call指令开始于节偏移0xe的地方，包括1字节的操作码0xe8，后面跟着的是对目标sum的32位PC相对引用的占位符。

相应的重定位条目r由4个字段组成：

```
r.offset = 0xf
r.symbol = sum
r.type   = R_X86_64_PC32
r.addend = -4
```

这些字段告诉链接器修改开始于偏移量0xf处的32位PC相对引用，这样在运行时它会指向sum例程。现在，假设链接器已经确定

```
ADDR(s) = ADDR(.text) = 0x4004d0
```

和

```
ADDR(r.symbol) = ADDR(sum) = 0x4004e8
```

使用图7-10中的算法，链接器首先计算出引用的运行时地址(第7行)：

```
refaddr = ADDR(s)  + r.offset
        = 0x4004d0 + 0xf
        = 0x4004df
```

然后，更新该引用，使得它在运行时指向sum程序(第8行)：

```
*refptr = (unsigned) (ADDR(r.symbol) + r.addend - refaddr)
        = (unsigned) (0x4004e8       + (-4)     - 0x4004df)
        = (unsigned) (0x5)
```

在得到的可执行目标文件中，call指令有如下的重定位的形式：

```
4004de:  e8 05 00 00 00          callq  4004e8 <sum>      sum()
```

在运行时，call指令将存放在地址0x4004de处。当CPU执行call指令时，PC的值为0x4004e3，即紧随在call指令之后的指令的地址。为了执行这条指令，CPU执行以下的步骤：

1) 将PC压入栈中

2) PC ← PC + 0x5 = 0x4004e3 + 0x5 = 0x4004e8

因此，要执行的下一条指令就是sum例程的第一条指令，这当然就是我们想要的！

2. 重定位绝对引用

重定位绝对引用相当简单。例如，图7-11的第4行中，mov指令将array的地址(一个32位立即数值)复制到寄存器%edi中。mov指令开始于节偏移量0x9的位置，包括1字节操作码0xbf，后面跟着对array的32位绝对引用的占位符。

对应的占位符条目r包括4个字段：

```
r.offset = 0xa
r.symbol = array
r.type   = R_X86_64_32
r.addend = 0
```

这些字段告诉链接器要修改从偏移量0xa开始的绝对引用，这样在运行时它将会指向array的第一个字节。现在，假设链接器已经确定

```
ADDR(r.symbol) = ADDR(array) = 0x601018
```