, LENGTH = 0x1000 }  
SECTIONS { ROM : { \*(.text) } >rom }  
  
输出节Phdr  
...