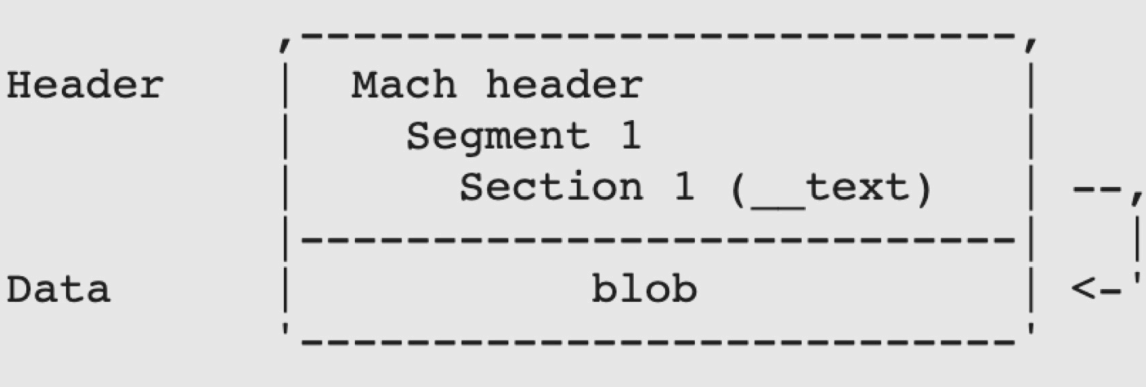
ABI Mach-O

`Mach-O(Mach Object)` 是 `macOS` 、 `iOS` 、 `iPadOS` 存储程序和库的文件格式。对应系统通过应用二进制接口（ `application binary interface` ，缩写为 `ABI` ）来运行该格式的文件。

`Mach-O` 格式用来替代 `BSD` 系统的 `a.out` 格式。 `Mach-O` 文件格式保存了在编译过程和链接过程中产生的机器代码和数据，从而为静态链接和动态链接的代码提供了单一文件格式。

段之前始终是4096字节或4 KB的倍数，其中4096字节是最小大小。
现在段是16 KB的倍数，在macOS_x86_64上是16k，在iOS上是32k。



Symbol Table

通过两个 `load commands` ：

- LC_SYMTAB: 当前 `Mach-O` 中的符号表信息。
- LC_DYSYMTAB: 描述动态链接器使用其他的 `Symbol Table` 信息。

用来描述 `Symbol Table` 的大小和位置，以及其他元数据。

`LC_SYMTAB`

用来描述该文件的符号表。不论是静态链接器还是动态链接器在链接此文件时，都要使用该 `load command` 。调试器也可以使用该 `load command` 找到调试信息。

symtab_command

定义 `LC_SYMTAB` 加载命令具体属性。在 `/usr/include/mach-o/loader.h` 中定义：

```
struct symtab_command {
  // 共有属性。指明当前描述的加载命令，当前被设置为LC_SYMTAB
  uint32_t cmd;
  // 共有属性。指明加载命令的大小，当前被设置为sizeof(symtab_command)
  uint32_t cmdsize;
  // 表示从文件开始到symbol table所在位置的偏移量。symbol table用[nlist]来表示
  uint32_t symoff;
  // 符号表内符号的数量
  uint32_t nsyms;
  // 表示从文件开始到string table所在位置的偏移量。
  uint32_t stroff;
  // 表示string table大小（以byte为单位）
  uint32_t strsize;
};
```

nlist

定义符号的具体表示含义：

```
struct nlist {
   // 表示该符号在string table的索引
```

```
    union {
       // 在Mach-O中不使用此字段
        char *n_name;
        // 索引
        long  n_strx;
    } n_un;
    unsigned char n_type;   /* type flag, see below */
    unsigned char n_sect;   /* section number or NO_SECT */
    short         n_desc;   /* see <mach-o/stab.h> */
    unsigned long n_value;  /* value of this symbol (or stab offset) */
};
```

`n_type`

1字节，通过四位掩码保存数据：

- `N_STAB(0xe0)`：如果当前的 `n_type` 包含这3位中的任何一位，则该符号为 `调试符号表(stab)` 。在这种情况下，整个 `n_type` 字段将被解释为 `stab value` 。请参阅 `/usr/include/mach-o/stab.h` 以获取有效的 `stab value` 。
- `N_PEXT(0x10)`：如果当前的 `n_type` 包含此位。则将此符号标记为私有外部符号 `__private_extern__(visibility=hidden)` ，只在程序内可引用和访问。当文件通过静态链接器链接的时候，不要将其转换成静态符号（可以通过 `ld` 的 `-keep_private_externs` 关闭静态链接器的这种行为）。
- `N_TYPE(0x0e)`：如果当前的 `n_type` 包含此位。则使用预先定义的符号类型。
- `N_EXT(0x01)`：如果当前的 `n_type` 包含此位。则此符号为外部符号.该符号在该文件外部定义或在该文件中定义，但可在其他文件中使用。

`N_TYPE`

`N_TYPE` 字段的值包括：
* `N_UNDF(0x0)`：该符号未定义。未定义符号是在当前模块中引用，但是被定义在其他模块中的符号。 `n_sect` 字段设置为 `NO_SECT` 。
* `N_ABS(0x2)`：该符号是绝对符号。链接器不会更改绝对符号的值。 `n_sect` 字段设置为 `NO_SECT` 。
* `N_SECT(0xe)`：该符号在 `n_sect` 中指定的段号中定义。

* `N_PBUD(0xc)`：该符号未定义，镜像使用该符号的预绑定值。 `n_sect` 字段设置为 `NO_SECT` 。
* `N_INDR(0xa)`：该符号定义为与另一个符号相同。 `n_value` 字段是 `string table` 中的索引，用于指定另一个符号的名称。链接该符号时，此符号和另一个符号都具有相同的定义类型和值。

`stab value` 包括：

```
#define N_GSYM  0x20    /* 全局符号: name,,NO_SECT,type,0 */
#define N_FNAME 0x22    /* procedure name (f77 kludge): name,,NO_SECT,0,0 */
#define N_FUN   0x24    /* 方法/函数: name,,n_sect,linenumber,address */
#define N_STSYM 0x26    /* 静态符号: name,,n_sect,type,address */
#define N_LCSYM 0x28    /* .lcomm 符号: name,,n_sect,type,address */
#define N_BNSYM 0x2e    /* nsect符号开始: 0,,n_sect,0,address */
#define N_OPT   0x3c    /* emitted with gcc2_compiled and in gcc source */
#define N_RSYM  0x40    /* 寄存器符号: name,,NO_SECT,type,register */
#define N_SLINE 0x44    /* 代码行数: 0,,n_sect,linenumber,address */
#define N_ENSYM 0x4e    /* nsect符号结束: 0,,n_sect,0,address */
#define N_SSYM  0x60    /* 结构体符号: name,,NO_SECT,type,struct_offset */
#define N_SO    0x64    /* 源码名称: name,,n_sect,0,address */
#define N_OSO   0x66    /* 目标代码名称: name,,0,0,st_mtime */
#define N_LSYM  0x80    /* 本地符号: name,,NO_SECT,type,offset */
#define N_BINCL 0x82    /* include file 开始: name,,NO_SECT,0,sum */
#define N_SOL   0x84    /* #included file 名称: name,,n_sect,0,address */
#define N_PARAMS  0x86  /* 编译器参数: name,,NO_SECT,0,0 */
#define N_VERSION 0x88  /* 编译器版本: name,,NO_SECT,0,0 */
#define N_OLEVEL  0x8A  /* 编译器 -O 级别: name,,NO_SECT,0,0 */
#define N_PSYM  0xa0    /* 参数: name,,NO_SECT,type,offset */
#define N_EINCL 0xa2    /* include file 结束: name,,NO_SECT,0,0 */
#define N_ENTRY 0xa4    /* alternate entry: name,,n_sect,linenumber,address */
#define N_LBRAC 0xc0    /* 左括号: 0,,NO_SECT,nesting level,address */
#define N_EXCL  0xc2    /* deleted include file: name,,NO_SECT,0,sum */
#define N_RBRAC 0xe0    /* 右括号: 0,,NO_SECT,nesting level,address */
#define N_BCOMM 0xe2    /* 通用符号开始: name,,NO_SECT,0,0 */
```

```
#define N_ECOMM 0xe4    /* 通用符号结束: name,,n_sect,0,0 */
#define N_ECOML 0xe8    /* end common (local name): 0,,n_sect,0,address */
#define N_LENG  0xfe    /* second stab entry with length information */

/*
 * for the berkeley pascal compiler, pc(1):
 */
#define N_PC    0x30    /* global pascal symbol: name,,NO_SECT,subtype,line */
```

`n_sect`

整数，用来在指定编号的 `section` 中找到此符号；如果在该 `image` 的任何部分都找不到该符号，则为 `NO_SECT` 。根据 `section` 在 `LC_SEGMENT` 加载命令中出现的顺序，这些 `section` 从1开始连续编号。

`n_desc`

`16-bit` 值，用来描述非调试符号。低三位使用 `REFERENCE_TYPE` ：

- `REFERENCE_FLAG_UNDEFINED_NON_LAZY(0x0)` ：该符号是外部非延迟（数据）符号的引用。
- `REFERENCE_FLAG_UNDEFINED_LAZY(0x1)` ：该符号是外部延迟性符号（即对函数调用）的引用。
- `REFERENCE_FLAG_DEFINED(0x2)` ：该符号在该模块中定义。
- `REFERENCE_FLAG_PRIVATE_DEFINED(0x3)` ：该符号在该模块中定义，但是仅对该共享库中的模块可见。
- `REFERENCE_FLAG_PRIVATE_UNDEFINED_NON_LAZY(0x4)` ：该符号在该文件的另一个模块中定义，是非延迟加载（数据）符号，并且仅对该共享库中的模块可见。
- `REFERENCE_FLAG_PRIVATE_UNDEFINED_LAZY(0x5)` ：该符号在该文件的另一个模块中定义，是延迟加载（函数）符号，仅对该共享库中的模块可见。

另外还可以设置如下标识位：

- `REFERENCED_DYNAMICALLY(0x10)` ：定义的符号必须是使用在动态加载器中（例如 `dlsym` 和 `NSLookupSymbolInImage` ）。而不是普通的未定义符号引用。 `strip` 使用该位来避免删除那些必须存在的符号（如果符号设置了该位，则 `strip` 不会剥离它）。

- `N_DESC_DISCARDED(0x20)`：在完全链接的 `image` 在运行时动态链接器有可能会使用此符号。**不要在完全链接的 `image` 中设置此位。**
- `N_NO_DEAD_STRIP(0x20)`：定义在可重定位目标文件（类型为 `MH_OBJECT` ）中的符号设置时，指示静态链接器不对该符号进行 `dead-strip` 。（请注意，与 `N_DESC_DISCARDED(0x20)` 用于两个不同的目的。）
- `N_WEAK_REF(0x40)`：表示此未定义符号是弱引用。如果动态链接器找不到该符号的定义，则将其符号地址设置为0。静态链接器会将此符号设置弱链接标志。
- `N_WEAK_DEF(0x80)`：表示此符号为弱定义符号。如果静态链接器或动态链接器为此符号找到另一个（非弱）定义，则弱定义将被忽略。只能将合并部分中的符号标记为弱定义。

如果该文件是两级命名 `two-level namespace image` （即如果 `mach_header` 中设置了 `MH_TWOLEVEL` 标志），则 `n_desc` 的高8位表示定义此未定义符号的库的编号。使用宏 `GET_LIBRARY_ORDINAL` 来获取此值，或者使用宏 `SET_LIBRARY_ORDINAL` 来设置此值。0指定当前 `image` 。 1到253根据文件中 `LC_LOAD_DYLIB` 命令的顺序表明库号。254用于需要动态查找的未定义符号（仅在OS X v10.3和更高版本中受支持）。对于从可执行程序加载符号的插件。255，用来指定可执行 `image` 。对于 `flat namespace images` ， 高8位必须为0。

`n_value`

符号值。对于 `symbol table` 中的每一项，该值的表达的意思都不同（具体由 `n_type` 字段说明）。对于 `N_SECT` 符号类型， `n_value` 是符号的地址。有关其他可能值的信息，请参见 `n_type` 字段的描述。

> *`Common symbols` 必须为 `N_UNDF` 类型，并且必须设置 `N_EXT` 位。 `Common symbols` 的 `n_value` 是符号表示的数据的大小（以字节为单位）。在 `C` 语言中， `Common symbol` 是在该文件中声明但未初始化的变量。 `Common symbols` 只能出现在 `MH_OBJECT` 类型的 `Mach-O` 文件中。*

section名称与作用

名称	作用
TEXT.text	可执行的机器码
TEXT.cstring	去重后的C字符串
TEXT.const	初始化过的常量

TEXT.stubs	符号桩。lazybinding的表对应项指针指向的地址的代码。
TEXT.stub_helper	辅助函数。当在lazybinding的表中没有找到对应项的指针表示的真正的符号地址的时候，指向这。
TEXT.unwind_info	存储处理异常情况信息
TEXT.eh_frame	调试辅助信息
DATA.data	初始化过的可变的数据
DATA.nl_symbol_ptr	非lazy-binding的指针表，每个表中的指针指向一个在装载过程中，被动态链接器搜索完成的符号
DATA.la_symbol_ptr	lazy-binding的指针表，每个表中的指针一开始指向stub_helper
DATA.const	没有初始化过的常量
DATA.mod_init_func	初始化函数，在main之前调用
DATA.mod_term_func	终止函数，在main返回之后调用
DATA.bss	没有初始化的静态变量
DATA.common	没有初始化过的符号声明(for example, int i;)

nm命令

打印 `nlist` 结构的符号表(symbol table)。

常用nm命令参数

```
nm -pa a.o
-a: 显示符号表的所有内容
-g: 显示全局符号
-p: 不排序。显示符号表本来的顺序
-r: 逆转顺序
```

```
-u: 显示未定义符号
-m:  显示N_SECT类型的符号(Mach-O符号)显示。
```