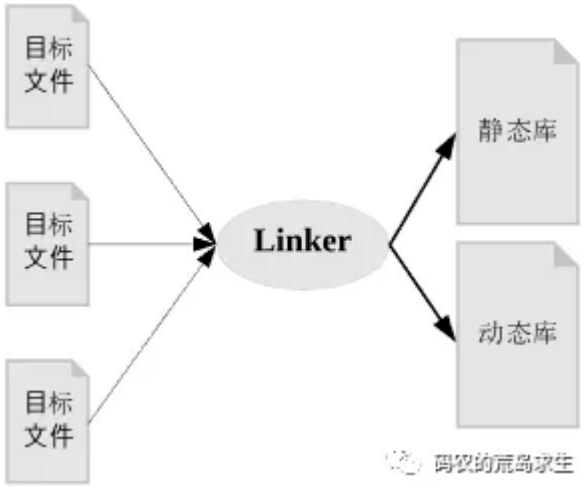
# 彻底理解链接器：三，库与可执行文件的生成

Original 码农的荒岛求生 码农的荒岛求生 2018-09-21 21:00

收录于话题
#链接器
6个 >

在链接器可操作的元素这一节中我们提到，链接器可以操作的最小单元为目标文件，也就是说我们见到的无论是静态库、动态库、可执行文件，都是基于目标文件构建出来的。目标文件就好比乐高积木中最小的零部件。

给定目标文件以及链接选项，链接器可以生成两种库，分别是静态库以及动态库，如图所示，给定同样的目标文件，链接器可以生成两种不同类型的库，接下来我们分别介绍。




## 静态库

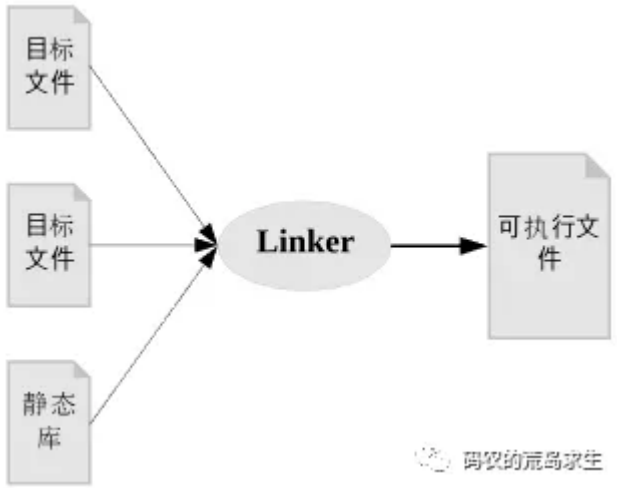
假设这样一个应用场景，基础设计团队设计了好多实用并且功能强大的工具函数，业务团队需要用到里面的各种函数。每次新添加其中一个函数，业务团队都要去找相应的实现文件并修改链接选项。使用静态库就可以解决这个问题。静态库在Windows下是以.lib为后缀的文件，Linux下是以.a为后缀的文件。

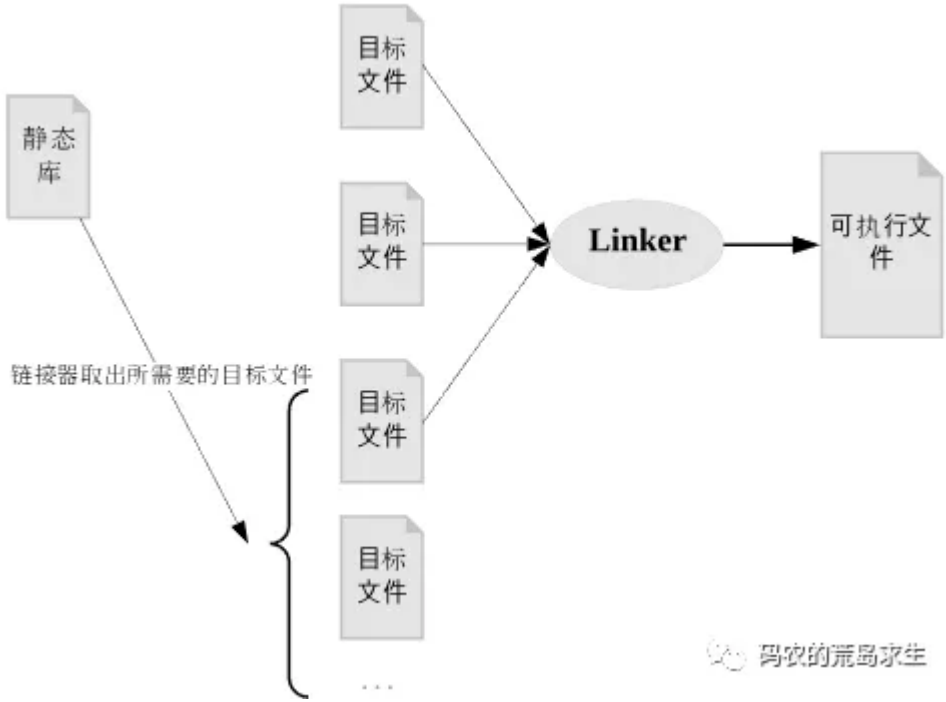
为解决上述问题，基础设计团队可以提前将工具函数集合打包编译链接成为静态库提供给业务团队使用，业务团队在使用时只要链接该静态库就可以了，每次新使用一个工具函数的时候，只要该函数在此静态库中就无需进行任何修改。

你可以简单的将静态库理解为由一堆目标文件打包而成， 使用者只需要使用其中的函数而无需关注该函数来自哪个目标文件（找到函数实现所在的目标文件是链接器来完成的，从这里也可以看出，不是所有静态库中的目标文件都会用到，而是用到哪个链接器就链接哪个）。静态库极大方便了对其它团队所写代码的使用。

## 静态连接

静态库是链接器通过静态链接将其和其它目标文件合并生成可执行文件的，如下图一所示，而静态库只不过是将多个目标文件进行了打包，在链接时只取静态库中所用到的目标文件，因此，你可以将静态链接想象成如下图2所示的过程。

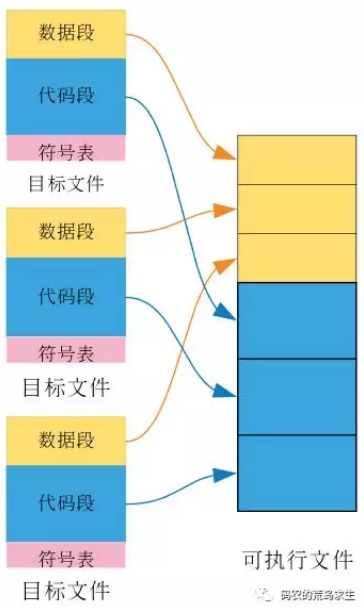





静态库是使用库的最简单的方法，如果你想使用别人的代码，找到这些代码的静态库并简单的和你的程序链接就可以了。静态链接生成的可执行文件在运行时不依赖任何其它代码，要理解这句话，我们需要知道静态链接下，可执行文件是如何生成的。

## 静态链接下可执行文件的生成

在上一节中我们知道，可以将静态链接简单的理解为链接器将使用到的目标文件集合进行拼装，拼装之后就生成了可执行文件，同时我们在目标文件里有什么这一节中知道，目标文件分成了三段，代码段，数据段，符号表，那么在静态链接下可执行文件的生成过程如图所示：




从上图中我们可以看到可执行文件的特点：

- 可执行文件和目标文件一样，也是由代码段和数据段组成。
- 每个目标文件中的数据段都合并到了可执行文件的数据段，每个目标文件当中的代码段都合并到了可执行文件的代码段。
- 目标文件当中的符号表并没有合并到可执行文件当中，因为可执行文件不需要这些字段。

可执行文件和目标文件没有什么本质的不同，可执行文件区别于目标文件的地方在于，可执行文件有一个入口函数，这个函数也就是我们在C语言当中定义的main函数，main函数在执行过程中会用到所有可执行文件当中的代码和数据。而这个main函数是被谁调用执行的呢，答案就是操作系统(Operating System)，这也是后面文章当中要重点介绍的内容。

现在你应该对可执行文件有一个比较形象的认知了吧。你可以把可执行文件生成的过程想象成装订一本书，一本书中通常有好多章节，这些章节是你自己写的，且一本书不可避免的要引用其它著作。静态链接这个过程就好比不但要装订你自己写的文章，而且也把你引用的其它人的著作也直接装订进了你的书里，这里不考虑版权问题 :)，这些工作完成后，只需要按一下订书器，一本书就制作完成啦。

在这个比喻中，你写的各个章节就好比你写的代码，引用的其它人的著作就好比使用其它人的静态库，装订成一本书就好比可执行文件的生成。

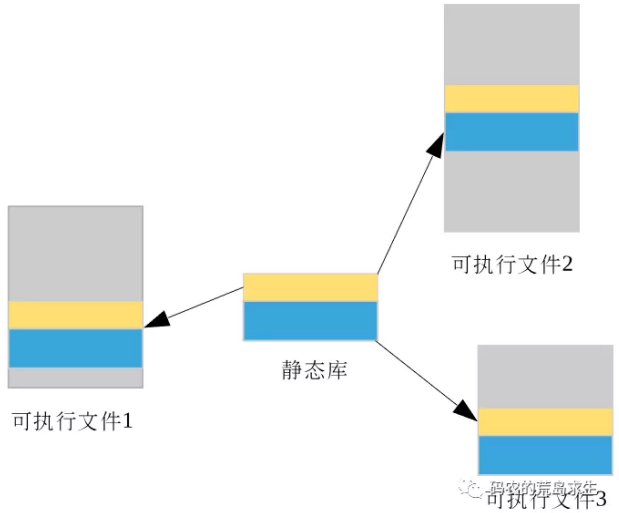
静态链接是使用库的最简单最直观的形式， 从静态链接生成可执行文件的过程中可以看到，静态链接会将用到的目标文件直接合并到可执行文件当中，想象一下，如果有这样的一种静态库，几乎所有的程序都要使用到，也就是说，生成的所有可执行文件当中都有一份一模一样的代码和数据，这将是对硬盘和内存的极大浪费，假设一个静态库为2M，那么500个可执行文件就有1G的数据是重复的。如何解决这个问题呢，答案就是使用动态库。

## 动态库

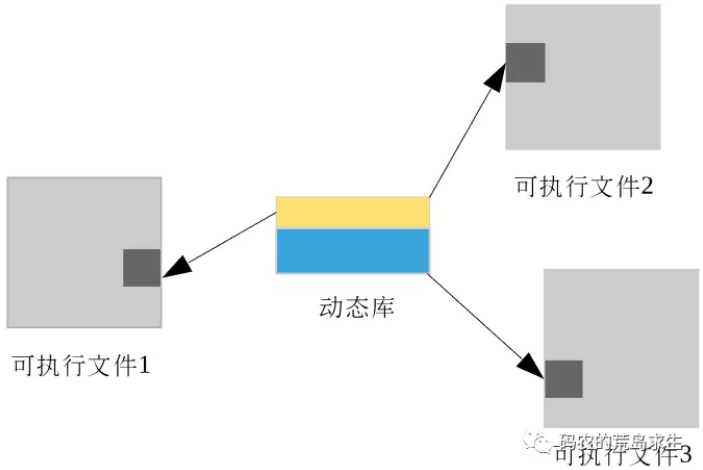
在前三小节中我们了解了静态库、静态链接以及使用静态链接下可执行文件是如何生成的。接下里我们讲解一下动态库，那么什么是动态库?

动态库(Dynamic Library)，又叫共享库(Shared Library)，动态链接库等，在Windows下就是我们常见的大名鼎鼎的DLL文件了，Windows系统下大量使用了动态库。在Linux下动态库是以.so为后缀的文件，同时以lib为前缀，比如进行数字计算的动态库Math，编译链接后产生的动态库就叫做libMath.so。从名字中我们知道动态库也是库，本质上动态库同样包含我们已经熟悉的代码段、数据段、符号表。只不过动态库的使用方式以及使用时间和静态库不太一样。

在前面几个小节中我们知道，使用静态库时，静态库的代码段和数据段都会直接打包copy到可执行文件当中，使用静态库无疑会增大可执行文件的大小，同时如果程序都需要某种类型的静态库，比如libc，使用静态链接的话，每个可执行文件当中都会有一份同样的libc代码和数据的拷贝，如图所示，动态库的出现解决了此类问题。




动态库允许使用该库的可执行文件仅仅包含对动态库的引用而无需将该库拷贝到可执行文件当中。也就是说，同静态库进行整体拷贝的方式不同，对于动态库的使用仅仅需要可执行文件当中包含必要的信息即可，为了方便理解，你可以将可执行文件当中保存的必要信息仅仅理解为需要记录动态库的名字就可以了，如图所示，同静态库相比，动态库的使用减少了可执行文件的大小。



从上面这张图中可以看出，动态库的使用解决了静态链接当中可执行文件过大的问题。我们在前几节中将静态链接生成可执行文件的过程比作了装订一本书，静态链接将引用的其它人的著作也装订到了书里，而动态链接可以想象成作者仅仅在引用的地方写了一句话，比如引用了《码农的荒岛求生》，那么作者就在引用的地方写上“此处参考《码农的荒岛求生》”，那么读者在读到这里的时候会自己去找到码农的荒岛求生这本书并查找相应的内容，其实这个过程就是动态链接的基本思想了。

到这里我们就可以回答之前提到过的问题了，helloworld程序中的printf函数到底是在哪里定义的，答案就是该函数是在libc.so当中定义的，Linux下编译链接生成可执行文件时会默认动态链接libc.so(Windows下也是同样的道理)，使用ldd命令就会发现每个可执行文件都依赖libc.so。因此虽然你从没有看到过printf的定义也可以正确的使用这个函数。

接下来我们讲解一下动态链接。

《彻底理解链接器：四，库与可执行文件的生成》，欢迎关注微信公众号，码农的荒岛求生，获取更多内容。

收录于话题 #链接器 6

‹ 上一篇

下一篇 ›

彻底理解链接器：二，符号决议
彻底理解链接器：四，库与可执行文件的生成