#### .o 就相当于windows里的obj文件 ，一个.c或.cpp文件对应一个.o文件  .a 是好多个.o合在一起,用于静态连接 ，即STATIC mode，多个.a可以链接生成一个exe的可执行文件  .so 是shared object,用于动态连接的,和windows的dll差不多，使用时才载入。

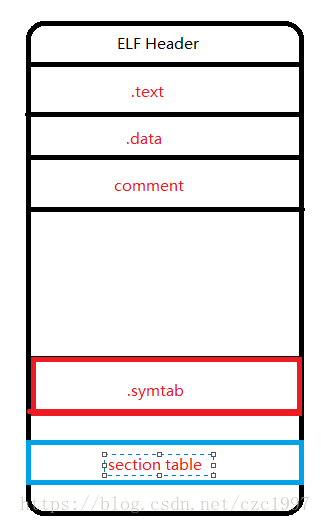
#### 编译及链接

https://blog.csdn.net/czc1997/article/details/81175399

（编译）.c -> .i ->.s->.o

->（链接）.exe

1. 编译到.o文件，还不能运行，因为它还依赖其他.o数据与指令



.o文件

ELF Header:记录这个obj信息

.text:指令

.data：初始化全局变量、静态变量、常量

.comment:未初始化数据或者初始化为0的数据（.bss和.comment共用）

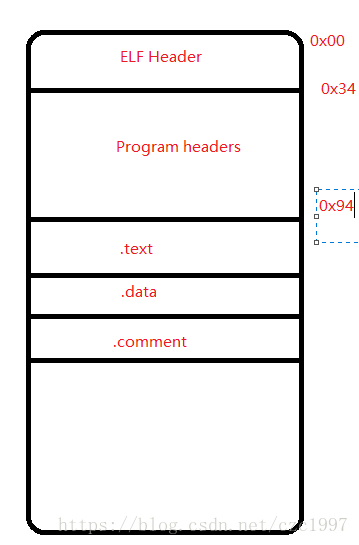
.symtab:符号表，记录数据、函数的符号（指令中通过该符号即可找到对应内容）

.section table:段表，记录所有段（ELH Header、.text、.data、.comment等）的信息（段偏移、段大小）

1. 链接：就是合并.o文件，生成.exe

<https://www.jianshu.com/p/a35f7514fec0>

**静态链接在编译期完成合并，动态链接在运行期完成合并**



.exe文件

ELF Header:main函数地址

.text:指令

.data：初始化全局变量、静态变量、常量

.comment:未初始化数据或者初始化为0的数据（.bss和.comment共用）

.symtab:符号表，记录数据、函数的符号（指令中通过该符号即可找到对应内容）

.section table:段表，记录所有段（ELH Header、.text、.data、.comment等）的信息（段偏移、段大小）

一个.c文件的执行会依赖其他.c文件，因此编译完后，还需要通过链接器将依赖的.c文件包含进来，这样程序才可以运行；

因为用到其他.c文件数据与指令，所以要将其他文件合并到当前.o文件中（包括：其他.o的.text、.data、.rodata、.bss、符号表）

符号表会进行重定位，因为其他文件的地址合并到当前文件时，其地址必定改变。

步骤：

①按段合并。

这里并不是单纯的段的叠加，而是将所有拥有相同权限（属性）的段合并在一个页面。例如，系统就把.data段和.bss段合并在了一起，因为它们都可读可写。将.rodata(用来存放字符串的段)和.text段合并，因为它们都只读。、

②调整偏移

将合并好的段进行偏移调整。

③合并符号表

④符号解析，并对属性为global/local进行判断

这一步非常重要，如果这个符号是local（static修饰过的）的，则不做任何处理。符号解析则是所有的obj文件中的符号表里的符号都要找到符号定义的地方。所以一般我们如果在编程时候使用了一个根本没定义过的变量，或者定义了同名变量，都是在这一步出错的。

那么我们在编译阶段发现data3失踪了，在经历符号解析之后它就会归来（当然，是不是原来那个它就不知道了）。在本工程中我只写了一个main.c源文件，系统将data3存在\*COM\*段中是说明系统觉得他有被强符号替代的可能性，所以在符号解析阶段系统就把它和其他文件中的同名符号（如果有的话）进行强弱比较，挑选出合适的符号。现在的data3本身就只有它一个叫data3，系统最后发现无可比较，就把它无罪释放啦:P

⑤给符号分配内存

⑥重定位

#### 编译完成后运行.exe

看一下程序运行的过程：

（1）创建虚拟地址空间到物理内存的映射

（2）创建页目录页表（CPU页表）

（3）加载代码段数据段（主存）

（4）创建进程（PCB、运行环境等）、线程（TCB、运行环境等）

（5）操作系统设置CPU上下文环境，将可执行文件入口地址（.exe文件的ELF Header）写入pc寄存器内

（5）CPU取指执行

PC：指令

用户态：执行用户空间指令，add mov等访问用户空间的指令

内核态：执行内核空间指令，即需要操作系统完成，例如：调度、系统调用、IO操作等（方式：系统调用、异常：缺页中断、外围设备中断）