lib文件格式

如果你试着做一个应用程序的连接器（Linker），就会发现，仅仅有目标文件是不够的。我们在连接程序时，不仅仅要用到目标文件，库文件也是必不可少的。

库文件是怎么样的结构呢？

其实，库文件的结构也很简单。它就是“一堆”目标文件的集合。把目标文件做成库以后，我们在使用目标文件中所实现的功能时，连接程序会自动在库文件里查找相应的目标文件，并使用它。这大大减少了我们对目标文件的管理工作，减轻了代码重用的负担。

Lib文件中的节

COFF格式中所用到的“节”的概念再次出现在Lib格式中。不过，Lib文件的节要简单得多。先让我们来看看它的整体结构：

如右图所示：

Signature

First Sec

Second Sec

Longname Sec

Obj Sec1

Obj Sec2

……

Lib格式只有四种类型的节（Section），即First Sec，Second Sec，Longname Sec和Obj Sec；其中Second Sec与Longname Sec是可选节，很多Lib文件中都没有。而开头的Singature只是一个标识，它相当于COFF目标文件中的魔法数字。它是一个长度为8的字符串，值为“!<arch>/n”。

First Sec，顾名思义，就是第一个节。它包含了库中所有的符号名以及这些符号所在的目标文件在库中的位置（绝对偏移）。

Second Sec就是第二节。它的内容和First Sec是相同的。不同的是，Second Sec是一个有序表，通过它来查找库中的符号比通过First Sec来查找要快很多。

Longname Sec是长名称节。这一节是一个字符串表。它包含了所有长目标文件名。如果后面的Obj Sec中没有给出相应的目标文件名，我们就要到这一节中来查找。

Obj Sec就是目标文件节。这些节中存储着不同的目标文件的原始数据。

在库文件中，每一节都有两个部分。一个部分是头，另一个部分才是该节的数据；数据紧跟在头的后面。头描述了该节数据的类型、长度等信息。这些头的格式都是相同的。其结构用C语言描述如下：

typedef struct {

char Name[16]; // 名称

char Time[12]; // 时间

char UserID[6]; // 用户ID

char GroupID[6]; // 组ID

char Mode[8]; // 模式

char Size[10]; // 长度

char EndOfHeader[2];// 结束符

} SectionHeader;

可以看到，头中的数据全都是字符串。用字符串的好处是可以提高格式的兼容性，因为在不同的机器上，数据的排列方式是不同的。有的机器是以Little-Endian方式工作，还有的是以Big-Endian方式工作，它们互不兼容（这两种方式的区别！？请看我的《COFF格式》一文，其中的文件头一节有说明）。用字符串就不会有这种问题（后面我们将会遇到）。但它也有不方便的地方，就是必须把字符串转换成数值，多了一个步骤。

在这个结构中，最常用的Name、Size以及EndOfHeader三个成员。Name就是节的名称啦！Size也很好理解，就是该节数据的长度。现在要注意的就是这个EndOfHeader成员了！这个成员标志着头的结束，其内容为“`/n”（注意，这里没有打错，是两个字符“`”和“/n”）。怎么样？有点奇怪吧？为什么要有这个结束符？每一节的头长度一定，每节中的数据长度也知道。按顺序向下读不行吗？答案是：不行！因为每一节之间存在间隙！通常是一个字节或零个字节。如果是零个字节倒好，按顺序向下读是OK的。可是如果不为零的话，这样读就要错位了。要知道错位没有，只好用一个结束符来定位了。如果在读头的时候发现结束符不对，那就要一个字节一个字节地向下查找，直到找到结束符，才能算是对齐了。切记！切记！

当然，通过First Sec或Second Sec中给出的偏移来读数据就不存在这个问题。不会发生错位，放心读吧！

现在让我们来看看每一节中的数据是什么样子。

First Sec

第一节，通常就是Lib中的每一个小节。它的名称是“/”。其数据部分的结构如下：

typedef struct {

unsigned long SymbolNum; // 库中符号的数量

unsigned long SymbolOffset[n]; // 符号所在目标节的偏移

char StrTable[m]; // 符号名称字符串表

}FirstSec;

第一个成员SymbolNum是符号的数量。注意！它是以Big-Endian方式储存的（x86平台上的数据是以Little-Endian方式储存的。这里应该注意转换。后面给出的convert函数可以在Little-Endian格式与Big-Endian格式之间进行相互转换）。

第二个成员SymbolOffset是一个数组，它的长度n就是符号的数量，也就是SymbolNum。这个数组储存了每一个符号所在的目标节的偏移。我们可以方便地通过它来查找符号所在的目标文件。注意！它也是以Big-Endian格式储存的。

第三个成员StrTable是一个字符串表，它的长度m就是SectionHeader.Size的值减去（SymbolNum+1）\*4。其结构很简单，就是一堆以‘/0’结尾的字符串（和COFF文件中的字符串表结构相同）。在有的系统中，它还可能是以“//n”这两个字符结尾的字符串的集合。

很简单的一个结构，不过有两个成员的长度是不定的。怎么才能方便地从Lib中读出这些数据，留给大家自己想吧！下面我只给出一个进行Little-Endian与Big-Endian互转的函数。

inline void convert(void \* p // 要转换的数据的指针

,size\_t size = 4 // 数据的长度，long为4，short为2

) {

char \* buf=(char\*)p;

char temp;

for ( size\_t i=0;i<size/2;i++ ) {

temp=buf[i];

buf[i]=buf[size-i-1];

buf[size-i-1]=temp;

}

}

Second Sec

现在看看第二节。

这一节与第一节很相似！它通常也就是Lib文件的第二个节。它的名字也是“/”（注意：文件中第一个叫“/”的节是第一节，第二个就是第二节）。不过它的结构与第一节有些不同，如下：

typedef struct {

unsigned long ObjNum; // Obj Sec的数量

unsigned long ObjOffset[x]; // 每一个Obj Sec的偏移

unsigned long SymbolNum; // 库中符号的数量

unsigned short SymbolIdx[n]; // 符号在ObjOffset表中的索引

char StrTable[m]; // 符号名称字符串表

}SecondSec;

第一个成员ObjNum是库中Obj Sec的数量。

第二个成员ObjOffset是一个偏移表，它记录了库中所有Obj Sec的偏移。这个表的记录数x就是ObjNum。

第三个成员SymbolNum与First Sec中的SymbolNum意义相同。

第四个成员SymbolIdx变成了一个索引，它记录了相应名称字符串在ObjOffset这个表中的位置，我们要通过两次索引才能找到我们所要符号的Obj Sec位置。它的项目数n为SymbolNum。但请注意，这个索引是unsigned short型，不再是unsigned long型。

第五个成员StrTable结构与First Sec中的一样。不过，它的长度m为SectionHeader.Size的值减去((ObjNum+1)\*4+(SymbolNum+2)\*2)。

值得注意的是，这里的所有数据都是Little-Endian格式的。千万不要弄错了！

Longname Sec

这个小节就是一个字符串表，它的名称为“//”，其结构同FirstSec.StrTable。这里就不多说了。

Obj Sec

这一节中的数据就是COFF文件的原始数据，把它读出来存成文件，就是一个COFF文件。它的格式请参考《COFF格式》一文。

要指出的是它的命名方式有些特殊。如果Obj文件的名称少于16个字符，它就会被保存在SectionHeader的Name成员中，以‘/’字符结尾。如果无法保存在Name成员中，则Name成员的第一个字符就为‘/’，之后再跟上这个名称在Longname Sec中的偏移。

例如：

!<arch>/n

……

LongName Sec:

This\_Is\_Long\_Name0001/0

This\_Is\_Long\_Name0002/0

……

Obj Sec1:

Name[16]：“shortname/”

……

Obj Sec2:

Name[16]：“/0” // 这里使用了第一个长文件名This\_Is\_Long\_Name0001

……

Obj Sec3:

Name[16]：“/22” // 这里使用了第二个长文件名This\_Is\_Long\_Name0002