# Introduction介绍

这一节描述对象文件的格式，被称为ELF(Executable and Linking Format).主要有三种类型的对象文件：

* 可重定位文件：保存适合于与其他对象文件链接以创建可执行文件或可共享对象文件的代码和数据。
* 可执行文件保存适合于执行的程序。
* 可共享对象文件保存适于在两种上下文环境中进行链接的代码和数据。 首先，链接编辑器可以链接它和其他可重定位或共享对象文件来来创建一个新的对象文件。其次，动态链接器将它与可执行文件和其他可共享对象进行组合来创建过程映像。

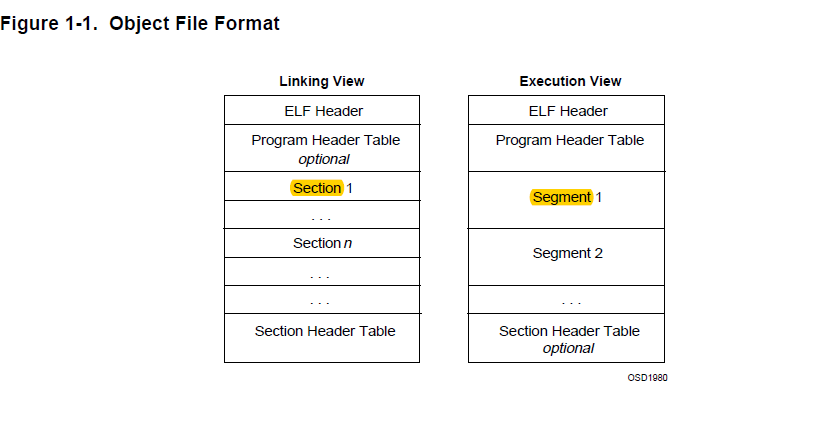
由汇编器和链接编辑器创建，目标文件是能够直接在处理器上执行的程序的二进制表示。

在介绍材料之后，本章重点介绍文件格式以及它与构建程序的关系。 第2章也描述了目标文件的部分内容，集中介绍于执行程序所需的信息。

# File Format对象文件格式

## 对象文件格式

对象文件参与程序链接（构建程序）和程序执行（运行程序）。 为了方便和高效，目标文件格式同时提供展示文件内容的两种视图，反映了两种活动（构建程序和运行程序）的不同需求。 如图1-1所示一个目标文件的组织。



## ELF header

ELF header位于目标文件的开始处，它保存描述对象文件的组织结构的“路线图”。Section包含用于链接视图的对象文件信息：指令Section (instructions)，数据Section (data)，符号表Section (symbol table)，重定位信息Section (relocation information)，等等。本节后面对各个Section进行详细描述。第二章也将介绍各个Segments和目标文件的程序执行视图。

## program header table

程序头表（program header table）（如果存在的话）告诉系统如何创建进程映像。用于构建进程映像的目标文件（执行程序）必须有程序头表（program header table）。可重定位文件（relocatable file）则并不需要程序头表（program header table）。

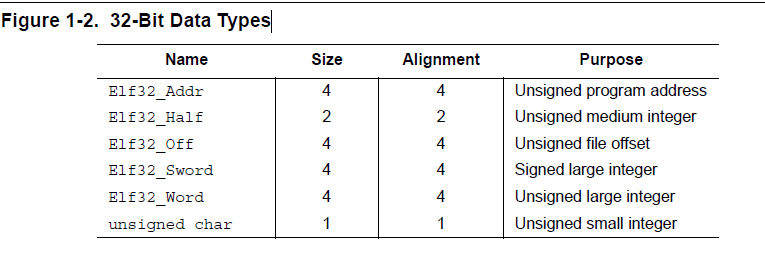
## section header table

段头表（section header table）包含描述各个Section的信息。每个Section在表中有一个对应的条目; 每个条目给出该Section的名称，大小等信息。 用于链接期间使用的目标文件必须具有段头表（section header table）; 其他目标文件可能有也可能没有。

注意。 虽然图中显示了程序头表(program header table)位于ELF头（ELF header）的后面，section header table位于各sections,的后面，实际的目标文件这几个部分的顺序关系可能与上图中展示的不同，可以确定的是ELF头（ELF header）一定位于文件的头部。

# Data Representation数据表示

如这里所描述的，目标文件格式支持具有8位字节和32位架构的各种处理器。 然而，它旨在可扩展到更大（或更小）的架构。因此，对象文件使用与机器无关的格式来表示它的控制数据，使得可以以通用方式来识别对象文件并解释其内容。 对象文件中的剩余数据使用目标处理器的编码，而不考虑创建文件的计算机。



对象文件格式定义的所有数据结构都遵循相关类的“自然”大小和对齐准则。如果必要，数据结构包含显式填充，以确保4字节对象的4字节对齐（32位处理器一次读取32位），将结构大小强制为 倍数为4，依此类推。数据也从文件的开头具有适当的对齐。 因此，例如，包含Elf32\_Addr成员的结构将在文件内的4字节边界上对齐。

出于可移植性的原因，ELF不使用位字段。

# Character Representations字符表示

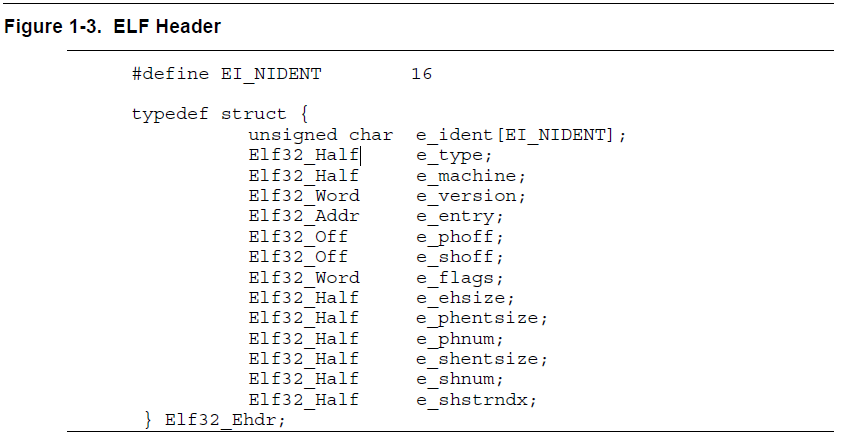
本节介绍ELF默认的字符表示，并为能够在不同系统之间移植的外部文件定义标准字符集。一些外部文件格式使用字符来表示控制信息。 这些单字节字符使用7位ASCII字符集。 换句话说，当ELF接口文档提到字符常量，例如'/'或'\ n'时，它们的数值应遵循7位ASCII指令。对于前面的字符常量，单字节值将是47 和10。

根据字符编码，0到127范围之外的字符值可能占用一个或多个字节。 应用程序可以对不同的语言使用不同的字符集扩展来控制它的字符集。 虽然TIS-conformance不限制字符集，但它们通常应遵循一些简单的准则：

* 0到127之间的字符值应对应于7位ASCII码。 也就是说，具有高于127的编码的字符集应该将的7位ASCII码作为子集。
* 值大于127的多字节字符编码应该只包含值在0到127范围之外的字节。也就是说，每个字符使用多个字节的字符集不应该“嵌入”类似于7位ASCII字符的字节 在一个多字节，非ASCII字符内。
* 多字节字符应该是自识别的。 这允许例如在任何多字节字符对之间插入任何多字节字符，而不改变字符的解释。

这些注意事项与多语言应用程序特别相关。

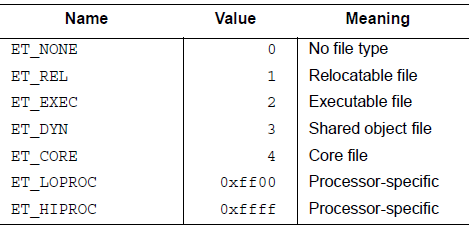
# ELF Header



## e\_ident

初始字节将文件标记为目标文件，并提供与机器无关的数据，用于解码和解释文件的内容。 完整的描述显示在下面的“ELF标识”中。

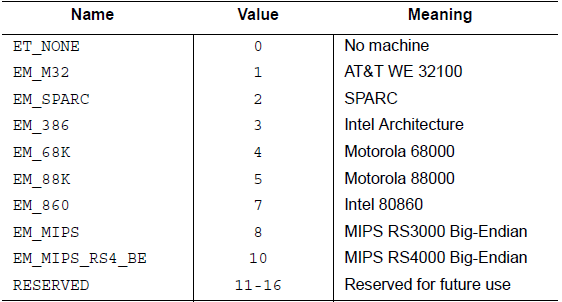
## e\_type

此成员标识对象文件的类型。

虽然Core文件的内容未指定，但保留类型ET\_CORE以标记这种文件类型。 从ET\_LOPROC到ET\_HIPROC（包括）的值保留用于处理器特定的语义。 保留其他值，并根据需要将其分配给新的对象文件类型。

## e\_machine

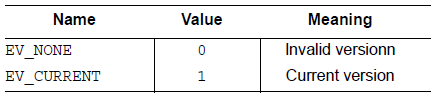
此成员的值指定单个文件所需的CPU体系结构。

保留其他值，并根据需要将其分配给新计算机。特定于处理器的ELF名称使用计算机名称来区分它们。 例如，下面提到的标志使用前缀EF\_; 一个名为WIDGET的标志

EM\_XYZ机器将被称为EF\_XYZ\_WIDGET。

## e\_version

此成员标识对象文件版本。



值1表示原始文件格式; 扩展将创建具有更高数字的新版本。 EV\_CURRENT的值，尽管在上面给出为1，将根据需要更改以反映当前版本号。

## e\_entry

该成员给出系统首次传输控制的虚拟地址，从而启动进程。 如果文件没有关联的入口点（entry point），则此成员保存为零。

## e\_phoff

该成员保存程序头表（program header table）的文件偏移量（以字节为单位）。 如果文件没有程序头表，则此成员保存为零。

## e\_shoff

该成员保存section header table的文件偏移量（以字节为单位）。 如果文件没有section header table，则此成员保存为零。

## e\_flags

此成员保存与文件相关联的处理器特定标志。 标志名的格式为EF\_machine\_flag。

## e\_ehsize

该成员保存ELF头的大小（以字节为单位）。

## e\_phentsize

该成员保存文件程序头表（program header table）中一个条目的大小（以字节为单位）;所有条目的大小相同。

## e\_phnum

此成员保存程序头表（program header table）中的条目数。 因此，e\_phentsize和e\_phnum的乘积给出了program header table的大小（以字节为单位）。 如果文件没有program header table，e\_phnum保持值为零。

## e\_shentsize

该成员保存段头section header的大小（以字节为单位）。 段头section header是section header table中的一个条目; 所有条目的大小相同。

## e\_shnum

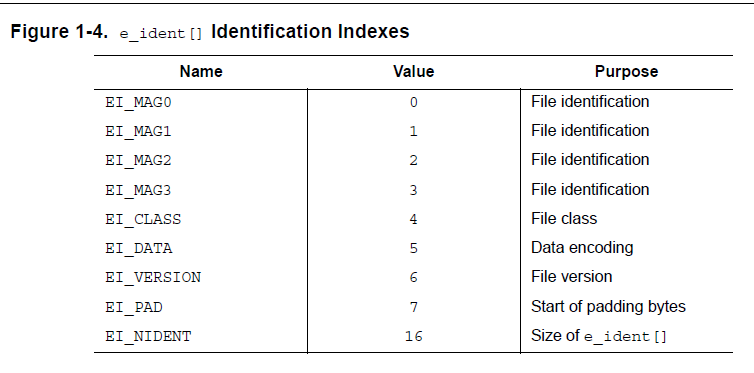
此成员保存section header table中的条目数。 因此，e\_shentsize和e\_shnum的乘积给出了section header table的大小（以字节为单位）。 如果文件没有section header table，e\_shnum保持值为零。

## e\_shstrndx

此成员保存与节名称字符串表（section name string table）相关联的条目在节标题表（section header table）中的索引值。 如果文件没有节名称字符串表（section name string table），则此成员保存值SHN\_UNDEF。 有关详细信息，请参阅下面的“Sections”和“String Table”章节。

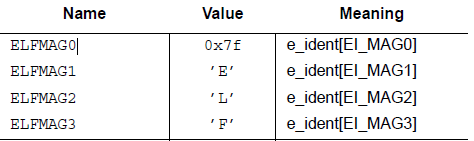
# ELF Identification

如上所述，ELF提供了一个目标文件框架来支持多种处理器，多种数据编码和多类机器。为了支持该对象文件系列，文件的初始字节指定如何解释文件，它独立于进行查询的处理器，并且独立于文件的剩余内容。ELF头（和目标文件）的初始字节对应于e\_ident成员。



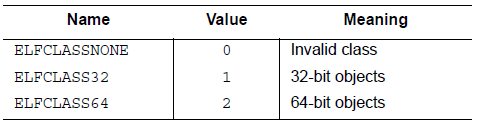
这些索引访问包含以下值的字节。

## EI\_MAG0到EI\_MAG3

文件的前4个字节组合起来表示文件的“magic number”，将该文件标识为ELF对象文件。

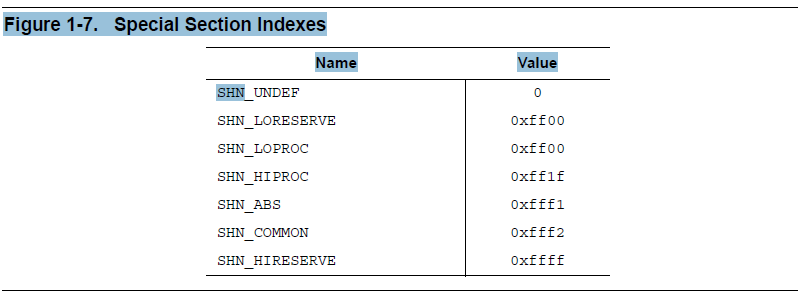
## EI\_CLASS

下一个字节e\_ident [EI\_CLASS]标识文件的类或容量。



# Sections

目标文件的section header table（段头表）允许定位该文件所有的section段。 section header table（段头表）是如下所述的Elf32\_Shdr结构的数组，section header table的索引是这个数组的下标。ELF header的e\_shoff成员给出从文件开头到section header table的字节偏移; e\_shnum告诉section header table包含多少条目; e\_shentsize给出每个条目的大小（以字节为单位）。



## SHN\_UNDEF

## SHN\_LORESERVE

此值指定保留索引范围的下限值。

## SHN\_LOPROC through SHN\_HIPROC

在SHN\_LOPROC到SHN\_HIPROC范围中的值为处理器特定的语义所保留。

## SHN\_ABS

此值指定相应引用的绝对值。 例如，相对于section号SHN\_ABS定义的符号具有绝对值，并且不受重定位的影响。

## SHN\_COMMON

相对于此section定义的symbols符号是common symbols,常用符号，例如FORTRAN 语言中的COMMON或C语言中未分配的外部变量。

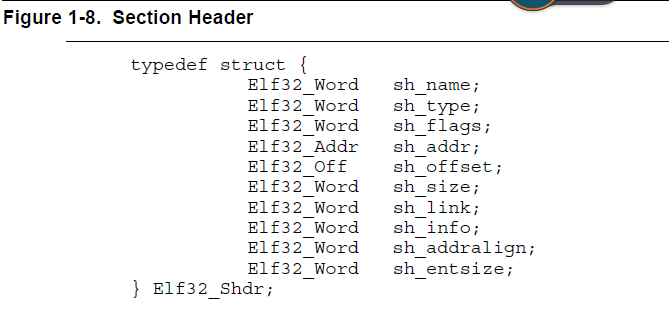
## SHN\_HIRESERVE

此值指定reserved indexes保留索引范围的上限。 系统保留SHN\_LORESERVE和SHN\_HIRESERVE之间的索引，包括; 这些值不被section header table段头表引用。也就是说，段头表不包含保留索引的条目。

在目标文件中，除了ELF头，program header table程序头表和section header table段头表，其它节的所有信息包含在section中。 此外，目标文件段满足几个条件。

* 对象文件中的每个节都有一个描述它的section header节头。 可能存在没有section的section header。
* 每个section占据文件中的一个连续（可能为空）字节序列。
* 文件中的section不能重叠。 文件中没有字节驻留在多个section中。
* 对象文件可能具有inactive space未使用的空间。 各种header头和section可能不会“覆盖”目标文件中的每个字节。未使用的空间的数据的内容是未指定的。

# Section header的结构如下：



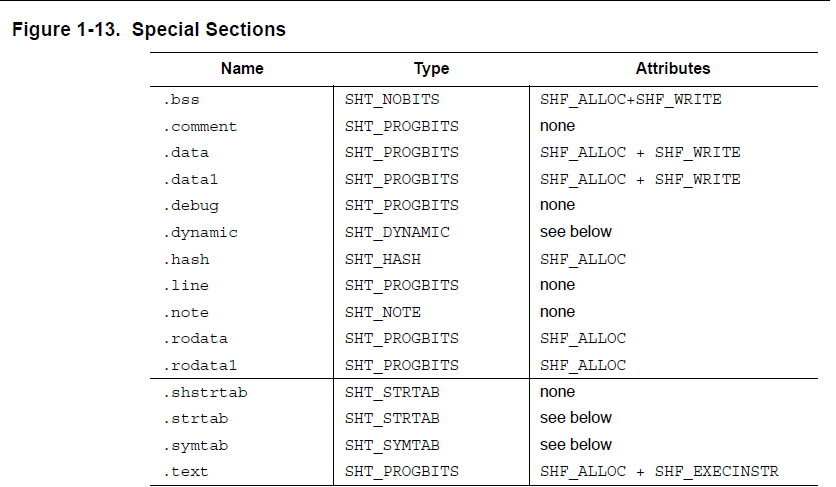
## sh\_name

此成员指定section的名称。 它的值是section header string table section的索引[见下面的“字符串表”），给出一个以null结束的字符串的位置。

## sh\_type

此成员对section的content内容和semantic语义进行分类。 Section类型及其说明如下所示。

# Special Sections

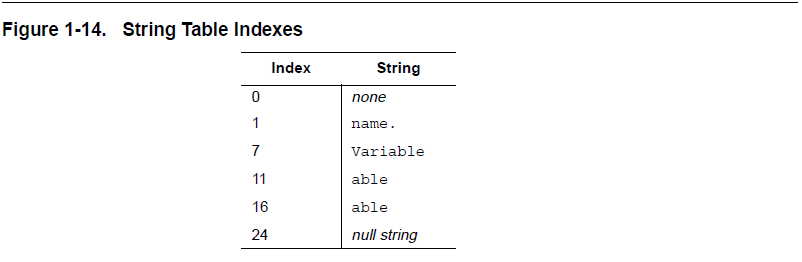
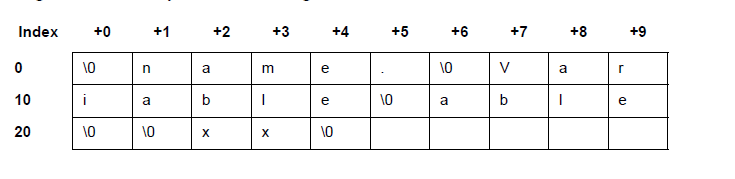


# String Table

本节介绍默认string table字符串表。string table字符串表段保存以空字符终止的字符序列，通常称为string字符串。 object file对象文件使用这些string字符串来表示symbol符号和section name节名称。 通过string table section的下标来引用一个将字符串。

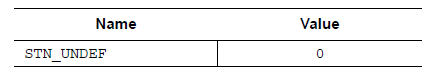
第一个字节，即索引为零，定义为保存一个空字符。 同样，string table字符串表的最后一个字节被定义为保持一个空字符，确保所有字符串的以空字符终止。 索引为零的字符串指定无名称或空名称，具体取决于上下文。 允许使用空的string table section字符串表部分; 它的section header段头的sh\_size成员将为零。 非零索引对于空string table section无效。

section header的sh\_name成员表示该section的名称，它的值是一个索引，表示该section的名称在section name string table中的索引值，section name string table在section header的索引值（位置）由ELF头部的e\_shstrndx成员指定。 下图显示了具有25个字节的字符串表以及与各种索引关联的字符串。

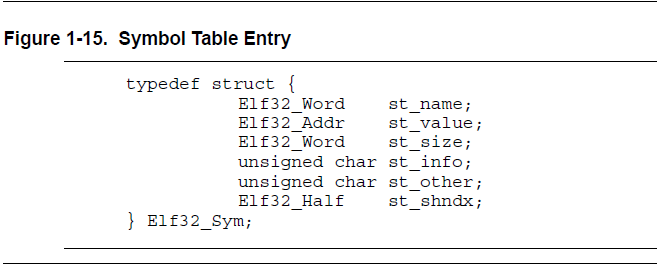
如示例所示，string table index字符串表索引可以引用section段中的任何字节。 字符串可能出现多次; 可能存在对子字符的引用; 并且单个字符串可以被引用多次。 还允许未被引用的字符串。

# Symbol Table符号表

object file对象文件的symbol table符号表包locate定位和relocate重定位程序的symbolic definitions符号定义和reference引用所需的信息。 符号表索引是此数组的下标。 索引0指定表中的第一个条目，并用作未定义的符号索引。 符号表的初始条目的内容在本节后面指定。



符号表条目的格式如下：



## st\_name

此成员的值是在object file目标文件的symbol string table符号字符串表中的索引，该表保存symbol names符号名称的字符表示。

## st\_value

该成员给出了相关联的symbol符号的值。 根据上下文，这可以是absolute value绝对值，地址等; 详细信息显示如下。

## st\_size

许多符号具有大小。 例如，数据对象的大小是对象中包含的字节数。 如果符号没有大小或未知大小，则此成员保留0。

## st\_info

此成员指定符号的类型和绑定属性。 下面显示了值和含义的列表。 以下代码显示如何处理值。

## st\_shndx

每个symbol table entry符号表条目都是相对于某个section段“定义”的;这个成员表示相关联的section在section header table节头表的索引值。 如图1-7和相关文本所述，一些段索引表示特殊含义。

符号的绑定决定了链接的可见性和行为。

