“……我下面要告诉你的是，Lilliput 和 Blefuscu 这两大强国在过去 36 个月里一直在苦战。战争开始是由于以下的原因：我们大家都认为，吃鸡蛋前，原始的方法是打破鸡蛋较大的一端，可是当今皇帝的祖父小时候吃鸡蛋，一次按古法打鸡蛋时碰巧将一个手指弄破了，因此他的父亲，当时的皇帝，就下了一道敕令，命令全体臣民吃鸡蛋时打破鸡蛋较小的一端，违令者重罚。老百姓们对这项命令极为反感。历史告诉我们，由此曾发生过六次叛乱，其中一个皇帝送了命，另一个丢了王位。这些叛乱大多都是由 Blefuscu 的国王大臣们煽动起来的。叛乱平息后，流亡的人总是逃到那个帝国去寻救避难。据估计，先后几次有 11 000 人情愿受死也不肯去打破鸡蛋较小的一端。关于这一争端，曾出版过几百本大部著作，不过大端派的书一直是受禁的，法律也规定该派的任何人不得做官。”（此段译文摘自网上蒋剑锋译的《格利佛游记》第一卷第 4 章。）

在他那个时代，Swift 是在讽刺英国（Lilliput）和法国（Blefuscu）之间持续的冲突。Danny Cohen，一位网络协议的早期开创者，第一次使用这两个术语来指代字节顺序[24]，后来这个术语被广泛接纳了。

对于大多数应用程序员来说，其机器所使用的字节顺序是完全不可见的。无论为哪种类型的机器所编译的程序都会得到同样的结果。不过有时候，字节顺序会成为问题。首先是在不同类型的机器之间通过网络传送二进制数据时，一个常见的问题是当小端法机器产生的数据被发送到大端法机器或者反过来时，接收程序会发现，字里的字节成了反序的。为了避免这类问题，网络应用程序的代码编写必须遵守已建立的关于字节顺序的规则，以确保发送方机器将它的内部表示转换成网络标准，而接收方机器则将网络标准转换为它的内部表示。我们将在第 11 章中看到这种转换的例子。

第二种情况是，当阅读表示整数数据的字节序列时字节顺序也很重要。这通常发生在检查机器级程序时。作为一个示例，从某个文件中摘出了下面这行代码，该文件给出了一个针对 Intel x86-64 处理器的机器级代码的文本表示：

```
4004d3:   01 05 43 0b 20 00         add    %eax,0x200b43(%rip)
```

这一行是由反汇编器（disassembler）生成的，反汇编器是一种确定可执行程序文件所表示的指令序列的工具。我们将在第 3 章中学习有关这些工具的更多知识，以及怎样解释像这样的行。而现在，我们只是注意这行表述的意思是：十六进制字节串 01 05 43 0b 20 00 是一条指令的字节级表示，这条指令是把一个字长的数据加到一个值上，该值的存储地址由 0x200b43 加上当前程序计数器的值得到，当前程序计数器的值即为下一条将要执行指令的地址。如果取出这个序列的最后 4 个字节：43 0b 20 00，并且按照相反的顺序写出，我们得到 00 20 0b 43。去掉开头的 0，得到值 0x200b43，这就是右边的数值。当阅读像此类小端法机器生成的机器级程序表示时，经常会将字节按照相反的顺序显示。书写字节序列的自然方式是最低位字节在左边，而最高位字节在右边，这正好和通常书写数字时最高有效位在左边，最低有效位在右边的方式相反。

字节顺序变得重要的第三种情况是当编写规避正常的类型系统的程序时。在 C 语言中，可以通过使用强制类型转换（cast）或联合（union）来允许以一种数据类型引用一个对象，而这种数据类型与创建这个对象时定义的数据类型不同。大多数应用编程都强烈不推荐这种编码技巧，但是它们对系统级编程来说是非常有用，甚至是必需的。

图 2-4 展示了一段 C 代码，它使用强制类型转换来访问和打印不同程序对象的字节表示。我们用 typedef 将数据类型 byte_pointer 定义为一个指向类型为“unsigned