中使用 public 和 private 声明一样。在 C 中，源文件扮演模块的角色。任何带有 static 属性声明的全局变量或者函数都是模块私有的。类似地，任何不带 static 属性声明的全局变量和函数都是公共的，可以被其他模块访问。尽可能用 static 属性来保护你的变量和函数是很好的编程习惯。

符号表是由汇编器构造的，使用编译器输出到汇编语言.s 文件中的符号。.symtab 节中包含 ELF 符号表。这张符号表包含一个条目的数组。图 7-4 展示了每个条目的格式。

code/link/elfstructs.c

```
1  typedef struct {
2      int   name;      /* String table offset */
3      char  type:4,    /* Function or data (4 bits) */
4            binding:4; /* Local or global (4 bits) */
5      char  reserved;  /* Unused */
6      short section;   /* Section header index */
7      long  value;     /* Section offset or absolute address */
8      long  size;      /* Object size in bytes */
9  } Elf64_Symbol;
```

code/link/elfstructs.c

图 7-4 ELF 符号表条目。type 和 binding 字段每个都是 4 位

name 是字符串表中的字节偏移，指向符号的以 null 结尾的字符串名字。value 是符号的地址。对于可重定位的模块来说，value 是距定义目标的节的起始位置的偏移。对于可执行目标文件来说，该值是一个绝对运行时地址。size 是目标的大小(以字节为单位)。type 通常要么是数据，要么是函数。符号表还可以包含各个节的条目，以及对应原始源文件的路径名的条目。所以这些目标的类型也有所不同。binding 字段表示符号是本地的还是全局的。

每个符号都被分配到目标文件的某个节，由 section 字段表示，该字段也是一个到节头部表的索引。有三个特殊的伪节(pseudosection)，它们在节头部表中是没有条目的：ABS 代表不该被重定位的符号；UNDEF 代表未定义的符号，也就是在本目标模块中引用，但是却在其他地方定义的符号；COMMON 表示还未被分配位置的未初始化的数据目标。对于 COMMON 符号，value 字段给出对齐要求，而 size 给出最小的大小。注意，只有可重定位目标文件中才有这些伪节，可执行目标文件中是没有的。

COMMON 和.bss 的区别很细微。现代的 GCC 版本根据以下规则来将可重定位目标文件中的符号分配到 COMMON 和.bss 中：

COMMON　*未初始化的全局变量*

.bss　*未初始化的静态变量，以及初始化为 0 的全局或静态变量*

采用这种看上去很绝对的区分方式的原因来自于链接器执行符号解析的方式，我们会在 7.6 节中加以解释。

GNU READELF 程序是一个查看目标文件内容的很方便的工具。比如，下面是图 7-1 中示例程序的可重定位目标文件 main.o 的符号表中的最后三个条目。开始的 8 个条目没有显示出来，它们是链接器内部使用的局部符号。

```
Num:    Value          Size Type    Bind   Vis      Ndx Name
  8: 0000000000000000    24 FUNC    GLOBAL DEFAULT    1 main
  9: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    3 array
 10: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND sum
```