理解字节序

作者： 阮一峰

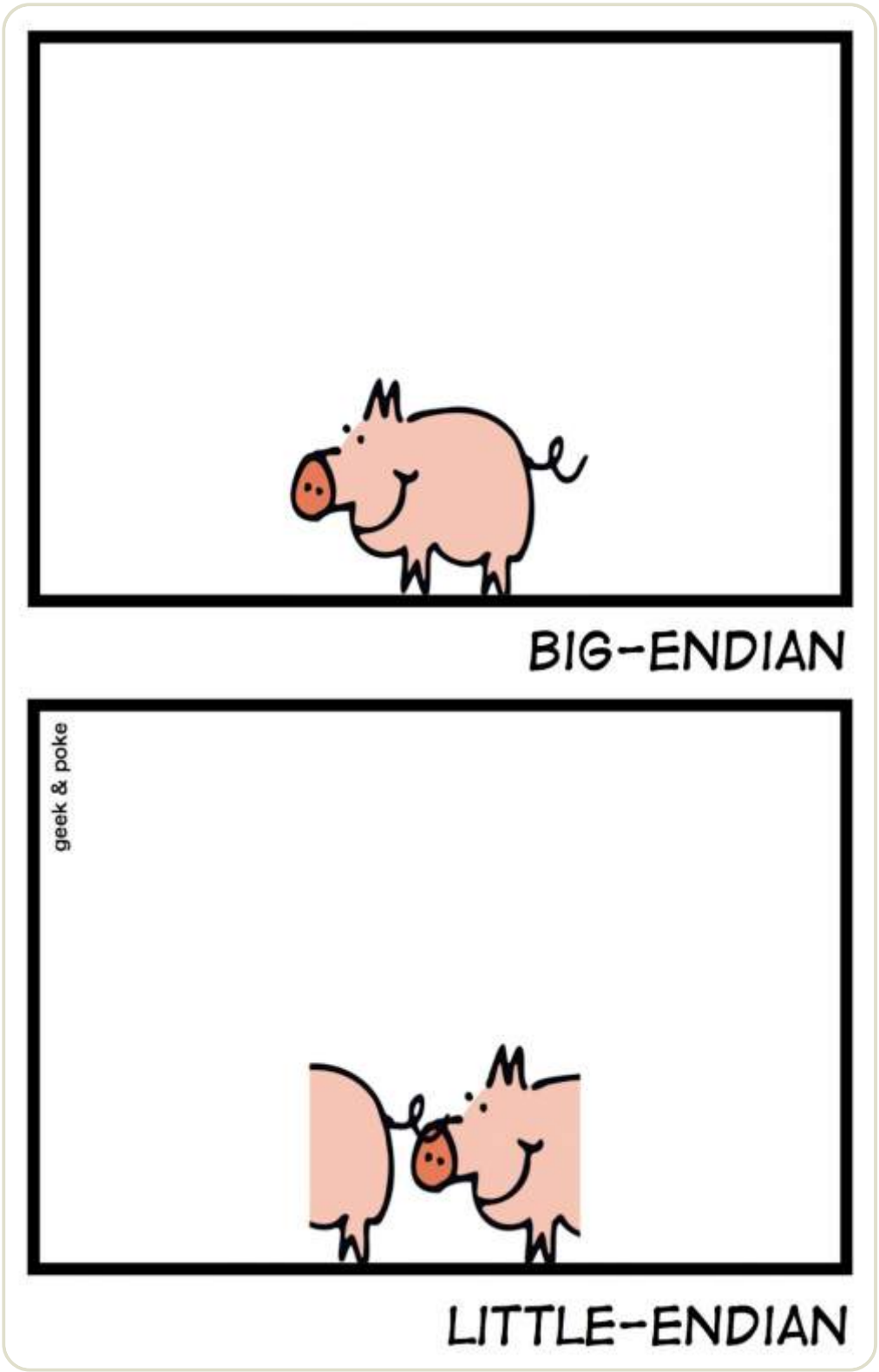
日期： 2016年11月22日

1.

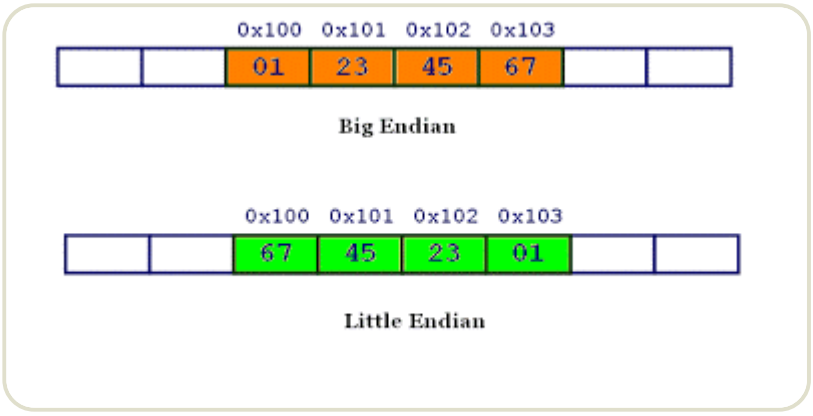
计算机硬件有两种储存数据的方式：大端字节序（big endian）和小端字节序（little endian）。

举例来说，数值 0x2211 使用两个字节储存：高位字节是 0x22 ，低位字节是 0x11 。

- **大端字节序**：高位字节在前，低位字节在后，这是人类读写数值的方法。
- **小端字节序**：低位字节在前，高位字节在后，即以0x1122形式储存。



同理， 0x1234567 的大端字节序和小端字节序的写法如下图。



2.

我一直不理解，为什么要有字节序，每次读写都要区分，多麻烦！统一使用大端字节序，不是更方便吗？

上周，我读到了一篇文章，解答了所有的疑问。而且，我发现原来的理解是错的，字节序其实很简单。

3.

首先，为什么会有小端字节序？

答案是，计算机电路先处理低位字节，效率比较高，因为计算都是从低位开始的。所以，计算机的内部处理都是小端字节序。

但是，人类还是习惯读写大端字节序。所以，除了计算机的内部处理，其他的场合几乎都是大端字节序，比如网络传输和文件储存。

4.

计算机处理字节序的时候，不知道什么是高位字节，什么是低位字节。它只知道按顺序读取字节，先读第一个字节，再读第二个字节。

如果是大端字节序，先读到的就是高位字节，后读到的就是低位字节。小端字节序正好相反。

理解这一点，才能理解计算机如何处理字节序。

5.

字节序的处理，就是一句话：

> "只有读取的时候，才必须区分字节序，其他情况都不用考虑。"

处理器读取外部数据的时候，必须知道数据的字节序，将其转成正确的值。然后，就正常使用这个值，完全不用再考虑字节序。

即使是向外部设备写入数据，也不用考虑字节序，正常写入一个值即可。外部设备会自己处理字节序的问题。

6.

举例来说，处理器读入一个16位整数。如果是大端字节序，就按下面的方式转成值。

```
x = buf[offset] * 256 + buf[offset+1];
```

上面代码中， buf 是整个数据块在内存中的起始地址， offset 是当前正在读取的位置。第一个字节乘以256，再加上第二个字节，就是大端字节序的值，这个式子可以用逻辑运算符改写。

```
x = buf[offset]<<8 | buf[offset+1];
```

上面代码中，第一个字节左移8位（即后面添8个 0 ），然后再与第二个字节进行或运算。

如果是小端字节序，用下面的公式转成值。

```
x = buf[offset+1] * 256 + buf[offset];
```

32位整数的求值公式也是一样的。

```
/* 大端字节序 */
i = (data[3]<<0) | (data[2]<<8) | (data[1]<<16) | (data[0]<<24);

/* 小端字节序 */
i = (data[0]<<0) | (data[1]<<8) | (data[2]<<16) | (data[3]<<24);
```

（完）

文档信息

- 版权声明：自由转载-非商用-非衍生-保持署名（创意共享3.0许可证）
- 发表日期： 2016年11月22日