objdump命令解析

objdump命令是Linux下的反汇编目标文件或者可执行文件的命令，它以一种可阅读的格式让你更多地了解二进制文件可能带有的附加信息。

最常用法： objdump -d main.o

objdump -S main.o 可以显示源程序及对应的汇编代码

objdump --help 获取帮助信息

参数选项：

--archive-headers

-a

显示档案库的成员信息,类似ls -l将lib\*.a的信息列出。

-b bfdname

--target=bfdname

指定目标码格式。这不是必须的，objdump能自动识别许多格式，比如：

objdump -b oasys -m vax -h fu.o

显示fu.o的头部摘要信息，明确指出该文件是Vax系统下用Oasys编译器生成的目标文件。objdump -i将给出这里可以指定的目标码格式列表。

-C

--demangle

将底层的符号名解码成用户级名字，除了去掉所开头的下划线之外，还使得C++函数名以可理解的方式显示出来。

--debugging

-g

显示调试信息。企图解析保存在文件中的调试信息并以C语言的语法显示出来。仅仅支持某些类型的调试信息。有些其他的格式被readelf -w支持。

-e

--debugging-tags

类似-g选项，但是生成的信息是和ctags工具相兼容的格式。

--disassemble

-d

从objfile中反汇编那些特定指令机器码的section。

-D

--disassemble-all

与 -d 类似，但反汇编所有section.

--prefix-addresses

反汇编的时候，显示每一行的完整地址。这是一种比较老的反汇编格式。

-EB

-EL

--endian={big|little}

指定目标文件的小端。这个项将影响反汇编出来的指令。在反汇编的文件没描述小端信息的时候用。例如S-records.

-f

--file-headers

显示objfile中每个文件的整体头部摘要信息。

-h

--section-headers

--headers

显示目标文件各个section的头部摘要信息。

-H

--help

简短的帮助信息。

-i

--info

显示对于 -b 或者 -m 选项可用的架构和目标格式列表。

-j name

--section=name

仅仅显示指定名称为name的section的信息

-l

--line-numbers

用文件名和行号标注相应的目标代码，仅仅和-d、-D或者-r一起使用使用-ld和使用-d的区别不是很大，在源码级调试的时候有用，要求编译时使用了-g之类的调试编译选项。

-m machine

--architecture=machine

指定反汇编目标文件时使用的架构，当待反汇编文件本身没描述架构信息的时候(比如S-records)，这个选项很有用。可以用-i选项列出这里能够指定的架构.

--reloc

-r

显示文件的重定位入口。如果和-d或者-D一起使用，重定位部分以反汇编后的格式显示出来。

--dynamic-reloc

-R

显示文件的动态重定位入口，仅仅对于动态目标文件意义，比如某些共享库。

-s

--full-contents

显示指定section的完整内容。默认所有的非空section都会被显示。

-S

--source

尽可能反汇编出源代码，尤其当编译的时候指定了-g这种调试参数时，效果比较明显。隐含了-d参数。

--show-raw-insn

反汇编的时候，显示每条汇编指令对应的机器码，如不指定--prefix-addresses，这将是缺省选项。

--no-show-raw-insn

反汇编时，不显示汇编指令的机器码，如不指定--prefix-addresses，这将是缺省选项。

--start-address=address

从指定地址开始显示数据，该选项影响-d、-r和-s选项的输出。

--stop-address=address

显示数据直到指定地址为止，该项影响-d、-r和-s选项的输出。

-t

--syms

显示文件的符号表入口。类似于nm -s提供的信息

-T

--dynamic-syms

显示文件的动态符号表入口，仅仅对动态目标文件意义，比如某些共享库。它显示的信息类似于 nm -D|--dynamic 显示的信息。

-V

--version

版本信息

--all-headers

-x

显示所可用的头信息，包括符号表、重定位入口。-x 等价于-a -f -h -r -t 同时指定。

-z

--disassemble-zeroes

一般反汇编输出将省略大块的零，该选项使得这些零块也被反汇编。

@file

可以将选项集中到一个文件中，然后使用这个@file选项载入。

关于符号表字段下面直接只介绍部分常用的：

.text：已编译程序的机器代码。

.rodata：只读数据，比如printf语句中的格式串和开关（switch）语句的跳转表。

.data：已初始化的全局C变量。局部C变量在运行时被保存在栈中，既不出现在.data中，也不出现在.bss节中。

.bss：未初始化的全局C变量。在目标文件中这个节不占据实际的空间，它仅仅是一个占位符。目标文件格式区分初始化和未初始化变量是为了空间效率在：在目标文件中，未初始化变量不需要占据任何实际的磁盘空间。

.symtab：一个符号表（symbol table），它存放在程序中被定义和引用的函数和全局变量的信息。一些程序员错误地认为必须通过-g选项来编译一个程序，得到符号表信息。实际上，每个可重定位目标文件在.symtab中都有一张符号表。然而，和编译器中的符号表不同，.symtab符号表不包含局部变量的表目。

.rel.text：当链接噐把这个目标文件和其他文件结合时，.text节中的许多位置都需要修改。一般而言，任何调用外部函数或者引用全局变量的指令都需要修改。另一方面调用本地函数的指令则不需要修改。注意，可执行目标文件中并不需要重定位信息，因此通常省略，除非使用者显式地指示链接器包含这些信息。

.rel.data：被模块定义或引用的任何全局变量的信息。一般而言，任何已初始化全局变量的初始值是全局变量或者外部定义函数的地址都需要被修改。

.debug：一个调试符号表，其有些表目是程序中定义的局部变量和类型定义，有些表目是程序中定义和引用的全局变量，有些是原始的C源文件。只有以-g选项调用编译驱动程序时，才会得到这张表。

.line：原始C源程序中的行号和.text节中机器指令之间的映射。只有以-g选项调用编译驱动程序时，才会得到这张表。

.strtab：一个字符串表，其内容包括.symtab和.debug节中的符号表，以及节头部中的节名字。字符串表就是以null结尾的字符串序列。

使用举例：

反汇编应用程序： objdump -d main.o

显示文件头信息： objdump -f main.o

显示制定section段信息(comment段) ：objdump -s -j .comment main.o