预编译
main.cpp
sum.cpp
#开头的命令
#program lib 指定程序运行时需要链接的库
#program link 指定入口函数，需要到链接阶段处理

编译过程↑

编译 → 生成特定架构下的汇编指令
结构分析
词法分析
语义分析
代码优化

汇编 → 二进制可重定位的目标文件（*.obj）
把汇编指令转化成纯粹的机器码
main.o
sum.o
*.o文件的格式组成

符号表的产生
符号什么时候分配虚拟地址？

编译过程中，符号是不分配虚拟地址的

elf文件头
.text
.data
.bss
.symbol → 符号表段
.section
..

链接过程 第一步 符号解析完成后

链接

链接：编译完成的所有.o文件 + 静态库文件

sum.o
main.o
*.a *.lib

步骤一：
所有.o文件段的合并
符号表合并后，进行符号解析

main.o sum.o
各个段表合并了 .text(=).text
.data(=).data .bss(=).bss

所有对符号的引用都要找到该符号定义的地方 → *.o/.a*

多报名称符号未定义 重定义

步骤二：链接的核心
符号的重定位（重定向）

符号解析完成后，给所有的符号分配虚拟地址 之后 符号的重定向

L代表local → 静态 → 链接器看不见

a.out (=) *.o.各种段组成的，

但a.out有program header段 有的有load 告诉系统这个段要加载到内存中

可执行文件的组成格式是什么样子的？

xxx.exe → windows
a.out → linux

```
tony@tony-virtual-machine:~/code$ ld -e main *.o
tony@tony-virtual-machine:~/code$ ls
a.out  main.cpp  main.o  sum.cpp  sum.o
tony@tony-virtual-machine:~/code$
tony@tony-virtual-machine:~/code$ objdump -t a.out

a.out:     文件格式 elf64-x86-64

SYMBOL TABLE:
00000000004000e8 l    d  .text  0000000000000000 .text
0000000000400130 l    d  .eh_frame  0000000000000000 .eh_frame
0000000000601000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .comment  0000000000000000 .comment
0000000000000000 l    d  .debug_aranges 0000000000000000 .debug_aranges
0000000000000000 l    d  .debug_info  0000000000000000 .debug_info
0000000000000000 l    d  .debug_abbrev  0000000000000000 .debug_abbrev
0000000000000000 l    d  .debug_line  0000000000000000 .debug_line
0000000000000000 l    d  .debug_str  0000000000000000 .debug_str
0000000000000000 l    df *ABS*  0000000000000000 main.cpp
0000000000000000 l    df *ABS*  0000000000000000 sum.cpp
000000000040011b g     F .text  0000000000000014 _Z3sumii
0000000000601008 g       .data  0000000000000000 __bss_start
0000000000601000 g     O .data  0000000000000004 data
00000000004000e8 g     F .text  0000000000000033 main
0000000000601004 g     O .data  0000000000000004 gdata
0000000000601008 g       .data  0000000000000000 _edata
0000000000601008 g       .data  0000000000000000 _end
```

```
tony@tony-virtual-machine:~/code$ cat main.cpp

// 引用sum.cpp文件里面定义的全局变量 以及 函数
extern int gdata;
int sum(int, int);

int data = 20;

int main()
{
        int a = gdata;
        int b = data;

        int ret = sum(a, b);

        return 0;
}
tony@tony-virtual-machine:~/code$ cat sum.cpp
int gdata = 10;
int sum(int a, int b)
{
        return a+b;
}
```

虽然没有定义，但也生成符号

// data .data

// main .text

```
tony@tony-virtual-machine:~/code$ objdump -t main.o

main.o:     文件格式 elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 main.cpp
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .note.GNU-stack  0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame  0000000000000000 .eh_frame
0000000000000000 l    d  .comment  0000000000000000 .comment
0000000000000000 g     O .data  0000000000000004 data
0000000000000000 g     F .text  0000000000000033 main
0000000000000000         *UND*  0000000000000000 gdata
0000000000000000         *UND*  0000000000000000 _GLOBAL_OFFSET_TABLE_
0000000000000000         *UND*  0000000000000000 _Z3sumii

tony@tony-virtual-machine:~/code$ objdump -t sum.o

sum.o:     文件格式 elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 sum.cpp
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .note.GNU-stack  0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame  0000000000000000 .eh_frame
0000000000000000 l    d  .comment  0000000000000000 .comment
0000000000000000 g     O .data  0000000000000004 gdata
0000000000000000 g     F .text  0000000000000014 _Z3sumii
```

多了一个外部依赖项

文件头

直接写这个，可以说明怎么查看这个文件

```
tony@tony-virtual-machine:~/code$ readelf -h main.o
ELF 头：
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  类别:                              ELF64
  数据:                              2 补码，小端序 (little endian)
  版本:                              1 (current)
  OS/ABI:                            UNIX - System V
  ABI 版本:                          0
  类型:                              REL (可重定位文件)
  系统架构:                          Advanced Micro Devices X86-64
  版本:                              0x1
  入口点地址:                        0x0
  程序头起点:                        0 (bytes into file)
  Start of section headers:          784 (bytes into file)
  标志:                              0x0
  本头的大小:                        64 (字节)
  程序头大小:                        0 (字节)
  Number of program headers:         0
  节头大小:                          64 (字节)
  节头数目:                          12
  字符串表索引节头:                  11
```

```
tony@tony-virtual-machine:~/code$ objdump -s main.o

main.o:     文件格式 elf64-x86-64

Contents of section .text:
 0000 554889e5 4883ec10 8b050000 00008945  UH..H..........E
 0010 f48b0500 00000089 45f88b55 f88b45f4  ........E..U..E.
 0020 89d689c7 e8000000 008945fc b8000000  ..........E.....
 0030 00c9c3                               ...
Contents of section .data:
 0000 14000000                             ....
Contents of section .comment:
 0000 00474343 3a202855 62756e74 7520372e  .GCC: (Ubuntu 7.
 0010 332e302d 32377562 756e7475 317e3138  3.0-27ubuntu1~18
 0020 2e303429 20372e33 2e3000             .04) 7.3.0.
Contents of section .eh_frame:
 0000 14000000 00000000 017a5200 01781001  .........zR..x..
 0010 1b0c0708 90010000 1c000000 1c000000  ................
 0020 00000000 33000000 00410e10 8602430d  ....3....A....C.
 0030 066e0c07 08000000                    .n......
```

g++ -c main.cpp -g
objdump -S main.o

```
main.o:     文件格式 elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
int sum(int, int); // sum *UND*

int data = 20; // data .data

int main()  // main .text
{
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 10             sub    $0x10,%rsp
        int a = gdata;
   8:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # e <main+0xe>
   e:   89 45 f4                mov    %eax,-0xc(%rbp)
        int b = data;
  11:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 17 <main+0x17>
  17:   89 45 f8                mov    %eax,-0x8(%rbp)

        int ret = sum(a, b);
  1a:   8b 55 f8                mov    -0x8(%rbp),%edx
  1d:   8b 45 f4                mov    -0xc(%rbp),%eax
  20:   89 d6                   mov    %edx,%esi
  22:   89 c7                   mov    %eax,%edi
  24:   e8 00 00 00 00          callq  29 <main+0x29>
  29:   89 45 fc                mov    %eax,-0x4(%rbp)

        return 0;
  2c:   b8 00 00 00 00          mov    $0x0,%eax
}
  31:   c9                      leaveq
  32:   c3                      retq
```

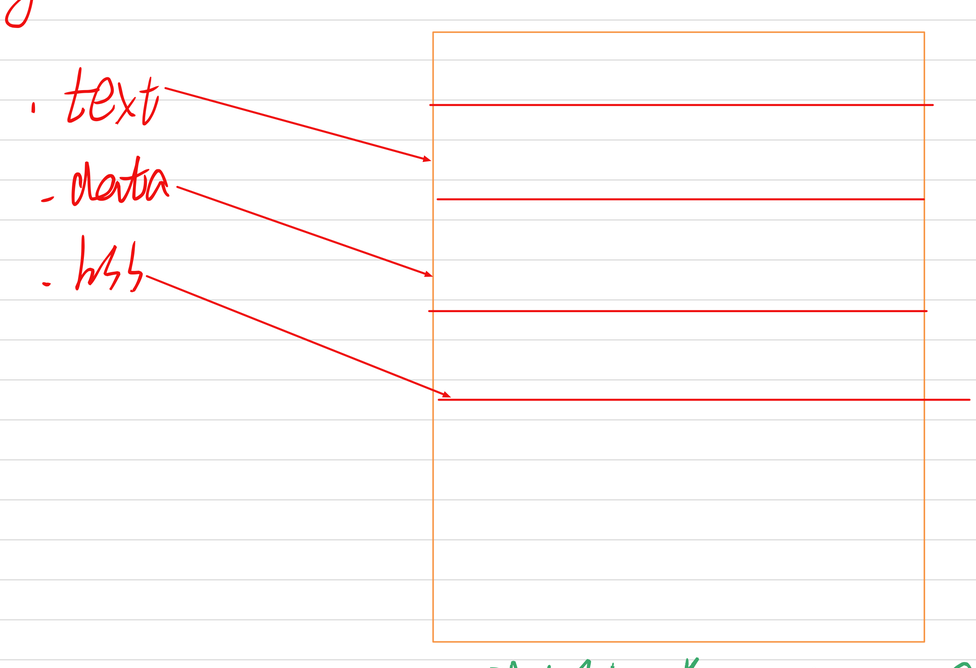
```
tony@tony-virtual-machine:~/code$ g++ -c sum.cpp -g
tony@tony-virtual-machine:~/code$ objdump -S sum.o

sum.o:     文件格式 elf64-x86-64


Disassembly of section .text:

0000000000000000 <_Z3sumii>:
int gdata = 10; // gdata .data
int sum(int a, int b) // sum_int_int  .text
{
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 7d fc                mov    %edi,-0x4(%rbp)
   7:   89 75 f8                mov    %esi,-0x8(%rbp)
        return a+b;
   a:   8b 55 fc                mov    -0x4(%rbp),%edx
   d:   8b 45 f8                mov    -0x8(%rbp),%eax
  10:   01 d0                   add    %edx,%eax
}
  12:   5d                      pop    %rbp
  13:   c3                      retq
```

a.out 在磁盘上，a.out有各种各样的段
elf header ⇐ 有程序的入口地址
program headers ⇐ .text .data 加载



进程的虚拟地址空间

MMU 虚拟地址 → 物理地址映射
系统出异常时，会执行地址映射的异常处理程序，分配物理内存